Weather with Cloudya:

A Full Stack Web Application to Make Weather Data More Accessible

Website:
Google Sites Informational Webpage, Weather with Cloudya App

GitHub:
BDA 600 Final Project

Video Link:
Video

Group Members

Miguel Ángel Bravo Martínez Del Valle

Fernanda Carrillo Escarcega

Riley Rutan

San Diego State University

BDA 600 Big Data Analytics Capstone Seminar

Instructors

Dr. Ming-Hsiang Tsou and Dr. Gabriela Fernandez

May, 2025

Table of Contents

1. Abstract

This project focuses on improving public access to real-time climate data by creating a user-friendly, AI-powered application. Our team developed a system using a Python backend, a custom React.js frontend, and FastAPI to connect the two, allowing users to navigate climate data and ask questions through the chatbot, Claudia. The application visualizes key weather variables including temperature, precipitation, cloud cover, humidity, and dew point allowing the user to explore multiple environmental conditions in their region.

2. Problem Statement

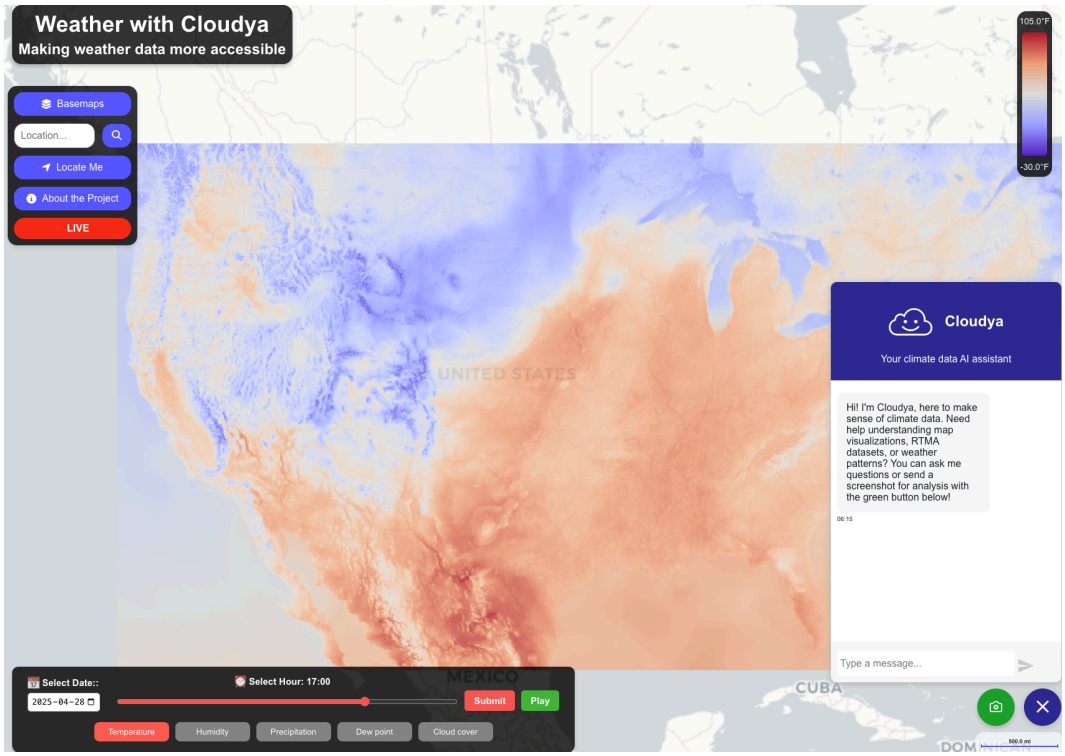
As climate change worsens, real-time climate data is more relevant than ever for making informed decisions. However, much of this data is presented in technical formats that are difficult to interpret for the general public creating a gap between available data and meaningful action. Our project addresses this problem by delivering an interactive application that visualizes important climate variables such as temperature, precipitation, cloud cover, humidity, and dew point. By integrating a conversational AI chatbot, we hope to simplify the user experience and make complex environmental data more accessible for public safety planning, climate education, and community awareness.

3. Project Objectives

The primary objective of this project is to create an accessible, interactive climate data platform that helps users better understand and respond to weather conditions. Our system is created to

simplify complex climate data and deliver real-time insights through intuitive visualization and an AI-powered chatbot. Our goals include:

- Collecting and integrating real-time climate data from sources such as RTMA, focusing on variables like temperature, precipitation, cloud cover, humidity and dew point.
- Designing and developing a Python-based backend and React.js frontend, connected via FastAPI, to create a seamless and responsive user experience
- Building a conversational chatbot that simplifies data, responds to user's questions, and provides recommendations based on current weather conditions.
- Developing an interactive dashboard that lets users explore climate patterns geographically, making use of geospatial visualizations and data filters.
- Ensuring system flexibility and scalability by containerizing the platform with Docker for deployment across different environments like the Raspberry Pi.



4. About the Data

The data that our application uses to create visualizations is the Real-Time Mesoscale Analysis (RTMA) from the National Oceanic and Atmospheric Administration's (NOAA) National Centers For Environmental Prediction (NCEP). This data product is gridded with high spatial resolution at 2.5 kilometers, and high temporal resolution at hourly intervals (De Pondeca et all 2011). The RTMA product is an analysis system for near-surface weather conditions, and includes many variables such as temperature, cloud cover, precipitation, dew point, and humidity, and includes coverage for the continental United States, Alaska, Hawaii, Puerto Rico, and Guam. This data is derived through a combination of surface recordings, satellite remote sensing, and model outputs to provide the most accurate results of near surface conditions. This data was designed to support real-time monitoring, emergency management, and short-term planning. The RTMA product is available through a public AWS S3 bucket through NOAA's Open Data Initiative, which provides data from April 2013 to the present. This allows our application to access data from over 4000 days, or over 91 terabytes of information. The RTMA dataset can be found at the following link:

https://registry.opendata.aws/noaa-rtma/

5. Literature Review

As climate change intensifies and the frequency of natural disasters increases, the need for accessible, real-time climate data has grown significantly. However, while large volumes of weather data are collected worldwide, there are multiple barriers in terms of access, usability, and interpretation, mainly for non-expert users.

Eggleton and Winfield (2020) analyze the challenges of managing open climate data, especially the issues of scale, standardization, and accessibility. Some institutions like the UK's Centre for Environmental Data Analysis (CEDA) often struggle with storing and serving huge amounts of weather information, with petabytes of data and terabytes being added daily. The authors emphasize the relevance of consistent metadata practices, standardized file formats, and adherence to FAIR principles (Findability, Accessibility, Interoperability, and Reusability) to make sure that climate datasets remain usable over time. While the focus of their work is mainly technical and back-end oriented, the implications are clear: when datasets are hard to store, structure, or access, it is even more difficult to translate them into meaningful insights for the general public or decision makers.

The gap between available climate data and its practical real world use is also highlighted in a large-scale review by Bhardwaj and Khaiter (2023), they analyzed over 700 publications about climate data analytics, and their findings are that only a small number of existing studies and tools focus on building user-facing platforms for analyzing or visualizing climate data. The majority of applications remain technical in nature, designed for experts and lacking accessible interfaces or interactive functionalities. The authors argue that even when data analytics and AI

have great potential for advancing climate understanding, the field of climate informatics is still in the early stages. They mention that there is a growing need for systems that not only analyze data, but also support real-time, user-friendly interaction and decision-making. This supports the focus of this project, that is to present complex environmental variables, such as temperature, humidity, cloud cover, and precipitation in a simplified, visual, and interactive format.

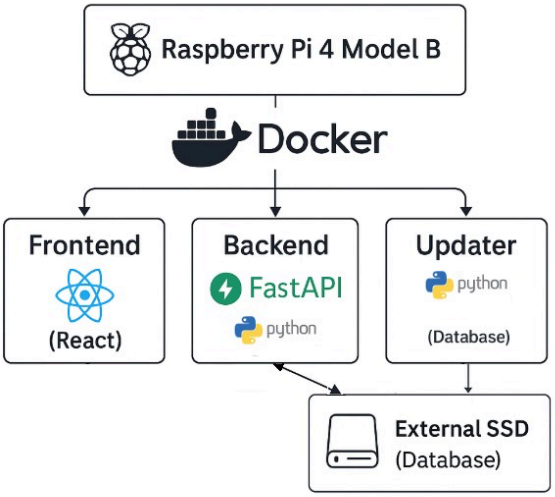
Additionally, Lumley et al. (2022) conducted a comparative analysis of over 20 web-based climate visualization tools. Their research showed that even when many dashboards exist, most of them are not designed with general users in mind. These tools often lack interactivity, provide limited variable coverage, and tend to fall short in supporting users through the analysis or decision-making process. In particular, the authors mention that tools almost never go beyond basic visualizations to offer conversational or interpretive assistance. They also provide a framework for climate visualization tools evaluation including purpose, data content, visual representation, interactivity, and web technology. This framework provides a useful foundation for designing and building interactive dashboards that are both scientifically rigorous and publicly accessible.

All of these studies emphasize the importance of climate data and the need for user-centered tools to make that data truly usable for the general public. While open data initiatives have increased availability, many existing platforms remain technical, static, and difficult for non-experts to navigate. This creates barriers for individuals and communities who need clear, timely information to understand local climate risks and make informed decisions.

This project addresses those gaps by developing a React-based geospatial dashboard paired with an AI-powered chatbot. The combination of the two allows users to visualize key environmental variables, such as temperature, precipitation, humidity, and cloud cover, and also to interact with the data through a conversational interface. By simplifying complex information and supporting real-time exploration, this tool moves beyond traditional dashboards and aligns with recent calls in the literature for more flexible, intelligent, and user-friendly systems that make climate data more actionable.

6. Methodology

The development of the Weather with Cloudya application is structured around five key components. First, a backend data pipeline was built using Python and FastAPI to handle data ingestion and API services. Second, a database system was implemented with automated update capabilities to manage and store incoming weather data for instant retrieval. Third, the frontend was developed using React.js to provide an interactive user interface. Fourth, Cloudya, an AI-powered chatbot, was integrated as a climate assistant and data interpreter. Finally, the application was deployed using Raspberry Pi hardware for web hosting and Docker to containerize each component for streamlined deployment and maintenance. This project structure can be seen in the figure below



6.1. Backend Data Pipeline and FastAPI Integration

The backend of our system is designed to efficiently handle the end-to-end process of ingesting, transforming, storing, and serving real-time weather data to the frontend dashboard and the Cloudya chatbot. To achieve this, we use the Real-Time Mesoscale Analysis (RTMA) dataset from NOAA's National Centers for Environmental Prediction, which provides hourly weather data with high spatial resolution across the United States. The focus of our pipeline is on five key variables: temperature, dew point, humidity, cloud cover, and precipitation. Data ingestion is automated using the Herbie Python library, which simplifies access to NOAA's GRIB2 files by managing the indexing and file retrieval process. Once downloaded, the raw RTMA data undergoes several preprocessing steps. Invalid or extreme values are filtered out, and the original rotated grid format is converted into a regular latitude-longitude grid through interpolation using SciPy's griddata function. The dataset is then cropped to the extent of the United States, applying a resolution of 0.05 degrees to strike a balance between detail and performance.

After interpolation, the data is stored in an xarray DataArray structure, which organizes values along consistent latitude and longitude dimensions, ensuring compatibility with geospatial tools and supporting subsequent reprojection. To integrate seamlessly with web-based mapping libraries such as Deck.gl, the data is reprojected from geographic coordinates (EPSG:4326) to the Web Mercator projection (EPSG:3857) using the rioxarray library. From here, two types of outputs are generated: (1) a visual PNG image that serves as a map overlay, created using matplotlib and flipped along the Y-axis to match Deck.gl's orientation, and (2) a structured JSON file containing down sampled or full-resolution grid values, geographic metadata, measurement units, a timestamp, and summary statistics such as minimum and maximum values. To improve loading performance and reduce bandwidth, data values are compressed using zlib and encoded in base64 before being embedded in the JSON response.

Once processed, the outputs are stored on an external SSD mounted to the Raspberry Pi, which functions as the persistent local database. To balance performance and storage, the system retains approximately 80 days of hourly data per variable. This rolling archive allows for historical exploration while keeping the system lightweight and efficient for edge deployment. When the frontend or chatbot requests data, the backend first checks for the file's existence in the local database. If the requested file is available, it is served instantly using ORJSON, a high-performance JSON parser that reduces latency. If not, the backend triggers the entire pipeline on-demand to fetch and generate the missing data in real-time.

This full pipeline—from ingestion to real-time delivery—is exposed through a FastAPI backend running in a Docker container on the Raspberry Pi. The backend responds to frontend queries

via RESTful endpoints, dynamically serving the appropriate JSON and PNG assets. It also supports Cloudya's requests by providing structured responses with climate variable data and associated metadata, enabling AI-assisted interpretation. Despite the limited hardware resources of the Raspberry Pi, the backend demonstrated high reliability and responsiveness thanks to its modular, optimized design and lightweight dependencies. This architecture ensures that users receive the most recent and relevant climate data while maintaining the flexibility to dynamically process and visualize additional requests.

6.2. Data Storage and Retrieval

After preprocessing and saving the RTMA data into structured PNG and JSON formats, a system for data storage and retrieval was developed to ensure efficient, flexible access for the dashboard and chatbot. This system was made to deliver both recently processed and dynamically generated real-time data, balancing speed, scalability, and flexibility.

To allow for fast access to the most recent weather data, the system maintains a rolling local database that saves the past 80 days of processed files. These files are stored in lightweight formats – compressed JSON files for data interactions and PNG images for visual overlays. When a user requests weather data, the system first checks if the requested data already exists in the local database. If the file is available, it is retrieved instantly using ORJSON, a high-performance JSON parsing library that ensures minimal latency, and flexibility.

If the user requests a file that does not exist in our database, the system triggers the RTMA processing pipeline to download, interpolate, reproject, and visualize it in the dashboard.

6.3. Application Frontend

The application frontend, or software that creates the user interface (UI) was developed with React js. React is a JavaScript library that is used for building fast and interactive UIs, where content is dynamically updated without having to reload the page. React is designed to break a webpage into reusable and self-contained components or files that make code easier to manage, and lets developers write HTML-like code within their JavaScript. In this project, we use React to create a smooth and dynamic frontend experience.

The development of the React frontend can be broken down into the features that the user can interact with to visualize RTMA. On the left side of the screen, there are several buttons in the menu. The Basemaps button allows the user to select from a variety of basemaps, from simple dark or light or dark basemaps, to a satellite imagery basemap. This allows users to customize the look of this application. There is a Location Search box, which allows the user to type in a city, state, or place to send the map to that location. The Locate Me uses the user's IP address to move the map and places a pin at the user's current location, allowing users to look at weather data in their area. The About the Project button takes the user to another page that gives more information about how the application was developed, as well as a brief video tutorial on how to use the application.

The core functionality of the application is the interactive web map that is built with Deck.gl, an open-source visualization framework that was developed by Uber. Deck.gl is designed to render large geospatial data as complex layers like points, heatmaps, and 3D objects with high performance and visually appealing results, and allows the user to seamlessly pan and zoom on

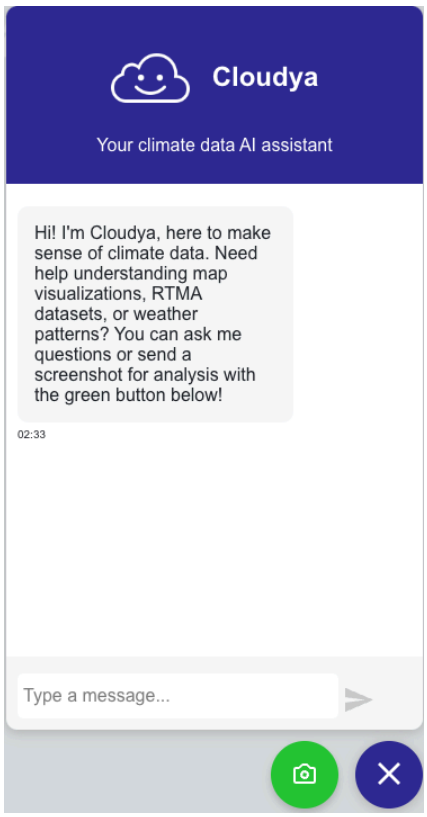
the map. When data is queried using the data selection menu, a heatmap of the data is displayed on the map. When a user hovers their mouse over a location, a tooltip appears that displays the data value, latitude, and longitude at that location. On the top right of the application, a color bar legend shows the units and color scale, and dynamically changes when different data variables are visualized.

The data selection menu is located at the bottom left of the application. The date selection widget allows users to select a previous or current date to query data for. If the selected date lies outside of the range that our database is currently storing, a popup will appear to let the user know that their query will take a bit longer than usual, as the Python backend will need to download and transform this data before it can be visualized on the map. The hour selection slider allows the user to select the hour of the day in Pacific Standard Time of the queried data. Below that, the variable selection buttons allow the user to choose which weather variable they would like to visualize. The red Submit button on the right side of the menu will send the query to the Python backend, and will be greyed out and display “Loading” until the data response is received and visualized on the map. The green Play/Pause button will continually increase the hour and submit to data query, to effectively animate the data visualizations over time.

Finally, the chat box on the right side of the screen proves the interface for the user to interact with Cloudya, and was built using the react-chat-widget npm library. The blue chat box launcher button allows the user to open and close the chat. The chat initially loads with a welcome message, where Cloudya introduces herself and suggests a few questions that a user could ask. The chat box will allow the user to type a message and send it to Cloudya, displays

when a response is loading, and shows her response message. The green screenshot button briefly closes the chat, takes a screenshot of the application, and sends it to Cloudya for analysis. While a response is loading, the screenshot button is greyed out so it cannot be clicked until a response is displayed.

6.4. Cloudya: AI-Powered Climate Assistant



Cloudya is an AI powered climate data assistant and interpreter, to translate complex climate data and visualization into user-friendly information. She was built to help users understand the data that they are visualizing, learning more about the data source, and provide additional insights and contexts to extreme weather events. Cloudya is built on Meta's Llama 3.2-Vision large language model (LLM). This model is multimodal, meaning it can take both text and images as an input. This is a key feature for Cloudya to interpret the data visualizations. Llama 3.2-Vision is designed for image reason, captioning, visual recognition, and answering general questions about an image. It has support for several languages and outperforms many popular open-source and closed multimodal models on industry benchmarks (Ollama). The application accesses this model with an API through OpenRouter, which acts as a unified interface for users to access and use a wide variety of LLM through a single API.

The green screenshot button was developed and placed just below the Cloudya chatbox. When pressed, a screenshot of the application is sent to the model. This allows Cloudya to see the data that the user is currently visualizing to help interpret the results to provide the user with a deeper spatial analysis of the data. Along with a prompt, the selected variable and units of measurement are passed directly to the model with the image input.

Cloudya's behavior when conversing with the user was developed through prompt engineering. This includes giving the model instructions to guide it towards desired outputs. The model is provided with context, instructions, examples, and relevant information about the RTMA dataset. To create the best climate data assistant and interpreter, Cloudya is given the following instructions:

- Be clear and concise using simple language and avoid technical jargon.
- Limit output to a few paragraphs to be readable within the chatbox.
- Do not make up or exaggerate details to avoid model hallucinations.
- Recognize the time and date of the data that is being visualized by inspecting the date selector at the bottom of the screen.
- Analyze and summarize the key spatial patterns and anomalies of data on the map.

6.5. Physical Deployment (Raspberry and External SSD)

A. Deployment Goals and Motivation

The primary deployment objective of this project was to build a fully self-hosted, lightweight, and scalable climate data application capable of running on low-power hardware. To achieve this, a Raspberry Pi 4 Model B with 8GB of RAM was selected as the production environment. This platform offers an affordable and energy-efficient solution for hosting real-time data services while maintaining sufficient performance for continuous backend operations, frontend rendering, and automated data updates.

Our vision extended beyond creating a one-time academic project. We aimed to develop a persistent, publicly accessible platform—something that would continue running indefinitely and be usable by real communities, not just for classroom demonstration. To that end, we prioritized deploying on hardware we could control and maintain independently, rather than relying on commercial cloud services such as AWS or Azure, which can incur significant long-term costs.

Deploying on a Raspberry Pi also aligned with our personal learning goals. We wanted to deepen our understanding of Linux-based systems, command-line operations, and systems administration, while gaining hands-on experience with Ubuntu Server and edge device deployment. This hands-on approach allowed us to build not only a robust application, but also the operational knowledge required to maintain and scale it effectively over time.

B. Dockerization Strategy

To ensure consistency, portability, and scalability across development and production environments, the entire application was containerized using Docker. Docker allowed us to package the backend (FastAPI), frontend (React), and the automated updater service into isolated containers, each with their own dependencies and configurations. This eliminated compatibility issues and made the system fully reproducible across different machines and operating systems, including ARM-based devices such as the Raspberry Pi.

The decision to use Docker was motivated by its ability to abstract away the underlying system differences. Regardless of whether the application was running on a local development machine, a cloud server, or a single-board computer like the Raspberry Pi, Docker ensured that all services behaved identically. This consistency was especially important given our use of open-source libraries that depend on specific Python and Node.js versions.

Each container in the system was built from a custom Dockerfile tailored to the specific needs of the service. The backend container includes a minimal Python environment with FastAPI, Uvicorn, and numerical libraries; the frontend container is based on a lightweight Node.js image and uses a static file server to serve the compiled React application; and the updater container runs a scheduled pipeline to download, process, and store climate data in an external SSD volume.

All three containers are orchestrated using docker compose, which defines their build instructions, runtime configuration, network settings, and shared resources. By defining a

dedicated Docker network, we enabled secure internal communication between services without exposing unnecessary ports to the public. This structure also made it easier to apply resource limits, implement restart policies, and scale the application to other devices in the future.

Backend Containers

The backend container, which runs a FastAPI application served with Uvicorn, was specifically designed to operate efficiently on the Raspberry Pi 4 Model B, which uses an ARM64 architecture. To ensure compatibility and reduce resource consumption, the Dockerfile for this service is based on a lightweight Python 3.10-slim image and includes only essential system dependencies such as gcc, make, and python3-dev. These packages are required to compile scientific libraries used in the data processing and API logic. The backend listens on port 8000 inside the container, and this port is exposed in the docker-compose.yml file to allow both the frontend container and external users (via API calls) to access the service. When the container is built and run on the Raspberry Pi using docker-compose up -d, the backend starts automatically and becomes available to respond to HTTP requests. In this environment, performance and memory usage were carefully considered, and the application proved capable of handling both local API traffic and external requests routed through the Cloudflare tunnel. The container is also configured with the restart: always policy to ensure resilience: if the Raspberry Pi reboots or the container crashes unexpectedly, the backend service restarts automatically without requiring manual intervention. Finally, the backend shares a common Docker network with the frontend and updater containers, allowing for seamless internal communication between

services. This setup enables the backend to serve processed climate data from shared volumes and expose structured JSON responses for interactive use in the application's dashboard.

Frontend Containers

The frontend container is responsible for serving the React-based user interface of the application. On the Raspberry Pi, this service was built using a minimal Node.js 18-slim image, selected for its small footprint and compatibility with ARM64 systems. Once the React application was compiled using npm run build, the container was configured to serve the static files using npx serve, a lightweight HTTP server ideal for production deployment. Internally, the container listens on port 3000, which is mapped to port 80 in the docker-compose.yml file, allowing users to access the web application without specifying a port in the URL. This configuration ensures that all web traffic directed to datamigos.win via the Cloudflare tunnel is correctly routed to the frontend. The Dockerfile was carefully optimized to exclude development dependencies and reduce image size, which is critical when working on hardware with limited storage and processing capabilities. As with the backend, the frontend container is part of the same internal Docker network, enabling it to communicate directly with the backend service using the hostname backend without relying on hardcoded IPs. It is also configured with the restart: always policy to maintain high availability, and all environment-specific configurations—such as API base URLs—are injected via .env files, ensuring that the frontend automatically adapts to the deployment environment. This setup allowed for a responsive, production-ready interface to run smoothly on the Raspberry Pi, providing users with real-time access to climate data visualizations and chatbot interactions.

Updater Container

The updater container is a dedicated service responsible for keeping the climate data repository continuously updated with the most recent RTMA files. Running independently from the backend and frontend, this container executes a Python-based pipeline every hour to fetch, process, and store high-resolution weather data for five key variables: temperature, dew point, humidity, cloud cover, and precipitation. The process begins with downloading GRIB2 files via the Herbie library, followed by interpolation and reprojection to align the data to a uniform Web Mercator grid. Once processed, the data is exported in two formats: a visual PNG image for map overlays, and a compact JSON file that includes compressed values (via zlib and base64), metadata, and statistical summaries for interactive use. These outputs are saved in a shared persistent volume mounted to the Raspberry Pi's external SSD. Although the system is capable of storing over two years of hourly data for each variable, doing so would require substantial disk space and considerable computational time, especially given the limitations of edge hardware. For this reason, the updater maintains a rolling window of approximately 80 to 100 days per variable, striking a balance between historical coverage and performance. The container is configured to run automatically using the restart: always policy and includes logging mechanisms to monitor its update cycles. This design ensures the application always has access to recent, high-quality environmental data without manual intervention, enabling both the dashboard and chatbot to operate with up-to-date information in real-time.

C. Environment Configuration

To ensure proper communication between services and correct adaptation to the deployment environment, environment variables were used extensively throughout the project. On the Raspberry Pi, a dedicated .env file was created to define production-specific configurations, such as the public-facing API URL (REACT_APP_API_URL) pointing to http://datamigos.win/api. This allows the frontend to correctly route requests through the Cloudflare tunnel to the FastAPI backend, rather than relying on localhost or other development addresses used during local testing. To prevent misconfigurations, the .env file was explicitly excluded from version control via .gitignore, ensuring that environment-specific values are not overwritten when pulling updates from the GitHub repository. This approach provided a clear separation between local development and production deployment, and was particularly important in the Raspberry Pi context, where all services run in containers that rely on consistent and persistent environment values. The .env file on the Raspberry Pi resides outside the containers but is passed into them during runtime via docker-compose, enabling seamless reconfiguration without needing to rebuild images. This setup not only ensured flexibility and maintainability, but also reduced the risk of exposing sensitive variables or breaking production functionality due to mismatched configurations between environments.

D. GitHub Integration and Workflow

To streamline the deployment process and maintain version control, the Raspberry Pi was securely connected to the GitHub repository using SSH authentication. This setup allowed us to pull the latest project updates directly onto the device without exposing credentials or manually

entering login information. An SSH key pair was generated on the Raspberry Pi and added to the GitHub account, enabling encrypted and passwordless communication with the repository. This configuration ensured that code updates, including changes to the backend, frontend, or updater, could be fetched reliably with a simple git pull. Additionally, the GitHub workflow was designed to respect environment-specific settings: the .env file, which contains production configurations, was excluded from version control to avoid being overwritten during updates. This approach gave us full control over how the application behaves in the Raspberry Pi environment without interfering with development setups. The combination of GitHub and SSH made the update process fast and secure, supporting a sustainable and scalable workflow for managing the system long-term. Once updates were pulled, rebuilt images could be generated and deployed via Docker, allowing changes to go live in just a few steps, directly from the terminal of the Raspberry Pi.

E. Cloudflare Tunnel Setup

To make the application securely accessible from the internet without exposing the Raspberry Pi's IP address, a Cloudflare Tunnel was implemented directly on the device. After setting up a Cloudflare account and configuring the domain datamigos.win, we installed the cloudflare client on the Raspberry Pi, choosing a version compatible with its ARM64 architecture. Authentication was completed via a secure login process, linking the device to the Cloudflare account. A tunnel was then created and assigned to the domain, and a configuration file (config.yml) was written to route traffic appropriately: all requests to datamigos.win are forwarded to the frontend running on port 80, while those including /api are redirected to the backend on port 8000. This

setup allows seamless and secure access to both the user interface and the API endpoints without the need to open any ports on the Raspberry Pi's local network. Because all services run locally in containers, the tunnel effectively bridges the gap between the public web and the internal Docker network, while maintaining strong security and isolation. This method not only reduced setup complexity compared to traditional reverse proxies or static IP services, but also ensured encrypted traffic and DDoS protection through Cloudflare's infrastructure, making it an ideal solution for deploying web applications from edge devices like the Raspberry Pi.

F. Automation with systemd

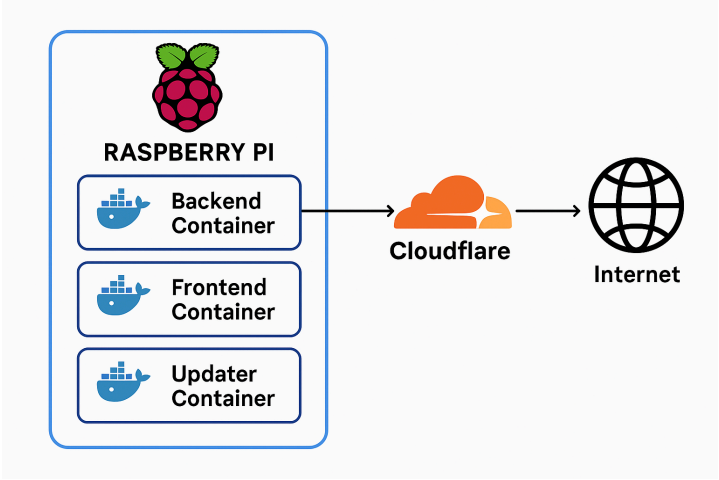
To ensure the system remains fully autonomous and resilient, key services on the Raspberry Pi were configured to start automatically upon boot using systemd. Two custom service units were created: one for the Docker containers (docker-compose-app.service) and another for the Cloudflare Tunnel (cloudflare.service). The Docker service ensures that the backend, frontend, and updater containers are started in detached mode (docker-compose up -d) as soon as the system is ready and Docker is initialized. It is configured with appropriate dependencies to wait for network availability before execution, and includes a restart policy to recover from unexpected failures. Similarly, the Cloudflare Tunnel service is set up to wait until Docker is fully operational before launching, using a pre-start condition that loops until docker ps confirms availability. This guarantees that the tunnel does not attempt to forward traffic until the frontend and backend are active. The tunnel is then executed with a specific UUID tied to the project's credentials, maintaining the secure connection to Cloudflare's edge network. Both services are enabled with systemctl to persist across reboots, and logging is directed to system

logs or designated files for easy debugging. This automated setup allows the entire application—data processing, visualization, and external access—to recover seamlessly from power outages or system restarts, making the Raspberry Pi a stable, long-term host for the project without requiring manual intervention.

G. Deployment Testing and Observations

Extensive testing was conducted on the Raspberry Pi to evaluate the performance, stability, and responsiveness of the full deployment. Despite the limited hardware resources compared to traditional cloud servers, the Raspberry Pi 4 with 8GB of RAM handled the system reliably, running all three containers—backend, frontend, and updater—concurrently without significant memory pressure. Initial performance tests showed that smaller JSON files and map overlays loaded quickly, while larger datasets occasionally introduced delays, particularly during decompression or JSON parsing on the frontend. These observations led to further optimizations, such as compressing data payloads and limiting the historical data range to improve responsiveness. The Cloudflare Tunnel proved to be highly stable, maintaining uninterrupted external access for days without intervention. Logs from the CloudFlare and docker-compose-app services confirmed that both services recovered automatically after controlled reboots and accidental shutdowns, validating the robustness of the automation setup. By simulating common failure scenarios, such as network disconnections and forced restarts, we confirmed that the system consistently returned to full operational status within seconds. These tests demonstrated that a Raspberry Pi, when properly configured and

optimized, is a viable and efficient platform for deploying real-time, interactive web applications with minimal maintenance requirements.

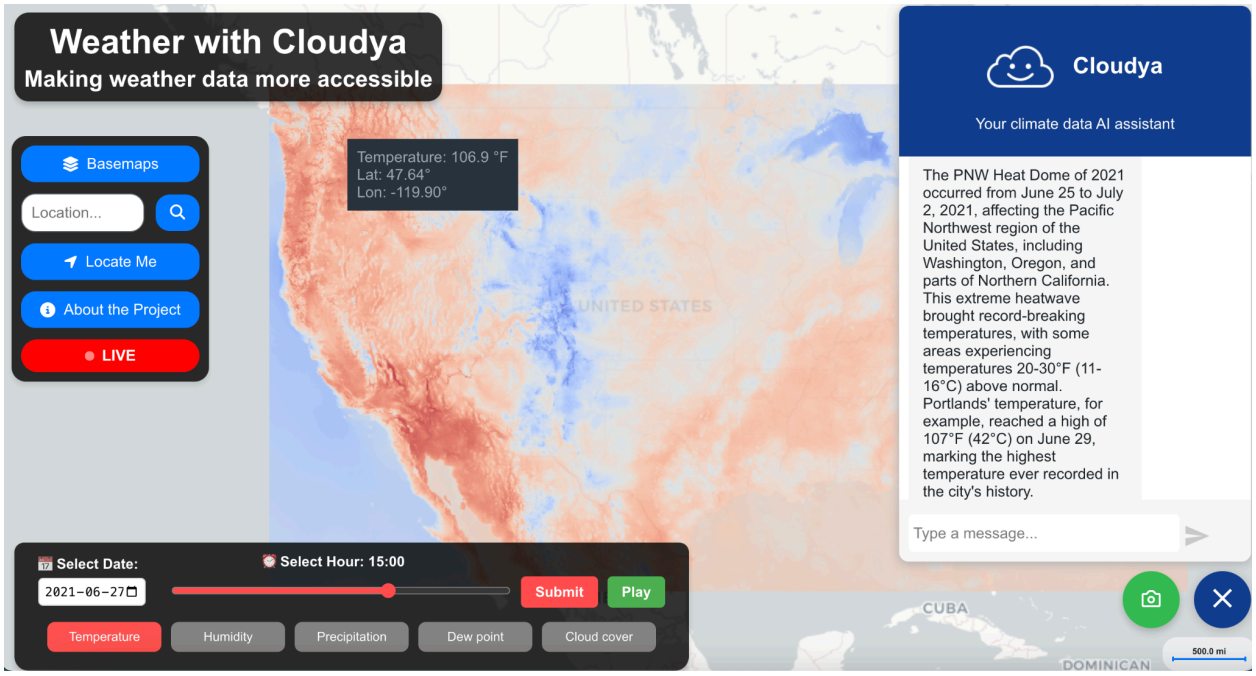


7. Results & Impact

The application, Weather with Cloudya successfully delivers an interactive climate data dashboard that allows for real-time exploration of environmental conditions across the United States. As shown in the image below, users are provided with a high-resolution map overlay that shows current temperature data using a gradient color scale. The visualization is provided using the reprojected files generated from RTMA data, allowing for accurate alignment with the base map and smooth integration into a web-based environment.

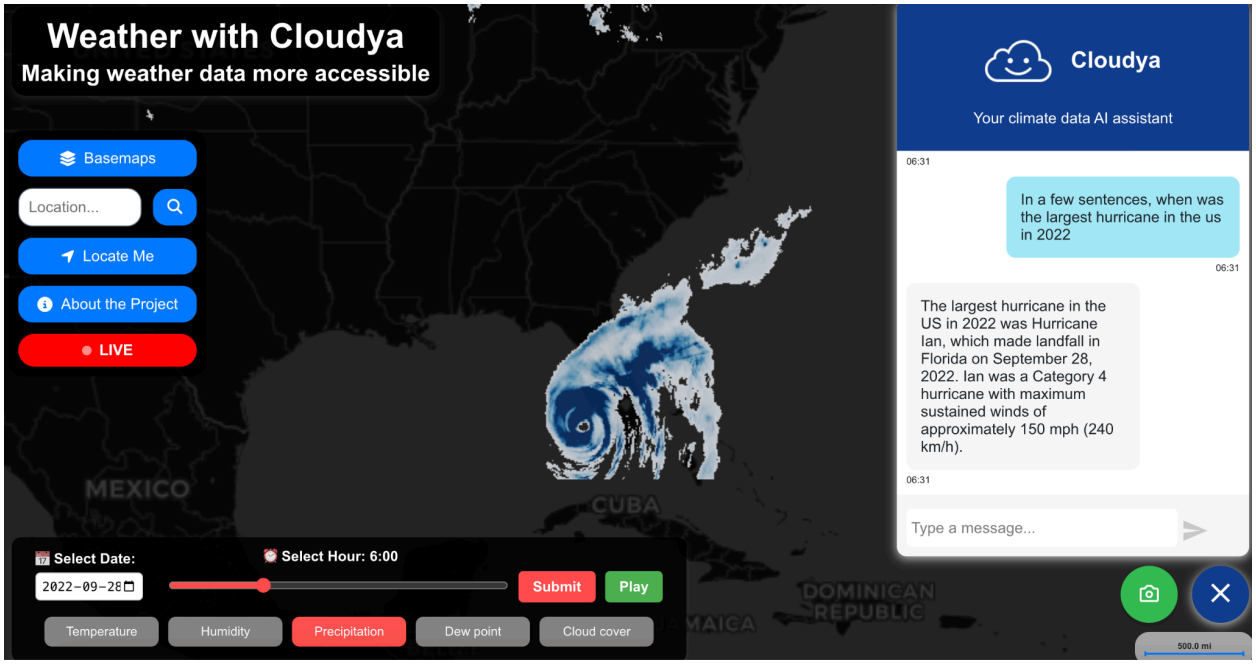
In this example the user requests for the temperature variable on June 27, 2021, when hovering over the image the temperature (106.9 F) is displayed at a location in the western U.S. confirming that the data is correctly processed and shown on the map. Then, Cloudya provides contextual explanations about the heat dome event in the U.S. during that time, combining real-time data with historical climate information. This example shows that both the data

pipeline and AI assistant are working as intended, supporting the project's goal of turning complex weather data into more accessible and easy to understand visualization.



In another example, the user first engages with Cloudya and asks about the largest hurricane in the U.S. in 2022. Cloudya accurately responds with a short summary, identifying Hurricane Ian and providing relevant information about its landfall date, strength (Category 4) and impacts.

The user then requests the date provided by Cloudya dashboard displaying precipitation data from September 28, 2022, at 6:00 am. This visualization captures a large-scale storm associated with Hurricane Ian just before it made landfall in Florida. This screenshot was taken while playing an animation of the precipitation variable, which is another example of the dashboard's feature. The map shows a well defined spiral of heavy precipitation centered in the Gulf of Mexico with darker blue areas representing higher rainfall intensity.



These examples demonstrate that the application is working as intended, successfully retrieving and visualizing real-time climate data while also providing users with relevant, AI-generated context. The combination of dynamic mapping and conversational support highlights the product's ability to make complex weather data more accessible, interactive, and informative.

8. Discussion and SWOT ANALYSIS

Developing this application revealed multiple key insights about working with real-time climate data and creating tools or accessibility. One of the biggest challenges was handling the format and structure of RTMA data. Since the raw files are stored in GRIB2 format, preprocessing involved several steps such as interpolation, reprojection, and cleaning before the data could be visualized effectively. Setting up this pipeline required balancing detail and performance, especially when generating high-resolution outputs in real-time.

Another challenge was optimizing the data storage and retrieval process to make the user experience smooth. By maintaining a rolling local database of recent data and using on-the-fly generation for older dates, the system was able to provide flexibility without compromising speed.

The interaction of the Claudya chatbot added a unique layer of interaction, turning a traditional data viewer into an educational tool. It became clear that even simple explanations could improve user experience and understanding by a lot, especially for people without a technical background.

Overall, the project demonstrated that it is possible to build a system that minimizes the gaps between raw climate data and everyday users. While the current version is functional and informative, there is room for improvement in areas like adding more variables, and increasing the chatbot ability to tailor its responses to the data shown on the map.

SWOT Analysis

Strengths	Weaknesses
● Full-stack application hosted on a local server eliminates ongoing cloud costs ● Real-time weather data retrieval from trusted RTMA sources ● Novel tool to help the average user visualize and understand big climate data with AI	● If Raspberry Pi hardware fails, system functionality will be temporarily disrupted ● Meta Llama 3.2 Vision model can occasionally provide unreliable results ● Database will eventually need scaling to support longer-term historical storage
Opportunities	**Threats**
● Expand to include additional weather variables, predictive models, or more detailed geospatial coverage ● Allow for time series analysis of weather variables across days, weeks, or months	● Power outages or SSD failure could interrupt service ● Cybersecurity vulnerabilities if public API access points are not properly secured

9. Conclusion

Weather with Cloudya demonstrates how real-time climate data, often locked behind complex formats and technical barriers, can be transformed into an accessible and engaging user experience. By combining automated data pipelines, intuitive geospatial visualizations, and an AI-powered assistant, our system empowers users to explore environmental conditions in a way that is both interactive and informative. This tool is more than a dashboard—it's a bridge between raw scientific data and everyday decision-making, built to serve communities, educators, planners, and individuals seeking to better understand their environment.

In the context of increasing climate volatility, democratizing access to high-resolution, real-time weather information is not just a technical achievement—it's a matter of equity and public safety. Deploying this system on a Raspberry Pi reinforces that vision by making the solution low-cost, scalable, and maintainable, even in settings without access to traditional cloud infrastructure. The project highlights how edge computing and open data can intersect to build sustainable, community-focused tools with real-world impact.

While this is the first iteration of Cloudya, the platform lays a strong foundation for future growth. With its modular design, robust backend, and flexible architecture, it is well-positioned to incorporate advanced features such as alerts, predictive models, and personalized user accounts. Ultimately, our goal was not just to create a successful capstone project, but to launch a system that continues to grow, serve users, and raise awareness of the role data can play in navigating a changing climate.

10. Future Work

While this project currently focuses on real-time data visualization and conversational interpretation, several avenues exist for future development. One key area is the incorporation of predictive analytics using historical RTMA data. With over a decade of hourly observations available, machine learning models could be trained to forecast short-term phenomena such as temperature spikes, precipitation extremes, or humidity-driven health risks like heat stress. Another valuable enhancement would be the integration of a user alert system—allowing individuals or organizations to receive real-time notifications based on thresholds relevant to their context (e.g., WBGT levels for outdoor workers or high dew point alerts for wildfire-prone

areas). In addition, expanding the system's capabilities to include user accounts would enable personalization features, such as saving preferred regions or accessing tailored summaries. More broadly, incorporating additional environmental datasets—such as air quality indices, UV radiation, or wind patterns—could further increase the dashboard's utility and public value. These enhancements would continue building toward the original goal of this project: to transform complex weather data into an intelligent, accessible, and actionable platform for everyday users and decision-makers.

11. References

Bhardwaj, E., & Khaiter, P. A. (2023). What data analytics can or cannot do for climate change studies: An inventory of interactive visual tools. *Ecological Informatics, 73*, 101918. https://doi.org/10.1016/j.ecoinf.2022.101918

Blaylock, B. K. (2024). Herbie: Retrieve Numerical Weather Prediction Model Data (Version 2025.3.1) [Computer software]. https://doi.org/10.5281/zenodo.4567540

De Pondeca, M. S. F. V., Manikin, G. S., DiMego, G., Benjamin, S. G., Parrish, D. F., Purser, R. J., Wu, W., Horel, J. D., Myrick, D. T., Lin, Y., Aune, R. M., Keyser, D., Colman, B., Mann, G., & Vavra, J. (2011). The Real-Time Mesoscale Analysis at NOAA's National Centers for Environmental Prediction: Current Status and Development. Weather and Forecasting, 26(5), 593-612. https://doi.org/10.1175/WAF-D-10-05037.1

Eggleton, F., & Winfield, K. (2020). Open data challenges in climate science. *Data Science Journal, 19*(52), 1–6. https://doi.org/10.5334/dsj-2020-052

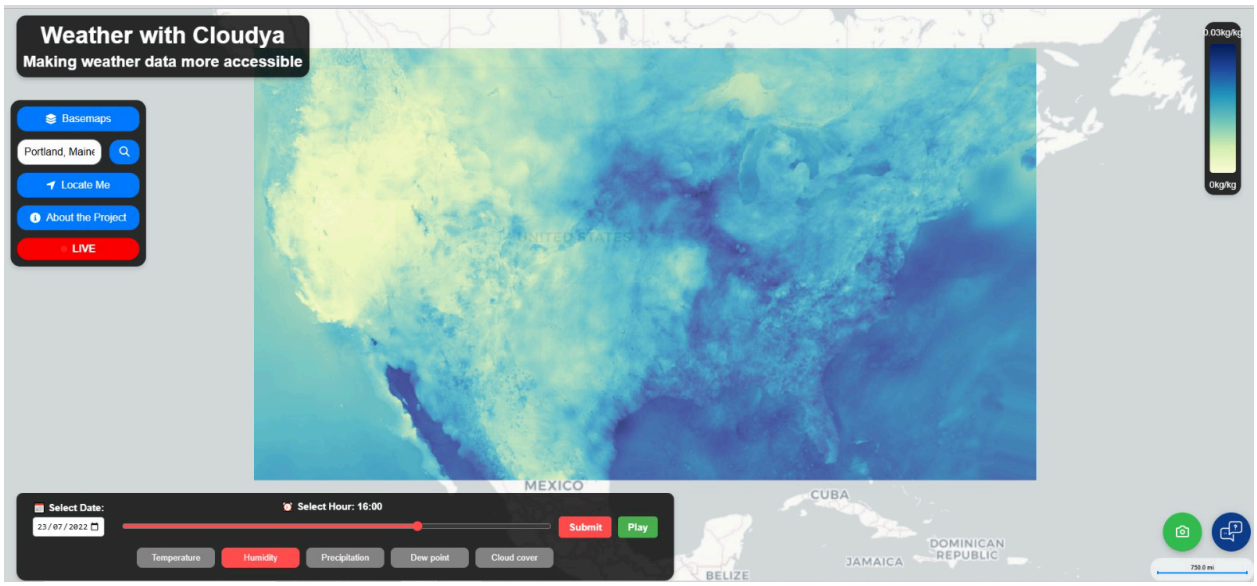
"llama3.2-vision." *Ollama*, ollama.com/library/llama3.2-vision.

Lumley, S., Sieber, R., & Roth, R. (2022). A framework and comparative analysis of web-based climate change visualization tools. *Computers & Graphics, 103*, 19–30. https://doi.org/10.1016/j.cag.2021.12.007

12. Appendix

Screenshots of Other Variables

Humidity



Cloud Cover