Term Project Report - Climate Weather Analysis of Brazil Using Apache Flink

611221201 Egor Morozov

2024/12/01 — 2025/01/10

Contents

1 Description

In this term project, I will use Apache Flink to analyze a data set of weather forecasts from various parts of Brazil. This data set includes some meteorological parameters including temperature, humidity, wind, etc. The main purpose is to extract meaningful data for visualization. Analysis to predict future weather trends.

This term project involves batch processing, Data and information visualization. Through the algorithms designed by myself, the data is visualized and explained in detail, demonstrating the ability of weather data analysis in modern extreme climates.

2 Goals

This term project is to use **Apache Flink** to perform efficient batch processing of large-scale weather data and analyze hourly weather data from all over Brazil. The dataset contains key parameters such as temperature, humidity and wind speed. This project aims to achieve the following goals:

1. **Extract Meaningful Data:**
    - Identify patterns and trends in hourly weather data for different regions of Brazil.
    - Explore the correlations between various meteorological factors and their impact on local and regional climate conditions.
2. **Long-Term Climate Monitoring:**
    - Analyze long-term trends in weather data to better understand how climate patterns change over time.
    - Contribute to climate research by identifying anomalies or changes that may indicate that Brazil is affected by global climate change.
3. **Data and Information Visualization:**
    - Convert meteorological data into visual formats such as charts to visualize complex weather trends and patterns.
    - Use visualizations to better interpret the relationships between various meteorological parameters and their changes over time.

Through the three objectives mentioned above, we can leverage the powerful data processing capabilities of distributed systems to help us analyze past weather changes and assess future weather trends.

3 Deign

3.1 Apache Flink Architecture

Apache Flink [2], is a powerful stream processing framework that supports both batch and real-time data processing.

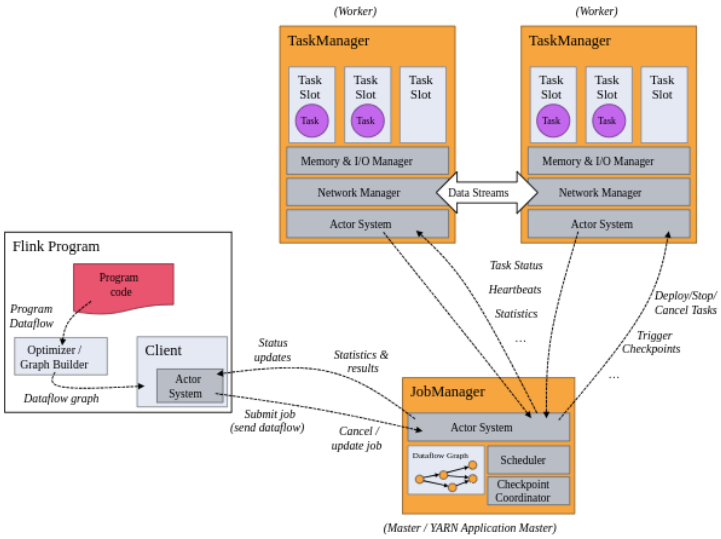


Figure 1: Flink Architecture

The core architecture of Apache Flink consists of the following components:

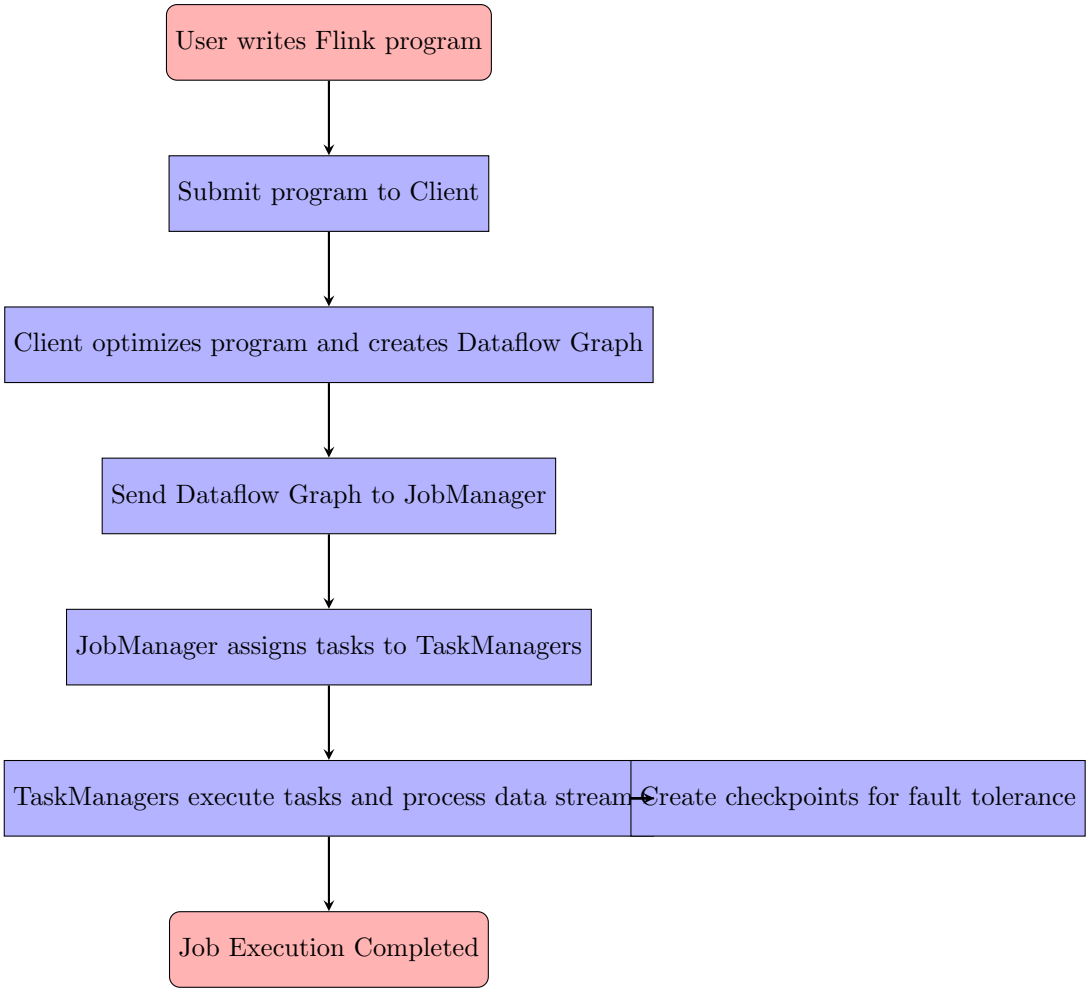
- **Flink Program:**
  - This is the user-defined program containing the logic of the data processing job.
  - The program is written using Flink's APIs and gets translated into a **Dataflow Graph**.
- **Client:**
  - The Client is the entry point for the Flink application.
  - It takes the program code, optimizes it, and submits the **Dataflow Graph** to the JobManager.
  - It interacts with the Flink cluster but does not participate in the actual execution of the job.
- **JobManager:**

  - The JobManager is the master node responsible for managing and coordinating the execution of a job.
  - Key components include:
    * **Scheduler:** Allocates tasks to TaskManagers and ensures efficient resource utilization.
    * **Checkpoint Coordinator:** Manages state snapshots to support fault tolerance.
    * **Actor System:** Facilitates communication within the cluster.

- **TaskManager:**
  - The TaskManager is the worker node in the Flink cluster.
  - It is responsible for executing tasks, which are parts of the overall Dataflow Graph.
  - Each TaskManager contains:
    * **Task Slots:** Logical resource divisions for running tasks.
    * **Memory & I/O Manager:** Handles task-specific memory allocation and input/output operations.
    * **Network Manager:** Manages data exchange between tasks over the network.
    * **Actor System:** Ensures communication between the JobManager and other TaskManagers.

- **Data Streams:**
  - These represent the data flow between TaskManagers during job execution.
  - Data streams are processed in a distributed manner.

Work Flowchart



3.2 Find average Temperature Algorithm

The algorithm for calculating average temperature is Bihia, which is a simple and efficient method for computing the mean of a set of numbers. The algorithm works by summing all the values in the dataset and dividing the total by the number of values. The pseudocode for the Bihia algorithm is as follows:

Algorithm 1 Find Average Temperature Algorithm

Require: Dataset D in CSV format, Parameters: *start_date*, *end_date*, *selected_state*

Ensure: Time-series chart showing average temperature trends

1: **Data Preparation:**
2: Load dataset D into Flink table T using `CREATE TABLE`.
3: Define schema with fields: `state`, `timestamp`, `temperature`, etc.
4: **Filtering:**
5: Filter table T to create a subset T_f where:

$$T_f = \{r \in T \mid r.\texttt{state} = \textit{selected\_state} \land \textit{start\_date} \leq r.\texttt{timestamp} \leq \textit{end\_date}\}$$

6: **Grouping and Aggregation:**
7: Group rows in T_f by `timestamp` (day) to compute the daily average temperature:

$$T_g = \texttt{GROUP\_BY}(T_f, \texttt{timestamp})$$

$$T_a = \texttt{AGGREGATE}(T_g, \texttt{avg}(\texttt{temperature}) \rightarrow \texttt{avg\_temperature})$$

8: **Sorting:**
9: Sort T_a by `timestamp` to ensure chronological order:

$$T_s = \texttt{SORT}(T_a, \texttt{timestamp})$$

10: **Result Conversion:**
11: Convert T_s into a Pandas DataFrame DF for analysis.
12: **Visualization:**
13: Plot DF as a time-series graph:

$$\texttt{plot}(DF.\texttt{timestamp}, DF.\texttt{avg\_temperature})$$

14: Save the graph as an image file.
15: **return** Time-series chart depicting average temperature trends. =0

3.3 Find Heavy Rain Days Algorithm

The **Find Heavy Rain Days Algorithm** is designed to identify and analyze the frequency of days with low rainfall (defined as rainfall below a specified threshold, $R = 20\,\text{mm}$) across different states. This algorithm uses a structured data pipeline to process weather records and visualize the distribution of heavy rain days as a percentage for each state.

The algorithm integrates scalable data processing capabilities, leveraging Apache Flink for filtering and aggregating weather data. It computes the number of heavy rain days for each state, converts the results into a format suitable for analysis, and visualizes the distribution using a pie chart.

This method allows for efficient analysis of large meteorological datasets and provides insights into regional rainfall patterns, enabling stakeholders such as meteorologists, policymakers, and researchers to understand and monitor weather trends effectively.

The algorithm follows these key steps:

1. Loading and preparing the dataset.
2. Filtering data to identify days with rainfall below the threshold.
3. Grouping records by state and calculating the count of heavy rain days.
4. Converting results into an analytical format (e.g., DataFrame).
5. Visualizing the results as a percentage distribution across states.

The detailed steps of the algorithm are outlined below.

Algorithm 2 Count and Visualize Heavy Rain Days by State

Require: Dataset D in CSV format with fields $\{\text{state}, \text{rainfall}, \dots\}$, Threshold $R = 20$ mm

Ensure: Pie chart representing the percentage of heavy rain days for each state

1: **Load Dataset:**
2: Load dataset D into a table T with schema $\{\text{state}, \text{rainfall}, \dots\}$.
3: **Filter Data:**
4: Create a filtered table T_f where:

$$T_f = \{r \in T \mid r.\text{rainfall} < R\}$$

5: **Group by State:**
6: Group T_f by the field `state`:

$$T_g = \texttt{GROUP\_BY}(T_f, \texttt{state})$$

7: **Count Heavy Rain Days:**
8: Compute the count of heavy rain days for each state:

$$T_c = \texttt{AGGREGATE}(T_g, \texttt{COUNT}(\texttt{rainfall}) \rightarrow \texttt{heavy\_rain\_days})$$

9: **Convert to DataFrame:**
10: Convert T_c into a Pandas DataFrame DF for visualization:

$$DF = \texttt{CONVERT}(T_c)$$

11: **Compute Percentages:**
12: Compute the percentage of heavy rain days for each state:

$$\texttt{percentage}_i = \frac{\texttt{heavy\_rain\_days}_i}{\sum \texttt{heavy\_rain\_days}}$$

13: **Visualization:**
14: Plot the percentages as a pie chart:

$$\texttt{PLOT\_PIE}(\texttt{labels} = \texttt{state}, \texttt{values} = \texttt{percentage})$$

15: **Save Chart:**
16: Save the pie chart as an image:

$$\texttt{SAVE\_CHART}(\texttt{output\_path} = \texttt{"heavy\_rain\_days\_pie\_by\_state.png"})$$

17: **return** Pie chart showing the percentage of heavy rain days for each state.
=0

3.4 Find Max Rainfall Algorithm

The Find Max Rainfall Algorithm is designed to determine and visualize the maximum recorded rainfall for each state within a weather dataset. This algorithm processes the data through a structured pipeline, leveraging Apache Flink's scalable data processing capabilities for efficient computation and analysis. The results are presented as a bar chart, highlighting the states with the highest rainfall.

This method enables researchers, meteorologists, and policymakers to identify regions experiencing extreme weather conditions, which can be crucial for disaster management, resource allocation, and climate studies. The algorithm is particularly effective for handling large-scale meteorological datasets and extracting meaningful insights.

The algorithm follows these key steps:

1. Loading the weather dataset and preparing the data for analysis.
2. Selecting the relevant columns (`state` and `rainfall`) for computation.
3. Grouping the data by `state` and calculating the maximum rainfall for each group.
4. Sorting the results in descending order of maximum rainfall to highlight the states with the highest values.
5. Converting the results into a format suitable for visualization.
6. Creating a bar chart to represent the maximum rainfall by state.
7. Saving the visualization as an image for reporting purposes.

The detailed steps of the algorithm are outlined below.

Algorithm 3 Find Maximum Rainfall by State

Require: Dataset D in CSV format with fields $\{\text{state}, \text{rainfall}, \dots\}$
Ensure: Bar chart representing the maximum rainfall recorded for each state

1: **Load Dataset:**
2: Load dataset D into a table T with schema $\{\text{state}, \text{rainfall}, \dots\}$.
3: **Select Relevant Columns:**
4: Extract the `state` and `rainfall` columns:

$$T_s = \texttt{SELECT}(T, \texttt{state}, \texttt{rainfall})$$

5: **Group by State:**
6: Group T_s by `state`:

$$T_g = \texttt{GROUP\_BY}(T_s, \texttt{state})$$

7: **Compute Maximum Rainfall:**
8: Compute the maximum rainfall for each state:

$$T_m = \texttt{AGGREGATE}(T_g, \texttt{MAX}(\texttt{rainfall}) \rightarrow \texttt{max\_rainfall})$$

9: **Sort Results:**
10: Sort T_m by `max_rainfall` in descending order:

$$T_o = \texttt{ORDER\_BY}(T_m, \texttt{max\_rainfall DESC})$$

11: **Convert to DataFrame:**
12: Convert T_o into a Pandas DataFrame DF for visualization:

$$DF = \texttt{CONVERT}(T_o)$$

13: **Visualization:**
14: Create a bar chart for the maximum rainfall by state:

$$\texttt{PLOT\_BAR}(DF.\texttt{state}, DF.\texttt{max\_rainfall})$$

15: **Save Chart:**
16: Save the bar chart as an image:

$$\texttt{SAVE\_CHART}(\texttt{output\_path} = \texttt{"max\_rainfall\_by\_state.png"})$$

17: **return** Bar chart showing the maximum rainfall for each state. =0

3.5 Relationship with Temperature and Humidity

The relationship between temperature and humidity was analyzed. First, load the data into the Flink table and filter out a subset of records with temperatures between 15°C and 35°C for analysis. The filtered data is then converted into a Pandas DataFrame. Produced a scatterplot of the correlation between temperature and humidity

Algorithm 4 Relationship with Temperature and Humidity

Let $E \leftarrow$ EnvironmentSettings.in_batch_mode() {Initialize batch environment settings.}
Let $T \leftarrow$ TableEnvironment.create(E) {Create table environment.}
Define data source: $D \leftarrow$ "data.csv"
Execute SQL query to create table `weather` with schema:

$$\begin{aligned} \text{station\_id} &: \mathbb{Z} \\ \text{state} &: \text{String} \\ \text{timestamp} &: \text{String} \\ \text{temperature} &: \mathbb{R} \\ \text{humidity} &: \mathbb{Z} \\ \text{pressure} &: \mathbb{R} \\ \text{wind\_speed} &: \mathbb{R} \\ \text{wind\_direction} &: \mathbb{Z} \\ \text{rainfall} &: \mathbb{R} \end{aligned}$$

Specify connector settings: CSV format with file path D.
Load table: $W \leftarrow T$.from_path("weather")
Sample data: $S \leftarrow W$.limit(10^4)
Filter rows:

$$F \leftarrow \{x \in S \mid 15 \leq x[\text{temperature}] \leq 35\}$$

Select columns:

$$C \leftarrow \{(x[\text{temperature}], x[\text{humidity}]) \mid x \in F\}$$

Define function `table_to_dataframe`(C):

    Execute table: $R \leftarrow C$.execute()
    Extract schema and rows: $(schema, rows) \leftarrow R$.get_schema(), collect()
    Return DataFrame: DataFrame($rows, schema$)

Convert C to DataFrame: $df \leftarrow$ `table_to_dataframe`(C)
Plot df:

    Set figure size to 8×6 inches.
    Scatter plot: $y \leftarrow$ humidity, $x \leftarrow$ temperature
    Use points: green, alpha transparency.
    Add title: "Temperature vs Humidity".
    Label axes: $x \rightarrow$ Temperature (°C), $y \rightarrow$ Humidity (%)
    Save plot: "temperature_vs_humidity_sampled.png"
    Display plot.

Output: "Chart saved as temperature_vs_humidity_sampled.png". =0

4 Implementation

In here, I will discuss the implementation details of the project.

4.1 Environment Description

This is our development environment.

Component	**Details**
Operating System	Ubuntu 24.04.1 LTS
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
RAM	16GB
Kernel	6.8.0-51
Python	3.10

Table 1: Development Environment Description

4.2 Docker

In the term proejct, we used Docker [1] to build a containerized environment for Apache Flink, Now let's me introduce the tools and platforms we used in the session.

What is Docker?

Docker is an open-source platform designed to facilitate the development, deployment, and execution of applications using **containerization** technology. It enables developers to package applications along with their dependencies into lightweight, portable containers, ensuring consistency across various environments.

Core Concepts of Docker

1. **Containers:** Containers are lightweight and portable execution environments that include an application and all its dependencies. They ensure that the application runs consistently regardless of the underlying system.

2. **Images:** An image is a static snapshot that contains the application's code, runtime environment, libraries, and other dependencies. Containers are instantiated from images.

3. **Dockerfile:** A Dockerfile is a script that defines the steps to build a Docker image, specifying the environment and dependencies required by the application.

Advantages of Docker

- **Cross-Platform Compatibility:** Docker ensures consistency between development, testing, and production environments, eliminating the "works on my machine" problem.

- **Rapid Deployment:** Containers start much faster than traditional virtual machines, enabling near-instantaneous application deployment.

- **Resource Efficiency:** Docker containers use fewer resources compared to virtual machines, making them ideal for high-density deployments.

- **Portability:** Containers are infrastructure-agnostic and can seamlessly run on local machines, cloud environments, or hybrid setups.

Common Use Cases for Docker

1. **Application Development and Testing:** Docker provides a consistent development and testing environment, reducing deployment errors.

2. **Microservices Architecture:** Each service can run in an isolated container, simplifying scalability and maintenance.

3. **Continuous Integration/Continuous Deployment (CI/CD):** Docker streamlines automated testing and deployment, accelerating delivery pipelines.

4. **Resource Isolation:** Docker ensures applications are securely isolated, minimizing interference and enhancing security.

4.3 Flink Docker Architecture

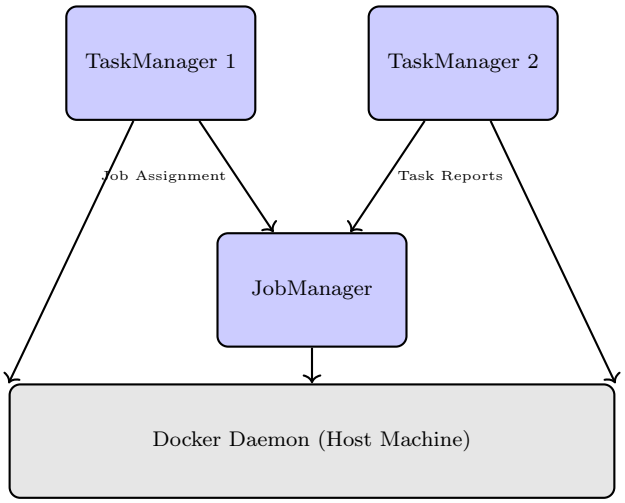


Figure 2: Compact Flink Cluster Architecture Using Docker

Basically, I write the Dockerfile to create the Flink Cluster, This is the Dockerfile I used in the project.

```
FROM flink:latest

RUN apt-get update && \
    apt-get install -y python3 python3-pip && \
    pip3 install apache-flink matplotlib
```

Base Image

```
FROM flink:latest
```

- The base image is the official **Flink** image tagged as `latest`. - This provides all necessary tools and dependencies to run Apache Flink.

Installing Python and Dependencies

```
RUN apt-get update && \
    apt-get install -y python3 python3-pip && \
    pip3 install apache-flink matplotlib
```

- `apt-get update`: Updates the package manager's index to ensure the latest versions of packages are available.

- `apt-get install -y python3 python3-pip`: Installs **Python 3** and **pip**, the package manager for Python. The `-y` flag ensures automatic confirmation for installation.
- `pip3 install apache-flink matplotlib`: Installs two Python libraries:

  - **apache-flink**: Provides the Python API for Apache Flink, allowing the development of Flink jobs in Python.
  - **matplotlib**: A popular Python library for data visualization, used to create charts and graphs, possibly for analyzing Flink's processed data.

Purpose of the Dockerfile

- Extends the official Flink image by adding Python support and additional tools.
- Enables Python-based development for Flink, making it suitable for developers working with Python APIs.
- Provides `matplotlib` for visualizing results from Flink's data processing workflows.
- Creates a lightweight and flexible container environment for data stream processing and visualization tasks.

4.4 Setup Flink Cluster

Based on the official Flink instructions, I wrote a shell script to automate the deployment process, allowing for the flexible configuration of the number of task managers.

```
#!/bin/bash

FLINK_PROPERTIES="jobmanager.rpc.address: jobmanager"
DEFAULT_TASKMANAGERS=2

# Function to display usage
function usage() {
    echo "Usage: $0 [NUM_TASKMANAGERS]"
    echo "       NUM_TASKMANAGERS: Optional, specify the number of TaskManagers to start (de
    echo "       Example: $0 3"
    echo "       Use '--help' to display this help message."
}

# Check for --help argument
if [[ "$1" == "--help" ]]; then
    usage
    exit 0
fi

# Get the number of TaskManagers from the first argument, or use the default
NUM_TASKMANAGERS=${1:-$DEFAULT_TASKMANAGERS}

# Validate NUM_TASKMANAGERS is a positive integer
if ! [[ "$NUM_TASKMANAGERS" =~ ^[0-9]+$ ]] || [[ "$NUM_TASKMANAGERS" -le 0 ]]; then
    echo "Error: NUM_TASKMANAGERS must be a positive integer."
    usage
    exit 1
fi

# Check if the Docker network 'flink-network' already exists
if ! docker network ls | grep -q "flink-network"; then
    echo "Creating flink-network..."
    docker network create flink-network
else
    echo "flink-network already exists. Skipping creation."
fi

# Start the JobManager container
echo "Starting JobManager container..."
docker run \
```

```
    -d \
    --rm \
    --name=jobmanager \
    --network flink-network \
    --publish 8081:8081 \
    --env FLINK_PROPERTIES="${FLINK_PROPERTIES}" \
    flink-python:latest jobmanager

# Start the specified number of TaskManager containers
echo "Starting $NUM_TASKMANAGERS TaskManager containers..."
for i in $(seq 1 "$NUM_TASKMANAGERS"); do
    docker run \
        -d \
        --rm \
        --name=taskmanager-$i \
        --network flink-network \
        --env FLINK_PROPERTIES="${FLINK_PROPERTIES}" \
        flink-python:latest taskmanager
    echo "TaskManager-$i started."
done

echo "Flink cluster setup completed. Access the JobManager UI at http://localhost:8081"
```

Script Breakdown

Default Variables

- `FLINK_PROPERTIES` specifies the JobManager's RPC address.
- `DEFAULT_TASKMANAGERS` sets the default number of TaskManagers to 2.

Usage Function

The `usage` function provides information about the script's usage, including the optional argument to specify the number of TaskManagers. If the `--help` argument is passed, the function displays the help message and exits.

Argument Handling

- The number of TaskManagers is retrieved from the first argument or defaults to 2.
- Validation ensures the argument is a positive integer. If invalid, an error message is displayed.

Docker Network Setup

The script checks if a Docker network named `flink-network` exists. If not, it creates the network to enable communication between Flink containers.

JobManager Deployment

The JobManager container is started with the following configurations:

- `--name=jobmanager`: Assigns the container name.
- `--network flink-network`: Connects the container to the Docker network.
- `--publish 8081:8081`: Exposes the JobManager UI on port 8081.
- `--env FLINK_PROPERTIES`: Sets the Flink properties.

TaskManager Deployment

The script launches the specified number of TaskManager containers in a loop:

- Containers are named `taskmanager-$i`, where `$i` is the loop index.
- Each container is connected to the `flink-network`.
- Flink properties are passed as environment variables.

Purpose of the Shell Script

- Automates the setup of a Flink cluster with configurable JobManager and TaskManager instances.
- Ensures seamless communication between Flink containers using a dedicated Docker network.
- Simplifies the deployment process for developers working with Apache Flink in a containerized environment.

5 Evaluation

The evaluation of the meteorological analysis focused on three key aspects: maximum rainfall, the distribution of heavy rain days, and temperature trends within specific states. The insights derived from these visualizations are detailed below:

5.1 Maximum Rainfall by State

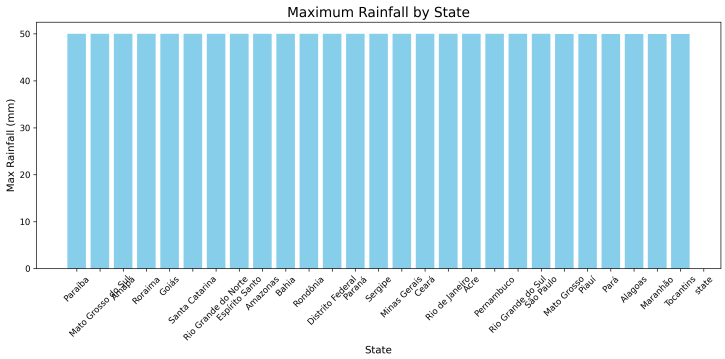


Figure 3: Maximum Rainfall by State

The bar chart illustrates the maximum recorded rainfall across various Brazilian states. The uniformity of the data suggests either consistent reporting thresholds or a limitation in data resolution. Further refinement of the dataset might be required to highlight variations in extreme weather patterns.

5.2 Heavy Rain Days Distribution (greater than 20mm)

The pie chart showcases the proportion of days with heavy rainfall below 20mm for each state. The distribution appears evenly spread, which could indicate a consistent meteorological pattern across states. However, potential bias in data collection or state-specific environmental factors influencing rainfall frequency should be further analyzed.

5.3 Average Temperature Trends in Bahia (Dec 1–7, 2024)

This line chart represents the temperature variations in Bahia, Brazil, over the course of a week. We can observe that the temperature fluctuates significantly, with the lowest being 15°C and the highest reaching 40°C. According to meteorological knowledge, this may be caused by the temperature differences between day and night.At the same time, it also reflects that Bahia has a typical tropical climate.

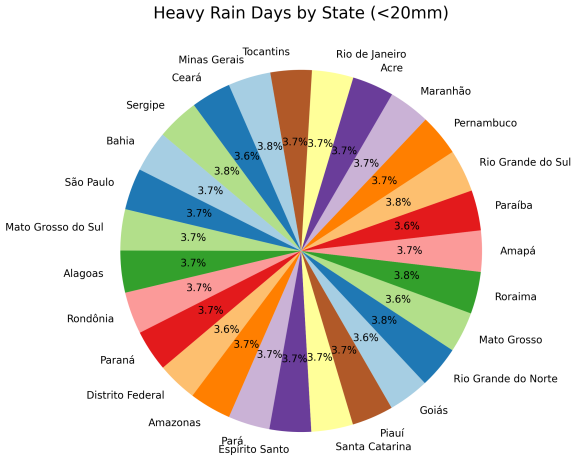


Figure 4: Heavy Rain Days Distribution

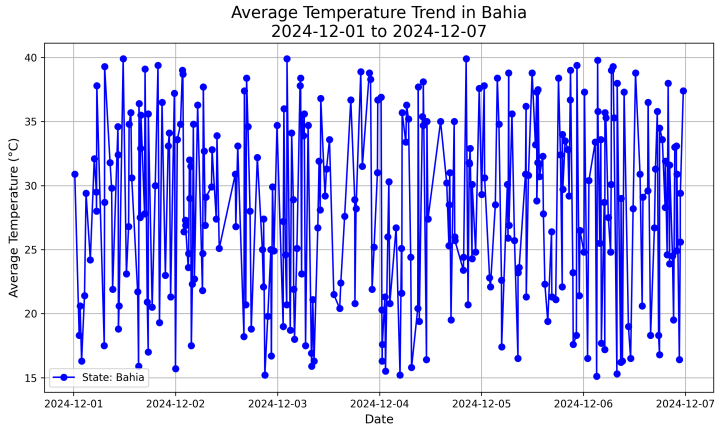


Figure 5: Average Temperature Trends in Bahia

5.4 Relationship with Temperature and Humidity (Dec 1–7, 2024)

The scatter plot in this chapter is about the relationship between temperature and humidity. We can draw the following conclusions.

Upon careful observation, certain specific temperature ranges might exhibit

higher or lower concentrations of humidity. For example:

- **Low Temperature Range (15°C–20°C):** There might be a higher concentration of data points with high humidity (e.g., 70%–100%) because cold air generally retains moisture more effectively.
- **High Temperature Range (30°C–35°C):** Data points might be more concentrated at lower humidity levels (e.g., 30%–50%) since higher temperatures often lead to faster evaporation of moisture.
- If the data is collected from a specific region, such as a tropical area, higher humidity levels are likely more common.
- Conversely, if the data is from a desert or arid region, lower humidity levels might dominate the dataset.

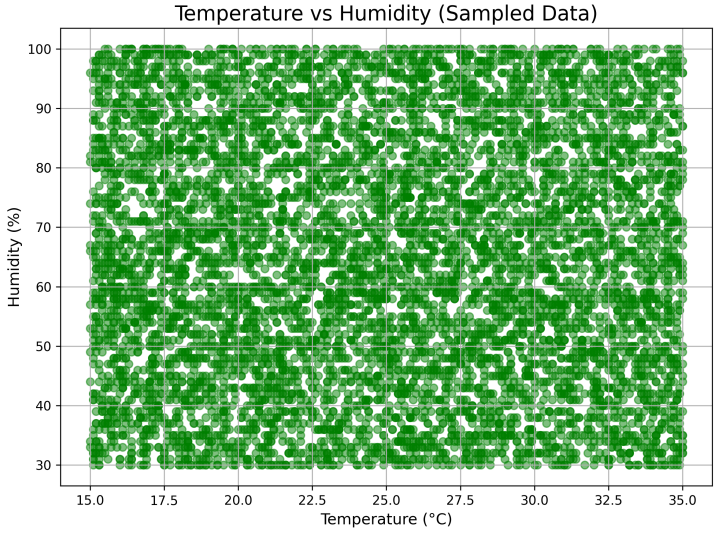


Figure 6: Relationship with Temperature and Humidity

6 Learning Experience

6.1 Technical Learning and Challenges

In this term project, I chose to use Docker from the very beginning of setting up the development environment, so as to learn how to containerize a distributed system to speed up the software development process, and to understand how the operating system communicates with containers and Flink

In addition to learning the tasks and mechanisms that each component is responsible for, at the beginning I used a small dataset to run the test results, but with the challenge of increasing the amount of data, you must consider whether the algorithm and data structure you designed are reasonable, otherwise the bath processing time will be long, which is not conducive to the operation of the whole system

6.2 Feedback

As for my personal impression, I think the term project designed by the professor was very good overall, and the open-mindedness allowed the students to freely use their creativity to construct their own ideas

Overall, I think this term project is very similar to simulating how to learn how to set up a development environment as an engineer from the very beginning, instead of just writing code, the second point is to learn how to learn to solve problems, how to make good use of resources to solve problems, I think English ability is very important, almost all technical communities are discussed in English-based languages, if you don't know English, you will encounter many difficulties during the development period, Finally, I think the most important point is to know how to express the question after the question, which is also the most confusing for many people, and the test is the degree of understanding of the knowledge background.

References

[1] Carl Boettiger. An introduction to docker for reproducible research. *SIGOPS Oper. Syst. Rev.*, 49(1):71–79, January 2015.

[2] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache flink: Stream and batch processing in a single engine. *The Bulletin of the Technical Committee on Data Engineering*, 38(4), 2015.