VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
HO CHI MINH UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

CAPSTONE PROJECT REPORT

BUILDING AN INTEGRATED DATABASE FOR SHORT TERM FORECASTING SYSTEMS IN HYDRO-METEOROLOGY

Major: Computer Science

THESIS COMMITTEE:	**Council 5**
SUPERVISORS:	**Lê Hồng Trang**
	Trương Quỳnh Chi
REVIEWER:	**Lê Thị Bảo Thu**
	—o0o—
STUDENT 1:	**Trần Hà Tuấn Kiệt (2011493)**
STUDENT 2:	**Nguyễn Đức Thuỵ (2012158)**

Ho Chi Minh City, 5/2024

Declaration

We guarantee that this research is our own, conducted under the supervision and guidance of Assoc. Prof. Dr. Lê Hồng Trang. The result of our research is legitimate and has not been published in any form prior to this. All materials used within this research are collected by ourselves, by various sources and are appropriately listed in the references section. In addition, within this research, we also used the results of several other authors and organizations. They have all been aptly referenced. In any case of plagiarism, we stand by our actions and are to be responsible for it. Ho Chi Minh City University of Technologytherefore is not responsible for any copyright infringements conducted within our research.

Ho Chi Minh City, May 19, 2024

Team Members

Trần Hà Tuấn Kiệt

Nguyễn Đức Thuy

Acknowledgments

We, the research group consisting of two members, would like to send our thanks toward everyone who has contributed to the success of this thesis.

First and foremost, we would like to express our gratitude to Assoc. Prof. Dr. Lê Hồng Trang . The dedication and extensive knowledge of our supervisor not only guided us through the challenges of the project but also helped us develop research skills and critical thinking.

We also want to extend special thanks to all the friends, colleagues, and family who supported us throughout the research process. The input and opinions of everyone enriched the content and quality of the project.

Last, but certainly not least, we appreciate each other - research partners accompanying us in every step of the project. Our collaboration and joint contributions have resulted in a product that we take pride in.

We believe that this project marks a significant step in our development and could not have been achieved without the support and contributions of everyone involved.

Abstract

In this document, we introduce a comprehensive proposal to integrate, coordinate, and monitor meteorological and hydrological data in Vietnam. The proposed system can serve as a foundation for building a national database specializing in meteorological information.

To clarify our proposal, we have built a comprehensive pilot version, in which we have simulated the process of collecting and transmitting data from the weather station at Nha Be. At the same time, we focused on the process of converting and organizing data storage to ensure transparency and optimal performance in information processing.

This research is a manifestation of significant efforts in optimizing the management of meteorological data in Vietnam. The proposed system not only addresses the challenges of integration and coordination but also lays the foundation for a robust national database, serving the diverse needs of research and applications in the field of meteorology.

Work Assignment

No.	Full name	Student ID	Work Assignment
1	Trần Hà Tuấn Kiệt	2011493	- Exploratory Data Analysis - System proposal and architecture - Research and Develop with new technologies - Implement system endpoints
2	Nguyễn Đức Thuỵ	2012158	- Exploratory Data Analysis - System proposal and architecture - Research and Develop with new technologies - Implement data processing flows

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem statement

In recognizing the need for National Meteorological Services (NMHSs) to improve their climate data and monitoring services, there is a deliberate focus on those aspects of climate data management wishing to make the transition to a modern climate database management system and, just as important, on what skills, systems and processes need to be in place to ensure that operations are sustained. In the context of the ever-growing complexity of climate change, the task of creating an integrated database for short-term forecasting systems in the fields of meteorology and hydrology poses a considerable challenge. The question at hand is how we can optimize the management of weather information from multiple sources and store it efficiently in a database. This optimization is crucial to ensure the provision of synchronized and high-quality information to support forecasting systems.

1.2 Motivation

Information about the weather has been recorded in manuscript form for many centuries. The early records included notes on extreme and, sometimes, catastrophic events and also on phenomena such as the freezing and thawing dates of rivers, lakes and seas, which have taken on a higher profile with recent concerns about climate change.

Specific journals for the collection and retention of climatological information have been used over the last two or three centuries (WMO 2005). The development of instrumentation to quantify meteorological phenomena and the dedication of observers to maintaining methodical,

reliable and well-documented records paved the way for the organized management of climate data. Since the 1940s, standardized forms and procedures gradually became more prevalent and, once computer systems were being used by NMHSs, these forms greatly assisted the computerized data entry process and consequently the development of computer data archives. The latter part of the twentieth century saw the routine exchange of weather data in digital form and many meteorological and related data centers took the opportunity to directly capture and store these in their databases. Much was learned about automatic methods of collecting and processing meteorological data in the late 1950s, a period that included the International Geophysical Year and the establishment of the World Weather Watch. The WMO's development of international guidelines and standards for climate data management and data exchange assisted NMHSs in organizing their data management activities and, less directly, also furthered the development of regional and global databases. Today, the management of climate records requires a systematic approach that encompasses paper records, microfilm/microfiche records and digital records, where the latter include image files as well as the traditional alphanumeric representation.

Before electronic computers, mechanical devices played an important part in the development of data management. Calculations were made using comptometers, for example, with the results being recorded on paper. A major advance occurred with the introduction of the Hollerith system of punch cards, sorters and tabulators. These cards, with a series of punched holes recording the values of the meteorological variables, were passed through the sorting and tabulating machines enabling more efficient calculation of statistics. The 1960s and 1970s saw several NMHSs implementing electronic computers and gradually the information from many millions of punched cards was transferred to magnetic tape. These computers were replaced with increasingly powerful mainframe systems and data were made available online through developments in disk technology.

Aside from advances in database technologies, more efficient data capture was made possible through the mid-to-late 1990s with an increase in automatic weather stations (AWSs), electronic field books (i.e. on-station notebook computers used to enter, quality control and transmit observations), the Internet and other advances in technology. Not surprisingly, there are a number of trends already underway that suggest there are many further benefits for NMHSs in managing data and servicing their clients. The Internet is already delivering greatly improved data access capabilities and, providing security issues are managed, we can expect major op-

portunities for data managers in the next five to ten years. In addition, Open Source7 relational database systems may also remove the cost barriers to relational databases for many NMHSs over this period.

1.3 Project Goal

It is essential that both the development of climate databases and the implementation of data management practices take into account the needs of the existing, and to the extent that it is predictable, future data users. While at first sight this may seem intuitive, it is not difficult to envisage situations where, for example, data structures have been developed that omit data important for a useful application or where a data centre commits too little of its resources to checking the quality of data for which users demand high quality.

In all new developments, data managers should either attempt to have at least one key data user as part of their project team or undertake some regular consultative process with a group of user stakeholders. Data providers or data users within the organization may also have consultative processes with end users of climate data (or information) and data managers should endeavour to keep abreast of both changes in needs and any issues that user communities have. Put simply, data management requires awareness of the needs of the end users.

At present, the key demand factors for data managers are coming from climate prediction, climate change, agriculture and other primary industries, health, disaster/emergency management, energy, natural resource management (including water), sustainability, urban planning and design, finance and insurance. Data managers must remain cognizant that the existence of the data management operation is contingent on the centre delivering social, economic and environmental benefit to the user communities it serves. It is important, therefore, for the data manager to encourage and, to the extent possible, collaborate in projects which demonstrate the value of its data resource. Even an awareness of studies that show, for example, the economic benefits from climate predictions or the social benefits from having climate data used in a health warning system, can be useful in reminding senior NMHS managers or convincing funding agencies that data are worth investing in. Increasingly, value is being delivered through integrating data with application models (e.g. crop simulation models, economic models) and so integration issues should be considered in the design of new data structures.

1.4 Project Scope

This project focuses on Ho Chi Minh City, a sprawling metropolis with a distinct climate that significantly impacts the daily lives of its residents. To provide the most valuable weather insights, we will implement a two-pronged approach.

Firstly, We will conduct research and collect data from the meteorological and hydrological monitoring stations in the city to ensure diversity and representation of local weather conditions. At the same time, we will proceed with preprocessing and standardizing the data to ensure accuracy and consistency of the dataset. Based on what we have collected, we will populate the data to proper formats and structures.

Secondly, this project entails the development of a comprehensive data platform. This platform will integrate a central database, designed to efficiently store and manage weather information from diverse sources. By consolidating these disparate data streams, we aim to create a unified and dependable resource for weather data. This platform is anticipated to significantly enhance the workflow of end-users and will be built with scalability in mind to accommodate future growth in data volume and user demand.

1.5 Stakeholders

1.5.1 The National Center for Hydro-meteorological Forecasting

The National Center for Hydrometeorological Forecasting (NCHMF), abbreviated for "Trung tâm Dự báo khí tượng thuỷ văn quốc gia" in Vietnamese, is an organizational unit under the General Department of Meteorology and Hydrology, Ministry of Natural Resources and Environment[4]. The National Hydro-Meteorological Forecasting Center has several crucial missions, including the establishment and presentation of standards and technical regulations for meteorological and hydrological forecasting, the operation of the national forecasting and warning system, monitoring and reporting on weather conditions and climate change, issuing and disseminating forecast bulletins and warnings, and participating in international meteorological agreements. Additionally, the center is responsible for conducting research, application, and technology transfer related to forecasting and warning, and implementing administrative reform and anti-corruption measures. These key missions contribute significantly to the center's

role in ensuring public safety and providing timely essential meteorological and hydrological information.

1.5.2 Nha Be Weather Radar Station

The Nha Be weather radar station is located on Nguyen Van Tao Street, Long Thoi Commune, Nha Be District, Ho Chi Minh City. This radar station has been in operation since 2004 and is managed by the Southern Region Hydro-Meteorological Center. Its mission is to monitor and supervise weather phenomena within a radius of 480 km from the center of Ho Chi Minh City, and to provide warnings and forecasts for dangerous weather conditions such as storms, tropical depressions, and thunderstorms within a radius of 300 km. Additionally, the station serves the purpose of forecasting rain and heavy rain for the Ho Chi Minh City area within a radius of approximately 120 km.

The Nha Be weather radar station is of the Doppler DWSR-2500C type, operating on the C-band frequency. It is manufactured by the U.S. company EEC and has the capability to scan signals within a radius of 300 km. In addition to being the forecasting center for the entire Southern region and Ho Chi Minh City, the Nhà Bè radar station also has the task of predicting rain and providing data for the flood control center of Ho Chi Minh City.

1.6 Solutions

Our product, the Weather Data Platform (WDP), is designed to be a one-stop shop for harnessing the power of weather data. It caters to the specific needs of a wide range of users, including meteorologists, academics, researchers, and developers from various fields. Whether you're a freelancer, a large corporation, or a non-profit organization, WDP can be your central hub for integrating, analyzing, and visualizing weather data – and much more.

1.7 Achievements and Contribution

Chapter 2

Theoretical basis

2.1 Database system and database management system

2.1.1 Traditional Database Systems

Database systems perform vital functions for all sorts of organizations because of the growing importance of using and managing data efficiently. A database system consists of a software, a database management system (DBMS) and one or several databases. DBMS is a set of programs that enables users to store, manage and access data. In other words, the database is processed by DBMS, which runs in the main memory and is controlled by the respective operating system.

A database is a logically coherent collection of data with some inherent meaning and represents some aspects of the real world. A random assortment of data cannot be referred to as a database. Databases draw a sharp distinction between data and information. Data are known facts that can be recorded and that have implicit meaning. Information is data that has been organized and prepared in a form that is suitable for decision-making. Shortly information is the analysis and synthesis of data. The most fundamental terms used in the database approach are identity, attributes and relationships. An entity is something that can be identified in the user's work environment, something that the users want to track. It may be an object with a physical or conceptual existence. An attribute is a property of an entity. A particular entity will have a value for each of its attributes. The attribute values that describe each entity become a major part of the data stored in the database.

Database Management System is a general-purpose software system designed to manage

large bodies of information facilitating the process of defining, constructing and manipulating databases for various applications. Specifying data types, structures and constraints for the data to be stored in the database is called defining a database. Constructing the database is the process of storing data itself on some storage medium that is controlled by the DBMS. Querying to retrieve specific data, updating the database to reflect changes and generating reports from the data is the main concept of manipulating a database. The DBMS functions as an interface between the users and the database ensuring that the data is stored persistently over long periods, independent of the programs that access it [9]. DBMS can be divided into three subsystems; the design tools subsystem, the run-time subsystem and the DBMS engine.

The design tools subsystem has a set of tools to facilitate the design and creation of the database and its applications. Tools for creating tables, forms, queries and reports are components of this system. DBMS products also provide programming languages and interfaces to programming languages. The run-time subsystem processes the application components that are developed using the design tools. The last component of DBMS is the DBMS engine which receives requests from the other two components and translates those requests into commands to the operating system to read and write data on physical media [3].

The database approach has several advantages over traditional file processing in which each user has to create and define files needed for a specific application. In these systems' duplication of data is generally inevitable causing wasted storage space and redundant efforts to maintain common data up-to-date. In the database approach data is maintained in a single storage medium and accessed by various users. The self-describing nature of database systems provides information not only about the database itself but also about the database structure such as the type and format of the data. A complete definition and description of database structure and constraints, called meta-data, is stored in the system catalog. Data abstraction is a consequence of this self-describing nature of database systems allowing program and data independence. DBMS access programs do not require changes when the structure of the data files is changed hence the description of data is not embedded in the access programs. This property is called program-data independence. Support of multiple views of data is another important feature of database systems, which enables different users to view different perspectives of databases depending on their requirements. In a multi-user database environment, users probably have access to the same data at the same time as well as they can access different portions of the

database for modification. Concurrency control is crucial for a DBMS so that the results of the updates are correct. The DBMS software ensures that concurrent transactions operate correctly when several users are trying to update the same data.

Using a DBMS also eliminates unnecessary data redundancy. In the database approach, each primary fact is generally recorded in only one place in the database. Sometimes it is desirable to include some limited redundancy to improve the performance of queries when it is more efficient to retrieve data from a single file instead of searching and collecting data from several files, but this data duplication is controlled by DBMS to prohibit inconsistencies among files. By eliminating data redundancy inconsistencies among data are also reduced [3]. Reducing redundancy improves the consistency of data while reducing the waste in storage space. DBMS allows data sharing to the users. Sharing data often permits new data processing applications to be developed without having to create new data files. In general, less redundancy and greater sharing lead to less confusion between organizational units and less time spent resolving errors and inconsistencies in reports. The database approach also permits security restrictions. In a DBMS different types of authorizations are accepted to regulate which parts of the database various users can access or update.

2.1.2 A Modern Approach to Data Systems

In contemporary computing environments, the dominance has shifted towards data-intensive applications, deviating from the traditional emphasis on compute-intensive tasks. The limiting factor for these applications seldom resides in the sheer computational power of the CPU; rather, the primary challenges typically revolve around the magnitude of the data, its intricate structures, and the rapidity with which it changes. Unlike compute-intensive operations that heavily rely on processing speed, data-intensive applications, dealing with extensive datasets, intricate data structures, or swiftly evolving information, necessitate adept strategies for storage, retrieval, and manipulation. Consequently, effectively addressing the multifaceted dynamics of data becomes paramount, highlighting the imperative for sophisticated data management and processing techniques to optimize performance in the face of these intricate challenges.

Why should we amalgamate these diverse elements within the overarching label of data systems? Recent years have witnessed the emergence of a plethora of novel tools for data storage and processing, each meticulously optimized for an array of distinct use cases [14]. Consider,

for instance, datastores that concurrently function as message queues (e.g., Redis) or message queues equipped with database-like durability assurances (such as Apache Kafka). The demarcation lines between these categories are progressively fading, reflecting a landscape where boundaries are increasingly ambiguous.

Moreover, a growing number of applications now present challenges of such magnitude or diversity that a solitary tool is no longer sufficient to fulfill all its data processing and storage requisites. Instead, the workload is deconstructed into tasks amenable to efficient execution by individual tools. These disparate tools are then intricately interwoven using application code, offering a nuanced and adaptable approach to the multifaceted demands of contemporary data management and processing.

In navigating the complex landscape of application development, the quest for reliability, scalability, and maintainability unveils a challenging yet essential journey. As we delve into the intricate patterns and techniques that permeate various applications, we embark on an exploration to fortify these foundational pillars in the realm of software systems.

Ensuring reliability involves the meticulous task of ensuring systems function correctly, even when confronted with faults. These faults may manifest in the hardware domain as random and uncorrelated issues, in software as systematic bugs that are challenging to address, and inevitably in humans who occasionally err. Employing fault-tolerance techniques becomes imperative to shield end users from specific fault types.

Scalability, on the other hand, necessitates the formulation of strategies to maintain optimal performance, particularly when facing heightened loads. To delve into scalability, it becomes essential to establish quantitative methods for describing load and performance. An illustrative example is Twitter's home timelines, which serve as a depiction of load, and response time percentiles providing a metric for measuring performance. In a scalable system, the ability to augment processing capacity becomes pivotal to sustaining reliability amidst elevated loads.

The facet-rich concept of maintainability essentially revolves around enhancing the working experience for engineering and operations teams interacting with the system. Thoughtfully crafted abstractions play a crucial role in mitigating complexity, rendering the system more adaptable and modifiable for emerging use cases. Effective operability, characterized by comprehensive insights into the system's health and adept management methods, also contributes to maintainability.

Regrettably, there exists no panacea for achieving instant reliability, scalability, or maintainability in applications. Nonetheless, discernible patterns and recurring techniques emerge across diverse application types, offering valuable insights into enhancing these critical attributes.

Data Transformation:

Data Transformation is a vital part of the data flow process, where data changes to meet specific requirements of the process or system. There are two main directions for implementing data transformation: batch processing and real-time processing.

In batch processing mode, data is processed in batches, often scheduled for processing at predefined intervals. This is suitable for tasks requiring the processing of large and complex datasets without an immediate response.

On the contrary, real-time processing is the method of processing data as soon as it arrives, without waiting for a large amount of data to accumulate. This is often preferred in applications demanding low latency, such as real-time event processing.

ETL:

ETL, an acronym for Extract, Transform, Load, stands as a fundamental methodology indispensable for orchestrating the seamless movement of data within storage ecosystems. This three-step process plays a pivotal role in shaping the lifecycle of data.

Firstly, in the "Extract" phase, data is sourced from diverse origins, ranging from databases to files and online services. This initial step lays the groundwork by retrieving relevant information from the varied reservoirs of data.

Subsequently, in the "Transform" phase, the extracted data undergoes a metamorphosis to align with the specific requirements of the target system. This transformative stage encompasses tasks such as data cleansing, format conversions, and even the computation of novel indices, ensuring the data is refined and tailored to suit its intended purpose.

Finally, the "Load" phase marks the culmination of the ETL process. At this juncture, the meticulously transformed data finds its destination, being loaded into the designated storage system. This storage system typically takes the form of a data warehouse or data lake, serving as the repository for the refined and purpose-adapted data. In essence, ETL encapsulates a

systematic and indispensable approach to managing the intricate journey of data within the expansive realm of storage systems.

Data Pipe:

A data pipe is an essential concept in deploying effective data flow. Built on the idea of automating the movement and transformation of data, data pipes serve as powerful workflow streams.

Through the use of data pipes, large data volumes can be processed flexibly and efficiently. Tasks such as error handling, performance monitoring, and even deploying new transformations can be automated, minimizing manual intervention and enhancing system stability.

Data Orchestration:

Data Orchestration is the intricate process of coordinating and managing multiple data processes, workflows, or services to achieve specific outcomes. At its core, it involves the meticulous definition and management of workflows, determining the sequential order, and dependencies among various data processes. Task execution is finely tuned through controllers that schedule and orchestrate tasks at optimal times and in a precise sequence, ensuring the desired outcomes are achieved. Controllers play a pivotal role in managing dependencies between tasks, orchestrating the execution of tasks only when their dependent tasks are successfully met. Robust orchestration systems provide comprehensive tools for monitoring workflow progress and logging pertinent information, offering insights to address and rectify any potential issues. Moreover, controllers optimize performance by strategically breaking down complex tasks into multiple subtasks, executing them in parallel to enhance overall system efficiency.

2.2 Basic of meteorology

Meteorology is the scientific study of the atmosphere that focuses on weather processes and forecasting. It involves the study of the behavior, dynamics, and physical properties of the earth's atmosphere and how these factors affect the weather and climate. Meteorologists use various tools and techniques, including weather satellites, radars, and computer models, to predict weather conditions and to understand the complex interactions within the atmosphere.

Meteorology is important for several reasons, such as predicting weather to warn of severe conditions, understanding climate change, and aiding in decision-making for agriculture, aviation, and other industries dependent on weather.

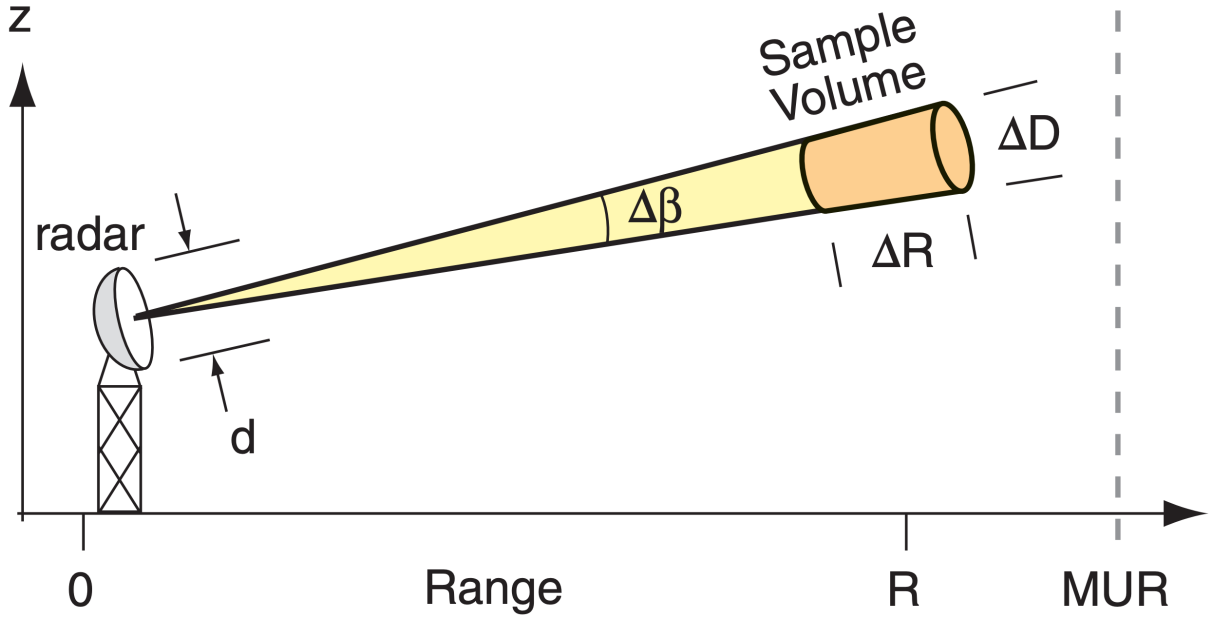


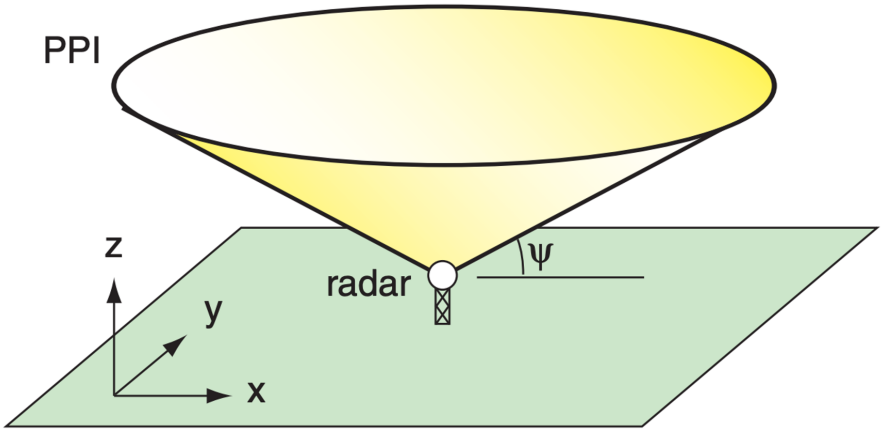
Figure 2.1: *A typical meteorological radar [15]*

2.2.1 Basic terminologies

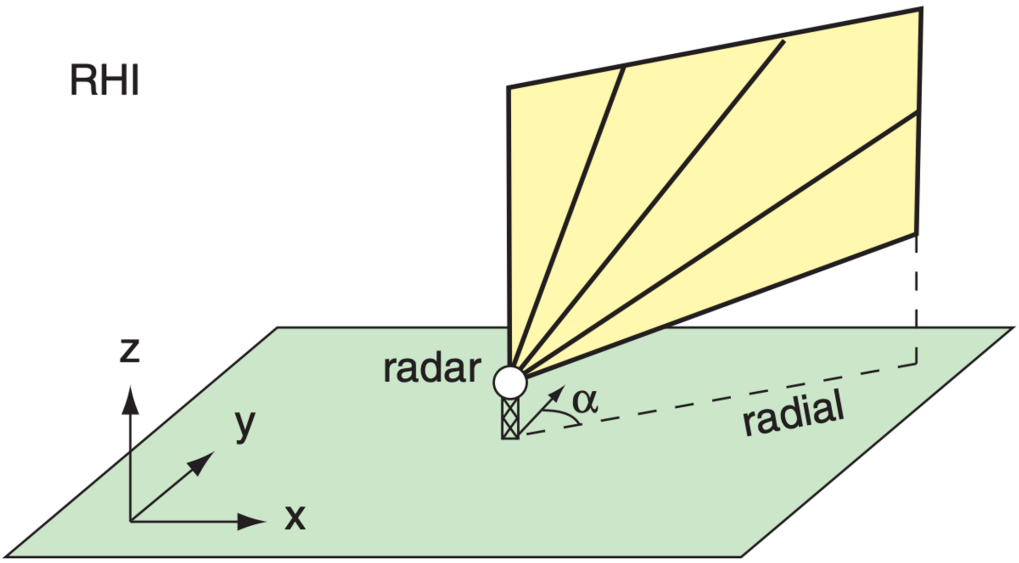
2.2.1.1 Weather Radar

Weather radar, short for weather surveillance radar, is a type of radar system used to detect and monitor precipitation, as well as other atmospheric phenomena such as the movement of severe weather systems. It plays a crucial role in meteorology and helps meteorologists track and forecast weather conditions. Normally, weather radars are programmed to scan in an azimuth of 360^o. For every round, the radar will scan at a different altitude. It usually takes about four to ten minutes for the radar to complete a full scan.

For PPI representation, the radar will scan the entire azimuth, but only at a certain altitude. The final result would be similar to a map on a flat surface. For RHI, in contrast, the radar retains the azimuth but increases in altitude. The collected result gives viewers more details about the height and sizes of a meteorologist event.



(a) *Plan-Position Indicator - PPI - [15]*



(b) *Range Height Indicator - [15]*

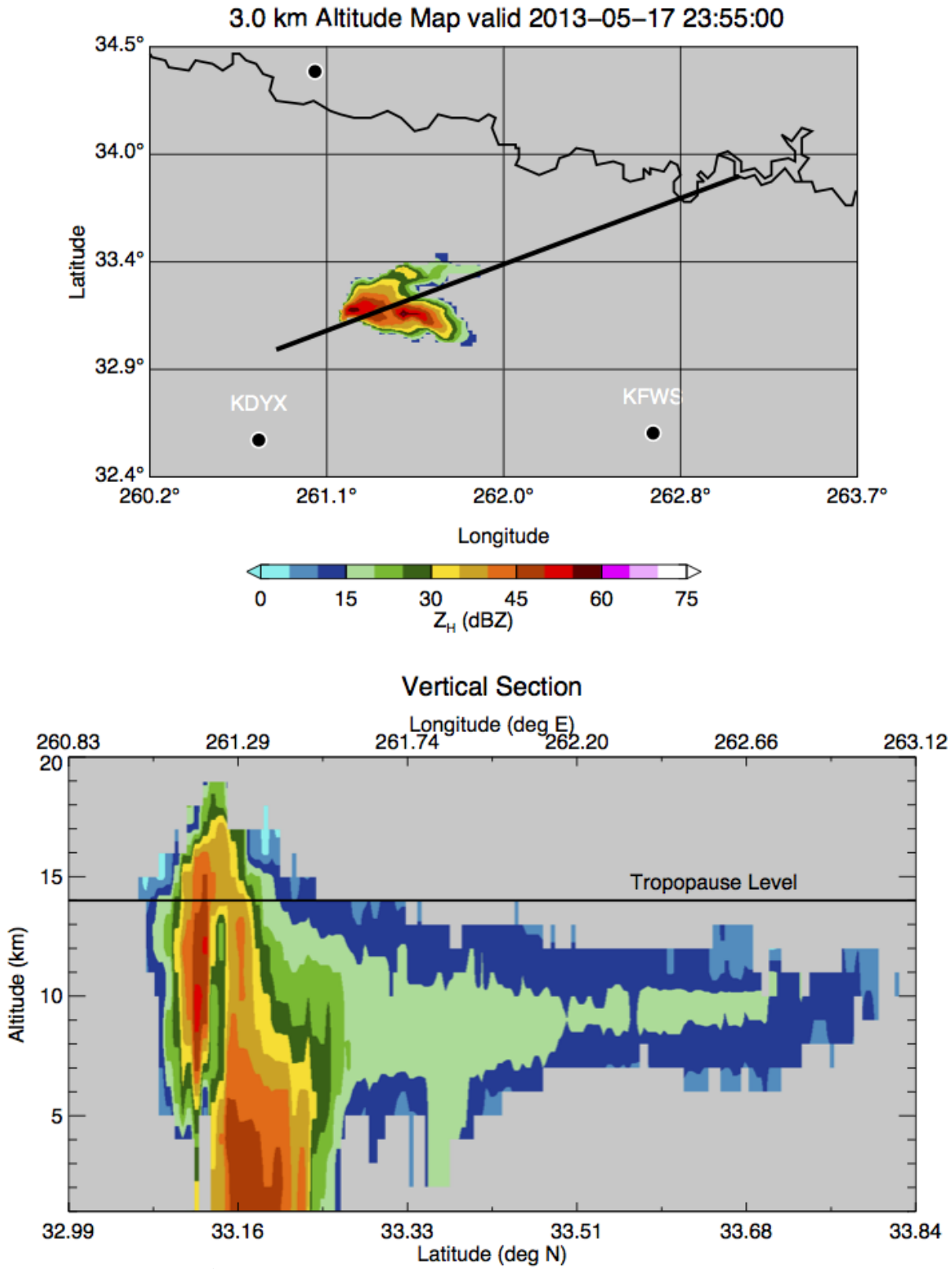


Figure 2.3: *Comparing the result between PPI and RHI - [2]*

2.2.1.2 Radar equation and Reflectivity

At a certain point in time, weather radar will emit a short pulse of radio wave ($\Delta t = 0.5 - 10\mu s$). Depending on the density of free molecules in the air (water vapor, smoke, ...), the energy of this wavelength will be partially absorbed. The wavelength intensity that the radar receives will be less than the intensity of the original wave. This ratio is expressed through **The radar equation** [15]:

$$\left[\frac{P_R}{P_T}\right] = [b] \cdot \left[\frac{|K|}{L_a}\right]^2 \cdot \left[\frac{R_1}{R}\right]^2 \cdot \left[\frac{Z}{Z_1}\right]$$

Which, the variables of the equation include:

- $|K|$ unitless:
  - $|K|^2 \approx 0.93$ for droplets
  - $|K|^2 \approx 0.208$ for ice crystal
- $R(\text{km})$: distance from the radar to the target
- $R_1 = \sqrt{Z_1 \cdot c \cdot \Delta t / \lambda^2}$: ratio of distance
- Z: Radar's reflectivity
- $Z_1 = 1\ \text{mm}^6\ \text{m}^{-3}$: Radar's unit reflectivity

From the radar equation, we can derive the formula for reflectivity:

$$\text{dBZ} = 10\left[\log\left(\frac{P_R}{P_T}\right) + 2\log\left(\frac{R}{R_1}\right) - 2\log\left|\frac{K}{L_a}\right| - \log(b)\right]$$

Meteorologists are usually interested in this number because it is proportional to the amount of precipitation.

Value (dBZ)	Weather
-28	Haze
-12	Clear air
25 - 30	Dry snow / light rain
40 - 50	Heavy rain
75	Giant hail

Table 2.1: *Relation between reflectivity and precipitation - Stull [15]*

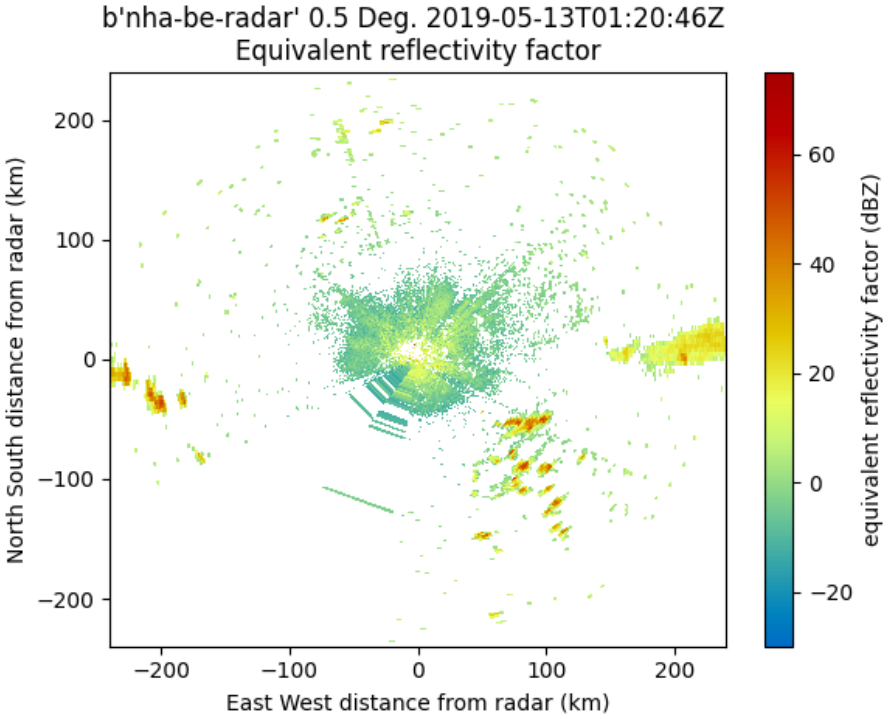


Figure 2.4: *Reflectivity from Nha Be radar*

2.2.1.3 Radial velocity

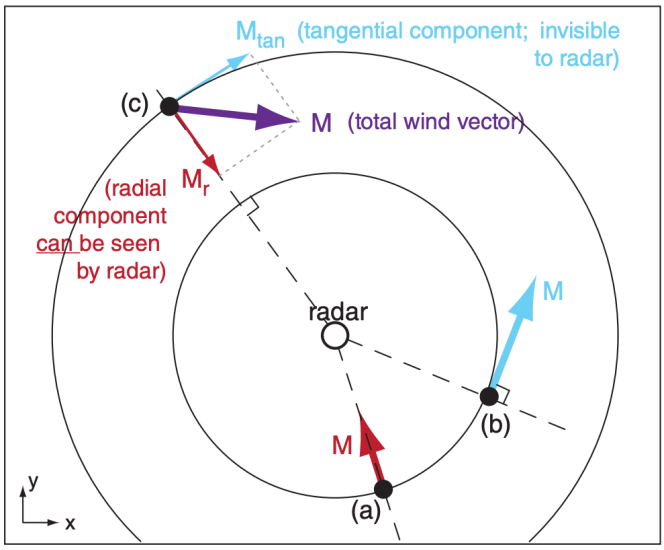


Figure 2.5: *Illustration for the velocity situations that a Doppler radar can observe. (a) When the wind direction at point M coincides with the radius of the circle centered at the radar, the radar can determine the velocity at this point. (b) When the wind direction is tangent to the circle, the radar cannot determine the velocity. (c) Analyzing the wind direction at M into two perpendicular velocities, the radar can only determine the velocity vector along M_r. - Stull [15]*

When the radio waves from these Doppler radars propagate to the molecules in the air, the displacement of these particles causes a phase shift between the transmitted and received signals. Radars rely on this information to calculate the wind velocity at various points in space.

2.2.2 Radar Products

2.2.2.1 Echo Top Height (ETH)

An echo top, in radar meteorology, signifies the highest altitude at which precipitation particles are detected. Essentially, it helps us pinpoint the maximum elevation angle where the intensity of precipitation, quantified by reflectivity, surpasses a predetermined threshold. This information is crucial for understanding the vertical extent of precipitation systems and their potential impact.

Overview

The modified ETH algorithm, developed by Lakshmanan et al. (2013) [Lakshmanan et al. (2013): https://journals.ametsoc.org/view/journals/wefo/28/2/waf-d-12-00084_1.xml], can

be applied to a NEXRAD PPI volume scan by following these steps:

1. **Finding Echo Top Height:**

   Locate the maximum elevation angle, denoted by θ_b, where the reflectivity, Z_b, surpasses a predefined echo-top reflectivity threshold, Z_T (e.g., 0 dBZ, 18 dBZ). If θ_b is not the highest elevation scan available in the volume, obtain the reflectivity value, Z_a, at the subsequent higher elevation angle, θ_a. Employ the following equation to calculate the echo-top height, θ_T:

   $$\theta_T = \frac{(Z_T - Z_a)(\theta_b - \theta_a)}{Z_b - Z_a} + \theta_b$$

2. **Handling Highest Elevation Scan:**

   If θ_b coincides with the highest elevation scan accessible, set $\theta_T = \theta_b + \beta/2$, where β represents the half-power beamwidth. This scenario arises when:

   - Far from the radar: Higher elevation scans possess shorter ranges compared to a baseline "surveillance" scan.
   - Very close to the radar: The highest elevation scan fails to capture the cloud's peak.

   Under these circumstances, θ_T corresponds to the top of the beam containing reflectivity greater than or equal to Z_T. In essence, the traditional echo-top computation is adopted when no data exists from higher elevation scans.

   This mathematical formulation can be expressed in LaTeX as:

   $$\theta_T = \begin{cases} \frac{(Z_T - Z_a)(\theta_b - \theta_a)}{Z_b - Z_a} + \theta_b & \text{if } \theta_b \text{ is not highest elevation scan} \\ \theta_b + \frac{\beta}{2} & \text{if } \theta_b \text{ is highest elevation scan} \end{cases}$$

   This equation incorporates a conditional statement to account for the two scenarios based on the availability of data from higher elevation scans.

2.3 Common Data format in meteorology

2.3.1 SIGMET data format - raw format (Vaisala)

Vaisala is a Finnish company specializing in environmental and meteorological instrumentation. The RAW format (also referred to as SIGMET in some documents [10]) is one of the storage formats developed by the company to organize data output from their radar devices.

Some notable points about this format include:

- The file content is divided into a **block**, each with a size of exactly 6144 bytes. This size aligns with the main storage size on older tape devices.
- The file typically consolidates data from all radar scanning sessions.
- Data records are organized within the scope of one block (6144 bytes). In the case of any remaining space in the block, it is padded with additional zeros.

With the aforementioned characteristics, the key advantages of the RAW format storage can be identified: [16]

- Compatibility with various tape types, which were commonly used devices in the past and are still widely used due to their cost-effective storage capacity.
- By storing data as blocks, SIGMET facilitates block-level error recovery in storage systems.

The main concern raised by the team is the mapping capability between the storage structure on the hard drive and the tape.

2.3.2 NETCDF data format - Network Common Data Form

NetCDF (Network Common Data Form) is a versatile file format designed explicitly for storing multidimensional scientific data. Within the netCDF library system, various binary formats are supported, each contributing to the flexibility and scalability of data management [13]. Notably, these formats include:

1. **Classic Format**: Initially used in the first version of netCDF and remains the default choice for file creation.

2. **64-bit Offset Format**: Introduced since version 3.6.0, this format supports larger variable and file sizes.

3. **netCDF-4/HDF5 Format**: Introduced in version 4.0, utilizing the HDF5 data format with some limitations.

4. **HDF4 SD Format**: Primarily supports data reading.

5. **CDF5 Format**: Synchronized support with the parallel-NetCDF project.

All these formats exhibit self-description, with a detailed header part describing the file structure, including data arrays and file metadata in the form of attribute name/value pairs. This design ensures platform independence, with issues such as endianness being flexibly addressed through software libraries.

Consider a specific example of storing essential meteorological parameters such as temperature, humidity, pressure, wind speed, and direction in netCDF files. This illustrates the capability of this format in handling diverse scientific datasets, providing a powerful and flexible means to manage multidimensional information.

```
→  titan2023 ncdump -h radar.nc
netcdf radar {
dimensions:
        time = UNLIMITED ; // (1748 currently)
        range = 1198 ;
        sweep = 5 ;
        string_length = 32 ;
```

Figure 2.6: *Radar information in NETCDF format. The total dimensions of the dataset are 2975, grouped into 4 distinct labels.*

Starting from version 4.0, the netCDF API introduces the ability to use the HDF5 data format. This crucial integration allows netCDF users to create HDF5 files, unlocking benefits such as significantly larger file sizes and support for unlimited dimensions. This step marks a significant move towards leveraging the extended advantages of the HDF5 format.

NetCDF Classic and 64-bit Offset Formats are international standards of the Open Geospatial Consortium [12], demonstrating the robustness and reliability in ensuring the compatibility and global scalability of the netCDF format.

2.4 Related technologies

2.4.1 Apache Airflow™

Apache Airflow™ stands as an open-source platform designed to manage data flow within systems associated with data. In the face of the escalating challenge of data pipeline management, Airflow emerges as a comprehensive solution, automating and optimizing data-related workflows effectively [1].

Airflow not only aids in defining and managing the start and end times of each data pipeline but also provides precise and detailed monitoring of the results of each task. This becomes particularly crucial when ensuring the integrity and reliability of the processed data.

With the ability to discern complex relationships between tasks through the Directed Acyclic Graph (DAG) model, Airflow empowers administrators with tighter control and flexibility in handling workflow processes. Its robust integration with logging systems facilitates detailed activity tracking, assisting in issue resolution and ensuring that every process aligns with expectations.

Simultaneously, the scheduling flexibility makes Airflow an excellent tool for time and resource management. Its strong integration with various data sources and extensibility through plugins allows Airflow to meet diverse needs in data processing and task automation.

Apache Airflow not only delivers robust performance but also brings flexibility and optimal technical features to data processing workflows. With its time management capabilities, powerful logging integration, scheduling flexibility, and scalability, Airflow stands as the top choice for enhancing performance and control in data processing workflows.

2.4.2 Kubernetes

Kubernetes, an open-source system for managing and deploying highly flexible applications in cloud and data center environments, has evolved into one of the most widely adopted tools in the field of Information Technology [8]. Originally developed by Google and later transferred to the Cloud Native Computing Foundation (CNCF), Kubernetes aims to automate the deployment, scaling, and management of containerized applications, alleviating the burden on developers and system administrators. The platform offers a unified foundation for deploying,

scaling, and managing containerized applications across multiple servers.

Kubernetes operates based on key concepts such as Pods, Services, ReplicaSets, and various other abstractions, creating a flexible environment for application deployment and management. This fosters an environment where developers can easily build applications, and system administrators can efficiently maintain them.

Beyond supporting traditional deployment models, Kubernetes paves the way for innovative strategies like Continuous Deployment (CD) and Microservices. With the ability to automate many aspects of the development and deployment process, Kubernetes plays a crucial role in constructing and sustaining complex, flexible, and scalable systems.

2.4.2.1 Kubernetes Components

Introducing essential concepts for managing and deploying applications, Kubernetes provides an effective and flexible environment. The main components of Kubernetes include Pod, ReplicaSet, Deployment, and Service.

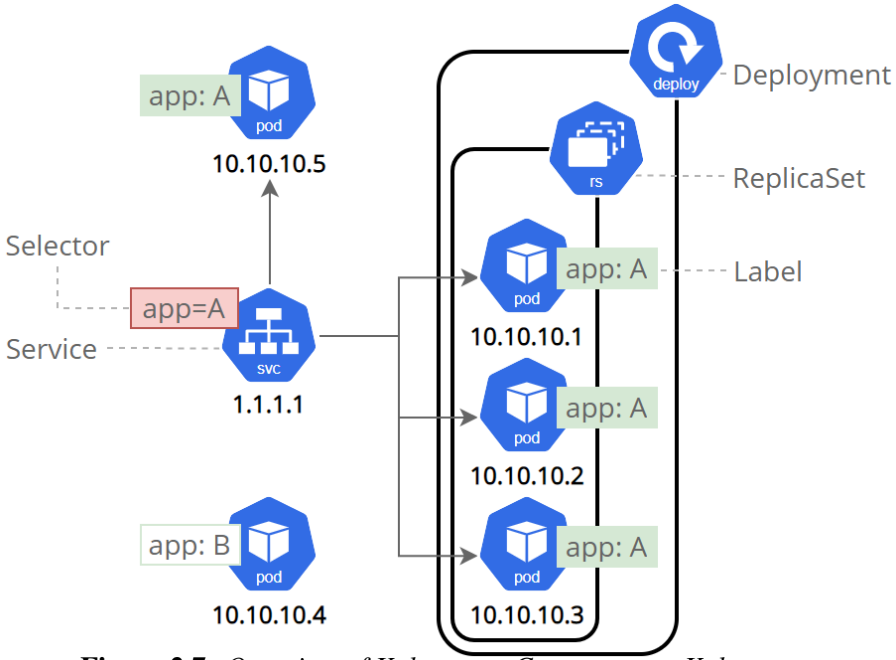


Figure 2.7: *Overview of Kubernetes Components - Kubernetes*

In Kubernetes, a **Pod** serves as the fundamental unit, representing a collection of containers that share a common workspace. Within the same Pod, containers collaborate by sharing network and storage resources, fostering interaction and enabling the construction of intricate applications.

The **ReplicaSet**, a crucial resource in Kubernetes, ensures a designated number of Pods operate in a specified manner. In the event of a Pod failure or shutdown, the ReplicaSet automatically initiates the creation of a new Pod to replace it. This mechanism ensures the application's stable state by guaranteeing a defined number of Pods are consistently operational.

For managing the deployment and updating processes of applications, **Deployment** is a key component in Kubernetes. It articulates the desired state of the application and orchestrates the updating of the ReplicaSet to achieve that state. Deployment provides versatile management capabilities, facilitating the deployment of new versions, rollbacks, and updates without disrupting the service.

The **Service** resource in Kubernetes furnishes an HTTP port to Pods, generating a unique IP address and DNS name for a cluster of Pods. This enables seamless communication among applications within the cluster and with external environments. Service effectively simplifies the intricacies of handling multiple Pods and IP addresses, offering a straightforward means of accessing services within the Kubernetes environment.

Typically, large-scale systems leverage Kubernetes in their software development and deployment processes. This adoption brings several advantages, including efficient resource management and self-recovery capabilities. Kubernetes optimizes resource utilization, ensuring optimal performance and reducing waste. Additionally, it automatically addresses issues during operations, enhancing high availability.

However, the technology is not without its challenges, including a steep learning curve for beginners. Mastery of diverse knowledge areas such as computer networking and containerization is necessary. Moreover, deploying and maintaining Kubernetes demands significant resources, both in terms of personnel and hardware, particularly for smaller organizations.

2.4.2.2 High Availability in Kubernetes

High Availability is a crucial factor in the success of any system. In Kubernetes, High Availability is achieved through the combination of several features, including self-healing, load balancing, and auto-scaling.

First, Kubernetes uses **Deployment**, a type of **Controller** for managing replicas. Through Deployment, we can easily perform horizontal scaling, which is the process of increasing the number of replicas of a Pod. This mechanism ensures that the application can handle numerous

requests without compromising performance. Moreover, in case of a Pod failure, a Deployment makes sure that a new Pod is created to replace it, ensuring the amounts of predefined replicas is always maintained.

At network layers, Kubernetes uses **Service** for communication between various components within or outside the cluster. Instead of directly accessing Pods, other components can access Services, which will redirect the request to the appropriate Pod. When Pods are replaced, the Service will automatically update the routing rules to ensure the request is sent to the correct Pod. Outside the cluster, Kubernetes also uses **Ingress** to manage external access to Services. Instead of specifying the direct node IP address, Clients can abstract it by using only the URL or hostname.

Finally, at the storage level, Kubernetes provides Persistent Volumes. By doing so, applications can be agnostic to the underlying storage infrastructure. This allows for easier management and scaling of storage resources. Not only that, depending on the provided **Storage Class**, Kubernetes makes sure that the data is replicated to multiple nodes, ensuring data availability in case of node failure.

To summarize, Kubernetes provides a robust set of features to ensure High Availability. By leveraging these features, we can build a highly available system that can scale with our load, while also being resilient to failures.

2.4.3 LROSE

LROSE (Lidar Radar Open Software Environment) is a project supported by the National Science Foundation (NSF) with the goal of developing common software for the Lidar, Radar, and Profiler community. The project operates based on the principles of collaboration and open source. The core software package of LROSE is a collaborative effort between Colorado State University (CSU) and the Earth Observing Laboratory (EOL) at the National Center for Atmospheric Research (NCAR) [7].

Originating from the need for a unified software environment for processing Lidar and Radar data in atmospheric science research [7], the project addresses complexities related to integrating data from various observation platforms, including Lidar, Radar, and Profiler. These components are designed to meet the specific needs of meteorologists and researchers working with remote sensor data.

LROSE is widely used in meteorological research, including studies related to cloud and precipitation processes, boundary layer dynamics, and other meteorological phenomena. The software supports the analysis of observation data collected from ground-based tools such as Lidar and Radar. LROSE seamlessly integrates with a variety of model and atmospheric analysis tools to optimize its capabilities. Researchers often integrate LROSE into numerical weather prediction models as well as other data assimilation techniques, creating a flexible and powerful system.

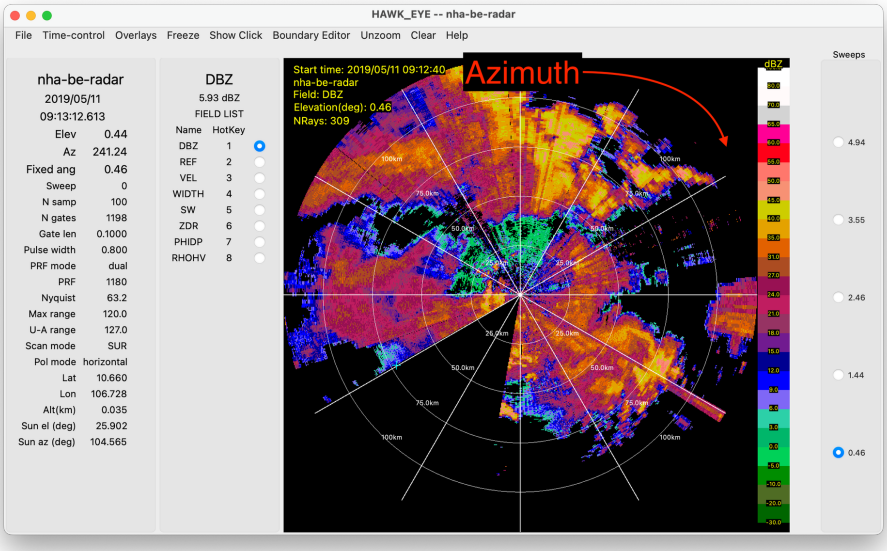


Figure 2.8: *Hawk Eye, Lidar and Radar visualization tool of LROSE*

The project actively encourages participation from a large scientific community, promoting the exchange of ideas, algorithms, and improvements for the software. Regular updates and contributions from users contribute to the continuous development and refinement of LROSE.

2.4.4 PyArt

2.4.5 ASP.NET Core

2.4.6 Min.IO

Chapter 3

System Analysis and Design

3.1 Exploratory Data Analysis

This section describes the processes and steps that our team took to study the data.

3.1.1 Describe provided data

The provided data resides inside two parent directories:

For the `convert` directory, it seems like the data there has been converted from the other `UF` files. Compared with the original, these files do not store as many necessary fields, with many meteorologist features being left out or not included. For instance, fields like **Reflectivity**, **Mean doppler velocity** and **Doppler spectrum widtah** only appear in the `UF` files, but not in the converted ones.

```
{ 'time': { 'units': 'seconds since 2019-05-13T01:20:07Z', } ... 'fields': {
    'total_power': { 'units': 'dBZ', 'standard_name':
    'equivalent_reflectivity_factor', 'long_name': 'Total power', 'coordinates':
    'elevation azimuth range', 'data': masked_array( data=[ [37.0, 27.5, 37.5,
    ..., --, --, --], [50.0, 41.0, 31.5, ..., --, --, --], [45.0, 41.0, 29.5,
    ..., 0.0, 4.0, 7.0], ..., [35.0, 32.0, 21.0, ..., 3.0, --, --], [33.5, 27.5,
    20.0, ..., --, --, --], [41.0, 31.5, 27.0, ..., --, --, --] ], mask=[
```

```
8    [False, False, False, ...,  True,  True,  True], [False, False,
    False, ...,
9     True,  True,  True], [False, False, False, ..., False, False, False
    ], ...,
10    [False, False, False, ..., False,  True,  True], [False, False,
    False, ...,
11    True,  True,  True], [False, False, False, ...,  True,  True,  True]
     ],
12    fill_value=1e+20, dtype=float32 ), '_FillValue': -9999.0 }, ... } }
```

Listing 3.1: A sample of metadata extracted from UF file

As a result, we decided to use the `UF` files for our study, while temporarily ignoring the `convert` directory.

3.1.2 Comparing PyART and LROSE

From the advice of our professor, at first, we decided to learn about LROSE and try to open the provided data. LROSE is a set of multiple tools and libraries for working with Radar data output. For macOS, support for Homebrew makes it easier to install these tools for local usage. However, for other platforms, downloading and compiling the source code is required, which increases the complexity and time needed to set up the environment.

The reference for LROSE is not very extensive, with many commands requiring special configs and parameters that have not been written down in the documentation. As a result, it took us a lot of time to figure out how to use the tools and libraries. In the end, though, our team was able to open the provided data and visualize it using the LROSE tools, whose name was `HawkEye`.

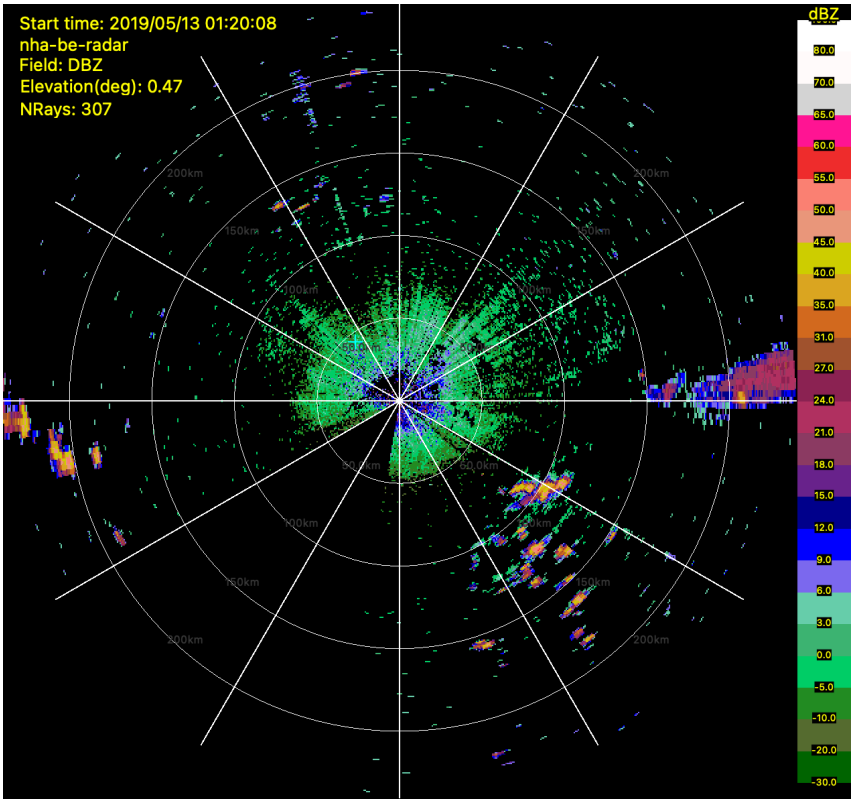


Figure 3.1: *Data visualization of Nha Be radar from HawkEye - Map of Vietnam*

During our investigation, we also encountered a different set of tools that can interact with the radar files, called PyART. When compared with LROSE-core, our team found that PyART contains many more advantages.

First, PyART is a Python library. This means that it is easier to install and use, especially for people who are already familiar with Python. Moreover, by supporting Python, our team can incorporate it into many popular ETL tools that use Python. Take Apache Airflow as an example. Instead of having to write a custom operator to call LROSE-core from the terminal, we can use a Task to perform the same action with PyART. Doing so would reduce much of the complexity of the workflow.

Secondly, with default configurations, PyART represents the data as an XArray [6] format, which is also a popular format for multidimensional data. As XArray is built on top of NumPy [5], this popular library can receive wider adoption when compared with many other formats. Integration with the existing library also means that we can use many other tools to work with the data. For instance, we can use Matplotlib to visualize the data; or use the library SciPy [17] to perform scientific calculations on the data. Because those tools require NumPy, we can use them directly with PyART without having to convert the data to another format.

Finally, PyART also has more comprehensive documentation when compared with LROSE. As newcomers to the subject of meteorology, we found that the documentation of PyART is much more accessible than LROSE. The documentation includes many examples and tutorials. Moreover, the documentation also contains many references to other resources, such as the book *Radar for Meteorological and Atmospheric Observations* [15]. Even though this is a subjective opinion, we believe that the documentation of PyART is much more helpful for further development.

With all the above ideas, our teams decided to use PyART for both the study and the development of the project.

3.1.3 Visualizing data using PyART

When installed, PyART also downloads another Python tool named Cartopy [11]. This library provides many useful features for our understanding of the data, such as the ability to plot the data on a map, or to convert the coordinates from one system to another.

Using these libraries, we can visualize our data directly from a Jupyter Notebook, similar to any other Python data visualization task. The area that the radar collects is a square, with the radar at the center. All of its length extends to about 300 kilometers in each direction.

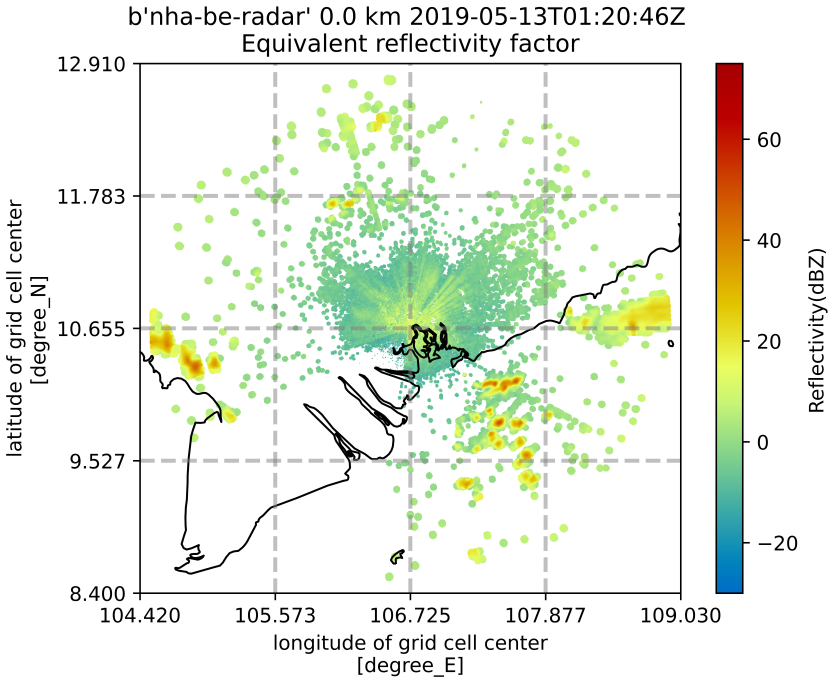


Figure 3.2: *Data visualization of Nha Be radar from PyART - Map of Vietnam - similar zoom to Figure 3.1*

3.2 Requirements

3.3 Functional Requirement

1. Data ingestion from various sources, including APIs and government data repositories.
2. Data storage and management, potentially utilizing databases or data lakes.
3. Data analysis and visualization capabilities tailored for meteorologists and researchers.
4. Integration with third-party platforms and APIs.
5. User management and access control mechanisms.
6. Reporting and alerting functionalities.

3.4 Non-Functional Requirements

1. **Reliability:**
   - Ensure high availability and stable operation of the platform.
   - Maintain data integrity, security, and privacy as top priorities.
2. **Scalability:**
   - Ability to scale and handle increasing volumes of data loads.
   - Leverage cloud-native technologies and containerization for easy scaling of infrastructure and applications.
3. **Performance:**
   - Rapid data processing and analysis capabilities.
4. **Usability:**
   - Intuitive and user-friendly interface tailored for different user groups.

5. **Security:**
   - Robust data protection and access control mechanisms.
6. **Compliance:**
   - Adhere to applicable regulatory standards, data privacy laws, and compliance requirements.
7. **Scaling Technology and Infrastructure:**
   - Adopt cloud-native architectures and containerization technologies (e.g., Docker, Kubernetes) to facilitate easy scaling of the platform's infrastructure and applications.
   - Leverage auto-scaling capabilities and elastic resources provided by cloud platforms to dynamically adjust capacity based on demand.
   - Implement microservices architecture and decoupled components for independent scaling of individual services or modules.

3.5 Data Requirements

How radar file are processed, cleaned, stored and used Storing data as blob - Radar Object (Minio) Caching

3.6 Data management

How data are stored? Storage - structured, unstructured, semi-structured Naming convention Data format

3.7 System Architecture

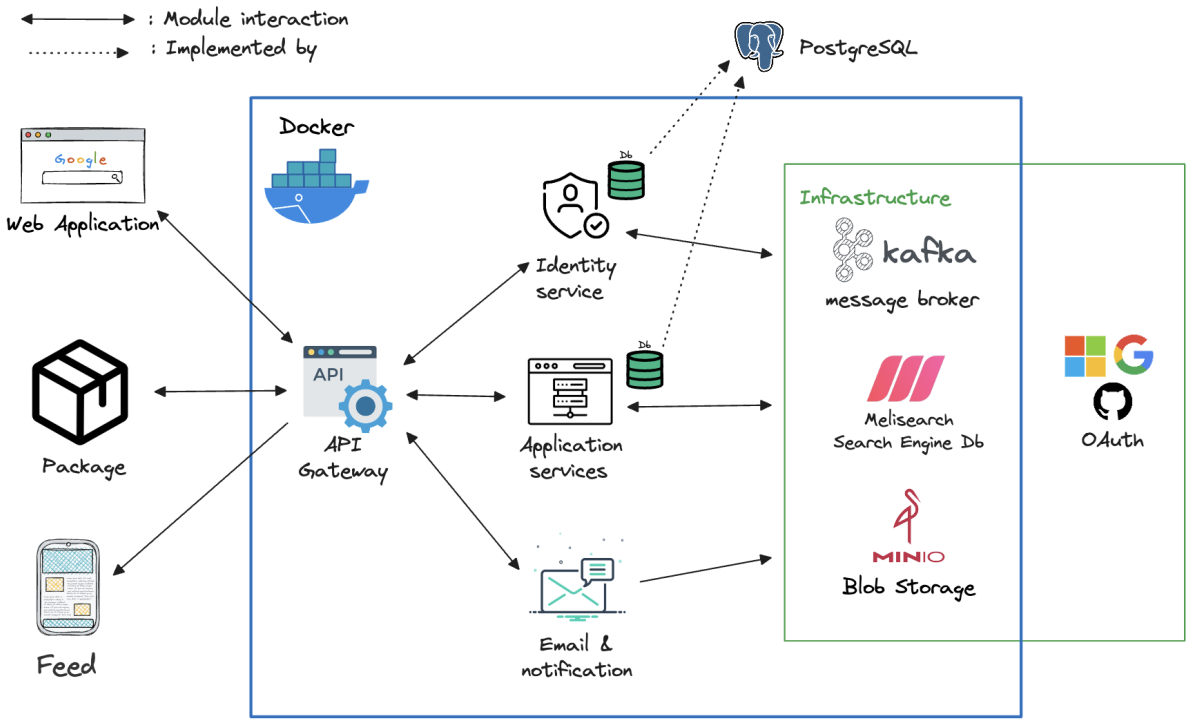


Figure 3.3: *System Architecture*

Based on the requirements set by stakeholders and after studying the existing system, the team proposes the design and implementation of a Weather Data Platform. Figure 3.3 illustrates the implementation of the system.

3.7.1 Microservices

In essence, a microservices architecture decomposes a large software application into a collection of smaller, self-contained services. Each service plays a specific role and owns a well-defined business capability. These services are loosely coupled, meaning they interact through well-designed APIs and operate independently.

Microservices architectures offer a compelling approach to building software. By decomposing large applications into smaller, independent services, they unlock agility, maintainability, and fault isolation. Each service owns a specific business function and interacts with others through well-defined APIs. This allows for faster development cycles, easier maintenance of individual services, and the freedom to choose the best technology for each job.

However, this power comes with a price. Managing a distributed system with numerous services inherently increases complexity compared to a monolithic application. Communication between services adds overhead to the system's performance. Perhaps the most significant challenge lies in ensuring data consistency across these independent services, requiring meticulous design and implementation. Carefully considering these trade-offs is crucial before embarking on a microservices journey.

While microservices boast impressive agility and maintainability, adopting this architecture isn't without its complexities. Managing a distributed system with numerous services inherently increases complexity compared to a monolithic application. Furthermore, communication between these services adds overhead to the system's performance. Perhaps the most significant challenge lies in ensuring data consistency across these independent services, requiring meticulous design and implementation.

3.7.2 Clean Architecture

Microservices and Clean Architecture can complement each other to create a modular, maintainable, and scalable system. Microservices architecture promotes the development of small, independent services that can be deployed, scaled, and updated independently. Each microservice encapsulates a specific business capability or domain, reducing the overall complexity of the system.

Clean Architecture, on the other hand, advocates for a layered approach to software design,

promoting separation of concerns and decoupling of components. By adhering to the principles of Clean Architecture within each microservice, developers can achieve a high degree of modularity, testability, and maintainability.

Clean Architecture separates the system into distinct layers, each with specific responsibilities. The Shared Kernel acts as the core, housing reusable components that benefit all layers. The Domain layer sits at the heart of the application, defining core entities and their behaviors. A common base class promotes code maintainability within this layer. The Application layer orchestrates request routing and establishes project-wide contracts for implementation in other layers. Clean Architecture emphasizes the testability of each layer through unit tests. Frameworks like xUnit simplify unit test creation.

The Infrastructure layer provides supporting services for the application. Cross-cutting concerns like logging reside here. User registration, authentication, and authorization functionalities are handled by the Identity layer. Persistence takes care of data access using patterns like repositories and Unit of Work. The Web Framework layer manages configurations for the web application, while the Web API layer delivers functionality to the user. Finally, plugins bridge the gap between monolithic and microservices architectures by promoting modularity.

Chapter 4

Implementation

4.1 Data Processing

The implementation phase of the team will be an extension to the current system, which is outlined in Figure 3.3.

In step 3, the team will set up a simple SFTP server. SFTP is a straightforward and widely used protocol, supported by numerous libraries and tools for communication based on this protocol. Additionally, compared to FTP, the mentioned protocol ensures security during data transfer. Depending on the permissions, the team may assist the observation station in constructing scripts to automatically forward processed files or allow manual file submission.

Once files are uploaded to the SFTP server, the team utilizes Airflow to orchestrate all existing ETL workflows in the overall system. Currently, the team pauses with a single DAG to process data from the Nha Be observation station. Airflow monitors newly added files on our SFTP server and initiates the ETL process. The choice of Apache Spark is based on the volume and complexity of the data. If the data size per new SIGMET file is manageable with Python alone, without the need for Spark, it will not be employed in this step.

Meteorological data, upon reaching the team's infrastructure, will be bifurcated into two main streams: Metadata, such as creation date, size, timestamp, etc., will be stored in a traditional Relational Database Management System (RDBMS). Specifically, PostgreSQL is chosen due to its popularity and the team's familiarity. Storing metadata here facilitates rapid query responses without direct access to the raw data. Common queries may include:

- What timestamps are being recorded? (e.g., from 21/11/2023 to 17/12/2023)

- At timestamp x, what are the geographical coordinates of the radar?
- What fields of data are currently stored?

Additionally, the database acts as an index, quickly identifying the storage location of raw data.

For specific hydrometeorological data, the team finds it inefficient to store them directly in conventional DBMS. Simultaneously, storing data in files still maintains a reasonable overall size. Therefore, the team decides to separate the raw data and store it directly in files. This approach, combined with the previously mentioned indexes, accelerates the retrieval process.

To facilitate data queries for models, machine learning, AI, etc., the team will develop a simple backend server using Python's FastAPI at step 5. In step 6, the backend receives query data in REST API format, queries the metadata DB and data files, and returns the achieved results. During different training instances, Machine Learning entities can connect to this server to retrieve data.

It's worth mentioning that the entire system will be developed and operated in a containerized manner and will be deployed on the Kubernetes platform. This reflects the system's ability to maintain high availability and ease of solution maintenance. In this illustration, the team will deploy it on a cluster of Raspberry Pi-embedded computers.

Lastly, in step 7, the team proposes an additional consideration. If suitable, the team may build a DataLoader to swiftly serve other model-making groups. Considering the popularity of Pytorch in AI, the team will initially approach this platform.

4.2 System

4.2.1 Programming languages and Libraries

C# was selected for this project because of its widespread popularity in enterprise applications and its capability to operate across various operating systems following the release of .NET Core. This feature significantly enhances deployment flexibility, allowing the application to be utilized in diverse environments.

Additionally, we have incorporated ASP.NET Core specifically version 8 into our technology stack. By leveraging ASP.NET Core, we benefit from a robust, well-supported framework

that facilitates the development of scalable and secure web applications. It seamlessly integrates with C#, enabling us to utilize a consistent programming environment while also exploiting features such as dependency injection, a vast ecosystem of middleware, and a strong configuration system that is suited to modern web applications.

ASP.NET Core 8 brings forward improvements in areas such as minimized startup times, reduced memory footprint, and enhanced security features, making it an ideal choice for developing scalable and secure web applications. The choice also underscores our commitment to developing applications that are both efficient and future-proof, ensuring that they perform optimally on both Windows and non-Windows platforms. This alignment with .NET Core's cross-platform capabilities ensures that our project remains versatile and adaptable to the evolving technological landscape.

Optionally, we have integrated the Nuxt framework into our technology stack. Nuxt is a progressive Vue framework that is used for building more robust and versatile web applications. It simplifies the development process by handling various aspects of the web infrastructure, such as server-side rendering, static site generation, and automatic code splitting. This inclusion enriches our application's interactivity and user experience, providing a seamless and dynamic interface for users.

4.2.2 Command Query Responsibility Segregation

The Command Query Responsibility Segregation (CQRS) is an architectural pattern that distinctively separates the tasks of reading data (queries) and writing data (commands) within a software application. This separation splits responsibilities into two main components:

- **Command Side**: This component manages operations that modify the system's state. It handles incoming commands from clients or external systems, conducts validations, and updates the data store accordingly. This side is essential for maintaining the integrity and accuracy of data modifications within the application.
- **Query Side**: Dedicated to data retrieval, this component processes all read requests. It fetches data from the appropriate sources, ensuring that the information provided is accurate and reflects the current state of the data store.

Key advantages of employing the CQRS pattern are:

- **Scalability**: CQRS allows for the independent scaling of the read and write components based on their respective workloads, which can significantly enhance system performance.
- **Flexibility**: With the separation of concerns, different storage and optimization strategies can be applied to the reading and writing processes. This flexibility enables the use of the most appropriate tools for each function, optimizing efficiency.
- **Event-Driven Architecture Compatibility**: The use of CQRS often complements event-driven architectures, where changes in the system's state are captured and managed as events. This compatibility ensures that the architecture is dynamic and responsive to changes in business requirements.

4.2.3 Project Structure

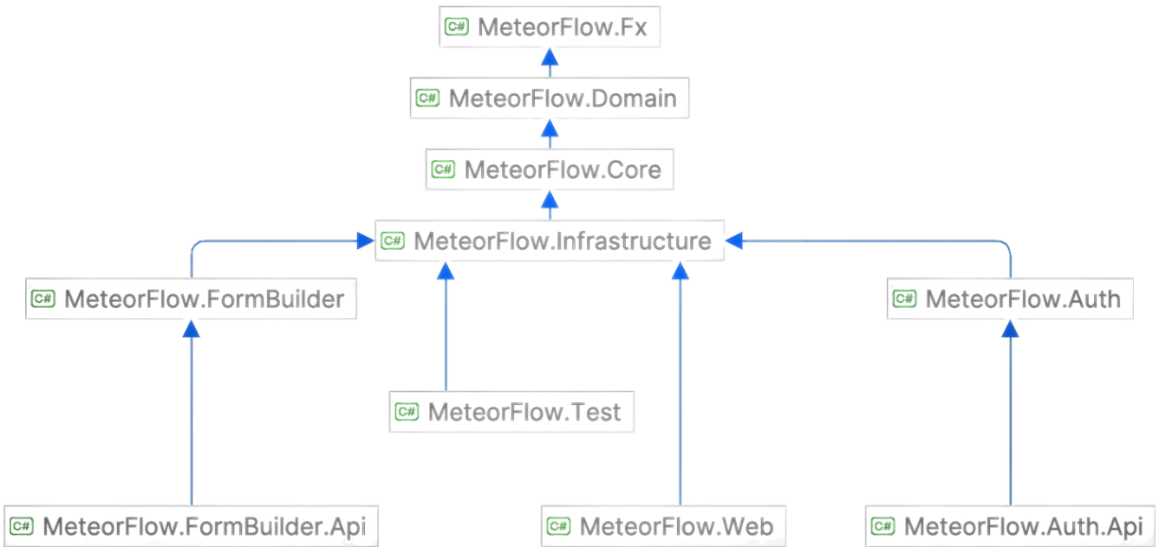


Figure 4.1: *Project Structure*

MeteorFlow is depicted as a modular framework comprising several interdependent components. Each library or module serves a distinct function, collectively supporting a robust and scalable application infrastructure. This section will delineate the roles and relationships of these components.

MeteorFlow.Fx enriches the application by introducing additional functionalities or user interface enhancements. These features, while not central to the primary business logic, signif-

icantly augment the application's overall capabilities, providing enriched user experiences and functional extensions.

MeteorFlow.Domain is tailored to articulate the business domain, encapsulating entities and rules essential to the application's domain logic. Its reliance on the Core module underscores the latter's foundational role within the architecture, affirming its influence over the domain-specific functionalities.

At the heart of the architecture, MeteorFlow.Core embodies the fundamental business logic and operations critical to the application. Its pivotal role is underscored by its influence on all peripheral modules, which depend on the Core for their foundational functionalities.

MeteorFlow.Infrastructure functions as the backbone for data access and manages cross-cutting concerns such as logging and caching. This module is crucial for the operational management of the application, interfacing seamlessly with the Core module to apply essential functionalities across various infrastructural tasks.

Built upon the MeteorFlow.Infrastructure, the Auth module specializes in security and authentication processes, managing user verification and credential handling. The module provides APIs to enable external access to their functionalities and seamlessly integrate with other systems and applications. Other modules currently or in the near future follow the same structure above to provide additional functionalities.

MeteorFlow.Web, primarily tasked with managing the application's API gateway, interacts with all other modules within MeteorFlow to ensure robust web operations and effective resource management. Its role is crucial in maintaining the integrity and performance of web-based services, underpinning the system's interaction with users and external systems.

4.3 API Specification

4.3.1 Core API

GET /api/core/definition

Tags	Definition
Description	Retrieves a list of all core definitions.
Responses	200: An array of core definitions returned successfully.

	401: Unauthorized access.

POST /api/core/definition

Tags	Definition
Description	Creates a new core definition.
Request Body	AppDefinitions schema.
Responses	201: Core definition created successfully. 400: Bad request if the data is invalid. 401: Unauthorized access.

GET /api/core/definition/{id}

Tags	Definition
Description	Retrieves a specific core definition by ID.
Parameters	id: UUID of the core definition.
Responses	200: Core definition returned successfully. 404: Core definition not found. 401: Unauthorized access.

DELETE /api/core/definition/{id}

Tags	Definition
Description	Deletes a specific core definition by ID.
Parameters	id: UUID of the core definition.
Responses	200: Core definition deleted successfully. 404: Core definition not found. 401: Unauthorized access.

POST /api/core/instance

Tags	Instance

Description	Creates a new instance.
Request Body	AppInstances schema.
Responses	201: Instance created successfully. 400: Bad request if the data is invalid. 401: Unauthorized access.

GET /api/core/instance/{id}

Tags	Instance
Description	Retrieves a specific instance by ID.
Parameters	id: UUID of the instance.
Responses	200: Instance returned successfully. 404: Instance not found. 401: Unauthorized access.

DELETE /api/core/instance/{id}

Tags	Instance
Description	Deletes a specific instance by ID.
Parameters	id: UUID of the instance.
Responses	200: Instance deleted successfully. 404: Instance not found. 401: Unauthorized access.

GET /api/core/setting

Tags	Setting
Description	Retrieves a list of all settings.
Responses	200: An array of settings returned successfully. 401: Unauthorized access.

POST /api/core/setting

Tags	Setting
Description	Creates a new setting.
Request Body	AppSettings schema.
Responses	201: Setting created successfully. 400: Bad request if the data is invalid. 401: Unauthorized access.

GET /api/core/setting/{id}

Tags	Setting
Description	Retrieves a specific setting by ID.
Parameters	id: UUID of the setting.
Responses	200: Setting returned successfully. 404: Setting not found. 401: Unauthorized access.

DELETE /api/core/setting/{id}

Tags	Setting
Description	Deletes a specific setting by ID.
Parameters	id: UUID of the setting.
Responses	200: Setting deleted successfully. 404: Setting not found. 401: Unauthorized access.

GET /configuration

Tags	FileConfiguration
Description	Retrieves the current configuration.

Responses	200: Configuration returned successfully. 401: Unauthorized access.

POST /configuration

Tags	FileConfiguration
Description	Updates the current configuration.
Request Body	FileConfiguration schema.
Responses	200: Configuration updated successfully. 400: Bad request if the data is invalid. 401: Unauthorized access.

4.3.2 Indentity API

GET /api/auth

Tags	Auth
Description	Retrieves authentication status based on the returnUrl parameter.
Parameters	returnUrl (string): The URL to return to after authentication.
Responses	200: Authentication status returned successfully.

POST /api/auth/login

Tags	Auth
Description	Logs in a user with provided credentials.
Request Body	LoginInfo schema: Includes username and password.
Responses	200: Login successful.

POST /api/users/{id}/passwordresetemail

Tags	Users
Description	Sends a password reset email to the user specified by ID.
Parameters	id (string, uuid): Unique identifier of the user.
Responses	200: Password reset email sent successfully.

PUT /api/users/{id}/password

Tags	Users
Description	Updates the password for the user specified by ID.
Parameters	id (string, uuid): Unique identifier of the user.
Request Body	PasswordSetter schema: Includes new password and confirmation.
Responses	200: Password updated successfully. 404: User not found.

POST /api/users/{id}/emailaddressconfirmation

Tags	Users
Description	Sends an email confirmation to the user specified by ID.
Parameters	id (string, uuid): Unique identifier of the user.
Responses	200: Email confirmation sent successfully.

4.4 Datastore

Chapter 5

Testing and Validation

5.1 Unit Test

5.2 Integration Test

During the Integration Test phases, our team makes sure that not only does our system run correctly at every part, but also that the system as a whole works as expected. To do that, we first start with a Test Plan.

5.2.1 MeteorFlow Test Plan

Project Name	MeteorFlow
Created Date	TBD
Release Date	TBD

Component	Start Time	Assignee	Notes
ETL files and REST API	TBD	Thuy Nguyen, Kiet Tran	Notes

5.2.2 MeteorFlow Test Report

Release Date: TBD

Chapter 6

Future enhancement

Improve into a Weather Data Platform (WDP)

The Weather Data Platform (WDP) is developed to become a comprehensive solution for harnessing the power of weather data. The system is designed to meet the specific requirements of meteorologists, scholars, academic researchers, and developers from various fields, including freelancers, businesses, and Non-governmental Organizations (NGOs). WDP serves as a centralized hub for integrating weather data, analysis, and various features.

With the desire to evolve into a Weather Data Platform, we aim for the perfection and diversification of weather information. Not just a data table but a comprehensive experience. In the future, in addition to continuing to build the proposed integrated database, we promise to further research to expand and develop this integrated database into a weather data platform with the following development directions:

1. **Non-linear Data:** Expand beyond basic data integration, focusing on providing non-linear, detailed, and multi-source data to help users explore more about the surrounding environment.

2. **Artificial Intelligence Understanding:** Use artificial intelligence to understand the language of weather, from minor changes to major events, creating a deep and intelligent weather information source.

3. **Interactive User Interface:** Not just accessing information but also interacting with weather forecasts. The user interface will be where users express curiosity and interact directly with the data.

4. **Geospatial Information Connection:** Leverage geographical information systems to provide a contextualized, localized view of weather forecasts. This helps users understand the impact of weather on their surroundings.

5. **Performance Optimization:** Ensure quick and consistent responsiveness under all conditions.

6. **Data Security:** Enhance data security to ensure the security and integrity of weather information.

7. **Advanced Forecasting System:** Research and integrate artificial intelligence to improve forecasting capabilities and provide accurate forecast information.

8. **Testing and Optimization:** Conduct system testing to ensure stability and address any potential issues. Optimize performance if necessary.

9. **Deployment and Maintenance:** Deploy the system and maintain a regular update cycle to ensure that it consistently delivers accurate and reliable weather information.

Chapter 7

Conclusion

In this study, we have successfully built a Proof-of-Concept with high applicability, aiming to streamline the steps in the conventional workflow. This is a crucial step towards optimizing and enhancing operational efficiency in both research and practical contexts.

Our goal was to create a flexible system with high adaptability, helping to simplify complex steps in the workflow. In doing so, we not only contribute to increasing efficiency but also alleviate the workload pressure on personnel, creating favorable conditions for creativity and focus on core tasks.

We didn't just stop at developing the system but also proposed flexible deployment strategies, emphasizing easy integration into the current working environment of those involved in information gathering and weather forecasting tasks.

References

[1] Airflow. What is airflow?, 2020. URL https://airflow.apache.org/docs/apache-airflow/stable/index.html. Last accessed by 16/12/2023.

[2] casey. Using range height indicator scan of radar, 2017. URL https://earthscience.stackexchange.com/questions/7222/using-range-height-indicator-scan-of-radar. Last accessed by 17/12/2023.

[3] Ramez A. Elmasri and Shamkant Navathe. *Fundamentals of Database Systems*. Addison Wesley, third edition, 1998.

[4] TRUNG TÂM DỰ BÁO KHÍ TƯỢNG THỦY VĂN QUỐC GIA, Jun 2203.

[5] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL https://doi.org/10.1038/s41586-020-2649-2.

[6] S. Hoyer and J. Hamman. xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1), 2017. doi: 10.5334/jors.148. URL https://doi.org/10.5334/jors.148.

[7] Brenda Javornik, Hector Santiago III, and Jennifer C. DeHart. The lrose science gateway: One-stop shop for weather data, analysis, and expert advice. In *Practice and Experience*

in Advanced Research Computing, PEARC '21. ACM, July 2021. doi: 10.1145/3437359.3465595. URL http://dx.doi.org/10.1145/3437359.3465595.

[8] Kubernetes. Overview, 2019. URL https://kubernetes.io/docs/concepts/overview/. Last accessed by 17/12/2023.

[9] G. Latisen and G. Vossen. *Models and Languages of Object-Oriented Databases*. Addison Wesley, 1998.

[10] Lrose. Radxconvert - lrose wiki, 2021. URL http://wiki.lrose.net/index.php/RadxConvert.

[11] Met Office. *Cartopy: a cartographic python library with a Matplotlib interface*. Exeter, Devon, 2010 - 2015. URL https://scitools.org.uk/cartopy.

[12] opengeospatial. Ogc standard netcdf classic and 64-bit offset. https://www.opengeospatial.org/standards/netcdf, 2011. Archived from the original on 2017-11-30. Retrieved 2017-12-05.

[13] Russ Rew, Glenn Davis, Steve Emmerson, Cathy Cormack, John Caron, Robert Pincus, Ed Hartnett, Dennis Heimbigner, Lynton Appel, and Ward Fisher. Unidata netcdf, 1989. URL http://www.unidata.ucar.edu/software/netcdf/.

[14] Michael Stonebraker and Uğur Çetintemel. 'one size fits all': An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering (ICDE)*, April 2005.

[15] Roland Stull. Weather Radars, 12 2022. [Last accessed by 17/12/2023].

[16] *RAW Product Format - IRIS Programming Guide - IRIS Radar*. Vaisala, 2024. URL https://ftp.sigmet.vaisala.com/files/html_docs/IRIS-Programming-Guide-Webhelp/raw_product_format.html.

[17] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat,

Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.