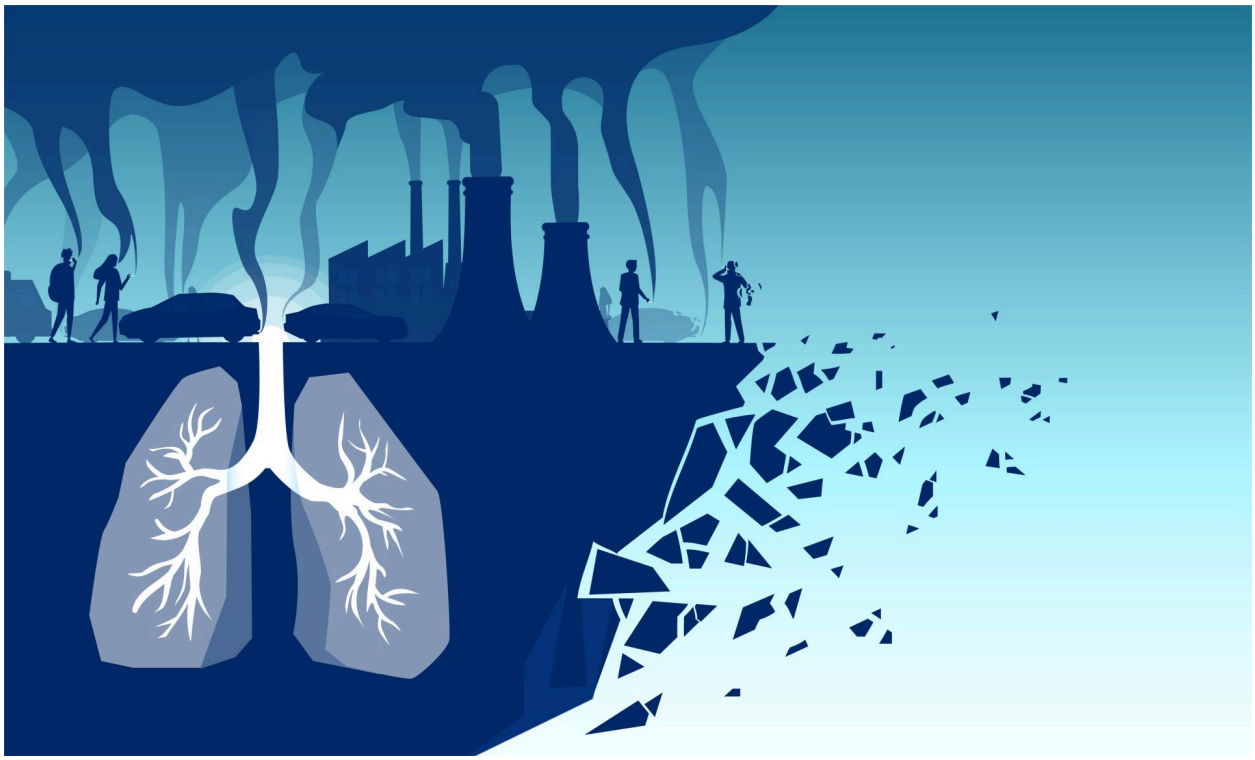
Big Data Analytics on the Cloud

Team number: 10

Team member:

Shanrui Huang - 1533562
Lingyi Zhang - 1470460
Jiaxing Wang - 1511557
Yunru Zhu - 1470423



Contents

1. Introduction **3**
2.System Architecture And Design **3**
2.1 Basic Components 3
2.2 Fission Deployment 4
2.3 Elasticsearch Deployment 5
2.4 Data Flow 6
3. Functionality **6**
3.1 Back_end 7
3.2 Frond_end 11
4. Scenarios **12**
4.1 Analysis of the Impact of Air Quality on Lung Diseases in Victoria 12
4.2 Analysis of the Impact of Wind Speed on Air Quality in Victoria 16
5. Evaluation of MRC and Tools and Processes for Image Creation and Deployment **20**
5.1 UniMelb Research Cloud 20
5.2 Tools and Processes for image creation and deployment 21
6. Error Handling **21**
7. Limitations **22**
8. Conclusion **22**
9. System Demonstration **23**
10. Team Member Contributions **23**
11. Appendix **23**

1. Introduction

The research question our group aims to discover is the correlation between air quality and lung diseases, and correlation between air quality and wind strength. We use real-time data including Victoria air quality data from National Air Quality Database, Victoria air quality data from Environment Protection Authority and Victoria weather data from Bureau of Meteorology Victoria. We also manually download COPD/Asthma/Respiratory System Disease data from SUDO.

In order to accomplish our task, we need to design an application based on Melbourne Research Cloud that utilises Kubernetes cluster, Fission framework, and Elasticsearch database to harvest, manage, process, and store data efficiently.

This report will describe the architecture, design, and functionality of the application we developed. Through this report, we will elaborate on the role and contribution of our team in the system development process and share the challenges and solutions we faced during the design and implementation process.We will also demonstrate the application through a video.

2.System Architecture And Design

2.1 Basic Components

- **Kubernetes:** Acts as the orchestration layer managing containerized applications. It handles the deployment and operation of application containers across clusters.

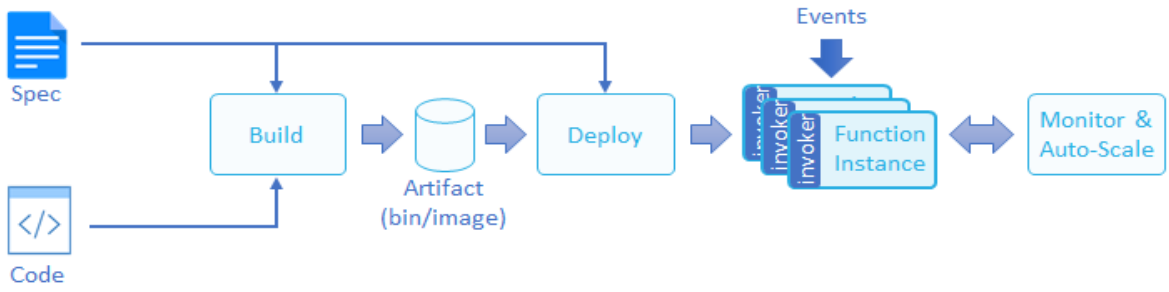
- **Fission:** A serverless framework running on Kubernetes, allowing the deployment of short-lived functions without managing server lifecycles. It is used to trigger data processing functions on demand, optimising resource usage and responsiveness.Fission takes advantage of the flexible and powerful orchestration capabilities of Kubernetes to manage and schedule containers. This allows Fission to focus on developing the functions-as-a-service (FaaS) feature.

- **Elasticsearch:** Utilised as the data storage backend, providing robust and real-time data indexing. The processed data is ingested into Elasticsearch via bulk API operations for efficient handling of large datasets.

- **Jupyter Notebook:** Acts as the frontend of the system. Retrieve data from Elasticsearch and perform the data visualisation.

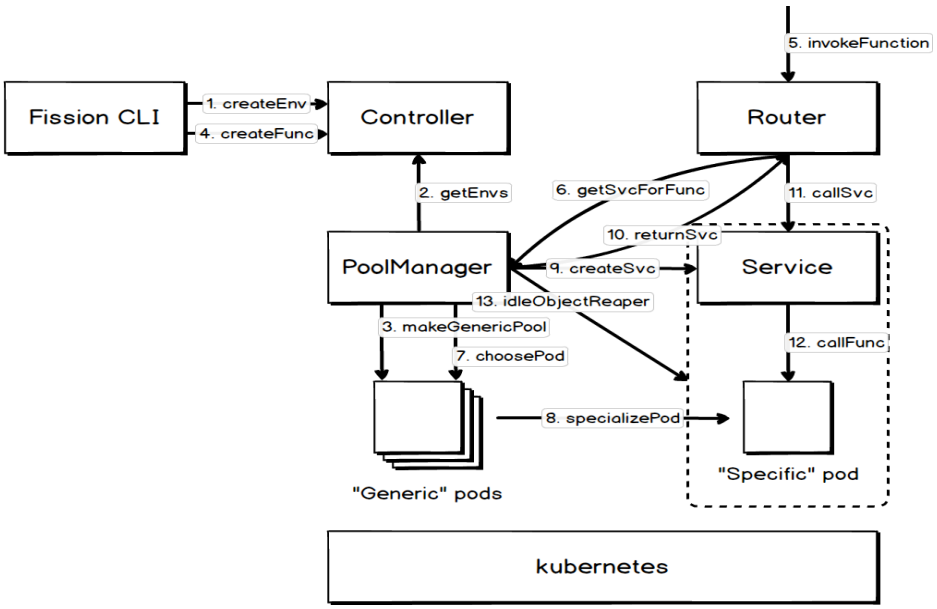
2.2 Fission Deployment

- **Lifecycle :**



(Figure 1:Lifecycle of fission)

- **Poolmanager:**

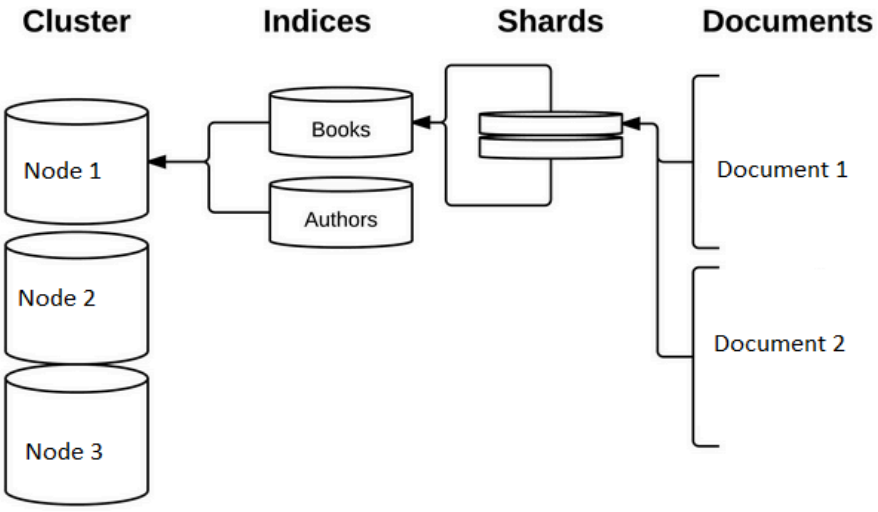


(Figure 2:Mechanism of Poolmanager)

1. Fission CLI is used to send a request to the controller, to build the required language environment for running the function.
2. PoolManager traverses the environment resources list, and creates a generic pod pool for each environment.
3. Use Fission CLI to send a request to the controller to create the function.
4. Router receives the request to trigger the function, and loads the corresponding information of the target function.
5. Router sends a request to the executor to obtain access to the function.
6. PoolManager randomly chooses a pod as the carrier to run the function from the generic pod pool of the runtime environment specified for the function.
7. Return the service information of the function to the router.
8. The request will finally be routed to the pod that runs the function.

- **HTTP Trigger**
  All requests sent to Fission functions are forwarded by a router. We use the K8s ingress mechanism to implement an HTTP trigger.

2.3 Elasticsearch Deployment



(Figure 3: Mechanism of Elasticsearch)

- **Index**
  An index is a collection of documents that have similar characteristics, it's the highest level entity we query against in Elasticsearch. We create separate indexes even for the same type of data if we apply aggregation before uploading to Elasticsearch.
- **Node**
  For our system we create one Master node and one Work node.The Master node controls the Elasticsearch cluster and is responsible for all operations including creating/deleting an index and adding/removing nodes. Whileas the Work node stores data and executes operations including search and aggregation.
- **Shard**
  Shards are the number of chunks Elasticsearch will subdivide the index into. Every index in our system has three shards.
- **Replica**
  Number of replicas represents the number of copies of the index.Every index in our system has one replica.
- **Kibana**
  Kibana is a data visualisation and management tool that supports a variety of types of plots. As long as the data is successfully uploaded to the corresponding elasticsearch index we can view through Kibana.

2.4 Data Flow

Services are deployed as fission functions, when triggered, harvest data from external APIs, then forward the data to the storage service.

Storage service is connected to Elasticsearch and upload data received to Elasticsearch, index mapping is defined in the service.

One jupyter notebook is connected to Elasticsearch, with the help of query DSL to get the data we want.

3. Functionality

The system we developed is a data analysis platform designed to process and analyse and compare large amounts of weather-related data with Sudo official data, especially to explore the relationship between air quality and other data. Key capabilities include real-time data ingestion, storage, search, and visualisation.

3.1 Back_end

- **Harvst**
  - **General structure**
    1. Set the index used to filter data:

In the data harvest related scripts, we need to first filter the data by setting specific query parameters. For example: "site, time" and other parameters, so that we can determine what type of data needs to be obtained based on the parameters in the HTTP request through the curl instruction in WSL.

```
site = request.args.get('site', '*')
date = request.args.get('date')
year = date[:4] if len(date) >= 4 else '2023'
if site == '*':  # all site
    site_filter = ''
```

2. Set target URL:

Define the target URL in each script, which points to the API of the data source, that is, specifies where to get the data.

```
url = f"https://naqd.eresearch.unimelb.edu.au/geoserver/wfs?s
```

3. Obtain data from the corresponding URL through the requests library:

Use the requests.get() method to get data from the URL defined in the previous step. This step is the core of the data collection process and pulls data from external data sources through HTTP GET requests.

```
response = requests.get(url)
```

4. Determine the URL that corresponds to be sent to the virtual machine:

A URL is defined in the script, which points to the interface deployed on the virtual machine. This interface is responsible for receiving data obtained from the data source.

5. Make a POST request to the virtual machine to call the cloud script:

```
store_url = 'http://router.fission/airqualitystore519'
```

Use the requests.post() method to send the obtained data to the URL of the virtual machine through an HTTP POST request. Then connect to the script that we upload to the corresponding upload data on the virtual machine.

```
store_response = requests.post(store_url,json=response.json())
```

- **Corresponding script**
  - airqualityharvest519.py
  - harvst_to_eachday.py
  - harvst_to_mean.py

  These three scripts are responsible for obtaining data on air quality from NAQD, and then sending the collected data to the cloud URL for subsequent data analysis and report generation.
  The difference between these three scripts is that the URLs on the cloud are different, so after the post request, they correspond to different subsequent upload scripts on the cloud.
  - harvst_BoM.py
    Regularly obtain meteorological data from the Australian Bureau of Meteorology (BoM), and then request post to the specified URL in the cloud
  - harvst_epa.py
    Data acquisition: Obtain environmental monitoring data from the Environmental Protection Agency (EPA), obtain the latest PM2.5 data in some designated site areas, and then request post to the specified URL in the cloud

- **Concept of Design**

In the early stages of the project, we designed a processing system for the back-end data part, which was divided into two main parts: data download and data upload:

- **Data download:** Use a Python script to grab air quality data from the specified website through API, and save the data as a local JSON file.
- **Data upload:** Read locally stored JSON files and upload the data to Elasticsearch for indexing and storage.

- **Problems encountered**
  During implementation, we encountered two main problems:
  - **Local data reading issues:** Since the virtual machine cannot access data in the local file system, this limits the ability to run data download and upload scripts in the virtual machine environment.
  - **Fission packaging problem:** We tried putting the data file and the script in the same folder, which allowed the script to find the local data, but it violated Fission's expectations and could not create a package, making it impossible to curl the url of the cloud upload data script.

- **Improved Design**
  In order to solve the above problems, we redesigned the data processing process, that is, redesigned the download data script, (stated above) to achieve the following goals:
  - **Direct data transfer:** Modify the data process so that the data can be uploaded directly from the source to the cloud Elasticsearch, and the data will not be stored in the local file system first.
  - **Improved system efficiency and reliability:** The new design solution not only solves the problems of data reading and backup, but also improves the efficiency and reliability of the entire system by reducing data processing steps.
- **Data Storage**
  - **General structure**
    1. Initialise the Elasticsearch client:

    First, each script initialises an Elasticsearch client for connecting to and interacting with the Elasticsearch cluster. Determine the service URL of the Elasticsearch cluster, which points to the Elasticsearch service deployed inside the Kubernetes cluster. During this process, SSL verification will be cancelled and the username and password for HTTP

authentication will be set to ensure connection security and access authorization.

```
es_client = Elasticsearch(
    'https://elasticsearch-master.elastic.svc.cluster.local:9200',
    verify_certs=False,
    ssl_show_warn=False,
    basic_auth=('elastic', 'elastic')
```

2. Get data by requesting the library:

Use the requests.get() method to get the data that needs to be uploaded from the data sent via HTTP POST in the previous harvest script. This step ensures that data received from different data sources can be received and processed correctly.

```
data = request.get_json(force=True)
```

3. Set the index of each message:

For each piece of data received, a specific index is set, which is used to filter and store the data in Elasticsearch. The setting of the index focuses on the type and purpose of the data, such as date, location and other key information.

```
actions = []
try:
    for feature in data["features"]:
        action = {
            "_index": "airquality_520",
            "_source": {
                "site_name": feature["properties"]["si
```

4. Batch processing and uploading data:

Use Elasticsearch's batch API to back up all prepared data into a list, and then upload it to Elasticsearch in batches. This batch processing method improves the efficiency of data upload, reduces the number of network requests, and ensures the stability and reliability of the data upload process.

```
    bulk(es_client, actions)
```

- **Corresponding script**

1. addobservations.py

This script receives data in JSON format from an HTTP POST request, and then batch uploads data such as "site_name", "time_stamp", "geo",

"ozone", "pm10" and "pm2p5" to Elasticsearch's airquality_520 index . This process enables efficient data transfer through Elasticsearch's batch API.

2. final_store_eachday.py

The script is responsible for receiving the data from the harvst_to_eachday script and formatting the date. Calculate the average of some observations, and then upload the processed data (such as "site_name", daily average of pm2.5, daily average of pm10) to the mean_eachday index of Elasticsearch.

3. store_sudo.py

This script reads data from the local file "sudodata.json" and uploads each data entry directly to Elasticsearch via an HTTP POST request. For each data entry, the script specifies a unique index and sets the necessary document ID (sudo/_doc/{doc_id}).

4. final_store_mean.py

This script processes the data received from the HTTP POST request and aims to calculate the average of pm2.5, pm10 for each year. Once the processing is complete, it uploads the average of these different sites into Elasticsearch's mean index.

5. store_epa.py

This script specifically handles air quality data obtained from the Environmental Protection Agency (EPA). Then process the site name, time point, and pm2.5 fields, and then upload these data to an epa index in Elasticsearch.

6. store_BoM.py

This script is a compilation of the harvst_BoM script to obtain BOM data, process wind speed, time, region and other information, and upload it to the BOM index of Elasticsearch.

3.2 Frond_end

- **General structure**
  1. Define Elasticsearch queries

     First we need to define an Elasticsearch query in the script, a dictionary in JSON format that specifies the fields and conditions to be retrieved from

the Elasticsearch index. This query is used to specify exactly what data needs to be obtained from the dataset.

```
query = {
    "query": {
        "term": {
            "site_name.keyword": "Mildura"
```

2. Use es.search() to query data
   Use the search() function in the Elasticsearch library to retrieve data based on the query defined in the first step. This method will perform a query on the specified index and return matching results.

```
response = es.search(index=index, body=query, size=10000)
```

3. Extract query results
   Finally, the returned data is converted into JSON format and written into a document.

```
with open("epa_data.json", "w") as f:
    for hit in hits:
        json.dump(hit["_source"], f)
        f.write("\n")
```

4. Save data to list
   At the same time, the returned data is stored in a Python list for further data analysis in jupyter notebook.

```
data = [hit['_source'] for hit in hits]
```

- **Corresponding script**
  1. get_epa_data.py
     This script queries the specified "Mildura" site data from the Elasticsearch index, saves the query results to the epa_data.json file, collects the data into a list named data, and returns this list.

  2. get_bom_data.py
     This script queries the Elasticsearch index for data from the "Mildura" site, and the results are stored in the bom_data.json file. The extracted data is also collected into the data list and this list is returned.

  3. getdata_air_disease.py
     This script integrates data from two indexes, queries data involving air quality and lung disease-related fields, cleans and normalises the data

after processing, and stores it in a DataFrame. The processed data is also collected into the data list and this list is returned.

4. getdata_air_everyday.py
   This script retrieves all records from the Elasticsearch index, writes the query results to the airquality_data.json file, collects the data into a list named data, and returns this list.

4. Scenarios

We selected two scenarios to demonstrate the system's ability to handle different types of analysis:

4.1 Analysis of the Impact of Air Quality on Lung Diseases in Victoria

- **Scenario Overview**

  In recent years, the number of people suffering from lung disease has increased in several regions of Victoria, requiring targeted public health interventions. This scenario aims to analyse the impact of air quality on different lung diseases in people in different regions of Victoria, and to explore whether changing air quality can reduce the rate of lung diseases in people.

- **Data Resource**
  1. **Air quality data**(PM2.5, PM10)
  2. **Sudo data**(Asthma count, Asthma rate, COPD count, COPD rate, Respiratory system disease count, Respiratory system disease rate)
- **Research Problem**

  Explore the changes in the PM2.5 content and PM10 content at different air quality testing sites within one year, as well as the statistics of people suffering from three lung diseases from the lung disease statistics site that overlaps with the air quality testing site. And explore the relationship between air quality and people's development of three lung diseases at these same sites.

- **Analytical Method**

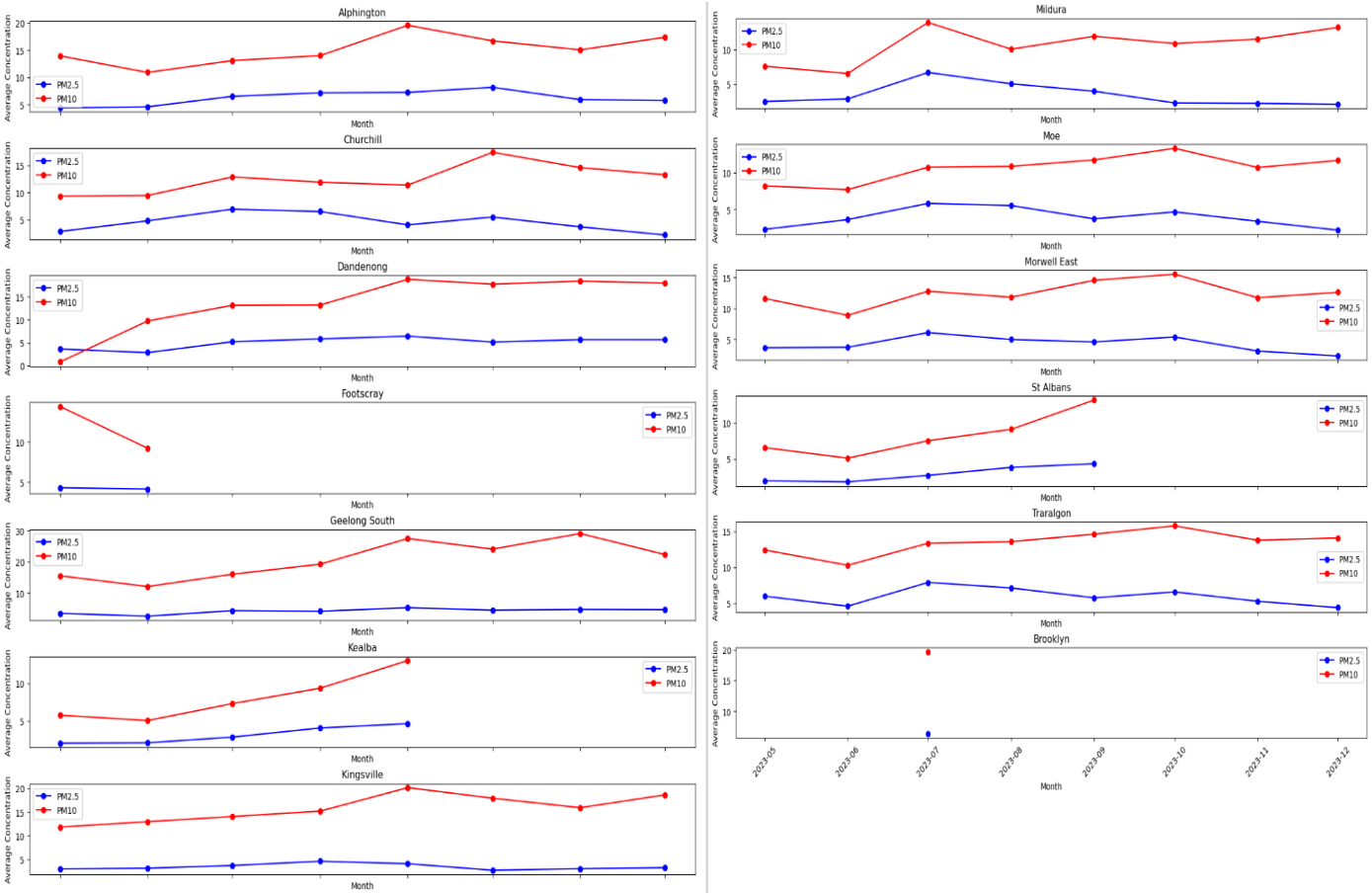
We created two python scripts to capture air quality data on ElasticSearch and air quality data from a site where the air quality detection site overlaps with the lung disease statistics site, as well as data on people with different lung diseases(Use the query statements to send a request through the ElasticSearch search API, specify the index and query conditions, and the API will return matching data results.). Also, we used jupyter Notebook to analyse and visualise the data.

Python Scripts: getdata_air_everyday.py

getdata_air_disease.py

Jupyter Notebook: data_analysis.ipynb

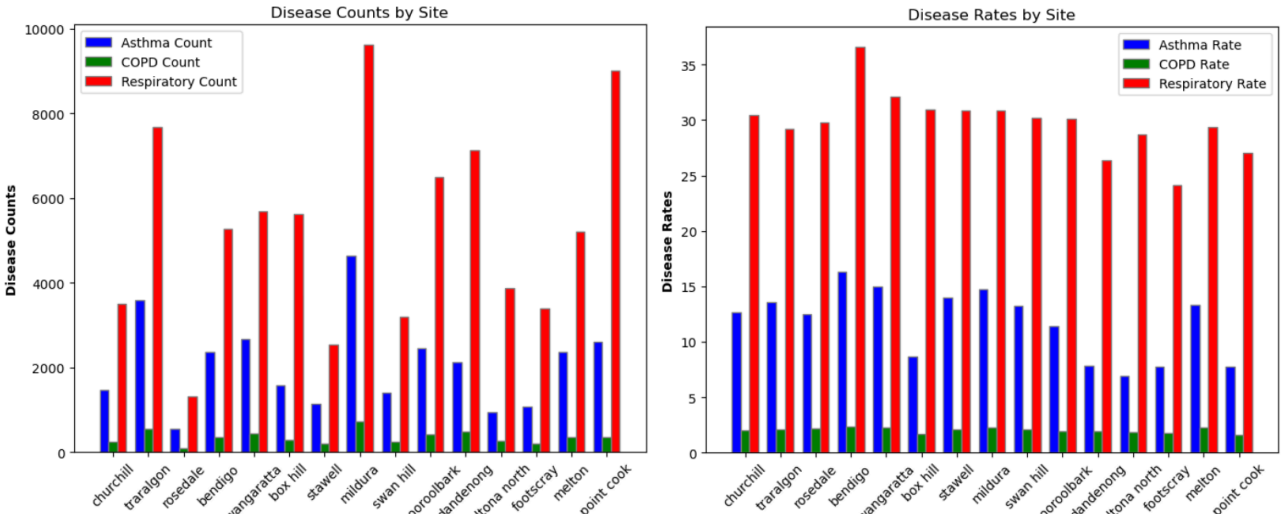
- **Results and Visualisation**
    1. Monthly Average PM2.5 and PM10 Levels at Each Site



(Figure 4:Monthly changes of PM2.5 and PM10 Levels at Each Site)

As can be seen from the line charts, PM2.5 and PM10 levels remain relatively stable in most regions, but there are some fluctuations. At all sites, PM10 concentrations are generally higher than PM2.5. This may be because PM10 includes PM2.5 and larger particles, then its total concentration is higher. At most sites, the trends in both types of particulate matter are similar, suggesting that they may be affected by the same environmental factors.

2. People Suffering from Different Lung Diseases at Each Site



(Figure 5:Different counts and rates of People Suffering from Different Lung Diseases at Each Site)

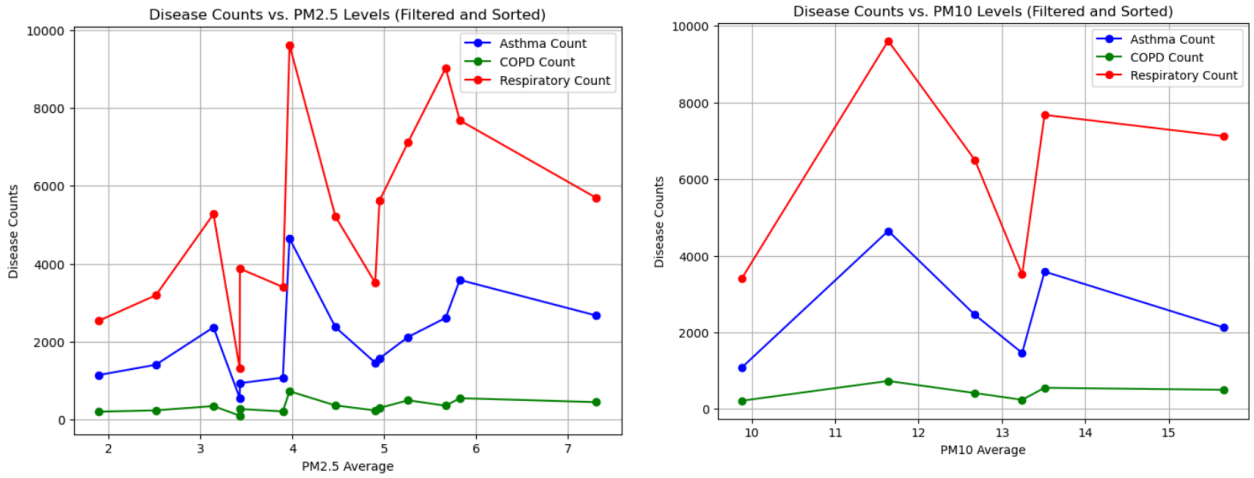
These two bar charts show the data we pulled from sudo - the number and percentage of people in different locations having different lung diseases.

Asthma: Asthma case numbers vary from site to site, with some sites such as 'Mildura' and 'Bendigo' showing very high counts.

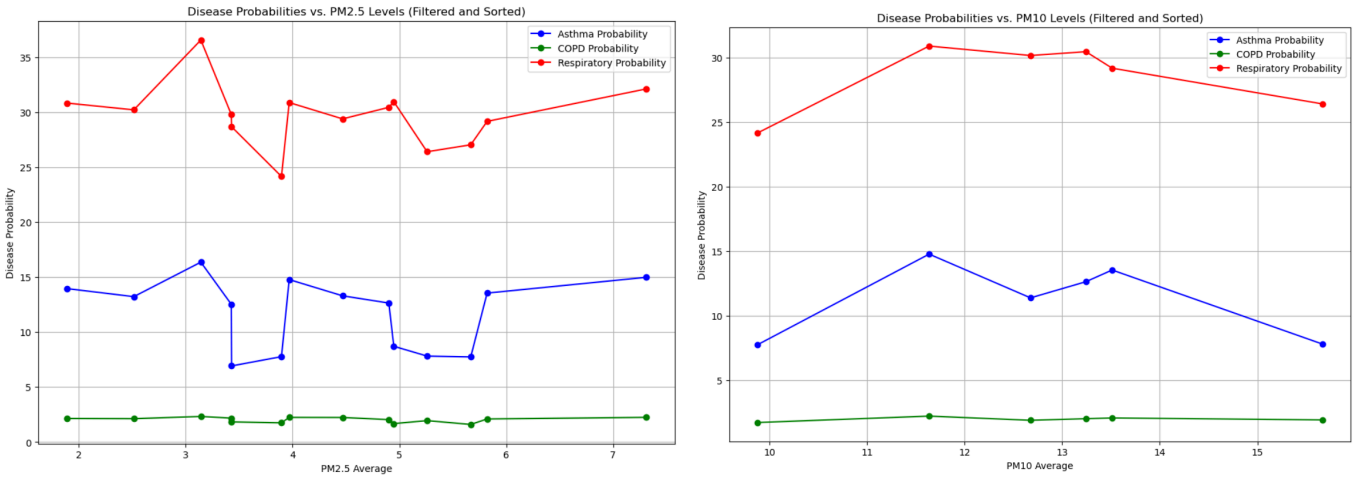
COPD: Case numbers and prevalence rates for COPD were generally lower than for asthma across all sites, but patterns across sites were broadly similar to those for asthma.

Respiratory disease: This condition always shows the highest counts, especially in sites like "mildura" and "bendigo".

3. Air Quality vs Lung Diseases



(Figure 6:Different counts of people with different lung diseases at different levels of PM2.5 and PM10)



(Figure 7:Different rates of people with different lung diseases at different levels of PM2.5 and PM10)

From these line charts, it can be seen that the number and rate of lung diseases in people under different levels of PM2.5 and PM10 fluctuate greatly, and there is no obvious positive or negative correlation. This suggests that air quality does not play a big role in people developing lung disease.

- **Challenge**

The python scripts we used to capture data on ElasticSearch cannot be directly called in jupyter notebook through import. We need to find its path and add it to the system path, then they can be imported and used normally.

- **Conclusion**

According to the results of data visualisation, we can conclude that air quality does not have a great impact on lung diseases among people in different regions. This may be due to a time mismatch between the air quality data and the data obtained from sudo on people's lung disease. Therefore, this scenario can only help us understand and analyse the changes in air quality at different sites and the number and rate of lung diseases among people. After data from a matching time period is obtained in the future, this scenario can still be used as a basis for exploring the relationship between air quality and lung diseases.

4.2 Analysis of the Impact of Wind Speed on Air Quality in Victoria

- **Scenario Overview**

In recent years, there has been an increase in the number of people suffering from lung disease in several regions of Victoria, which might be caused by changes in air quality. This scenario aims to analyse the impact of wind speed on air quality to help formulate more effective air quality management and policies and improve public health.

- **Data Resource**

1. **EPA data**(PM2.5)
2. **BOM data**(Wind speed)

- **Research Problem**

  Explore the changes in PM2.5 content at a certain air quality detection site over a period of time, as well as the changes in wind speed at the same air quality detection point within the same time period. Also explore the relationship between them.

- **Analytical Method**

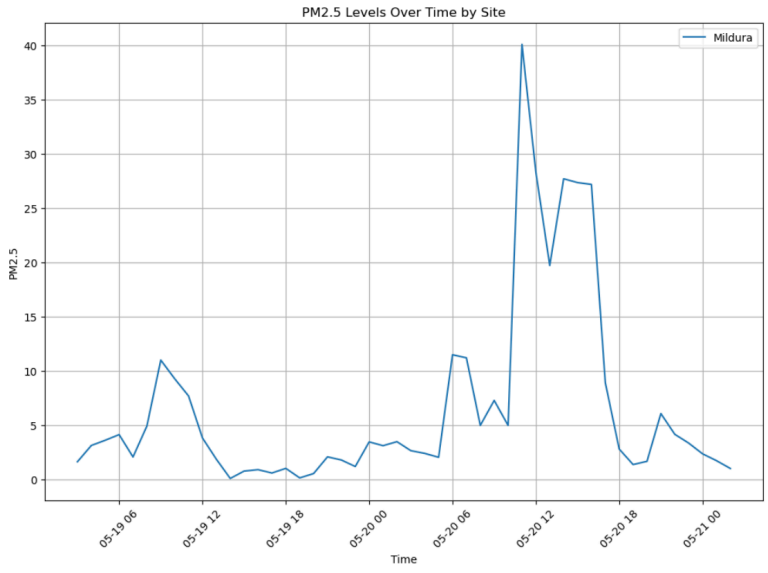
  Same as the previous scenario, we created two python scripts to capture air quality data at a certain site on ElasticSearch over a period of time and wind speed change data from the same site at the same time (Use the query statements to send a request through the ElasticSearch search API, specify the index and query conditions, and the API will return matching data results.). Also, we used jupyter Notebook to analyse and visualise the data.

  Python Scripts: get_epa_data.py

  get_bom_data.py

  Jupyter Notebook: scenario2.ipynb

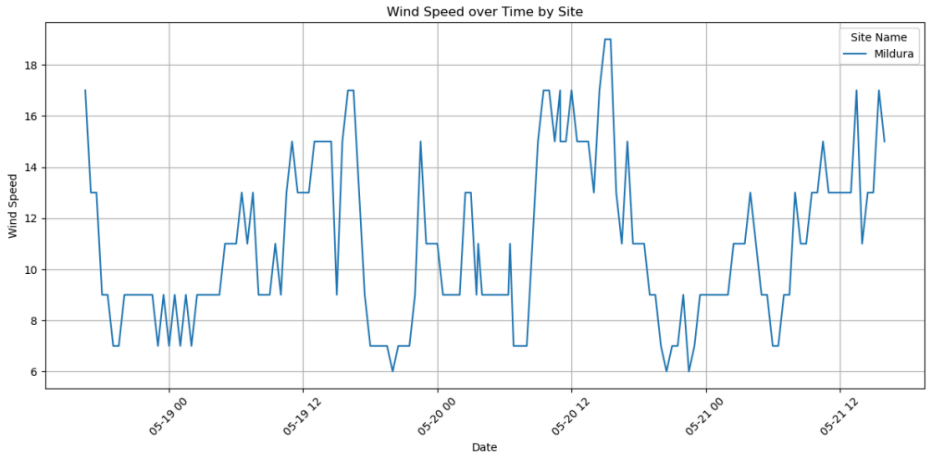
- **Results and Visualisation**
  1. Air quality over time at different sites



(Figure 8:PM2.5 level over time at Mildura site)

Figure 8 shows the PM2.5 concentration changes at the Mildura site from 00:00 on May 19th to 00:00 on May 21st. It can be seen that the PM2.5 concentration increased slightly in the morning of May 19th, and then gradually dropped to close to 0 at noon that day. In the early morning of May 20th, the PM2.5 concentration increased slightly again, but remained at a low level. On the morning of May 20th, the PM2.5 concentration suddenly rose sharply, reached a peak, and then dropped rapidly. Overall, the air quality in this region is good most of the time, but there are cases where pollutant concentrations rise sharply in short periods of time, and further investigation into the causes and measures are needed to improve air quality.

2. Wind speed at different sites over time

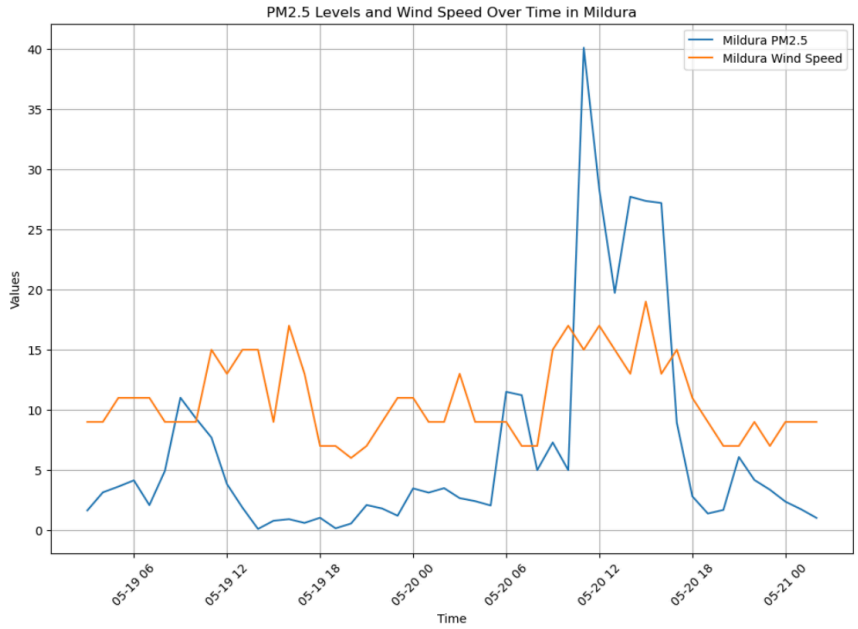


(Figure 9:Wind speed over time at Mildura site)

Figure 8 shows the wind speed changes at the Mildura site from 00:00 on May 19th to 00:00 on May 21st. It can be seen that the wind speed experienced significant fluctuations during this period, especially the significant increase in the morning of May 20th and the early morning of May 21st. Fluctuations in wind speed may have an impact on air quality, with higher wind speeds helping to disperse and dilute pollutants, while

lower wind speeds may cause pollutants to accumulate. This could help further research into the impact of wind speed on air quality.

3. Comparison of wind speed and air quality at a site



(Figure 10:PM2.5 level and wind speed over time at Mildura site)

As shown in Figure 10, on May 19th, there seemed to be no obvious correlation between PM2.5 concentration and wind speed, and fluctuations in wind speed did not significantly affect PM2.5 concentration.

On May 20th, there was a clear inverse correlation between wind speed and PM2.5 concentration. When wind speed rises rapidly in the morning, PM2.5 concentration also reaches its peak shortly afterwards. This could be due to high wind speeds bringing pollutants into Mildura from other areas, causing a sharp rise in PM2.5 concentrations.

At noon on May 20th, as the wind speed decreased, the PM2.5 concentration also began to decrease, further indicating that changes in wind speed have a significant impact on PM2.5 concentration.

- **Challenge**

  We failed to retrieve the data merged based on time and site_name at the same time during querying data from ElasticSearch. Thus, we could only retrieve the data of the same site separately, and then merge the data based on time in the jupyter notebook.

- **Conclusion**

  Based on the data visualisation results, we can conclude that changes in wind speed have a significant impact on PM2.5 concentration, especially in the case of sharp changes in wind speed. Higher wind speeds may carry pollutants from other areas into the monitored area, causing an increase in PM2.5 concentrations. Lower wind speeds may cause pollutants to accumulate locally, raising PM2.5 concentrations.

  Therefore, in the future this scenario can help us better understand the impact of wind speed on air quality and guide future air pollution control and environmental management strategies.

These scenarios help demonstrate the system's functional and technical advantages, verify its implementability, promote users' understanding of it, and stimulate innovative thinking to find more creative and practical solutions.

5. Evaluation of MRC and Tools and Processes for Image Creation and Deployment

5.1 UniMelb Research Cloud

- **Pros**
  1. Resource Availability. We have been allocated 11 virtual CPUs and 700 GB of volume storage,giving us freedom to discover any topics that we are interested in.
  2. Open-Source. Openstack is supported by a large community, we are allowed to access a wide range of plugins.
- **Cons**

1. Complexity. The initial setup is very challenging, as new users,it costs us a relatively long time. Our group even deployed the cluster multiple times since there were a lot of mistakes made.
2. Access Limitations. We find that the connection to Melbourne Research Cloud highly depends on the University Internet.As long as we leave the campus, even with VPN the connection could become unstable. One group member even can't connect to our group cluster outside of the campus.

5.2 Tools and Processes for image creation and deployment

- **Pros**
    1. Efficiency. Fission allows us to create/delete/update/monitor different events in a short time,the framework is very time-efficient.
    2. User-Friendly. The fission command is very easy to understand and interpret.
- **Cons**
    1. Additional Library Installation. It is very difficult to deploy extra library we might need on the environment.We have to follow a very strict standard in order for the additional library/module/function to be loaded.

6. Error Handling

- **“Error sending request to functions”**
  This is a very common error message we get during the process of developing the backend of the application.It simply implies that our request is not sent to the function normally. The most common reason for this error is that the connection is very unstable and the function response time exceeds the built-in upper limit and we only need to resend the same request several times. However, there could be many potential reasons to see this error message, once it is because the package for a certain function fails.
  The best way to address this error is to check the function log and debug step by step,and solve one bug each time.
- **Install additional library**
  Installing an extra python library in the environment is very complex. When we design the bulk api during the storing data to Elasticsearch phase, the script needs to use the “bulk” function in elasticsearch library helpers module. So initially we follow the normal import standard

```
from elasticsearch8 import Elasticsearch, helpers
```

However, it keeps returning a “500 internal error” message when we send a request to this function. Through countless experiments we found that the reason is we can’t directly import a module.

```
from elasticsearch8 import Elasticsearch
from elasticsearch8.helpers import bulk
```

For some reason we have to directly import the function we wish to use. And with everything else remaining the same the function can operate normally.

- **Passing data from harvest script to store script**

  Initially we are confused about where the fission functions execute. We think that since we define a local python script to the function,the function has to be executed locally, so we pass the data to a local url.

  ```
  store_url = 'http://127.0.0.1:9090/store'
  ```

  However, later we find that as long as the function is created, it is in the cluster. For example, a team member can send request to a function without the corresponding python script. So, we correct the url and data can be successfully pass from first script to second.

  ```
  store_url='http://router.fission/store’
  ```

7. Limitations

Our application faces significant limitations in terms of data availability from Sudo. The data from Sudo is limited and not recorded on a daily timeline while the data we get from NAQD is on a daily basis. Most relevant datasets on SUDO are updated annually. This infrequent update cycle results in less-than-ideal visualisation outcomes across different scenarios.

Additionally, public datasets from the EPA and BOM are also restricted,we are only allowed to retrieve data of the last three days.Also unlike the University of Melbourne's NAQD website,external websites like BOM are lack of dynamic API capabilities, we can't define a date-filter for it, it's hard to effectively utilise data from these sites.

8. Conclusion

In summary, our project successfully developed an application to explore the correlation between air quality and lung disease, as well as air quality and wind speed, and the correlation between air quality and wind power through COPD/asthma/respiratory disease data via epa, BoM , naqd, and manual download from SUDO.
Our application is built on the Melbourne Research Cloud, leveraging a Kubernetes cluster, Fission framework and Elasticsearch database. They enable us to efficiently collect, manage, process and store large amounts of relevant data.

9. System Demonstration

Video:https://youtu.be/KYUvyEorko4

Our video cannot be broadcast live because our cluster lost connection a few days before the due date.

Github:https://github.com/yunru111/comp90024_team10.git

10. Team Member Contributions

Jiaxing Wang: backend
Shanrui Huang:backend,frontend
Lingyi Zhang:frontend
Yunru Zhu: frontend

11. Appendix

1. https://www.alibabacloud.com/blog/fission-a-deep-dive-into-serverless-kubernetes-frameworks-2_594902
2. https://www.linkedin.com/pulse/empowering-kubernetes-logging-efk-stack-guide-fission-labs-on6dc/
3. https://www.knowi.com/blog/what-is-elastic-search/