CLUSTER AND CLOUD COMPUTING ASSIGNMENT 2
GROUP 43

BIG DATA ANALYTICS ON THE CLOUD

Yiqing Song
Department of Data Science
Student ID 1344675
yiqsong@student.unimelb.edu.au

Keti Zhang
Department of Data Science
Student ID 1401706
keti.zhang@student.unimelb.edu.au

Zijuan Lin
Department of Data Science
Student ID 1372754
zijuanl@student.unimelb.edu.au

Xuan Ji
Department of Data Science
Student ID 1405130
xuji3@student.unimelb.edu.au

Hantong Li
Department of Information Technology
Student ID 1399648
hantong.li@student.unimelb.edu.au

May 2024

Contents

1 Introduction

In this project, our group developed a data analysis system based on Twitter, climate, and environmental data using a cloud computing platform called the Melbourne Research Cloud provided by the university of Melbourne. Using cloud computing technology and allocating computing resources allows our system to analyze air pollution in different regions. At the same time, the project supports the prediction of the mental health status of the residents by combining real-time acquisition of environmental data with historical Twitter data. The interactive front-end design allows users to access important information through intuitive graphical information while reducing the complexity and cost of analyzing results.

This efficient, well-built system uses the latest cloud technologies to undertake large-scale data management with Kubernetes, Fission, and Elasticsearch. To be precise, the system can integrate several datasets from SUDO, Twitter, EPA, and BoM sources to perform analyses and visualize several scenarios. Some of the primary objectives are to develop a data ingestion pipeline across Fission, where streaming data processing and storage would be stored in Elasticsearch, local uploads for storage of static data in Elasticsearch, and a Jupyter Notebook front-end that would be interactive and easy on the eyes. Scenarios can occur due to the change in weather, the citizen's mental health, and the changing weather and air pollutants.

The expected results from this project include a scalable and reusable big data analytics platform on which citizens can visualize climate change in their city and receive potential alerts for health monitoring under different climatic conditions.

2 System Architecture

2.1 Overall Architecture

In Figure 1 ,the cloud computing system is initialized as the client devices connect through the Unimelb VPN to reach the MRC cloud. The clients then establish connections over SSH to the Bastion Node, acting as the secure entry of this cloud computing system. The next step is to distribute instances in the system: one for the Bastion Node, one for the Master Node, and three for ElasticSearch and components of it. Each volume is assigned to each ElasticSearch Working Node and Kibana Working Node. The ElasticSearch Working Nodes, together with the Kibana Working Nodes, are assigned with one volume each. The Master Node is assigned with an 8GB volume, while the ElasticSearch Working Nodes have 100GB volumes for database storage.

The streaming data pipeline for this system involves setting up of functions in Fission. These functions are set up to be invoked on a time trigger, which triggers the data fetching process—this way, data is fetched from the streaming APIs or URLs like EPA and BoM and securely saved into ElasticSearch. The data stream ensures that the system keeps updated with the latest data for analysis and visualization. Everything set up will be

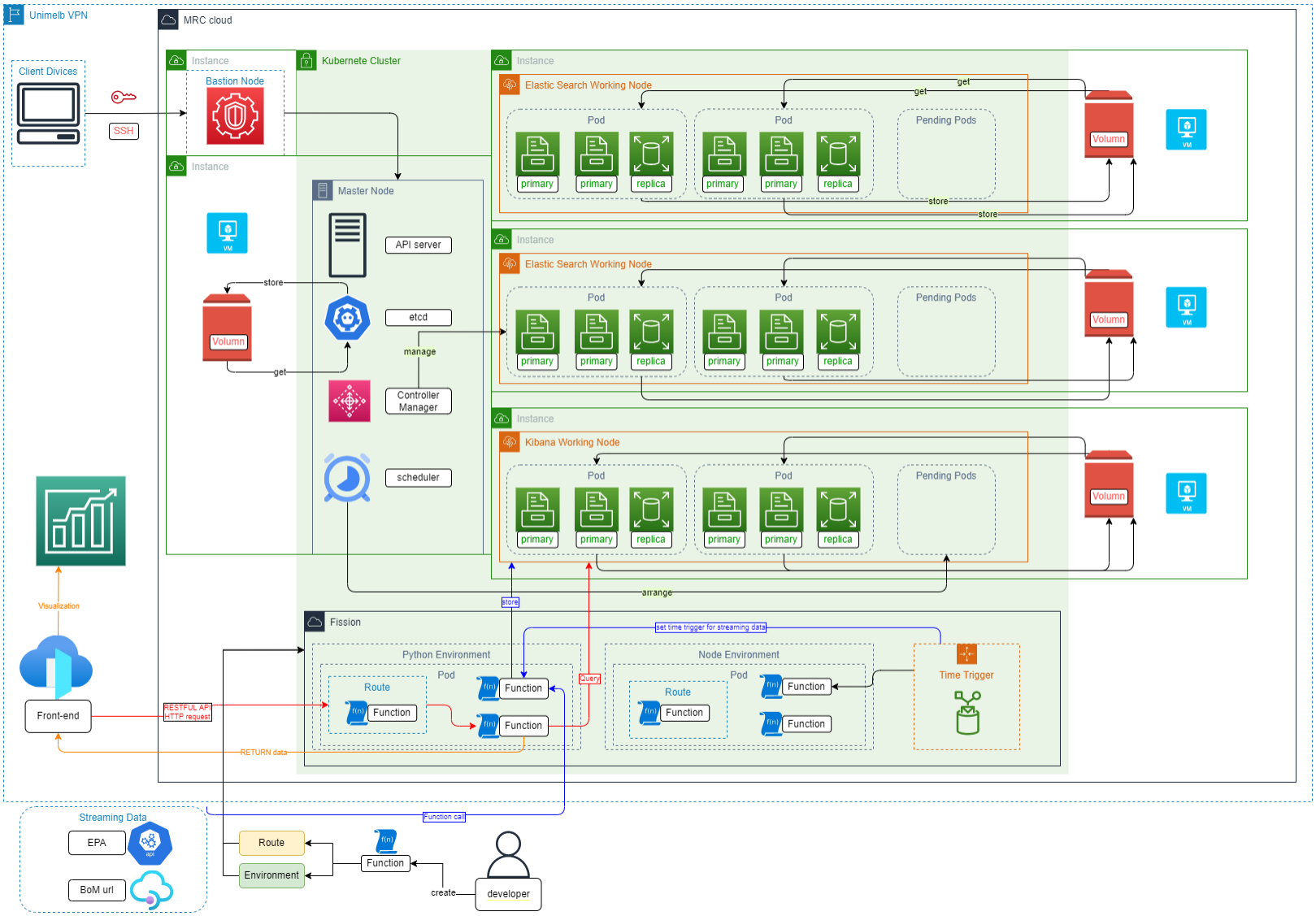

Figure 1: Overall Structure

interacted with through the front end using RESTful API HTTP. The front end actually interacts with the setup by making RESTful API HTTP requests to get the data for visualization, which actually flows back from the front end through the API routes defined in Fission and finally into the visualization tools that present the analysed data to the end users.

2.2 Use of Kubernetes

The full orchestration is set up by means of Kubernetes, which orchestrates and manages the deployment, scaling, and operation of containerized applications running on a cluster. This orchestrator makes it possible to use Kubernetes to achieve the desired state of high availability and reliability through automated monitoring of node and pod health. Workloads are rescheduled where necessary.

This ability of the cloud-based solution was further reinforced once Elasticsearch and Fission were introduced within the Kubernetes ecosystem, enlisting the strengths of both technologies. Below, this section elaborates on the use of Elasticsearch and Fission in conducting efficient data management, real-time processing, and scalable deployment of microservices.

Efficient Way for Data Storage and Retrieval: Elasticsearch, embedded in the core of the mainstream architecture, serves the data input and retrieval from this massive volume of data. The Elasticsearch nodes working in this architecture have been deployed as containers in the Kubernetes cluster. Thus, the activities, including indexing and managing the data, must maintain good performance and high scalability.

High Availability: It is defined to have multiple Elasticsearch working nodes, each having primary and replica pods. It is the job of Kubernetes to reschedule these pods if there are failures related to nodes. It does not, however, prevent service downtime.

Scalability: Elasticsearch can be horizontally scaled with the orchestration capabilities of Kubernetes. New instances of Elasticsearch nodes can be easily brought up or down based on demand, in order to ensure the right usage of resources and responsiveness to varying loads.

Automated Management: Kubernetes will automate the deployment of Elasticsearch nodes, scale them, and manage the nodes to handle tasks regarding rolling updates, rollbacks, and health checks to increase performance consistency and decrease human involvement.

3 Data Sources and Processing

In this section, the datasets used in this project are introduced in detail, along with how they were processed and managed. By introducing the data structure and containment, the reader can understand the details of the dataset and how this project connects to real-life, real-world data.

3.1 Dataset Descriptions

The datasets involved in this data analysis project, their sources, their structure, and the methods used to store and manage them in Elasticsearch are described in the following sections. The datasets covered include

3.1.1 SUDO

The pollen datasets that come from SUDO contain pollen concentration data located in four different regions of Australia (Campbeltown in New South Wales, Canberra in Australian Capital Territory, Parkville in Victoria, and Locklear in Queensland). Our dataset is pollenized by region, with each region's data stored separately in a separate JSON file with the same data structure. Each JSON file contains a FeatureCollection of geographic points, with each feature representing a specific location within the region. The metadata for each feature includes pollen concentration information (e.g., Asteraceae pollen and other types of pollen), the number of days of data collection, and the specific coordinates of the location.

3.1.2 EPA

This analysis system uses a dynamic dataset the Environmental Protection Authority provides that continually retrieves data from its API . The dataset contains many environmental data, including coordinates (latitude and longitude), timestamps of data collection, and air pollution indices, including PM2.5 and particulate matter, temperature, wind speed, and humidity.

3.1.3 BoM

This is meteorological data in dynamical format provided by the Bureau of Meteorology, providing information through the API regularly; this dataset involves the correlation between wind patterns, pollen dispersal, and weather factors related to air pollutants. The meteorological variables include temperature, relative humidity, wind direction, rainfall, and other parameters. Meanwhile, all data comprises metadata related to geographic coordinates, a data collection timestamp, and a detailed measurement of various weather conditions.

3.1.4 Other Source

VIC DATA - Microclimate Sensor Locations
The dataset named sensor in our database includes the Microclimate Sensor Locations dataset from VIC DATA, which provides comprehensive information on the locations of microclimate sensor devices installed throughout Melbourne, along with the climate data they captured in 2019. This static dataset includes critical metadata such as device ID, installation time, and various climate measurements including precipitation, wind speed, wind direction, gust speed, vapor pressure, atmospheric pressure, relative humidity, and air temperature. Additionally, the dataset provides geographic coordinates (latitude and longitude) for each sensor, enabling detailed spatial analysis and mapping for the project.

Spartan - Twitter
The Twitter dataset used in this project is a large-volume static JSON dataset from SPARTAN's project for analyzing social media sentiment and word usage, with an original

dataset size of about 120GB. This dataset contains comprehensive information about tweets, including metadata, author ID, creation time, tweet content and sentiment analysis, and some data containing geographic information.

3.2 Data Preprocessing

The 120 GB of tweets data is downloaded from Spartan, we take 5% from the sample to handle this very big size. We believe it to be representative enough for meaningful analysis, but it reduces processing load. We would apply this approach to two scenarios: sentiment analysis and text content analysis.

We therefore extract the user IDs, with geo data making it possible to geographically correlate the source and convert the geo data into random so that the regions will be in correlation with the regions covered by EPA data to allow geographically correlated analysis of tweets together with environmental data. Information from the created tweets will be analyzed using timestamps, the created_at field.

In the two JSON files that have been described, the data is arranged in the two types: sentiment data and the data with texts. The sentiment data file is attributed to user IDs, geolocation, timestamp, and sentiment scores of tweets. Separating the sentiment data from the text data helps to avoid one index becoming too large and slowing down query performance. The file containing the text data includes the user IDs, geolocations, timestamps, and the text content of the tweets. Such a kind of separation guarantees both the efficiency and scalability of the system as the amount of data increases.

This stream data is further processed with a number of scripts that handle data extraction from samples and transformation. The data extraction script reads the 120 GB part of the Twitter file and extracts a 5% sample. The sampled data is further processed per sentiment analysis by the script with the processing logic. The data structuring script structures the data into the two JSON files and makes sure it has the necessary columns to uphold the information needed in each file for the two analyses. With the data divided into two JSON files, it is quite flexible to easily undertake sentiment and text content analysis on huge sets without any limitations of the original data size.

4 Analytical Scenarios

In this section, we will explore in detail the various analytical options used in this project to investigate the topic of air pollution on the mood levels of the population. This section explores the relationship between environmental factors and how these factors ultimately affect human life and emotions. All of the analytical work in this project is based on datasets, including pollen data, air pollutant data, and Twitter mood data collected from multiple data sources.

4.1 Scenario Descriptions

In this project, our theme is the analysis of the multidimensional impact of environmental factors on human life and emotions.By analyzing pollen data, air pollutant data and Twitter sentiment data, we revealed the complex relationship between environmental factors and residents' emotional expressions.

Scenario 1 is to analyze the influence of weather factors on pollen concentration and establishing the association between pollen concentrations and weather parameters such as temperature, humidity, wind speed, and rainfall. The first step in data visualization was to amalgamate the predicted pollen data with the recent weather data in a single table for further data visualization. Our research group first established what the prediction model of historical pollen data and weather data looked like in order to predict recent pollen concentration. Then, we combined the predicted pollen data with the recent weather data into a single table for further analysis and visualization. In the data visualization process, the relationship between each weather element and pollen concentration data was first preliminarily analyzed using the Pearson correlation coefficient. Then the trends and correlations between weather factors and pollen concentrations were demonstrated by visualization tools such as time series plots, scatter plots and bubble plots. Through the above image analysis, we expect to be able to identify the effects of weather factors such as temperature, humidity, wind speed and precipitation on pollen concentration. This will also help us to get a broad understanding of how seasonal changes and extreme weather events, for example, affect the level of pollen concentration in the air.

Scenario 2 is to study the impact of air pollution on the mood of the population. Due to the limited amount of data and uneven time distribution, we disrupted the timing of the air pollutant data and the Twitter data to ensure that the two types of data occurred during the same time period. At the same time, due to the mismatch between the two data volumes, the volume of Twitter data was much larger than that of air pollutants, which made the analysis difficult, and so the pollutant type was uniformly assumed to be PM2.5. First, we created an interactive map integrating the air pollution data and the Twitter data in order to gain a preliminary understanding of the impact of air pollution on people's emotional states on a geographic level. We then created a Time Series, Scatter Plot, Bar Chart, Box Plot, and Heatmap to further explore the relationship. In this scenario, we expect to be able to explain the extent to which different pollutants affect the public's mood and to help formulate recommendations for the public's physical and mental health in relation to air pollution.

Scenario 3 analyzes the sentiment of Melbourne residents on Twitter. We use the Twitter data to generate word clouds to visualize and analyze trending topics and public sentiment on social media. The output of Scenario 3 can be analyzed in combination with the previous two scenarios, such as observing the emotional state and expression of residents during times of high pollen counts, or monitoring the emotional dynamics of residents in real time during times of high air pollution.

4.2 Results Presentation

4.2.1 Scenario 1

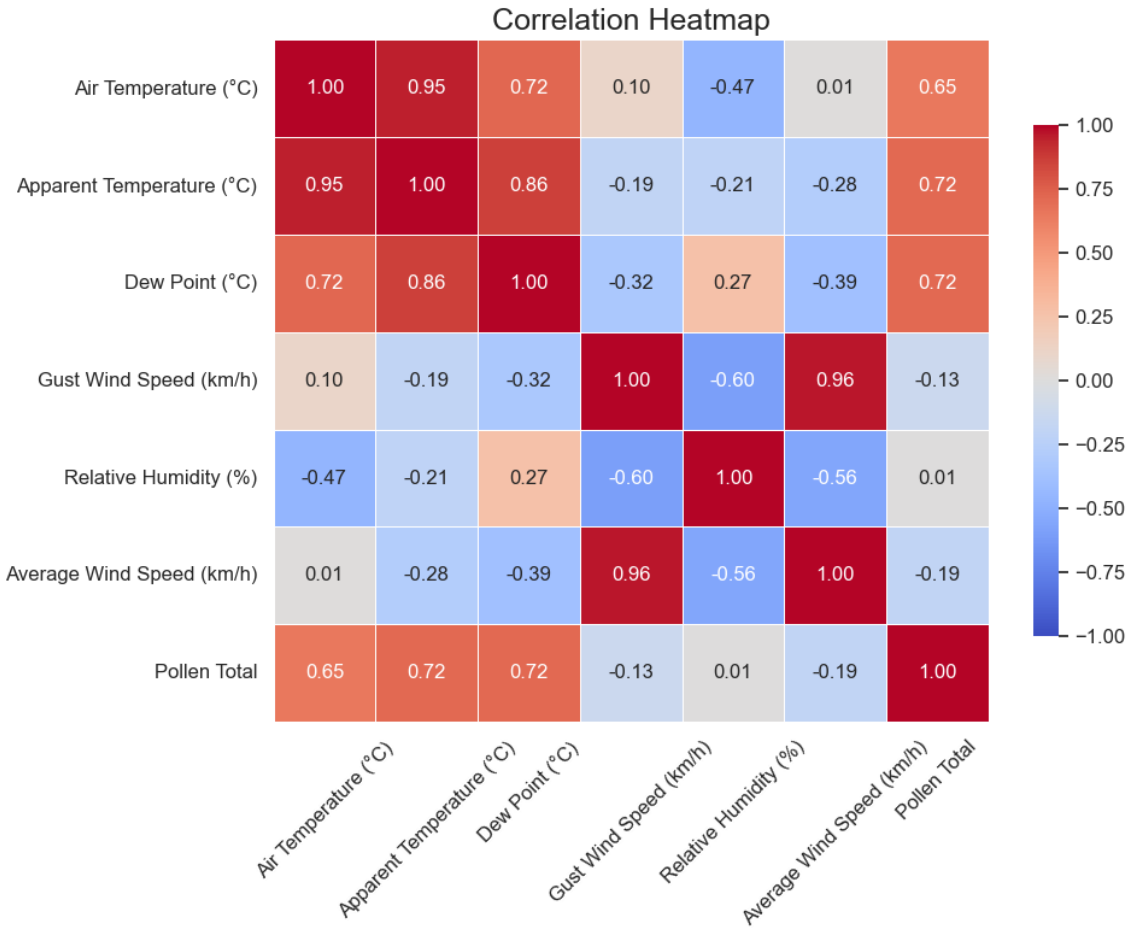

Figure 2: Heat map in Scenario 1

Figure 2 reveals the correlation between different meteorological elements and total pollen concentration. The heat map shows that among all meteorological factors considered, temperature and pollen concentration show the most significant positive correlation. On the other hand, there is a slight negative correlation between wind speed and pollen concentration. However, it should be pointed out that this negative correlation is weaker than the positive correlation of temperature. It is worth noting that the correlation between relative humidity and pollen concentration is almost negligible. It can be seen that changes in humidity have limited impact on pollen levels and are not the main controlling factor compared to temperature and wind speed.

In figure 3, there are significant fluctuations and outliers in the total pollen concentration data. Specifically, there are multiple high-value abnormal points outside the upper whisker of the data, which means that during a specific period of time, the pollen concentration will have an obvious peak or abnormal increase. The occurrence of this

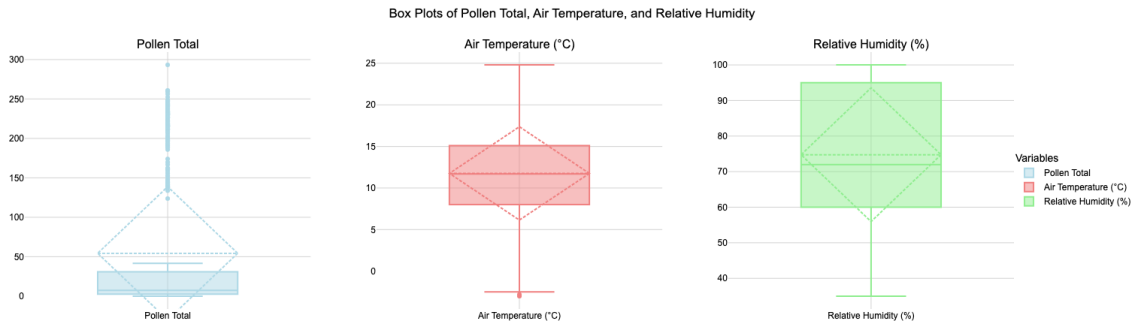

Figure 3: Box Plot in Scenario 1

phenomenon may be closely related to seasonal changes in plant growth and specific weather conditions. In contrast, the distribution of temperature and relative humidity data is relatively concentrated and there are no obvious outliers, which indicates that the changes in both remain relatively stable during the study period. Temperature and humidity are important meteorological elements and the focus of our research. Their stable data distribution is conducive to accurately understanding the relationship between them and pollen concentration.

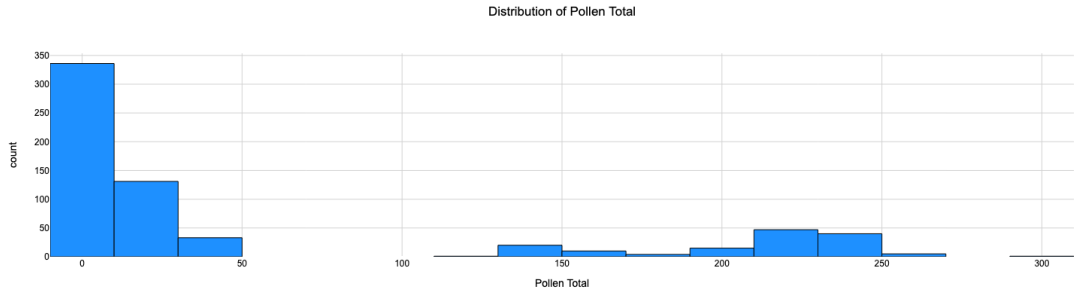

Figure 4: Bar Chart in Scenario 1

It can be seen from figure 4 that the pollen concentration is low most of the time, concentrated in the range of 0 to 50. Moderate and high concentrations of pollen are rare but may occur under certain weather conditions.

In Figure 5, we can see that pollen concentration varies by location. Rocklea has the highest and most volatile pollen concentrations, while Canberra and Parkville have lower concentrations. Campbelltown is central. Rocklea showed significant fluctuations, while pollen concentrations at other sites were relatively stable.

The Figure 6 shows that pollen concentration is lower at lower temperatures, increases

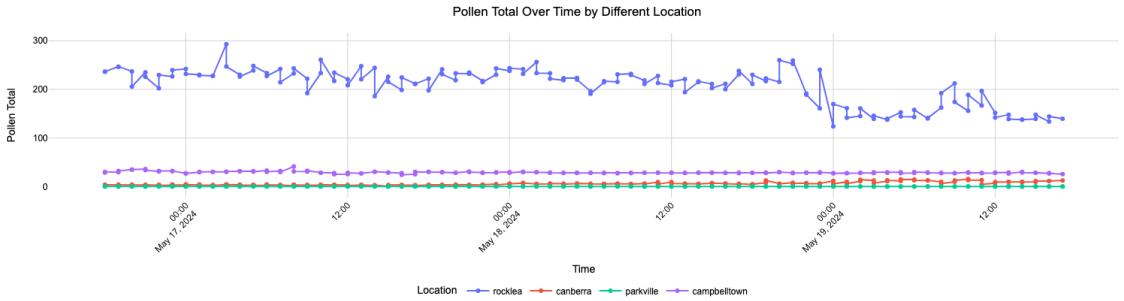

Figure 5: Time Series Plot for Pollen in Scenario 1

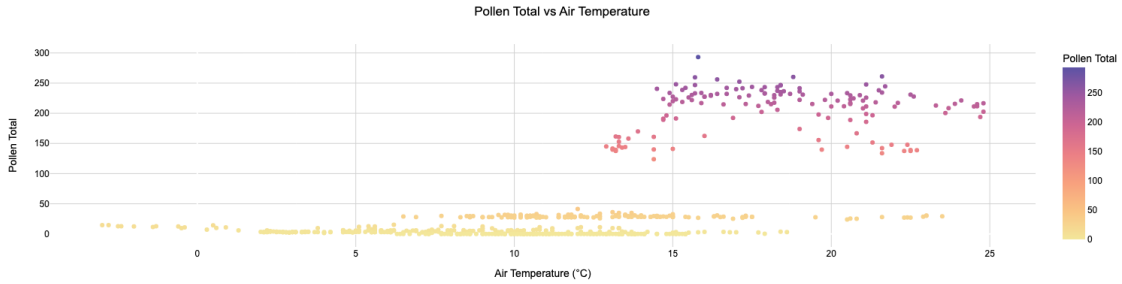

Figure 6: Scatter Plot of Air Temperature vs Pollen in Scenario 1

significantly at higher temperatures, and levels off at higher temperatures. Warmer temperatures may encourage plant flowering and pollen release.

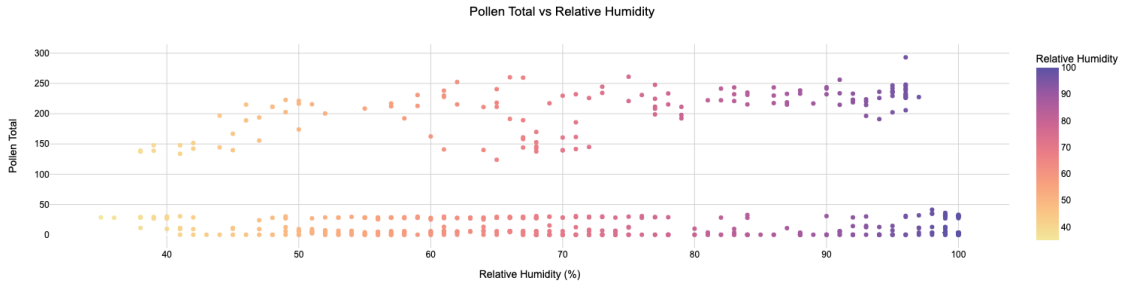

Figure 7: Scatter Plot of Humidity vs Pollen in Scenario 1

The figure 7 shows that pollen concentration increases significantly at moderate humidity but decreases above 60% humidity. Moderate humidity may aid pollen release and spread, while higher humidity may cause pollen to settle.

In figure 8, we can observe that the positive relationship between gust wind speed and

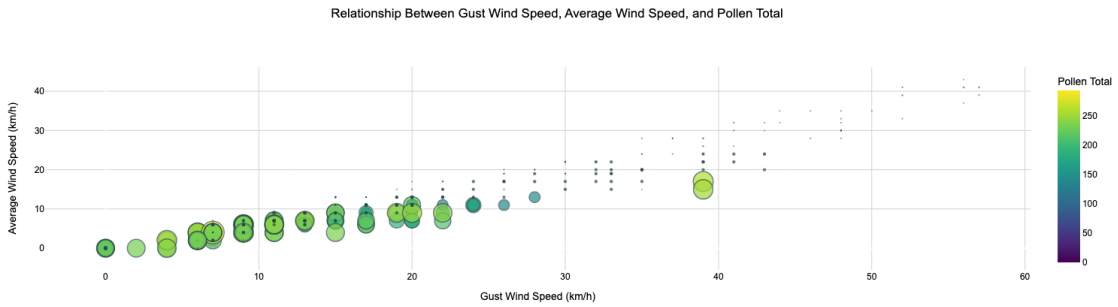

Figure 8: Bubble Plot of Wind Speed vs Pollen in Scenario 1

average wind speed. Moderate wind speeds help pollen spread, while very high wind speeds may cause pollen to settle quickly, reducing the concentration of pollen in the air.

4.2.2 Scenario 2

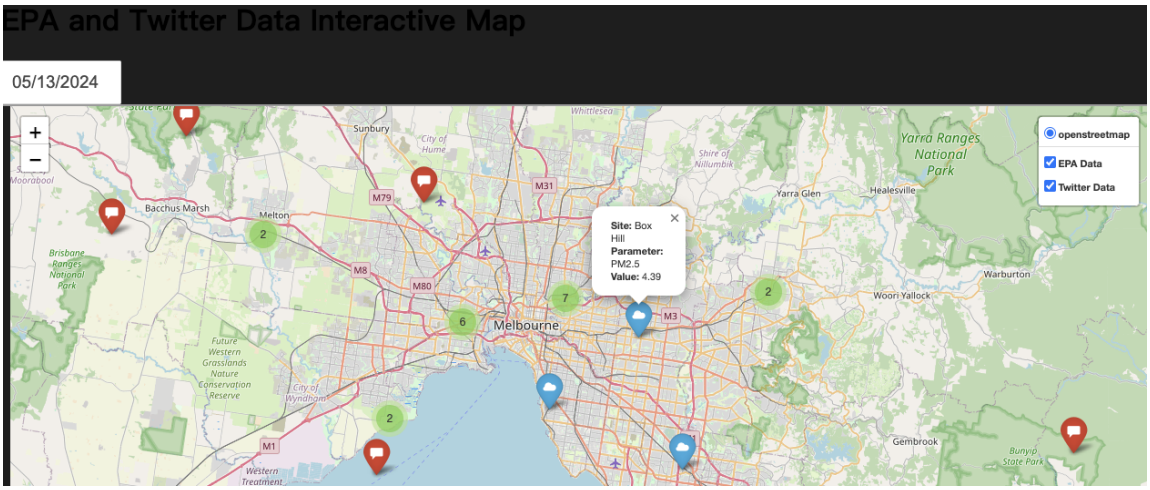

Figure 9: EPA and Twitter Data Interactive Map in Scenario 2

Through figure 9, we can initially understand the geographical correlation between air pollution and residents' emotions. For example, in areas with higher air pollution, there may be more expressions of negative emotions on social media.

Through figure 10 and figure 11, we can deeply analyze the correlation between air pollution and emotions. Scatter plot can help us observe the direct correlation between air pollution indicators and sentiment scores, revealing the correlation between the two. A time series chart can show how this relationship changes over time, helping us identify trends and possible anomalies within a specific time period.

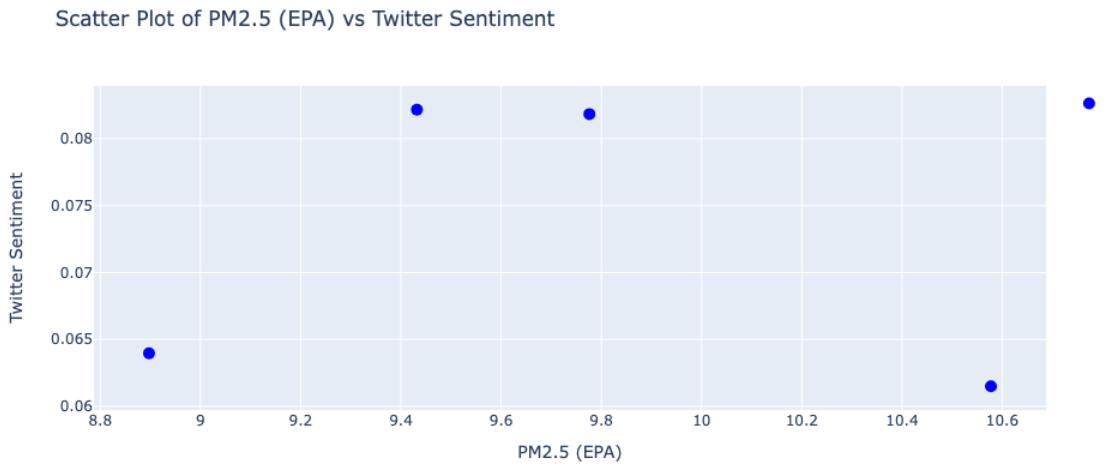

Figure 10: Scatter Plot of PM2.5 vs Twitter Sentiment in Scenario 2

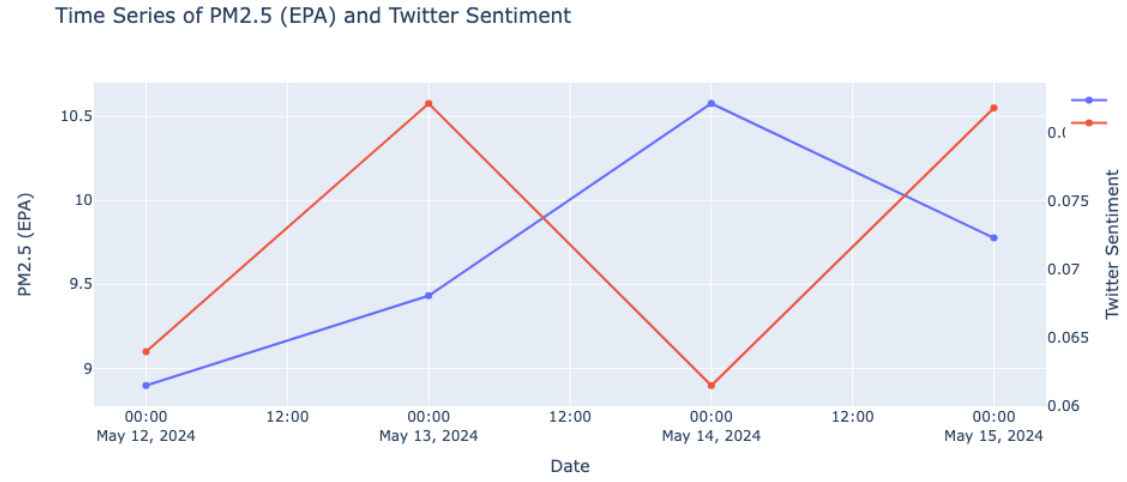

Figure 11: Time Series Plot for PM2.5 & Twitter Sentiment in Scenario 2

4.2.3 Scenario 3

Word cloud charts can visually display residents' emotional states and expressions during a specific period. For example, figure 12 is a word cloud picture in December 2021. Flinders may be related to Flinders Street in Melbourne, implying that important events or activities occurred in the area in December. There has also been a lot of talk about cancellations or delays, possibly involving travel, events or other plans over the holidays. The frequent occurrence of words like love, thank, good, and happy shows that the emotions in tweets in December are generally more positive, which may be related to the joyful atmosphere of the Christmas holiday.

Figure 12: Word Cloud Chart for Twitter Sentiment in Scenario 3

5 System Implementation

5.1 System Functions

```
(ccc) ravpn-200-student-10-8-1-236:~ takashimi$ fission fn list
NAME                ENV    EXECUTORTYPE MINSCALE MAXSCALE MINCPU MAXCPU MINMEMORY MAXMEMORY SECRETS CONFIGMAPS NAMESPACE
bompollen           python poolmgr      0        0        0      0      0         0                              default
campbelltownstream  python poolmgr      0        0        0      0      0         0                              default
canberrastream      python poolmgr      0        0        0      0      0         0                              default
data-extract        python poolmgr      0        0        0      0      0         0                              default
epastreaming        python poolmgr      0        0        0      0      0         0                              default
melbournestream     python poolmgr      0        0        0      0      0         0                              default
queenslandstream    python poolmgr      0        0        0      0      0         0                              default
twitter-nlp         python poolmgr      0        0        0      0      0         0                              default
```

Figure 13: Functions of System

In this section, we introduce the main functions of our system to handle various analytical tasks.

bompollen:
Thid is the primary function for Scenario 1 which focuses on modeling the relationship between weather factors and pollen concentration.

This function first makes an API request to BoM for retrieving the latest meteorological observationsThe data is then parsed and stored in a structured format by converting date formats, handling missing values, and extracting relevant features. Meanwhile, there is a connection has been established with Elasticsearch to obtain historical weather and pollen data. The historical data was processed and merged so that only the relevant records that matched the date range and location were included.

The merged dataset trains a random forest regression model, which shows the rela-

tionship between weather factors and pollen concentrations. The predictions were formatted as JSON and then returned to the frontend for further analysis and visualization.

This feature plays a important role in Scenario 1 by accurately predicting pollen concentrations. It enabling users to understand the impact of weather on pollen levels.

data-extract:
After receiving the request, this function would check the existence of the required JSON. If it is missing, it indicates a problem exists, and the function would return an error response while extracting the query body and index name from the request.

After previous step, the function would establishes a connection to the Elasticsearch and executes a search query on the specified index. The query results are retrieved and formatted as a JSON response and would return to the front end. If errors occur during this process, the function catches the exception and returns an error response.

This function help the frontend to query data from Elasticsearch while ensuring efficient and secure data retrieval.

twiter-nlp:
The Twitter-nlp function analyzes Twitter data for natural language processing and returns the results to the frontend. The function is implemented with both of the Flask and NLTK libraries.

After the request, the function would validate the time parameters to ensure the right supported range. After that, the function sends a query to retrieve Twitter data from Elasticsearch based on the specified date range, and it gets easier. The fetched data is preprocessed by NLTK, which includes tokenizing the text, removing stop words, and lemmatizing tokens. The preprocessed text is then analyzed by counting words' frequency. As a result, the formatted JSON would be returned to the front end.

This feature helps a lot with analyzing social media sentiment and trends, which allows users to gain insight into opinions and statements on Twitter.

campbelltownstream& **canberrastream**& **melbournestream**& **queenslandstream**:
These functions fetch real-time weather data from the BoM API endpoint and stream data directly to Elasticsearch for real-time monitoring and analytics. The functions have been designed using Python and therefore conduct a series of operations to pull the most recent weather observations for the localities of Campbelltown, Canberra, Melbourne, and Queensland through the BoM API, with a reformat of date formats and missing values. They then interact with Elasticsearch and prepare the bulk payload for indexing. Finally, these batch payloads are sent for indexing into Elasticsearch, making the weather from above four places always in real-time monitoring status with the latest data information for analysis and visualization.

epastreaming:
This function retrieves real-time air pollution data from the EPA API and streams it to Elasticsearch for monitoring and analysis. It's built in Python and uses the EPA API to pull the newest air quality measurements. The response that is received processes the returned data by reformatting the date information and handling missing data. Afterwards, this opens a connection to Elasticsearch and sets up an indexing bulk payload so that data can be sent. The data is sent to Elasticsearch, which continues the process of indexing air quality data in real time, holding the data further for analysis and visualization.

5.2 UniMelb Research Cloud

The University of Melbourne Research Cloud is an essential research facility that provides scalable and virtualized computational resources. This report elaborates on pros, cons, tools, and a process to create images for deployment in the MRC.

5.2.1 Advantages

The user-friendly benefits of the MRC are that it has been intentionally made for ease of use, with very limited background knowledge in the technical field of cloud computing; therefore, one can easily create and manage virtual machines and other resources. The platform is open to collaborative development, hence allowing researchers to work on similar projects with many other researchers across the globe. Furthermore, there is greater usability as real-time tracking of the resources and applications enables users to monitor their performance and problems quickly.

Another advantage of MRC is that it has OpenStack integrated within it. OpenStack is an open-source cloud computing platform that helps in resource management for, for instance, resources like Elasticsearch and Fission. This will ensure a common and relatively strong framework of resource management, hence reducing the unpredictability from the user's point of view. Researchers manage their applications and data without bothering about infrastructure low-level technicalities. This abstraction makes the development process a lot easier, with the user worrying very little about the actual technicalities of cloud management.

5.2.2 Challenges

However, MRC also has a few challenges. A key one involves the complexity in testing during the development phase. In the case of malfunction, the response and error messages given are inadequate to pin the issue. This aspect is most often experienced in cases of node failures and trying to pinpoint the source of the problem in the development phase. When employing Fission functions, there can be remnants of resources that pile up, leading to memory issues. These remnant resources also block the system memory from being used to roll out a new package, and their cleanup is manual to keep it running

Some future tools and processes that are going to be used will make this image creation and deployment in MRC more reliable and efficient. Very easily under the MRC, one can create and manage Virtual Machine images. Users can configure VMs with all the applications and dependencies required, and even save such configurations as custom images, which can further be used over and over to spin up new instances of VMs without needing to go through the total process of setting them up, thus maintaining standard environments.

Another useful tool is snapshot management. Snapshots of VMs taken on a regular basis can capture the state of the VM at any time, which subsequently can be restored if needed. This forms a vital ability to recover data each time a node crashes or one needs to restore a previous configuration. Automated processes for backup to regularly back up data and restoration in case of failure should also be there to maintain data integrity and availability.

Monitoring and alert systems are of much importance in keeping track of the health and performance of VMs and the rest of the resources. Monitoring and logging tools such as Prometheus, Grafana, and the ELK Stack (Elasticsearch, Logstash, Kibana) offer the possibility that such robustness enables the user to identify problems in real time and solve them. Alerts can consequently be set for any anomalies that take place, like high memory or node failures, in order to take corrective action timely.

Docker and Kubernetes help to normalize node crash recovery. Docker can put applications into containers with all ease moving and deploying. Containers can easily shift between environments without deviation and will always run above any kind of infrastructure. Kubernetes now happens to be an orchestration tool that is managing these containerized applications at a large scale. It is automated for deployment, scaling, and operation, ensuring efficiency and resiliency of the application.

In conclusion, MRC brings a lot of benefits in terms of usability, collaboration, and simplified management and brings challenges in the area of testing and managing the resources. The challenges for these can be mitigated by implementing the rigidly followed monitoring, backup, and recovery process, thereby making the platform operation a seamless and time-bound process for the researchers.

5.3 Front-End Jupyter Notebook

We perform data visualization in Jupyter Notebook. Notebooks make data analysis interactive and dynamic. We use notebooks to collect, preprocess and combine all the different data sources such as weather, pollen, air quality and Twitter related topological data and put them into their respective scenarios for analysis. At the same time, in Jupyter Notebook, we also conducted visual correlation analysis and developed interactive visualization tools with the support of Matplotlib, Seaborn, Plotly and other libraries. We help explore data dynamic trends and relationships between data in more detailed ways, such as time series

charts, scatter plots, bubble charts, word clouds, etc. Within these notebooks, we also embedded the Dash application to make interactive dashboards where users can interact with the data, filter results, and dynamically visualize aspects of the data.

5.3.1 Scenario 1

Scenario-1 attempts to explore the correlation between weather conditions and pollen counts by analyzing weather data and pollen concentration levels in four cities. The first step is to draw correlation heat maps of different weather variables and pollen sums. Next are box plots and distribution plots to understand the pollen concentration status and climate differences in the four cities. We then visualized the pollen concentration and weather condition results using time series plots, scatter plots, and bubble plots to identify key weather factors that influence pollen levels.

To more visually display and analyze pollen data and its relationship to weather factors, an interactive Dash application was created that summarizes all the above charts.It is versatile, not only for existing data sets, but also scalable when processing more data. The general application is divided into three main independent modules: Pollen Distribution, Pollen Over Time and Pollen with Variables. The Pollen Distribution module allows the user to filter for a specific date range and location and observe a histogram of total pollen. The Pollen Over Time module allows users to choose what they want to plot from variables such as total pollen count, relative humidity, or air temperature. Users can also select different locations and time ranges to obtain time series plots of selected variables. The Pollen and Variables module allows users to select different locations and time ranges, as well as different meteorological variables such as humidity, air temperature, and wind speed. By viewing scatter plots of total pollen counts against these variables, users can analyze the correlation between them and gain a deeper understanding of the impact of each factor on pollen concentration.

5.3.2 Scenario 2

Scenario-2 explores the impact of air pollution levels in Melbourne on residents' mood. In this scenario, we built two Dash applications, aiming to transform complex data into clear and intuitive visual presentations while giving users flexible analysis capabilities. These applications not only present a multi-dimensional perspective on current data, but are also scalable to adapt to the growth of future data sets and the evolution of analysis needs. Through the graphical interface and human-computer interaction design, users can efficiently and conveniently explore the relationship between air quality and public sentiment.

The first is the interactive map of air pollution data and Twitter data, which integrates EPA's air quality monitoring data with Twitter data through a geographic information system (GIS), providing users with a multi-dimensional and multi-scale data exploration experience. Users can analyze the correlation between air pollution and emotions from a geographical perspective. Users can use the zoom function to independently adjust the

spatial resolution of the map, which can not only drill down detailed data for specific areas, but also grasp the overall pattern and macro trends. Through the rich variable selection menu, users can flexibly set their data preferences, such as EPA's real-time monitoring data of air pollutants or Twitter data. The time dimension can also be controlled by the user. By specifying a specific date, the spatial data distribution picture of that day can be presented. In this application, we use intuitive visual elements to mark different data types. The blue pushpin represents an EPA monitoring site, and its value represents the current pollutant concentration reading at the site; the red mark reflects the emotional expression of Twitter users corresponding to the geographical location. By integrating and presenting air quality data and public sentiment data, users can efficiently and intuitively explore and judge the potential correlation between them. Then there is the EPA and Twitter data analysis part. This application displays the Melbourne area's air pollution data (EPA) and Twitter sentiment data through multiple data visualization modules, helping users digitally understand the relationship between air pollution and sentiment more clearly. Users can select specific pollutants (such as PM2.5) for analysis. You can also select a specific date range to view pollutant data and Twitter sentiment data within that time period. The application includes Time Series plot, Scatter Plot, Bar Chart, Box Plot and Heatmap. Time series graphs are used to show the changing trends of pollutant levels and mood index over time. The correlation graph further verifies the correlation between the two, visually displaying the degree and direction of the correlation between the amount of pollutants and the mood index. The bar chart is used to compare the distribution of mood index under different pollutant amounts. Box plots are used to display the distribution and variation range of mood index under different pollutant amounts.

5.3.3 Scenario 3

Scenario-3 focuses on visualizing Twitter data using word cloud charts to capture trending topics and sentiments on the social network. Word cloud can identify the most discussed topics on social networks during the study period, while also providing insight into overall public sentiment during this period and how it fluctuated based on different events or discussions.

5.4 Back-End Fission

As previously mentioned, our backend system is architecturally designed around a serverless framework, predominantly utilizing Fission for its implementation. This potent tool has been seamlessly integrated into our Kubernetes cluster to facilitate efficient deployment and robust management. Detailed within this report, the Fission client setup in our infrastructure is strategically aligned with our objectives to boost scalability and minimize operational complexity. Fission serves as the cornerstone of our backend, crucial for achieving the performance standards we expect. It distinctly orchestrates our system into the following categories:

- **Data Streaming into Elastic Search DB (Ingress):** This section explains our methodology for recording data into the Elastic Search Database using Fission.
- **Data Queries from Frontend (Egress)**:Here, we discuss how Fission enables frontend team members to access the database seamlessly and securely through a RESTful design, bypassing traditional authorization mechanisms.
- **Data Processing:** We delve into the rationale and methods for data processing within our backend.
- **Testing:** This part outlines our strategies for testing Fission functions both pre- and post-deployment.

These four components collectively form the backbone of our scenario analyses, with nuances tailored to the specific uses of each. Therefore, we have decided to structure our exposition of the Fission backend around three real-world scenarios. For each scenario, we will elaborate on these four aspects to provide a detailed and comprehensive view of Fission's application within our system.

5.4.1 Scenario 1

For scenario 1 (4.1.1), we require the pollen data for the four cities,which is Melbourne, Queensland, New South Wales, and Canberra. We also require the streaming weather data for the above-mentioned cities from the Bureau of Meteorology (BoM). The simplified version below was realized to make it possible to carry out effective analysis:

Data Streaming
First, for static pollen data for the four cities, the JSON files have been read locally but without performing the needed pre-processing of data. This process involves a check on data validation that corresponds to the pre-set index mapping on Elasticsearch. This is going to involve looking at the structure of the JSON files to look at the field names and data types compared to the expected schema. We need to ensure the data is tailored to the Elasticsearch index mapping for the indexing and subsequent searching to be performed accurately. However, after validation, it shows that it will be a very tedious and unproductive process to index the data to individual documents. Then, we decided to send the bulk request with the bulk API given by Elasticsearch. In this way, multiple indexing requests can be made in a single call to the API, which makes the process of indexing much faster. These batches reduce the overhead of submitting each request individually and, at the end, result in efficient indexing.

We followed a similar preprocessing mechanism to match the needed index mapping data in the Elasticsearch of streaming weather data from BoM, which has to be updated daily. Since the data is very dynamic in nature, we had to perform this operation automatically on a regular basis. The latter is enabled inside the Fission function package by opening a route in which updating these processes is activated by the timer. As these updating points are spaced out by the timer, this will avoid the simultaneous sending of data from all data sources, which would create heavy loads on the Elasticsearch nodes,

causing the system to be constantly unstable or even crash. These updating points are spaced at different sets of times, balancing the load throughout the nodes and thereby keeping the system stable and with good performance.

Data Queries
We have architected the backend to be RESTful so that the front end can fetch data from various regions quickly and more efficiently. It allows the front end to make posting requests with the region name as a parameter. The back end would respond to the request using Flask, gather required data from Elasticsearch, and return the data. Flask would receive a POST request, process it, and extract the region name for which it would query Elasticsearch so that the information retrieved is about the specified region, which reduces data transfer and increases response time. We used Fission functions to quickly analyze the data further before results are sent to the front end. In this way, as data analysis gets done on the backend, there is a lower quantum of the work that has to be done on the front end for data processing to be done extensively. This not only lowers the processing load at the client side but also speeds up the data retrieval procedure, thus enhancing user experience.

Data Processing
We aligned the pollen and practical data in both timing and locations. First, we retrieved historical weather data used in previous studies on environmental correlates of pollen data, and then we queried pollen data from the Elasticsearch index. The two datasets were cleaned and processed to secure quality and ascertain that the data is consistent. After cleaning and pre-processing the data, we integrated the two datasets by aligning the data points based on the same period and locations. This provided us with one integrated dataset that would build pollen levels with corresponding weather conditions and a solid background for further analysis. With the integrated dataset ready, we turn to apply machine learning techniques to train the model. Next, we used machine learning for the identification of any such classification which would, in due course, allow us to establish a relationship between the pollen concentration and weather conditions. In the training of the model, cross-validation was used as a technique for the fine-tuning of the model parameters for the optimization of performance, so that the accuracy and reliability of the model output are high.

It essentially trained a machine learning model with real BoM weather data to predict, from this data, what the future pollen count might be. The processing will avail real-time weather data for dynamic forecasting, which will in turn give current information concerning the presence of pollen to the users.

Testing
It was very challenging during the testing phase to manage to fission functions in the cluster and create routes for frontend access. Local testing was also a task because we could not really tell if the users were right on uploading the right data. We have addressed some of these challenges with some measures that, in fact, led to the needed accuracy and dependability in our system. For the local testing, we had to validate the data returned for

every region. In this regard, it was incumbent upon us to do a massive validation check that the data we fetched from Elasticsearch was similar to our expectation. With this, we would, of course, go to the next steps with the assurance that data handling processes were correct. We implemented error handling in our functions even before setting the cluster management. We used try-catch blocks to catch any exceptions. What it does is catch any errors that can possibly come just in case the front end provides invalid values. As soon as it has picked one, our system returns a useful error message that will help us to know the source of the error. And definitely, the feedback loop aids hugely in debugging and fine-tuning our function.

5.4.2 Scenario 2

In Scenario-2, our focus shifts to analysing the interplay between Twitter data and pollution statistics. The frontend team needs an API that can fetch data from both indices.

Data Streaming
In this scenario,we explore the interaction of pollution statistics with Twitter sentiment data. Resampling—a 5% sample taken from a 120-GB file of Twitter data stream. Later, we parsed the file to extract user IDs, geolocations, timestamps, and sentiment scores, finally organizing it into a JSON file. At the same time, using fission functions with time triggers, we streamed pollution data from EPA API. This allows the frontend team to better collate and correlate the data from both the indices, resulting in great overall analysis.

Data Queries
Unlike what we did in Scenario-1, the frontend team could access raw data directly from Elastic Search while preceding any pre-processing. To facilitate this, we established an HTTP-triggered Fission route accessible via a URL. This route, set up as a 'POST' method, allows the frontend to communicate directly with the backend through the API. We developed a generalized backend query function called 'data-extract' precisely for this purpose. Communication between the client and server is managed through Flask, which supports the reception of two parameters: 'index name' and 'query.' This setup enables the backend to execute search queries within ElasticDB using the Python Elasticsearch package. We chose Flask over a RESTful URL design because Elastic queries are often lengthy and complex. Flask provides a more robust and error-tolerant method for handling intricate queries that might otherwise become cumbersome and error-prone if transmitted through URLs.

Data Processing
As outlined, this function serves primarily as a query tool without any backend data processing. It is designed not just for this specific scenario, but for any situation requiring direct data retrieval. The intent is to return the data unchanged, preserving its original form for accurate analysis.

Testing
Due to the inherent client-server interaction, local testing of this function before deployment is not feasible. Our approach involved initially uploading a lightweight version of the function to facilitate preliminary communication tests between the backend and frontend. After confirming the primary connection, we extended the body of the function and performed further tests. All tests were performed directly in the frontend environment by executing data queries to ensure the function would run correctly under real-world conditions.

5.4.3 Scenario 3

In Scenario 3, we aim to support the front end in analyzing Twitter text data over specific periods. To facilitate this, the backend provides efficient APIs for data retrieval and handling substantial natural language processing (NLP) operations.

Data Streaming
In Scenario 3, we managed and analyzed a large pre-collected Twitter dataset for text content analysis. We extracted a 5% sample from a 120 GB Twitter file, processed it to include user IDs, geolocations, timestamps, and tweet text content, and organized this data into a JSON file. This ensures efficient querying and supports the front-end in analyzing Twitter text data over specific periods. Detailed information can be found in the Data Processing section.

Data Queries
This scenario simplifies the querying process compared to Scenario 2. Instead of complex database queries, the front only needs to send two integer parameters: 'year' and 'month.' These parameters are passed to the backend through a RESTful API using a Fission route configured as a 'GET' request. This method is chosen because it does not modify server-side data and is solely for data retrieval. Upon receiving the parameters, the backend substitutes them into a pre-defined query to perform an Elasticsearch operation that filters Twitter data according to the specified date and time.

Data Processing
In order to save front-end RAM resources, most of the computational workload of this scenario, including that related to text processing, is managed by the back-end.We employ the NLTK Python library for comprehensive text preprocessing, which includes removing stopwords, lemmatization, stemming, and tokenization. These steps are intended to cleanse the data of irrelevant elements, allowing the analysis to focus on significant words. However, this static approach sometimes mistakenly filters out relevant information due to the dynamic nature of internet-based language, indicating potential areas for methodological refinement. After processing, we calculate the frequency of each token and send these results back to the frontend in JSON format, providing a streamlined and manageable dataset for further analysis.

Testing

Testing in such client-server scenarios is always a challenge. Our development process again emulates Scenario 2 but in a far more progressive style: we start with basically a structure of code, then update and test in a repeat manner. Such an iterative approach ensures that each level of the function is sturdy and works before moving on to more complex implementations, so ultimately it optimizes both the development timeline and the efficacy of deployed functions.

5.5 ElasticSearch

For this project, Elasticsearch forms the backbone in terms of data storage and querying needs, and the foundation of efficient data storage and analytics. Indexes were created in Elasticsearch for keeping the data by storing the data sets from different sources. Every index works as a database, systematically organizing data.

Specifically, data is stored in these indexes as documents, and each document represents a unit of data. The sharding technique is employed to split the indexes into smaller, more manageable parts distributed across multiple nodes to improve query performance and scalability. At the same time, to ensure fault tolerance and high availability, sharded copies are used on different nodes, allowing the system to handle failures without losing data. Overall, Elasticsearch can be viewed as a cluster of multiple nodes that manage indexes.

5.5.1 Database Design

From Figure 14, the system contains seven databases deployed in four instances. Twitter data is stored in an index named start with tweet, segmentation data containing the four locations in Australia is stored in an index named pollen, EPA data obtained using the API is stored in an index beginning with epa, BoM data obtained using the API is stored in an index beginning with bom, and data from VIC DATA on the Microclimate Sensor in Melbourne is stored in an index named sensor.

bom-000001	green	open	3	1	11924	3.88mb
epa-000001	green	open	3	1	5018	1.43mb
metrics-endpoint.metadata_current_default	green	open	1	1	0	450b
pollen	green	open	3	1	832	247.14kb
sensor	green	open	3	1	37280	6.15mb
temperatures	green	open	3	1	820	653.76kb
twitter	green	open	3	1	470097	131.61mb
twitter-word	green	open	3	1	470097	267.3mb

Figure 14: Index of ElasticSearch

5.5.2 Data Insertion

Creating Elasticsearch Indices
In order to upload each of the datasets involved in this project into elastic research, each dataset required the creation of an Elasticsearch index customized to the specific structure and requirements of each data type.

As the first step, we created Elasticsearch indexes for the pollen and sensor datasets using Shell scripts which defined the index structure and settings, including mapping and slicing configurations. During the data import process, we found that this upload method is not effective and needs long time to upload a small size file. To solve this problem, we tried to upload the file directly in Elasticsearch. This way is easy to use and allows users to upload files directly from a specified local path. However, we found that this direct upload approach had a limitation: it could not customize index settings.

Ultimately, we developed and adopted a hybrid approach which finds the balance and the flexibility while uploading the Twitter dataset. We used a Python script to create the Elasticsearch index and import the data with settings and mappings specific to the Twitter data. It is including defining slice numbers, replicas.

Importing Data into Elasticsearch
In this project, we experimented with 3 data import strategies. Initially, we used JavaScript files to load the data into Elasticsearch while dealing with pollen and sensor data. In the case of pollen data, the process consisted of the following steps: read configuration values to access Elasticsearch by securely loading credentials from the configuration file; loading the pollen data from a JSON file specific to each region; finally, the data data sent to Elasticsearch by HTTPS requests, and each entry is published in the specified Elasticsearch index.

Then, we tried to directly upload through Elasticsearch, and give up since performance limitations in processing the sensor data. This approach proved efficient for the same file upload time, directly scaling down by more than 3/4 of the make practice, but this limited our ability to customize the index settings.

Based on the performance limitations observed in previous strategies, we directly used Elasticsearch to upload data, which proved more efficient but limited our ability to customize the indexing settings.

We adopted a better approach using Python scripts while dealing with the Twitter dataset. This approach allowed us to preprocess the data, create custom indexes, and ensure efficient data ingestion. In detail, the twitterToElastic.py scripts loaded data from local JSON files first and then processed entries to filter out missing or invalid geo-coordinates and ultimately indexed the processed data into Elasticsearch indexe

By exploring those 3 upload methods, we overcame the limitations of direct up-

loads and shell scripts and finally found a balance between efficiency and configurability. We set environment variables for Elasticsearch credentials throughout the process and protected these files with appropriate permissions to ensure data security and integrity.

5.5.3 Custom Routing

For this project, we employed custom Elasticsearch routing to optimize query performance, especially on comparative pollen data within different regions. The dataset used in this study contained information about four different regions: Campbelltown, Canberra, Parkville, and Rocklea. While ingesting, each piece of data attributed to an individual region had a dedicated routing hashing value. This allowed queries that target a particular region to be sent directly to the relevant shard, thus narrowing the span of the query and improving the search process. We allow custom routing for optimization purposes in order to facilitate easier and effective regional data comparison.

5.5.4 Life Cycle Policy

Elasticsearch Life Cycle Policy is a collection of management actions performed on an index. These stages range from Hot, where an index receives write operations and demands high performance; Warm stage, where you read more data than you write, therefore having fewer write operations with moderate performance requirements; Cold stage, when the index is accessed very infrequently but still gets queried, and finally, the lowest performance demand held. There is also the Delete Stage, which automatically removes the index after reaching a certain age to save space on your disk.

In our setup, we defined separate life cycle policies for bom and epa data streams, named **bom-policy** and **epa-policy**, respectively, although they are currently identical. For each policy, the **hot** phase is triggered when an index reaches either 20GB in size or is 30 days old. In this phase, the index receives write operations and requires high performance. The **warm** phase begins 30 days after the index creation, during which the data is mainly read, and the index is allocated to nodes with the box_type set to **warm**, with an additional replica added for redundancy. The **cold** phase starts 90 days after creation, where the index is infrequently accessed but retained, and it is frozen to save resources. Finally, the **delete** phase is reached at 120 days, where the index is automatically deleted to free up storage.

It ekes out hardware resources at different stages of a Life Cycle Policy, actually saving costs and optimizing the use of resources. It keeps system performance high because the rollover mechanism prevents the indices from becoming large. The index automatic life cycle management, thus, reduces the difficult level in manual operations and considerably raises efficiency. The Life Cycle Policy secures the conformance of the organization's data retention policy. It eases the automatic actions of eliminating out-of-date data and keeping only necessary and current data in the repository of the system.

6 Error Handling

6.1 Error Mechanisms

As our team, primarily composed of individuals with backgrounds in mathematics or statistics and limited programming knowledge or experience with Linux, embarked on building this system, we inevitably faced numerous challenges, including a variety of errors and bugs.

Coding Errors
Many of the issues we encountered were type errors, which are typically straightforward and can be resolved by inspecting the error messages and code. These errors often arise from minor oversights during the coding phase and can generally be categorized as coding errors. Although fundamentally, all issues stem from the code; this category includes errors that are directly traceable and fixable based on the error messages provided.

Data Errors
Issues such as missing values and inconsistent data formats are prevalent in data science and are particularly challenging. We addressed these through standard methods like the NLP text cleaning techniques used in Scenario 2. Our approach to data cleaning and preprocessing was to apply established methods rather than developing context-specific solutions, focusing more on system design and implementation.

Cluster and Backend Errors
Errors associated with our cluster backend, especially those related to Fission, were more complex. These ranged from packaging and building up functions to their actual implementation. Each step in this process could potentially introduce errors, which we will detail in the following section.

6.2 Common Issues and Solutions

Space Errors
Initially, each function was built with its own set of packages. While this approach was more straightforward, it led to redundant installations of the same packages across multiple functions, significantly increasing space consumption. We consolidated several functions to optimize space to share a single package configuration, differentiating them through specific entry points.

Internal -500 Errors
These were among the most challenging to debug due to their ambiguous nature. Despite encountering this error frequently, there needed to be a reliable pattern for resolution. Our strategy involved a methodical exclusion process, analyzing our code and the system set up to isolate and resolve the issue.

HTTP Status 405/404 Errors

These errors were common during client-server interactions through URLs. A 404 error typically indicated a problem with the input URL, while a 405 error usually stemmed from using an incorrect method (e.g., confusing GET and POST).

By addressing these errors systematically, our team has gained valuable experience in troubleshooting and refining our system despite the steep learning curve imposed by our diverse academic backgrounds and limited initial programming expertise.

7 Conclusion and Future Work

7.1 Summary

Overall, in this assignment, we designed, built, and implemented a compact computational cluster of four nodes inherited from the Melbourne Research Cloud. Our cluster is managed by the kubelet system and achieved serverless implementation through Fission. Additionally, we utilized an Elasticsearch database as part of our backend architecture for handling data ingress and egress. We also present three scenarios to illustrate the capabilities of our cluster architecture, system design, and storytelling—more importantly, our understanding of the cluster mechanisms.

The major benefits of our system include, but are not limited to, scalability. This advantage is derived from our serverless design and the Fission package management, which handles multiple Fission functions under one package. This setup enables packages to be easily moved, redeployed, and updated, also saving on disk storage. Flexibility is achieved by utilizing multiple open-source platforms within our cluster system, which can be customized as needed. Efficient communication is maintained as our backend and frontend smoothly transfer information, utilizing techniques specifically designed for each context to optimize frontend resources such as RAM and computational power.

However, it should be noted that our team has a very limited programming background. Reaching this stage is indeed an achievement, yet there are weaknesses in our system design and overall procedure. These include inefficient development progress, as our team members often relied on manual testing rather than employing cleverly designed scripts appropriate for the context. This is an area we aim to improve in future projects.

7.2 Roles and Contributions

Yiqing Song 1344675
In this project, my main tasks included developing scenarios, creating and managing Elasticsearch indexes, writing and maintaining JavaScript, shell scripts, and Python files. I was also responsible for setting and managing lifecycle policies, participating in testing Fission functions, and organizing the project repository on GitHub to ensure smooth collaboration and efficient project management.

Keti Zhang 1401706
Set up the Kubernetes cluster, created the Fission client, environment, and Elasticsearch environment, and added the rest of the team's public keys to our bastion host to enable connection. Uploaded Fission packages and the setups, created the data-extract function that enables the frontend to query data, and solely finished Scenario 3 both backend and frontend. Occasionally participated in debugging random errors. Contributed around 1400 words to this report.

Zijuan Lin 1372754
Responsibilities include uploading data to the Elasticsearch database, creating indexes, and ensuring efficient data ingestion. Collected data from VIC DATA and SUDO sources, defined project scenarios, and documented vital findings. Developed database scripts, designed research protocols, and wrote project reports to ensure a comprehensive and clear presentation of the work results.

Xuan Ji 1405130
In this project, my main contributions are in data collection, scenario development and front-end data visualization, and the creation of a demonstration video. I was involved in identifying and collecting necessary data sources. Additionally, I was responsible for designing and implementing front-end data visualizations, using Jupyter Notebooks to create interactive and user-friendly visualizations.

Hantong Li 1399648
In this project, I was primarily responsible for developing Fission functions and conducting data analysis, including managing data transmission and retrieval from Elasticsearch. I collaborated closely with the team member in charge of Elasticsearch to design the index map structure, ensuring efficient data handling and smooth development. This collaboration was essential for overcoming challenges and optimizing our system's performance

8 Appendix

Github: https://github.com/Lucasbeast/COMP90024-SM1-Project

Demonstrating video: https://youtu.be/uyMgNHQLCEs