Group 4

Cluster and Cloud Computing: COMP90024 - TreeInsights

Pranav Pai (1536042)
Faculty of Science
ppai@student.unimelb.edu.au

Saad Sheikh (1384242)
Faculty of Engineering and IT
saadullahs@student.unimelb.edu.au

Rongcong Lin (1520403)
Faculty of Science
rongcongl@student.unimelb.edu.au

Kaan Gocmen (1544249)
Faculty of Engineering and IT
kgocmen@student.unimelb.edu.au

Abhigyan Singh (1506476)
Faculty of Science
abhigyans@student.unimelb.edu.au

May 22, 2024

Contents

List of Figures

List of Tables

Abstract

This document pertains to TreeInsights, a cloud-based analytics solution. The tool is designed to intelligently display trends between trees and various factors, including natural resources like weather, as well as population density. The report aims to provide insights on how we can improve our understanding of Australia's diverse tree ecology system and make informed decisions regarding urban forestation that reduce environmental impact. We hope this will give a good starting point to build a resilient Australia.

Abbreviations

Abbreviation	**Meaning**
ABS	Australian Bureau of Statistics
API	Application Programming Interface
AWS	Amazon Web Services
BOM	Bureau of Meteorology
EPA	Environmental Protection Agency
ES	Elasticsearch
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
K8s	Kubernetes
LB	Load Balancer
MOP	Method of Procedure
MRC	Melbourne Research Cloud
REST	Representational State Transfer
SUDO	Spatial Urban Data Observatory
URL	Uniform Resource Locator
VPN	Virtual Private Network
WSL	Windows Subsystem for Linux
YAML	Yet Another Markup Language

Table 1: Abbreviations and their Full Forms

1 Introduction

For sustainable urban development and environmental conservation, understanding the relationships between environmental factors and urban greenery is essential. TreeInsights, a cutting-edge cloud-based analytics platform, serves this purpose by explaining complex relationships between urban trees and factors like weather and the population density within Melbourne's urban landscapes. This report talks about the technology that is used behind the platform and also focuses on very interesting scenarios about the urban forests of the Inner Melbourne area. Ultimately, this application aims to harness and analyze large data sets that capture the essence of these interactions.

TreeInsights uses real-time weather data sourced from the Bureau of Meteorology, alongside a dataset of 70,000 trees obtained from the Melbourne Victoria government data site. Additionally, we have incorporated population density data from the SUDO website. **The main purpose of our analysis is to understand and optimize urban forestry planning.**

Our platform leverages ElasticSearch for data management, Fission for real-time data processing, and Kubernetes for scalable deployment. TreeInsights backend, featuring a RESTful design, integrates seamlessly with our Jupyter Notebook for rich data analytics and visualizations. Additionally, we have incorporated Streamlit, an open-source Python framework, to develop a dynamic front-end application that facilitates direct user interaction with our software. This efficient system architecture enhances our ability to deliver actionable ecological insights, supporting sustainable urban forestry decisions. A detailed overview of the technical design is discussed later.

2 Scenario Overview

In this section, we delve into four distinct scenarios to showcase the capabilities and applications of TreeInsights in urban forestry planning and management. By examining these scenarios, we aim to illustrate the comprehensive analytical potential of TreeInsights in fostering sustainable urban development and environmental conservation.

Scenario 1: Overview of Urban Forest Plan
This scenario aims to introduce the extent of urban forest and identify the most common tree species by genus and common name. This analysis utilizes data from the City of Melbourne Open Data portal for urban forest information and the municipal boundary polygon from DATA VIC. The scenario development involves two key steps: first, visualizing the land cover of the urban forest on a map to provide an intuitive spatial representation; second, determining the most numerous tree species and presenting this information in a visually appealing table. This approach offers a clear understanding of the urban forest's scale and diversity.

Scenario 2: One of the benefits of urban trees: Oxygen Production
Urban vegetation, particularly trees, provides numerous benefits that enhance environmental quality and human health in urban areas. Among these benefits, oxygen production is often highlighted (Nowak et al., 2007). This analysis investigates the significance of oxygen production provided by urban forests in every suburb of Melbourne. To conduct the analysis, data were collected from various sources: Urban Forest Data and Suburb Boundary Polygons from CITY OF MELBOURNE OPEN DATA, Municipal Boundary Polygon from DATA VIC, and Population Data From SUDO. The scenario development involves two steps: first, estimating the oxygen production of trees in each suburb; second, estimating how much residents' oxygen consumption would be offset by them.

Methodology: Tree oxygen production varies by tree size. The estimation method of oxygen production by every tree is based on its diameter breast (dbh). According to the data from Minneapolis, (Nowak et al., 2006), trees 1-3" (2.54 - 7.62cm) dbh produced 2.9 kg O2 / year; trees 9-12" (22.86 - 30.48cm) dbh: 22.6 kg O2 / year; 18-21" (45.72 - 53.34cm) dbh: 45.6 kg O2 / year; 27-30" dbh (68.58 - 76.2cm) dbh: 91.1kg O2 / year; and greater than 30" (76.2cm): 110.3 kg O2 / year. In this analysis, the mean value of oxygen production between upper dbh interval and lower dbh interval is used to interpolate the oxygen production of these dbh intervals that were not stated in the literature. However, the estimation error is unknown. Estimation error includes the uncertainty of tree height, tree health and growth.

An average adult human oxygen consumption rate of 0.84 kg / day (306.6 kg / year) (Perry and LeVan c. 2003) was used to multiply the estimated resident population (ERP) and then estimate how much human oxygen consumption would be offset by urban forest oxygen production annually in each suburb (oxygen consumption offset rate = sum of oxygen production by each tree / (average oxygen consumption by residents * population size) * 100%).

Scenario 3: The change in urban forest oxygen production relative to residential population oxygen consumption in 10 years (since 2021)
Greater Melbourne's population has increased from 4.44 million to 5.21 million over the past 10 years, according to ABS data. Understanding how urban forest oxygen production changes based on trees' growth and oxygen consumption by the growing population can inform urban planning and environmental policy for ensuring sustainable development and improved air quality. To conduct this analysis, various data sources were used, including Urban Forest Data and Suburb Boundary Polygons from CITY OF MELBOURNE OPEN DATA, and population data from SUDO. The scenario development involves three steps: first, estimating the oxygen production by trees as their diameter grows; second, estimating the oxygen consumption due to population growth; and third, finding the offset rate by dividing oxygen production by oxygen consumption and utilizing a line plot to visualize the trend.

Methodology: The average dbh growth is 0.61 cm/year in a park-like structure (e.g., parks, cemeteries, golf courses) (deVries, 1987, as cited in Nowak et al., 2007). This growth rate was used along with the estimation of oxygen production criteria from scenario 1 to estimate the oxygen production of urban trees over different years. However, the estimated error is unknown and could be influenced by factors such as tree health and the number of newly planted trees. The average growth rate of residential population in the City of Melbourne from 2009 - 2018 was approximately 2.73% annually based on the census data published by Australian Bureau of Statistics (ABS). This growth rate was used to simulate the population growth in each suburb from 2021 - 2023. The year 2021 was chosen as the starting point for this scenario because the urban tree data were recorded in that year. And the period from 2009 to 2018 was selected as the reference interval to avoid the effects of the pandemic.

Scenario 4: Useful tree life prediction model
Our "Useful Tree Life Prediction" model is a critical scenario within our urban tree forestry analysis application. By leveraging detailed weather data such as rainfall, wind speed, temperature, and relative humidity, this model provides accurate predictions of the useful life expectancy of various tree species within the melbourne victoria government dataset. The weather data we have used in this scenario is from the Melbourne (Olympic Park) weather station as it is the one closest to all the urban trees that we have used for our analysis. With insights on how long a species of tree can survive under given weather conditions, urban planners can optimize green spaces for longevity and ecological benefits. Environmentalists will also be able to leverage this tool to make informed decisions about tree plantation and maintenance.

3 System Architecture & Design

The TreeInsights platform is designed to integrate and analyze environmental data relevant to urban forestry in Melbourne. It manages data from collection to visualization, offering deep insights into how urban greenery interacts with environmental factors.

3.1 System Architecture

The architecture of TreeInsights optimizes data flow, accessibility, robust data management and analytics. The system's components are as follows:

Data Collection: The platform gathers environmental information of real-time weather conditions from the Bureau of Meteorology. Additionally, tree statistics and soil data are sourced from the Melbourne Victoria government data portal, while population data is obtained from SUDO (Spatial Urban Data Observatory).

Data Processing and Storage: Fission, a serverless framework, processes incoming data streams in real-time, transforming and normalizing data before storing it in Elasticsearch. This ensures efficient processing and quick retrieval for analysis. Data Management with Elasticsearch: Elasticsearch acts as the primary repository for processed data, enabling efficient data management and supporting real-time analytics through complex queries.

Kubernetes Deployment: The deployment utilizes Kubernetes to manage the orchestration of containerized services within the system, further the applications are deployed following a serverless architecture using Fission and Elasticsearch. This setup promotes scalability, optimizes resource allocation, and maintains system reliability across different load conditions.

RESTful API Integration: This project incorporates a RESTful API directly within the Jupyter Notebook, which is designed to fetch and manage data from an Elasticsearch database. This integrated approach allows for seamless data manipulation and visualization directly within the analytical environment of the notebook, enhancing the efficiency and accessibility of data-driven insights.

Version Control Git: We have used GitLab as a key collaboration tool which helped us for our project's success. It simplifies efficient code sharing and reviews among team members, enhancing proper communication ,version control and reducing error.

This architecture supports TreeInsights' core functionalities that ensures flexibility for future expansions without compromising performance. The integration of such technologies like Kubernetes, Elasticsearch, and Fission enables the system to handle complex data interactions and provide valuable insights effectively.

3.2 System Design & Implementation

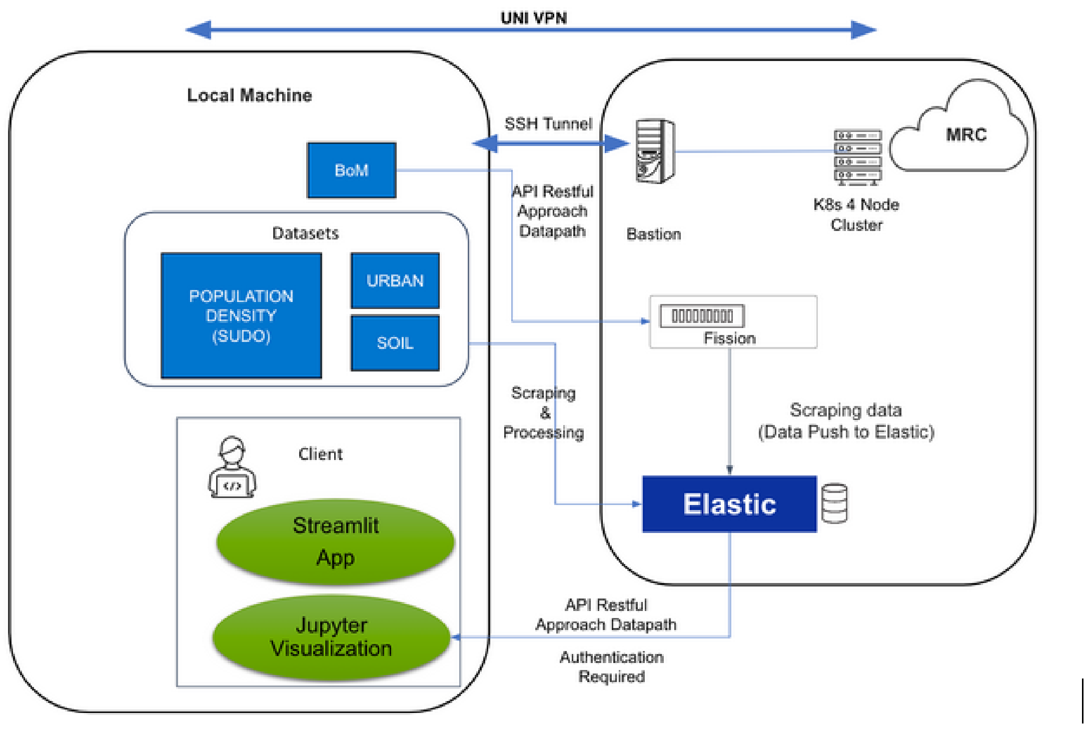


Figure 1: System Design and Architecture Diagram

Figure 1 illustrates the architecture and the data flow within the system that integrates our data collection, processing, storage and visualization. Here is a detailed explanation of the key components of the workflow:

1. **Local Machine:** All developers have to be connected to the University of Melbourne VPN to get secured access to the Melbourne Research Cloud via their terminal. This ensures data transfer between both of the entities. The local machine contains the data that is used for our collection and analysis process. It also acts as a client with tools like Jupyter NB and Streamlit App that is used by the end-user to interact with our application.

2. **Connection:** Once connected to the VPN the main developer who created the cloud infrastructure adds the public keys of other developers to the bastion host. Bastion host is a special-purpose server designed to isolate and secure connection to the private network from an external network. With this bastion connection we perform ssh tunneling, acting as a gateway to our K8 clusters (1 master node and 3 worker nodes) in MRC.

3. **MRC:** The MRC contains the K8 clusters (details in section 4) and applications that run on these clusters. Elasticsearch for data storage and management and Fission that acts as a channel to send data to elasticsearch.

4. **Data Flow:** The Data is collected from various sources (mentioned in section 7) on the local machine. The bastion host paves a pathway for Fission **(port:9090)** to fetch this data (scrap and process) and then send it to Elasticsearch. This is done by using a RESTful API, "bom_fission.py" (details in section 5). This RESTful API uses a request method like GET to fetch data from the BoM using the API key. This data is then stored in Elasticsearch **(port:9200)**. Kibana **(port:5601)** is an open-source data visualization and exploration tool designed to work with Elasticsearch. To use all of these applications, port forwarding from our local machine to the Kubernetes services is necessary. This is done by using a command like kubectl port-forward service/kibana-kibana -n elastic 5601:5601 (for Kibana app).

5. **Data processing and visualization:** The data stored in ES is accessed and fetched using a RESTful design (details in section 9) that is implemented within the Jupyter NB (and streamlit app). This design provides a robust framework for creating web services that are easy to use and scalable. Therefore the client side is exposed to these RESTful services that facilitate effective communication within the clients and the servers. Here, although not necessary, authentication to use these services are implemented within our JupyterNB. Appropriate ES password is required to fetch data from the database. This systematic workflow completes the design of the application.

4 Cloud Infrastructure

The project utilizes the Melbourne Research Cloud which is a Openstack based cloud hosted by the University of Melbourne, it uses Magnum container orchestration via OpenStack and further relies on fission serverless framework to deploy and manage applications like analytics platform. This setup ensures the necessary scalability and resilience for processing large datasets. Essential components of this infrastructure include various instances and volumes dedicated to the collection, processing, and storage of data.

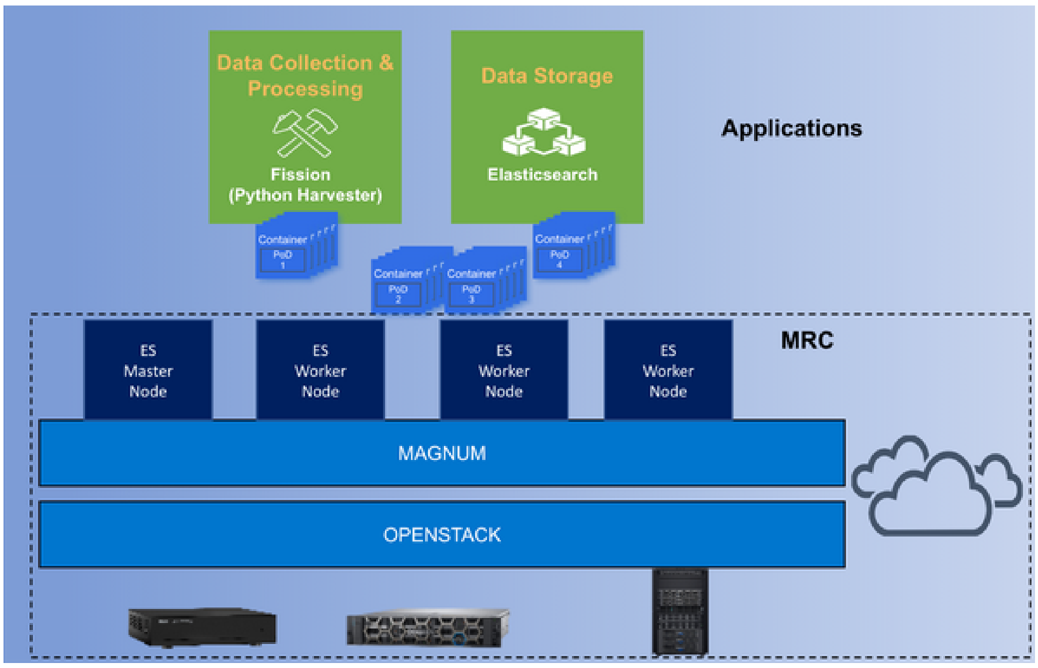


Figure 2: Cloud Infrastructure Diagram

Key Components and Configuration

1. **Openstack:** Openstack provides the underlying infrastructure for MRC. It manages and provisions a large number of VMs and handles scheduling and lifecycle of infrastructure resources.
2. **Magnum:** Magnum is a openstack project that provides a container orchestration engine (COE) for deploying and managing clusters such as Kubernetes and docker. It simplifies the process of setting up these clusters within MRC. By using Magnum, MRC ensures that container clusters are isolated and secure. It is also responsible for setting up a secure connection channel between the master and worker nodes.
3. **Instance Setup:** Our environment includes a bastion host for secure external SSH access, alongside a master node and multiple worker nodes forming a Kubernetes (K8s) cluster

(As seen in the system design diagram). This setup enhances security by controlling external access to the cluster nodes.

4. **Kubernetes Nodes:** The system uses Kubernetes to orchestrate containerized applications. The master node (**elastic-p7bnpocfz4my-master-0**) oversees the cluster's state and coordinates the distribution of tasks across worker nodes (**elastic-p7bnpocfz4my-node-0, -1, -2**), which are hosting applications for data processing.
5. **Volume Management:** Persistent block volumes from OpenStack Cinder are linked to these nodes, ensuring data remains available and consistent through various container operations, which is vital for application reliability and data integrity.

Tools and Processes for Image Creation and Deployment

1. **Image Creation:** Docker is used within the MRC to create, manage, and deploy containerized applications. Docker images are configured with all necessary software and libraries, ensuring consistency across development and production environments.
2. **Deployment Methods:** The integration of Kubernetes and Docker streamlines the deployment mechanism. Kubernetes manifests are utilized to set the desired conditions for Docker containers, which Kubernetes' scheduler then manages automatically. This arrangement leverages Docker's efficient container framework alongside Kubernetes' strong orchestration features.
3. **Helm:** Helm is a package manager for Kubernetes. It manages and deploys K8 applications. eg: Elasticsearch.

Pros and Cons of Using UniMelb Research Cloud

Pros:

- **Customizability and Control:** MRC provides a flexible environment which is suitable for academic research needs, allowing detailed control over computing resources.
- **Cost-Effectiveness:** As a university-managed resource, MRC is typically more cost-effective for academic projects than commercial cloud services.
- **Integration with Academic Resources:** MRC is naturally integrated with other University of Melbourne resources, including data repositories and high-speed academic networks.
- **Security and Compliance:** MRC adheres to strict security protocols and compliance standards, ensuring that sensitive research data is protected and managed according to best practices in data security.

Cons:

- **Resource Limitations:** Unlike commercial providers like AWS or Azure, MRC has limited resources in terms of scalability and available services.
- **Technical Support:** Support might be less immediate compared to commercial clouds, which can offer 24/7 professional support.
- **Feature Set:** MRC may lag behind commercial services in the diversity of advanced cloud features and integrations offered.
- **Ease of use:** MRC services are planned from the ground up , it means we need detailed complex procedures to use them unlike public cloud counterparts where this can be as easy as an API call and few clicks.

5 Backend design

In this section we discuss how the backend infrastructure of our application works. We have used Fission as our main technology and tried to implement an Ansible code for maintenance and logging purposes. This ensures a robust and efficient backend system.

5.1 Fission

Fission is an open-source framework designed to create and manage serverless functions or "microservices" in the Kubernetes environment. It allows developers to implement short-lived functions that deploy without any sort of build and deployment pipeline that is usually used. This significantly reduces the amount of time required to deploy applications. We are also able to use HTTP requests, timers or other event sources to run this function.

In our project we have used fission to collect data from BoM and Mastodon and then fetch and load it into Elasticsearch through a restful manner (using elasticsearch client). This is done by deploying the main harvester function (bom_fission.py). This harvester is packaged and deployed using a YAML specification file. This specification file automates the process of creating packages, functions and event timers. Let's go through the specification files before we touch upon the main harvester function.

YAML Specification Files:

- **Fission-deployment-config.yaml -** This file is created during fission spec initialization. It manages the required settings for deploying applications within the Fission environment. This also helps the deployment align with the serverless architecture. This file contains the main unique id that is used by fission to track and manage resources.
- **Fission-bomdatafetcher.yaml -** This is the main specification YAML file that defines the package, function and the time-trigger. The package is created by using the "bash sbuild.sh" command when we use fission spec apply. The names of the files that need to be packaged are also mentioned, which includes, bom_fission.py, requirements.txt and __init__.py. Then we go on to define the function ('bomdatafetcher'). Information of the ConfigMap and Secrets is also added here so that the main function can call sensitive data like username and password while processing the data to push it to ES. Next the TimeTrigger ('bomdatafetcher-timer') is defined which basically uses a cron scheduler to run the function every 30 minutes, referencing the fission function. Hence every half an hour the data is collected from BoM and pushed to ES.

  All these are defined under the same namespace ('default') that was created at the start ('python'), that uses builder image ('fission/python-builder').
- **route-bomdatafetcher.yaml -** In this file the HTTPTrigger is defined. The configuration that we have used essentially sets up a web-accessible endpoint that, when accessed

via GET requests to '/fetch-bom-data', triggers the 'bomdatafetcher' function. This is a component of a serverless application that executes code in response to web requests.

BoM Harvester Python files:

- **Bom_fission.py -** This is the main harvester python file, which acts as a RESTful API to collect data from BoM and push it to Elasticsearch. This python file is also used to scrap the incoming data.

  This flask application defines a route '/fetch-bom-data' that listens for GET requests, when the request is received the function gets triggered. The function then attempts to fetch weather data from the BoM website using 'requests.get'. The response from the BoM website is parsed through 'BeautifulSoup'. Each row in the observation table is processed to extract the weather data. The function then initializes the elasticsearch client using the url and provided credentials. For each row the function creates a dictionary containing the parsed values. This dictionary is then indexed into the Elasticsearch cluster using 'client.index'. This data is stored in the 'melbourne_weather' index.
- **Requirements.txt -** The text file contains all the packages of python that are required to be installed in the virtual environment while the function is being deployed.
- **build.sh -** This is a shell script used to build the application. It installs the python dependencies that are mentioned in the requirements.txt file.
- **__init__.py -** This file is used to mark a directory as a Python package. It is mainly used for our project testing purposes. The '__all__' list defines the public interface of the package. So when called from the testing file by "from bom_fission import *" these functions in the list that are used in the harvester file are accessible directly.

5.2 Ansible

Ansible is an open-source automation tool used for configuration management, application deployment and task automation. It essentially simplifies repetitive tasks that can be performed through automation. This makes it easier for developers to manage infrastructure at a large-scale.

In our project, our ansible playbook ('report_playbook.yaml') generates reports for Elasticsearch by fetching credentials from secrets and retrieving the cluster health status. The 'inventory.yaml' file defines the ansible connection 'localhost' as the target host with a local connection. Due to lack of time, we have implemented only a monitoring service with Ansible.

6 Database Management

In this section, the database design of utilizing Elasticsearch is explained in our application. Elasticsearch is a powerful search and analytics engine, and its efficient utilization requires careful consideration of mapping structures and data organization.

After collecting the data from various sources, we create a mapping for every dataset. Mappings play an essential role in defining the structure of documents stored in Elasticsearch indices. Utilizing CURL commands, we define mappings for JSON documents to be indexed. These mappings specify the data types, properties, and indexing behavior for each field, ensuring consistency and facilitating efficient search and retrieval operations. For live-streaming data (for example data from Bureau of Meteorology), we mine the data from the website a certain period of time (in the case of Bureau of Meteorology, thirty minutes) and use the same curl command to ensure that the data is inserted into relevant dataset.

We visualise indexed data by the interface provided by Kibana. Furthermore, Kibana's visualization capabilities enable us to gain insights from the indexed data swiftly and intuitively, Usage of Elasticsearch is safe, as every access to Elasticsearch needs keys, passwords or tokens, that's why we can be sure that our database is manipulation-proof.

Pros and Cons of Using ElasticSearch:

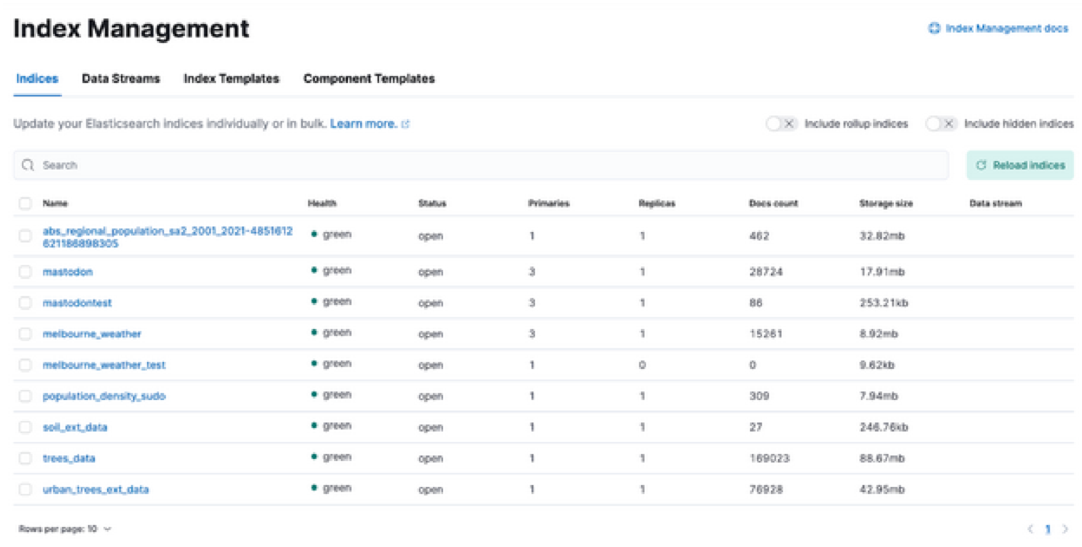
Pros:

- It has integration with other components of the Elastic Stack, including Kibana for visualization and Logstash for data processing which is really great for data management and analytics.
- Its searching capabilities allow complex queries for large volumes of complicated data, easing comprehensive data analysis.
- Elasticsearch is excellent at real-time data ingestion, enabling timely insights and responses to changing data streams.
- Its integration with Kibana, makes it a great tool for visualizing and analyzing the data.

Cons:

- Setting up and configuring Elasticsearch clusters is somehow complex, requiring some expertise, and takes a lot of time. Elasticsearch is not easy to learn for novice users. It should be made more user friendly, and directed to the general population.
- Elasticsearch requires a significant amount of resources, especially in terms of memory and CPU usage, particularly as data volumes grow.
- Ensuring data consistency in the environment is challenging. It requires careful planning and implementation of mappings.

Here is a screenshot of our database. We have created multiple indices that were mapped to the documents being pushed to elasticsearch, either through fission or manually using a python function.



Index Management

Update your Elasticsearch indices individually or in bulk. Learn more.

Name	Health	Status	Primaries	Replicas	Docs count	Storage size	Data stream
abs_regional_population_sa2_2001_2021-4851612621186898305	● green	open	1	1	462	32.82mb	
mastodon	● green	open	3	1	28724	17.91mb	
mastodontest	● green	open	3	1	86	253.21kb	
melbourne_weather	● green	open	3	1	15261	8.92mb	
melbourne_weather_test	● green	open	1	0	0	9.62kb	
population_density_sudo	● green	open	1	1	309	7.94mb	
soil_ext_data	● green	open	1	1	27	246.76kb	
trees_data	● green	open	1	1	169023	88.67mb	
urban_trees_ext_data	● green	open	1	1	76928	42.95mb	

Rows per page: 10

Figure 3: Elasticsearch Database

7 Data Collection

In this section we discuss the sources that were used to collect the data used for our story.

7.1 SUDO Data Collection

SUDO (Spatial Usage Data Observatory) is a system designed by University of Melbourne to collect and analyze spatial usage data for academic or research purposes.

Excessive data mining has been done to analyze the right datasets for our user story, our initial focus for SUDO data sets were on following three user stories.

- **User story#1:** Better Fire Safety planning using Emergency Water supply points.
- **User story#2:** Improve online security by correlating social and financial data.
- **User story#3:** Social Media to improve labor market efficiency.

After deep dataset analysis, we abandoned these user stories as there were not relevant and useful datasets therefore we selected Tree and Population data due to following reasons:

Pros:

- SUDO Tree/Population datasets can easily co-relate with our other data sets BOM.
- SUDO is a good tool for spatial data analytics on Suburb/location basis.

Cons:

- Datasets are old and hence some results may not reflect the reality.
- Many related datasets of interest are not retrieved , it needs to be looked for best use of this data for future research.
- The User Interface is not that user friendly, meaning it takes a considerable amount of time to find the desired dataset.

7.2 Streaming Data Collection

We collected streaming data from BoM from requesting its website and EPA and Mastodon via its API.

- **BoM:** The meteorological data from BoM's website `http://reg.bom.gov.au/vic/observations/melbourne.shtml` is refreshed every 5 minutes and its data can be collected through the utilization of requests, therefore there is no existing API for BoM since there is need for the development of it. See section 8.1 for how the data is processed.

- **EPA:** for EPA, we need to create account from `https://portal365.epa.vic.gov.au/sign-up/` and create subscription , the EPA provides a REST API for each of query with API key.
- **Mastodon:** for Mastodon, we need to apply the API key on ADO , `https://www.ado.eresearch.unimelb.edu.au/` and then we can make REST API to a specific server based on the API key.

7.3 External Data Collection

External data ("urban_trees_ext_data" and "soil_ext_data") were collected from the `https://data.melbourne.vic.gov.au/pages/home/`City of Melbourne government website. These datasets contained the precise number of trees planted around the inner melbourne area and the soil types present in these areas. This was a good enough reason for us to pick these datasets and co-relate not only with each other but also the SUDO and BoM datasets. Although we have talked about using soil data for our analysis, we dropped this scenario due to time constraints.

8 Data Scraping & Storage Mapping

In this section we will go ahead and describe how we have processed all the data-sets that we collected and used for the analytics. We have also described the main attributes that each data contains and their description.

8.1 BoM data

The data from BoM is processed with two different ways:

1. **Sending a request to the html file "`http://reg.bom.gov.au/vic/observations/melbourne.shtml`":** With the help of the libraries BeautifulSoup and requests, the data from the website is fetched and then scraped. While scraping, the coordinates of every station are injected into the data to allow us to correlate the data to spatial (like SUDO) data. The data of 21 stations (see below) are fetched with the attributes (see below) and the corresponding mapping file of BoM data is prepared to push the data to ElasticSearch.

Category	**Attribute**	**Description**	**ES Mapping Type**
Station	station	Name of the observation station	keyword
	latitude	Latitude of the observation station	float
	longitude	Longitude of the observation station	float
Temperature	temp	Ambient temperature	float
	app_temp	Steadman Apparent Temperature	float
	high_temp	Daytime high temperature (6 am to 9 pm)	float
	low_temp	Overnight low temperature (6 pm to 9 am)	float
	dew_point	Temperature for dew to form	float
	delta_t	Delta-T (Wet Bulb Depression)	float
Time	current_date_time	Date and time of the observation	date (date_hour_minute)
	high_time	Time of the "Daytime high temperature"	keyword
	low_time	Time of the "Overnight low temperature"	keyword
	high_wind_gust_time	Time of the "Highest Wind Gust"	keyword
Highest Wind Gust	high_wind_gust_dir	Direction of the Highest Wind Gust	keyword
	high_wind_gust_kmh	Speed (km/h) of the Highest Wind Gust	integer
	high_wind_gust_kts	Speed (knots) of the Highest Wind Gust	integer
Wind	wind_dir	Direction of the wind	keyword
	wind_gust_kmh	Wind gust (km/h) over 3 seconds	integer
	wind_gust_kts	Wind gust (knots) over 3 seconds	integer
	wind_speed_kmh	Wind speed averaged over 10 minutes (km/h)	integer
	wind_speed_kts	Wind speed averaged over 10 minutes (knots)	integer
Rain	rain	Precipitation since 9 am	float
Pressure	press_msl	Atmospheric pressure reduced to mean sea level	float
Humidity	rel_hum	Relative humidity in the air	float

Table 2: Data Attributes For BoM Data

List of stations: Melbourne (Olympic Park), Melbourne Airport, Avalon, Cerberus, Coldstream, Essendon Airport, Fawkner Beacon, Ferny Creek, Frankston Beach, Frankston (Ballam Park), Geelong Racecourse, Laverton, Moorabbin Airport, Point Cook, Point Wilson, Rhyll, Scoresby, Sheoaks, South Channel Island, St Kilda Harbour RMYS, Viewbank

In the end we used the data that only contains Melbourne (Olympic Park). This is because we had a few pivots of our original analysis and hence didn't need the data for other stations. Urban forestry analysis is done within the area of Inner Melbourne, hence only this station is included in this area.

2. **Fetching every station's json file (no longer used):** Within the first way (html web-scraping method), we noticed that there exists some links to json files that show stations data for the last three days. Using these links, we sent requests to these json files. However we dropped this idea of getting the json files, since the table's data (from html) is useful enough when you fetch the data every thirty minutes.

8.2 SUDO data (Population density)

The population data found on SUDO about the population density from ABS (Australian Bureau of Statistics) is downloaded as a JSON file. Then we noticed that the data contains lots of unwanted information (columns) which would create complexity in the data. This complexity also meant that the creation of the mapping would become extreme. Those attributes are removed from the data. Moreover, the data included data from every suburb in Victoria. Therefore, the data from "Rest of Victoria" is deleted, in other words only the data from "Greater Melbourne" stays. Finally, the corresponding mapping of the data for the ElasticSearch is prepared and therefore, the population data is pushed under the "population_density_data" data field in our ElasticSearch server. As the name of the data field suggests, in this data we are interested in the population density compared to other information.

Category	**Attribute**	**Description**	**ES Mapping Type**
ID	id	Id of the data	keyword
	primary_index	Primary index of the data	integer
Location	geometry	(Multi)Polygon of the Statistical Area 2	geo_shape
	state_name_2016	State name of the record (Victoria)	keyword
	sa4_name_2016	Statistical Area 4 name	keyword
	sa3_name_2016	Statistical Area 3 name	keyword
	sa2_name_2016	Statistical Area 2 name	keyword
	gccsa_name_2016	Greater Capital City Statistical Area name	keyword
	state_code_2016	State code of the record	integer
	sa4_code_2016	Statistical Area 4 code	integer
	sa3_code_2016	Statistical Area 3 code	integer
	sa2_maincode_2016	Statistical Area 2 main code	keyword
	gccsa_code_2016	Greater Capital City Statistical Area code	keyword
Count	erp_2001	Estimated Residential Population (2001)	integer
	erp_2011	Estimated Residential Population (2011)	integer
	erp_2021	Estimated Residential Population (2021)	integer
Density	pop_density_2021_people_per_km2	Population density of the suburb (2021)	float
Area	area_km2	Area (km sq.) of the suburb	float

Table 3: Data Attributes For Population Data

8.3 External Data

To facilitate the analysis with more quality of data we added Tree Urban Forest data which is gathered from the Meblourne government website. The data is in JSON format and has information like scientific_name, genus, family, year_planted, date_planted, and the geospatial points. The attributes have been filtered according to the requirements of our analysis in Jupyter NB. The following table gives the overview of the data attribute and mapping for better understanding of the data.

Attribute	Description	ES Mapping Type
com_id	Contains the tree id	integer
common_name	Contains common name used for the tree	keyword
scientific_name	The scientific name of the tree	keyword
genus	Tells the type of genus the tree has	keyword
family	To which family the tree belongs to	keyword
diameter_brest_height	It is the Diameter, breadth, height of the tree	integer
year_planted	The year in which the tree was planted	date (format: 'yyyy')
date_planted	The exact date on which the tree was planted	date (format: 'yyyy-MM-dd')
age_description	The age of the tree from the time it was planted	text
useful_life_expectency	The estimated lifetime of the tree in years (words)	text
useful_life_xpectency_value	The estimated lifetime of the tree in years (number)	integer
located_in	The location where the tree is located	keyword
uploaddata	The date on which the data was uploaded	date (format: 'yyyy-MM-dd')
Coordinate Location (lon,lat)	Contains the coordinate longitude and latitude points	geo_point
latitude	The latitude point of the tree	double
longitude	The longitude point of the tree	double
easting	Vertical lines running from the top to bottom and divide the map from west to east	double
northing	Horizontal lines running left to right and divide the map from north to south	double
Geolocation	Contains the geospatial point with longitude and latitude values	geo_point

Table 4: Data Attributes For Tree Data

8.4 Elasticsearch Queries

System was validated using a number of API calls using CURL, following queries have been validated:

- Get all doc id for melbourne_weather
- Add rain temp and wind speed in doc ID melbourne_weather
- Get records in doc ID melbourne_weather
- Update records in doc ID melbourne_weather
- Query the latitude form Melbourne_weather

In addition we validated using complex queries following functionalities:

1. **Conditional query based on whether rain exists and temperature $>$2:**

   Below cURL command sends a GET request to an Elasticsearch instance to search the melbourne_weather index for documents that have a rain field and a temp field greater than $25.0°C$. It sorts the results by the rain field in descending order, limits the output to

10 documents, and computes the average value of the rain field. The request uses basic authentication and formats the response using jq for readability.

```
curl -X GET -k "https://127.0.0.1:9200/melbourne_weather/_search" \
  --header 'Content-Type: application/json' \
  --data '{
    "query": {
      "bool": {
        "must": {
          "exists": { "field": "rain" }
        },
        "filter": [
          { "range": { "temp": { "gt": 25.0 } } }
        ]
      }
    },
    "sort": [
      { "rain": { "order": "desc" } }
    ],
    "aggs": {
      "average_rain": {
        "avg": {
          "field": "rain"
        }
      }
    },
    "size": 10
  }' \
  --user 'elastic:password' | jq '.'
```

2. **Nested query matching the conditions of particular stations , temperate and date:**

   This command is used to filter and retrieve weather data using nested query . It filters data for a particular station meeting specific temperature criteria more than 1.0°C. By using a boolean query with must conditions,This is particularly useful for narrowing down large datasets to relevant records based on specific conditions.

```
    curl -X GET -k "https://127.0.0.1:9200/melbourne_weather/_search" \
  --header 'Content-Type: application/json' \
  --data '{
    "query": {
        "bool": {
            "must": [
                {
                    "range": {
                        "temp": {
                            "gt": 1.0
                        }
                    }
                },
                {
                    "match": {
                        "station": "Point Cook"
                    }
                }
            ]
        }
    }
}' \
  --user 'elastic:password' | jq '.'
```

9 Frontend design

In our project we have used two frontend softwares to showcase our findings. The Jupyter Notebook is used for most of our scenario analysis. However we have also ventured in wrapping one of our scenarios which is "Useful tree life expectancy prediction" (discussed in section 10) on streamlit app. Streamlit is an open-source Python library designed for creating web applications for machine learning and data science projects. It is easy to use without much web development skills. We have planned to use this App for more dashboards that visualizes key metrics from our datasets. Apps built here can also be deployed and shared with others.

Jupyter Notebook on the other hand is a very convenient tool for deeper exploratory analysis with raw data. We have used this tool for all of our scenarios that required spatial mapping. It is a perfect tool for interactive coding, data analysis and sharing computational workflows.

9.1 RESTFul Design

To efficiently manage and analyze large volumes of data stored in Elasticsearch, we fetched data to our frontend applications through using RESTful methods. This approach allows us to leverage HTTP requests to interact with Elasticsearch, making the process more versatile and easier to integrate with various tools and environments. By using RESTful APIs, we can perform queries, scroll through extensive datasets, and retrieve the necessary information directly into our Jupyter Notebook environment for further analysis.

One of the main reasons we chose to fetch data using RESTful methods is the flexibility it provides. Unlike elasticsearch clients that might need specific setups and dependencies, RESTful APIs use standard HTTP methods like GET and POST. These methods are widely supported and can be easily used with libraries like requests in Python.

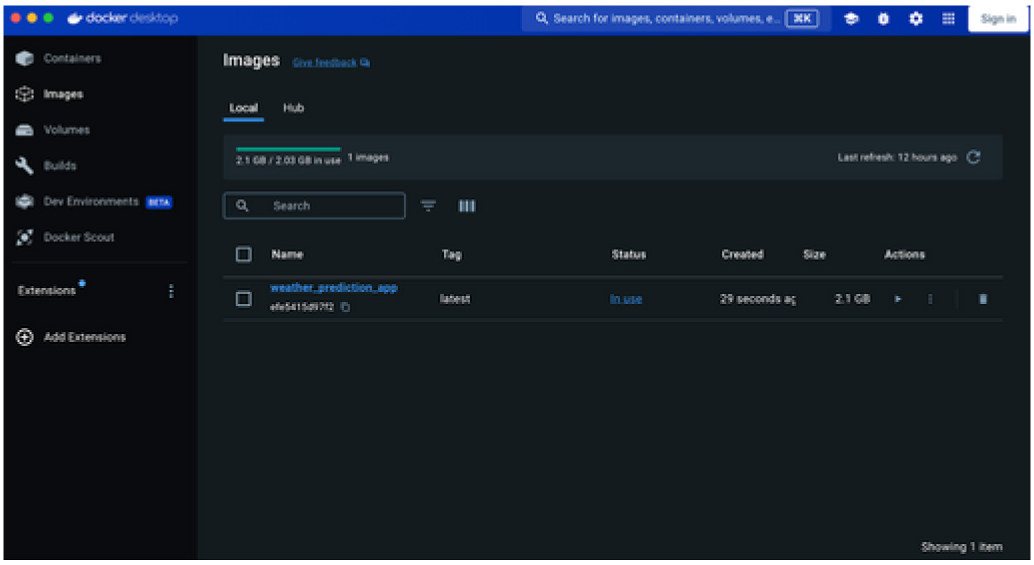
We chose not to use Fission to fetch our data and instead opted for direct interaction with the Elasticsearch url for several reasons. Firstly, interacting directly with the Elasticsearch URL using RESTful methods is straightforward and eliminates the need of an additional abstraction layer, that just adds to the overhead cost. By using RESTFul APIs directly, we can perform our queries more efficiently and with great control. This ensures faster data retrieval.

9.2 Docker

Docker is a platform that allows developers to automate the deployment of applications inside lightweight containers. These containers include everything needed to run the application, such as code, libraries, system dependencies etc.

We have created a dockerfile to define an environment in which our application runs. The docker file specifies the base image, working directory, system dependencies, Python packages and the commands to run our Streamlit app and Jupyter Notebook. Here we exposed ports **8501** and **8888** (8501 for streamlit and 8888 for Jupyter) in the container and while running it, it is essential to map those ports to our local host machine. This ensures that the application is able to fetch data from elasticsearch.

The main use of docker here is to contain our complete application in one single container and create a consistent and a reproducible environment for running our data analytics tasks. This ensures that the application will run smoothly in the user's local machine or even a production environment (given that he has access to the ports and passwords). The user also must have docker installed and enabled to build and run the application.



2.1 GB / 2.03 GB in use 1 images Last refresh: 12 hours ago

Name	Tag	Status	Created	Size	Actions
weather_prediction_app efe5415d97f2	latest	In use	29 seconds ag	2.1 GB	

Figure 4: Docker

10 Data Analytics & Visualization

In this section, we provide a detailed discussion of how we visualized the data.

10.1 Kibana Analytics

Kibana Analytics provides different tools for analysis of data, like creating dashboards, maps, etc for visualization of the data. We used this to create the dashboard for a pictorial view of 'Population Density of Inner Melbourne' as Layer 1 superimposed with 'Tree Density Map View' to get the visual representation of the data and try to find the relationship between the two.

Here are the Kibana plots of our data:

1. **Tree analysis vs population density** shows that "Carlton" and "Parkville" have maximum tree density for population size; this correlation can support proper population planning to meet adequate weather conditions .

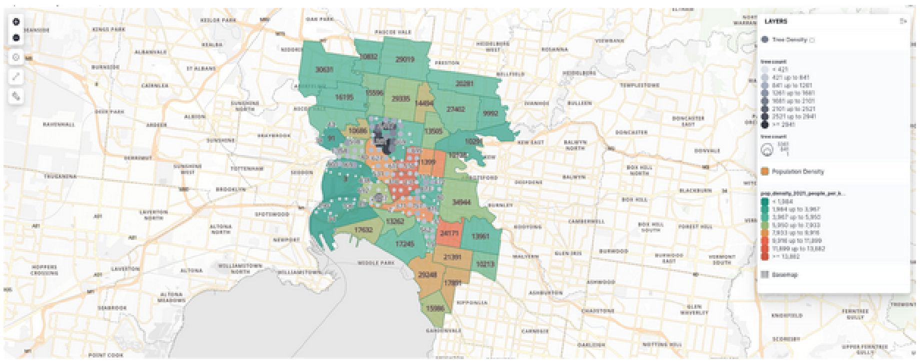


Figure 5: Tree vs Population

2. **The Estimated Residential Population data** from 2001 to 2021 show how drastically the population has increased over time and from this we can also infer that for melbourne city it is the highest increase in population compared to other areas around Melbourne.

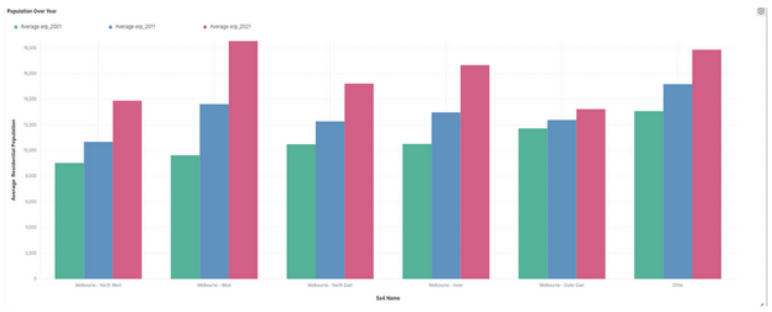


Figure 6: Estimated Residential Population; in 2001, 2011 and 2021

3. **BoM plot for Rain/Wind speed and temperature** explains that periods of high rain are accompanied by high wind speed and low average temperatures.

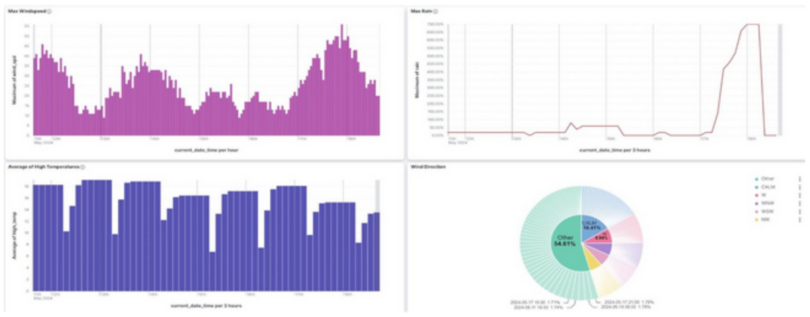


Figure 7: Docker

10.2 JupyterNB Analytics

We will be going through the "Urban_Forestry_Analysis.ipynb" notebook. An overview of Urban Forest Plan in City of Melbourne:

1. **Visualizing the land cover of the Urban Forest Plan**

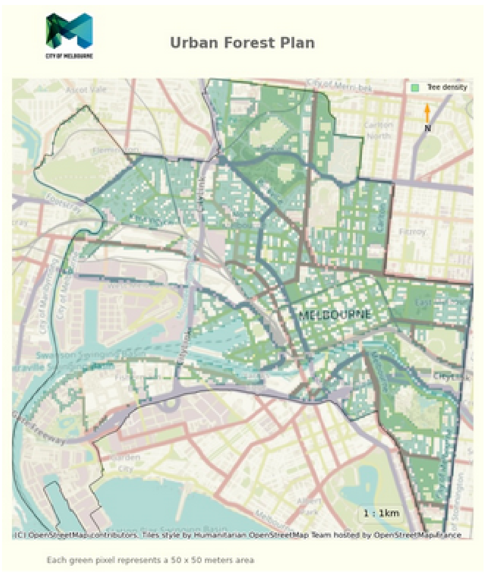
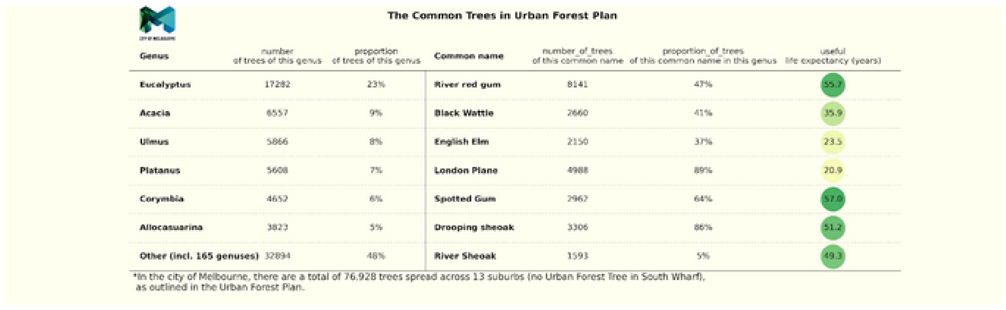

CITY OF MELBOURNE
Urban Forest Plan
Tree density
N
MELBOURNE
1 : 1km
(C) OpenStreetMap contributors. Tiles style by Humanitarian OpenStreetMap Team hosted by OpenStreetMap France
Each green pixel represents a 50 x 50 meters area

Figure 8: Map of Urban Forest

The analysis of urban forest scale in the City of Melbourne reveals distinctive spatial patterns across the cityscape (Map of Urban Forest). The eastern side predominates with extensive coverage, while the western regions, particularly those adjacent to water bodies, exhibit sparse tree planting. Additionally, certain green spaces such as Royal Botanic Gardens, Yarra Park in the south-east, and Royal Park, Melbourne Zoo in the north-east are excluded from the Urban Forest Plan.

2. **Finding the most numerous tree species**



The Common Trees in Urban Forest Plan

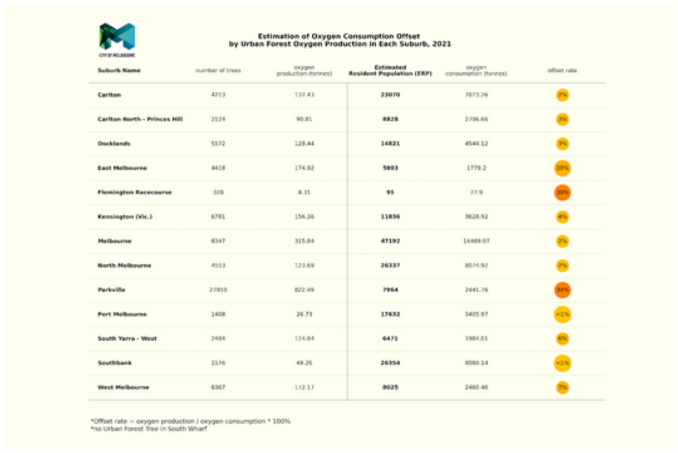
Genus	number of trees of this genus	proportion of trees of this genus	Common name	number_of_trees of this common name	proportion_of_trees of this common name in this genus	useful life expectancy (years)
Eucalyptus	17282	23%	River red gum	8141	47%	55.7
Acacia	6557	9%	Black Wattle	2660	41%	35.9
Ulmus	5866	8%	English Elm	2150	37%	23.5
Platanus	5608	7%	London Plane	4988	89%	20.9
Corymbia	4652	6%	Spotted Gum	2962	64%	57.0
Allocasuarina	3823	5%	Drooping sheoak	3306	86%	51.2
Other (incl. 165 genuses)	32894	48%	River Sheoak	1593	5%	49.3

*In the city of Melbourne, there are a total of 76,928 trees spread across 13 suburbs (no Urban Forest Tree in South Wharf), as outlined in the Urban Forest Plan.

Figure 9: The Common Trees in Urban Forest Plan

Until 2021, 76,928 trees were planted in the City of Melbourne area under Urban Forest Plan. The trees encompass 170 genera and 572 species. Among them, eucalyptus is the most numerous one, comprising 23% of the total. The River Red Gum, a species within the eucalyptus genus, accounts for almost half the number of all eucalypts, and is the most common tree across all species. Additionally its average useful life expectancy is a notable 55.7 years (as seen in the figure, The Common Trees in Urban Forest Plan).

3. **One of the benefits of urban trees: Oxygen Production**



Estimation of Oxygen Consumption Offset
by Urban Forest Oxygen Production in Each Suburb, 2021

Suburb Name	number of trees	oxygen production (tonnes)	Estimated Resident Population (ERP)	oxygen consumption (tonnes)	offset rate
Carlton	4213	137.43	23070	7073.26	2%
Carlton North - Princes Hill	2124	90.81	8828	2706.66	3%
Docklands	5572	128.44	14821	4544.12	3%
East Melbourne	4418	174.92	5803	1779.2	10%
Flemington Racecourse	328	8.31	91	27.9	30%
Kensington (Vic.)	6781	156.26	11836	3628.92	4%
Melbourne	8347	315.84	47192	14469.07	2%
North Melbourne	4553	123.69	26337	8074.92	2%
Parkville	27910	822.49	7964	2441.76	34%
Port Melbourne	1408	26.73	17632	5405.97	<1%
South Yarra - West	2484	124.04	6471	1984.01	6%
Southbank	2176	49.26	26354	8080.14	<1%
West Melbourne	6367	172.17	8025	2460.46	7%

*Offset rate = oxygen production / oxygen consumption * 100%
*no Urban Forest Tree in South Wharf

Figure 10: Estimation of Oxygen Consumption Offset

Oxygen production by trees varies among cities based on differences in the number of trees and their diameter distributions (Nowak et al., 2007). The percentage of residents' oxygen consumption offset by urban forests depends on both the residential population and total oxygen production. Suburbs with high human population densities, such as Melbourne Central and Southbank, tend to have a relatively lower proportion of their oxygen consumption offset by their urban forests (Estimation of Oxygen Consumption Offset). In contrast, suburbs with low residential populations and high tree densities, like Parkville, exhibit the highest offset rates. An acre of trees with 100% canopy coverage can provide enough oxygen for eight people (Nowak et al., 2007).

4. **The change in urban forest oxygen production relative to residential population oxygen consumption in 10 years.**

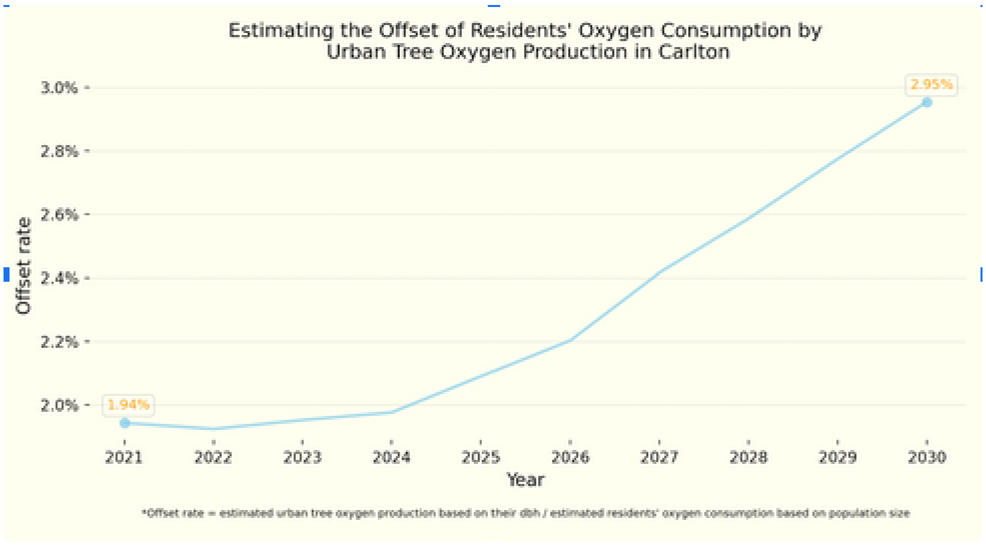
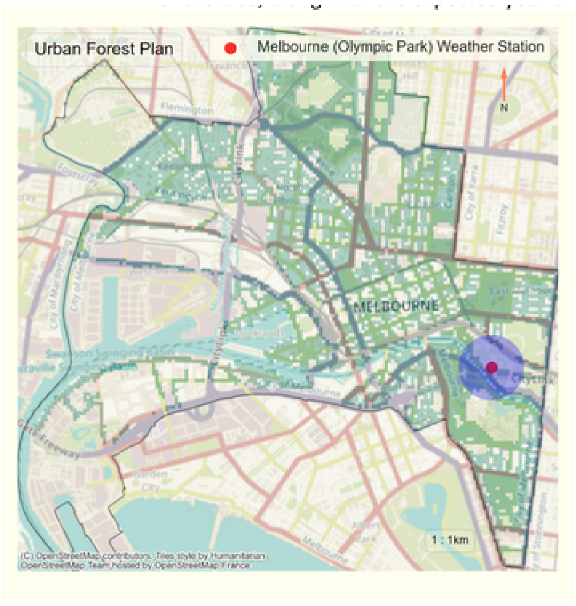


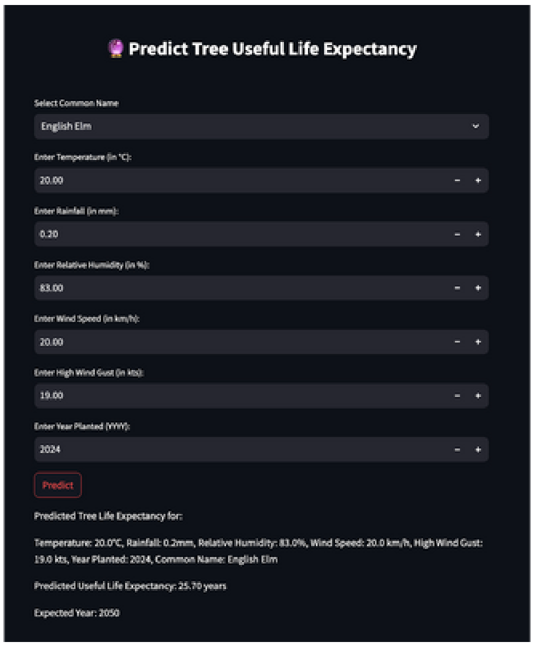
Figure 11: Offset Rate vs. Year

An example from Carlton, figures of all suburbs are stored in the gitlab repository. path: frontend/output Figure 1: This plot shows an increasing tendency of residential population oxygen consumption would be offset by urban forest oxygen production in Carlton annualy. As the diameter of these planted trees grows annually the rate of production of oxygen also grows, based on Nowak estimation. 4

5. Useful Tree Life Prediction Model

As shown below, the map shows the location of the weather station around the inner melbourne area (red dot). The second image is the model that can predict the useful life expectancy of a given species of tree. This prediction is based on the machine learning algorithm called Random Forest Regression. This algorithm works by creating multiple decision trees during the training phase of the model and outputs the mean prediction of individual decision trees. Trees grow under complex and often non-linear conditions influenced by various environmental factors such as weather, soil quality, and urban conditions. Random Forest can effectively capture and model these non-linear relationships, providing more precise predictions. This model is integrated in an interactive tool where the user can input the average weather conditions of the lifespan of the tree to get the predicted useful life expectancy of the tree, along with the expected year of end-life.


Urban Forest Plan
● Melbourne (Olympic Park) Weather Station
N
MELBOURNE
1 : 1km
(C) OpenStreetMap contributors. Tiles style by Humanitarian OpenStreetMap Team hosted by OpenStreetMap France


Predict Tree Useful Life Expectancy

Select Common Name
English Elm

Enter Temperature (in °C):
20.00

Enter Rainfall (in mm):
0.20

Enter Relative Humidity (in %):
83.00

Enter Wind Speed (in km/h):
20.00

Enter High Wind Gust (in kts):
19.00

Enter Year Planted (YYYY):
2024

Predicted Tree Life Expectancy for:

Temperature: 20.0°C, Rainfall: 0.2mm, Relative Humidity: 83.0%, Wind Speed: 20.0 km/h, High Wind Gust: 19.0 kts, Year Planted: 2024, Common Name: English Elm

Predicted Useful Life Expectancy: 25.70 years

Expected Year: 2050

Figure 12: Melbourne Olympic Park Weather Station and Prediction Model UI

11 Fault Tolerance & Testing

In this section we talk about how fault tolerant our software is and what tests we have implemented throughout the development process to ensure its reliability and robustness.

11.1 Fault Tolerance

Based on testing we identified several failover scenarios that should be addressed when deploying the solution in scale , these are as follows:

- Application frontend do not use parallelism , as application host connects to MRC via a single bastion host connection , any impact on this connection will make system not functional.
- Our Kubernetes cluster on MRC using single control node which is a single point of failure for the cluster.
- On application level we created multiple indexes in elastic with different shards , in case of a single shared access by all nodes means it can become a bottleneck when the elastic database size increases.
- For datasets we are connecting to external data sources like BOM etc over internet and reachability to those servers can make system malfunction.

We have not implemented scalability in our system but scalability can impact system availability if the load is not balanced equally between all the nodes.

Both reliability and availability of the system can be improved by considering following factors as seen in Figure below.

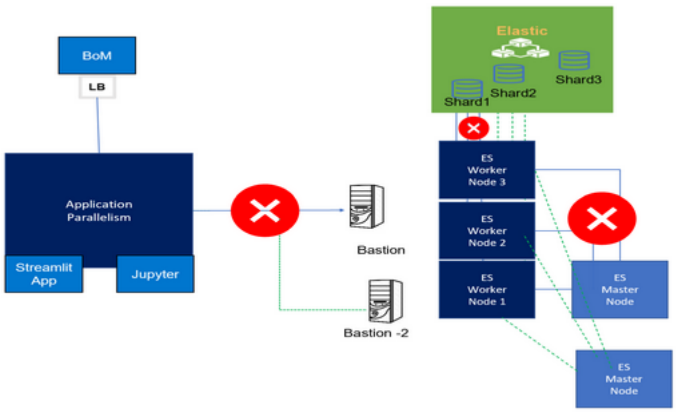


Figure 13: Fault Tolerance

- **Load Balancer:** The load balancer ensures that the failure of one data source or application instance does not overwhelm the system, by rerouting traffic to available instances.
- **Redundant Nodes:** By implementing high availability architecture for master nodes we can enhance availability as redundant nodes can compensate, preventing data loss and service disruption.
- **Sharding:** Data sharding across multiple nodes helps in distributing the load and ensures that the failure of a single shard does not compromise the entire dataset.
- **Bastion Hosts:** The use of multiple Bastion hosts ensures that secure access to the system is maintained even if one Bastion host fails.

11.2 Testing

All software is prone to malfunctions. Testing our software for each leg in our project helps in preventing issues from escalating further. We have tested our projected in multiple ways, listed as follows:

- **Unit Testing:** This is a type of software testing where each component of the software application is tested in isolation from the other parts of the application. In our project, we have unit tested the fission API (bom_fission.py). The fission API contains many functions that are used to parse values from the weather data to handle certain checks. For example the 'test_parse_int' function in the test file tests the 'parse_int' function to ensure it correctly parses integers and handles invalid input. Similarly the 'test_parse_time_valid' function tests the 'parse_time' function to ensure the time is converted into 24-hour format. All the components from the "bom_fission.py" have been tested and passed.
- **Integration testing:** Integration testing tests the entire application from start to finish. In our project, the first Integration testing is used to test how the data flow works from fetching data to Elasticsearch. We have first created a test document just like the weather data that is fetched from Fission. Further we have pushed this document to elasticsearch using HTTP requests and then queried this data. At the end we have deleted the created document. This test file not only checks the correctness of the data flow that we have implemented but also tests the http methods used for the API. A failed test case is also implemented which tries to query a URL with an incorrect document ID, this is necessary to know if our database handles incorrect urls correctly. This type of testing ensures that the entire application stack, including network communication and external systems, work together as expected. We have done the same testing by using es-client.
- **ConfigMap Testing:** The "BoM_ConfigMap_test.py" defines a unit test for verifying the connection to the Elasticsearch instance using the configuration stored in ConfigMap and Secrets within the Kubernetes cluster. We have used functions that handle potential errors such as missing keys, file not found and YAML parsing errors. This ensures that the configuration files are properly read and any issues are identified promptly.

12 Error Handling

This section outlines the errors and roadblocks encountered by each team member during the project and details how these challenges were addressed. We managed to overcome most of our issues by leveraging extensive documentation, utilizing Google for research, and consulting ChatGPT for additional insights.

Once the K8 cluster is set and running on the cloud, the user (i.e the developers of the project) can face errors while connecting to the instance. These errors can occur due to multiple reasons and can cause serious delay in building and running the application.

The following are the errors we faced while interacting with the MRC, and how we solved them.

1. **Connecting to the wrong project:** The MRC instance for each team has a specific project, the one that is default with the name 'pt-80447' and the other with 'unimelb-comp90024-2024-grp-#grp number'. If sourced to the default group the user will encounter the bad-request error as this is an empty project with no instance. So team members have to make sure of the source to the assigned project and download that source file on the local machine.
2. **OpenStack Client not installed on the user machine** Once the K8 cluster is set on the MRC, it is very important the local machine has the required OpenStack Client as the MRC uses OpenStack (client/host) for hosting the K8 cluster. If the user does not have the openstack client installed the user will face the ImportError where the required dependencies are not installed on the local machine.
3. **Config file error for Tunneling via the Bastion node:** Tunneling via the Bastion node can result in an error if the config file which is provided is not in the correct directory ($\tilde{/}$.kube/¡config_file¿). Also it is important to know that the hidden folder ".kube" should be located under the home directory in order for tunneling to work.
4. **VPN connection issue On LINUX:** While working on the project from a remote location the user needs to use the VPN to establish a secure connection to the MRC. For a Linux user if the config file is not set as per the university's VPN guidelines. The user can face the failed VPN connection resulting in unauthorized access to the K8 cluster.
5. **WSL can not connect to VPN:** We used the solution provided but it was finally addressed after we added [WSL2] in our config file.
6. **HTTPS Time Out:** This occurs when the user tries to connect directly to the K8 cluster. As Bastion node is used to create a secured gateway to connect to the K8 cluster, the user has to make sure for every terminal opened it is sourced to the project and is connected to the bastion node.

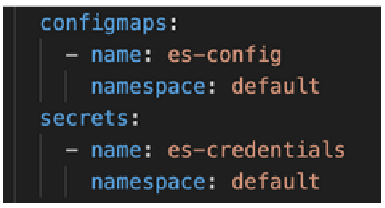
We did not encounter any major roadblocks/errors while setting up fission and Elasticsearch on the cloud. However we were faced with challenges as we started using these technologies specifically for our project.

Here are some roadblocks as follows:

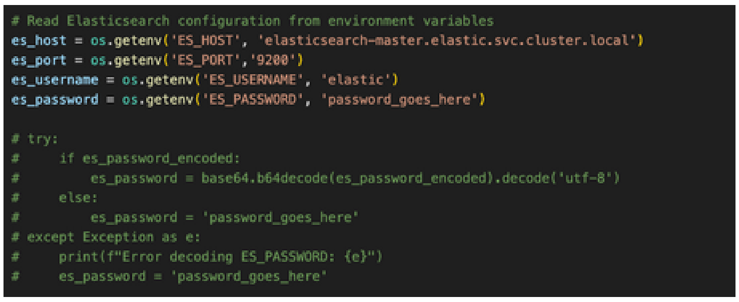
1. **Incorrect deployment of fission:** While testing fission function after applying the fission specs, we encountered numerous issues that related to the RESTful api package itself. The most common was incorrect use of the requirements.txt file. Initially, we were unaware of the python libraries that had to be installed on the virtual machine for the fission api to work as intended.

2. **Build.sh:** During the creation of the fission specs file, we couldn't understand how to run our build file that would install the required python libraries during the process of applying the fission specs using fission spec apply. After digging through some documentation, this was resolved by adding buildcmd: "bash sbuild.sh" command in the source of the spec file: function-bomdatafetcher.yaml.

3. **ConfigMap and Secrets:** We could have used a normal configmap file to map our elasticsearch credentials. However we used ConfigMap only for the url specifics (ES_HOST and ES_PORT) and used Secrets to store the username (ES_USERNAME) and (ES_PASSWORD). Initially when we experimented with ConfigMap we just used all four in the ConfigMap file, this worked on our fission function as expected. Then when we moved on to the current method explained above, we encountered issues that we still couldn't resolve. Our secret file contains a base64 encrypted version of our elasticsearch password. We



```
configmaps:
  - name: es-config
    namespace: default
secrets:
  - name: es-credentials
    namespace: default
```



```
# Read Elasticsearch configuration from environment variables
es_host = os.getenv('ES_HOST', 'elasticsearch-master.elastic.svc.cluster.local')
es_port = os.getenv('ES_PORT','9200')
es_username = os.getenv('ES_USERNAME', 'elastic')
es_password = os.getenv('ES_PASSWORD', 'password_goes_here')

# try:
#     if es_password_encoded:
#         es_password = base64.b64decode(es_password_encoded).decode('utf-8')
#     else:
#         es_password = 'password_goes_here'
# except Exception as e:
#     print(f"Error decoding ES_PASSWORD: {e}")
#     es_password = 'password_goes_here'
```

Figure 14: Error Handling - ConfigMap and Secrets

have this data called in our spec file: function-bomdatafetcher.yaml. And we also tried to encode the password in the fission api after calling it (commented code, line 23 to 30, bom_fission.yaml). Yet, this technique failed, when I tried to print the variables it was showing NULL. This might possibly be due to the incorrect way of storing/calling the ConfigMap and Secrets data in the spec file. In the end, when we progressed through the project, we didn't want to touch our backend once it was already up and running as expected.

4. **ES date-time mapping:** The date-time mapping can be particularly tricky. The formats listed in the Elasticsearch date-time documentation (available here: Elasticsearch Date Formats) are intended for querying data using command languages like KQL within the UI. This does not imply that developers must format their date-time data accordingly when pushing data. It is easy for developers to get confused and apply these formats during the scraping of their date-time attributes. However, all date/time/date-time data should be formatted as date+time, which in our case is '%Y-%m-%dT%H:%M'.

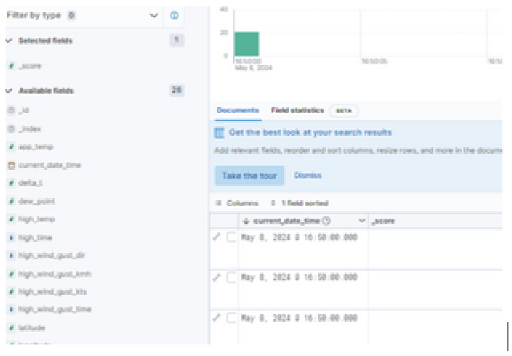
5. **Change in ES password:** Midway through the project, we changed the elasticsearch password on the website, this created havoc as we did not know what we were supposed to edit/re-deploy on our cloud to reflect this password change. Our fission functions stopped functioning. We tried to redeploy our elasticsearch, fission and kibana pods, but that did not work. We then released after a day's research that we had to update the helm and redeploy elasticsearch. So we deleted elasticsearch and redeployed using helm. However we still were faced with one more problem, which was with Kibana explained in the next point.

6. **Kibana re-install issue:** While we tried to use the command given on gitlab (comp90024), helm uninstall kibana -n elastic. This didn't quite work as there were still kubernetes resources like configmap, role, rolebinding, serviceaccount that were used to manage Kibana. So whenever we tried to re-install kibana using helm, we were prompted with errors stating that these resources have been already created. Hence we had to delete these by using commands like kubectl delete rolebinding -n elastic pre-install-kibana-kibana.

7. **Elasticsearch Data Push:** Once the mapping is pushed to the Elastic search for creating the Index, the data needs to be in the same mapping format if not the system throws errors as the index and the data have different format and the data does not get uploaded on the ES. Also as the static data from SUDO is JSON file the size should be less than 100mb to upload it in one go, if it is more than that the data does not get pushed to the ES. While using the Curl commands it's important to note if the mappings are not in bulk api form and we use bulk api parameter in the command it throws request 400 error as wrong use of API. This was then worked around by using a basic python function (export_sudo.py) which used es client to push our data to ES.

8. **Kibana Analytics:**

   - **Join:** in Kibana dashboards we wanted to correlate data from different indices , our aim was to plot the tree data against the melbourne_weather however the dashboard only works if there are common attributes across both data sets , we created a cross reference index as well but it did not give any results , we found the only solution is tonre-index and to re-scrap to build new Index which is not implemented.



Add relevant fields, reorder and sort columns, resize rows, and more in the docum

current_date_time	_score
May 8, 2024 @ 16:50:00.000	
May 8, 2024 @ 16:50:00.000	
May 8, 2024 @ 16:50:00.000	

Figure 15: Kibana UI

   - **Geospatial Plotting:** while creating the map from the kibana analytics it is necessary to have data with geospatial points either on the Elastic search or on the local machine if uploaded directly. If the data field does not have geospatial points the error occurs while trying to create a map and do not have further option to explore on the interface.

13 User Guide

This section provides comprehensive instructions and insights into the functionalities and usage of our project. Whether you are a novice user or an experienced developer, this guide will help you navigate and utilize the various aspects of our project efficiently.

13.1 Useful Links

Click → **Our GitLab Project Repository**
Click → **A Video Demonstration**

13.2 System Deployment & Frontend User Journey

Here is a walkthrough on how to use our software (without docker):

1. **Git Clone** Repository to your local machine
2. Open terminal, cd <project-directory>
3. Create python env: ***python -m venv myenv***
4. Activate env: ***myenv\Scripts\activate*** (or ***source myenv/bin/activate*** on Mac)
5. Go to the /frontend directory
6. Install requirements: **pip install -r requirements.txt**
7. Need to connect to MRC and open port 9200 (after bastion)
8. Run **jupyter notebook** to get access to our NB and use the token provided in the terminal.
9. Run **streamlit run Weather_Prediction_App.py** to run the streamlit application.

14 Team Collaboration

Team Members	**Responsibilities**
Pranav Pai	Cloud Infrastructure / Backend / Analytics / Team - Coordinator
Rongcong Lin	Analytics and Story-Telling / Backend Support
Kaan Gocmen	Elasticsearch / Data Scraping / LaTeX
Abhigyan Singh	Elasticsearch / Kibana Analytics
Saad Sheikh	Data Collection / Kibana Analytics / Report and GitLab Manager

Table 5: Team Collaboration

15 Conclusion

In this project, our team developed a cloud-based analytics application that elucidates the impact of environmental factors, such as weather, as well as population, on urban forestation. Our findings supported the notion that weather patterns directly influence tree health. Additionally, we highlighted the necessity of increasing afforestation rates in response to Melbourne's growing population to maintain a sustainable environment.

We gathered data from multiple sources using REST-based APIs, facilitating the extension of functionality through a modular microservices model. For analytics, we deployed Elasticsearch on a 3-node Kubernetes cluster, providing cluster-level redundancy. However, our application currently faces scaling and redundancy limitations. For our analysis and data visualization, we utilized both Jupyter Notebooks and Streamlit. Jupyter Notebooks allowed us to perform deep exploratory analysis, while Streamlit provided an interactive web-based interface for presenting our findings.

The tool we developed offers invaluable support to a diverse range of professionals. For city planners and urban developers, it provides critical data and predictive insights essential for designing sustainable urban environments, optimizing green space allocation, and planning for future urban expansion. Environmental scientists and ecologists benefit from robust analytical frameworks that facilitate the study of interactions between urban greenery and environmental factors, leading to improved research and policy recommendations. Public health officials and community advocates can utilize the tool to understand the benefits of urban forests, promoting green initiatives that enhance air quality and public health outcomes. Lastly, arborists and urban forestry managers can leverage the predictive model to manage tree populations proactively, ensuring the long-term sustainability and health of urban forests.

In the future, we plan to enhance our analysis by integrating data from Fire and Emergency services, providing a more comprehensive understanding of environmental impacts on urban forests.

References

[1] Nowak, D.J., Hoehn, R., Crane, D.E., Stevens, J.C., & Walton, J.T. (2006). Assessing Urban Forest Effects and Values: Minneapolis' Urban Forest. Resource Bulletin NE-166. U.S. Department of Agriculture, Forest Service, Northeastern Research Station, Newtown Square, PA. 20 pp. https://doi.org/10.2737/NE-RB-166

[2] Perry, J., & LeVan, M.D. (2003). Air Purification in Closed Environments: Overview of Spacecraft Systems. U.S. Army Natick Soldier Center. https://www.researchgate.net/publication/23903409_Air_Purification_in_Closed_Environments_An_Overview_of_Spacecraft_Systems

[3] LaTex: https://www.overleaf.com/1921847849dsxrhqcmtgck#7186d6

[4] Docker Inc. (2024). Docker Project Website. Retrieved April 23, 2024, from https://www.docker.com

[5] Magnum: https://docs.openstack.org/magnum/latest/