Cloud Computing
Project Document
Respect Weather - weather app

Jakub Piwko, 313451
Kacper Skonieczka, 313505
Grzegorz Zakrzewski, 313555

July 14, 2024

1 Introduction

The document is a specification of a weather app implemented using Google Cloud Platform. It covers the description of the project and details all requirements: user interface, storage, and large-scale assumptions. We also present a diagram of micro-services and API that provides communication between crucial endpoints of our system. Lastly, we describe the promised level of system quality in accordance with SLA, SLO, and SLI standards.

2 Project description

Respect Weather is a web weather application designed to provide access to short-term weather forecasts and historical data about atmospheric conditions. Although Respect Weather currently serves data for approximately one thousand predefined locations in Europe and North Africa, it is designed to easily expand its capabilities. Additionally, basic authentication is implemented, and users have the ability to add locations to their *favourites* list.

If we didn't run out of credits, our app is available *here*.

The entire application source code can be found *here*, with the Terraform file named *main.tf*.

2.1 Services description and diagram

Respect Weather is built on top of several GCP services. A diagram of the services layout and connections is presented in Figure 1. One can distinguish four main parts of our project design: ETL, storage, API, and UI. Let's take a closer look at services contained in each part.

ETL The ETL process supplies our weather application with meteorological data. The key component here is a PySpark Job, managed by Dataproc Workflow. Dataproc is a service for running Apache Hadoop, Apache Spark, and other open-source frameworks. A Dataproc Workflow is an operation that runs previously configured jobs on a cluster, fully managed by GCP. We have configured our ETL PySpark Job as the only job of a Workflow Template. This template is triggered twice a day at 08:00 and 20:00 by a Cloud Scheduler job. The downloaded and prepared data is loaded into a BigQuery table. Scripts

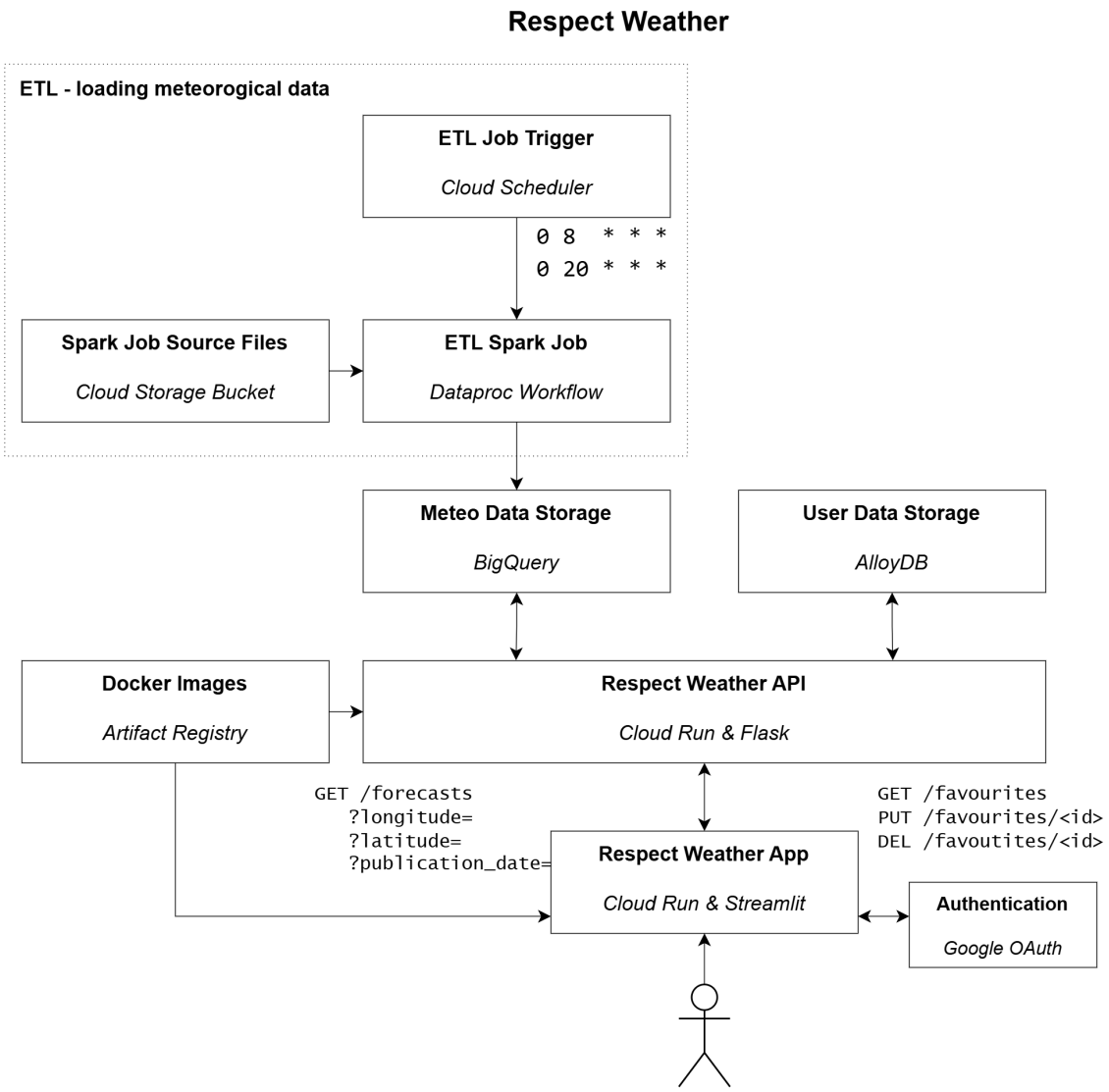

Figure 1: Diagram of microservices and connections.

required for running the Spark job are stored in a Cloud Storage bucket. More information about storage and the data is provided in Section 4 and Appendix A.

Storage Two GCP services are used to store data in our application: BigQuery and AlloyDB. BigQuery is a data warehouse used in our project to store potentially large volumes of meteorological data. AlloyDB is a PostgreSQL database managed by GCP and stores data about the users of our application. Storage characteristics are provided in Section 4.

API The application exchanges data with storage through an API. It was implemented with Python and Flask, built as a Docker image, and stored in Artifact Registry. The latest image is downloaded and served with Cloud Run. API details are described in Section 3.

UI The user interface in our application is implemented with Streamlit, an open-source Python framework for building dashboards and data apps. Although Streamlit is not a tailored solution for every

possible problem, it served us well and allowed us to quickly build a good-looking front-end for our application. As with the API, the Streamlit app was prepared as a Docker container and is served via Cloud Run. A description and some screenshots are provided in Section 5.

3 API

Respect Weather API provides several endpoints for user interaction and data retrieval:

- `GET /forecasts?longitude=&latitude=&publication_date=` - allows users to download weather forecasts for a specific geographical location and forecast publication date. The longitude and latitude arguments are floats, and the publication date should be in the format: `YYYY-MM-DD`. In response, the dataframe is returned with MIME type `text/csv`. The dataframe always includes eleven columns: `time`, `valid_time`, `latitude`, `longitude`, `number`, `u10`, `v10`, `tp`, `tcc`, `t2m`, and `prmsl`. Variable names are described in Appendix A. If no data is found in the database, an empty dataframe (only the header row) is returned.
- `GET /favourites` - returns a list of the user's favorite location IDs as JSON.
- `PUT /favourites/<location_id>` - saves the location ID to the user's set of favorite locations.
- `DELETE /favourites/<location_id>` - drops the location ID from the user's set of favorite locations.

Originally, our API was also intended to handle user authentication. However, during the project development, we decided to switch to Google OAuth 2.0 and allow users to log in using their Gmail accounts. Authentication is required for the `/favourites` endpoints. Requests to this endpoints must include the `Authorization: Bearer <id_token>` header field with the token obtained from the OAuth 2.0 process.

The connection between the Respect Weather API and databases is established using Python libraries provided by Google. For BigQuery, it is `pandas-gbq`, and for AlloyDB, it is `google-cloud-alloydb-connector`. Although it is not mentioned in diagram 1, a special Virtual Private Cloud network with Private Service Access connection had to be prepared to allow the connection between the Respect Weather API and AlloyDB.

4 Storage

The application's data management strategy utilizes a two-pronged approach to handle different types of data efficiently:

- **BigQuery** is chosen for storing weather forecasts, which consist predominantly of structured and tabular data. Currently, we are using a subset of weather forecast data published by GEFS, but if we were to decide to utilize all possible data, the volume would increase up to 100GB per day. BigQuery is adept at managing this volume of data, even more so with table clustering over the publication date. BigQuery's design caters to the application's need for eventual consistency, aligning with the scheduled ETL processes that update the weather data twice daily. Weather forecast data will be stored in a single table in BigQuery (partitioned by publication date). Write operations will be very infrequent, so we may see it as a read-only storage.
- **AlloyDB**, on the other hand, is deployed for *user-related data* management, encompassing for now user-defined favorites. This database guarantees strong consistency, ensuring that user data

is immediately and accurately available, which is critical for user interaction. The scalability of AlloyDB, as a fully managed PostgreSQL database for the most demanding enterprise workloads, is tailored to accommodate the growing number of app users, supporting both read and write operations efficiently.

This delineation in storage solutions allows for the optimised handling of both the large-scale weather data analytics and the dynamic user data management, ensuring a robust and seamless application experience.

5 UI

In this section, the user interface is described, including implemented features and appropriate screenshots.

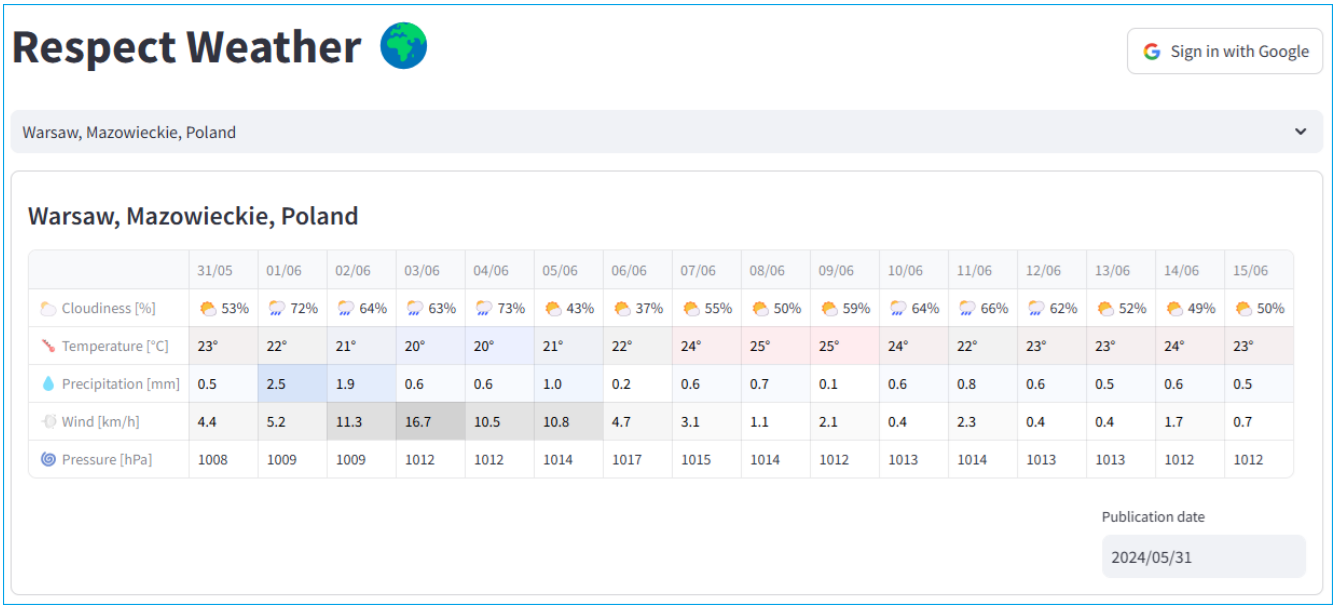

Figure 2: Basic appearance of the application.

The UI in the Respect Weather app is implemented with Python and Streamlit. It was an easy and fast way to arrange and connect the necessary widgets. After entering the website, the user sees basic appearance of the application - Figure 2. On the right side is a sign-in button, which allows users to authenticate using their Gmail account. Below, there is a dropdown list that allows users to select one of the predefined locations (which were obtained from *here*). The default location is Warsaw, Poland. After selecting a location, the weather is displayed in the form of a data frame. This data frame has five rows, corresponding to weather forecast features: cloudiness, temperature, precipitation, wind, and pressure. Each column in the data frame matches dates for which forecasts are provided—up to 16 days ahead. Emojis in the "cloudiness" row and colors in the other rows make the interface more user-friendly and help recognize outlying weather conditions. On the bottom left side of our app, there is a date picker, which allows users to check historical weather forecasts published on a specific day.

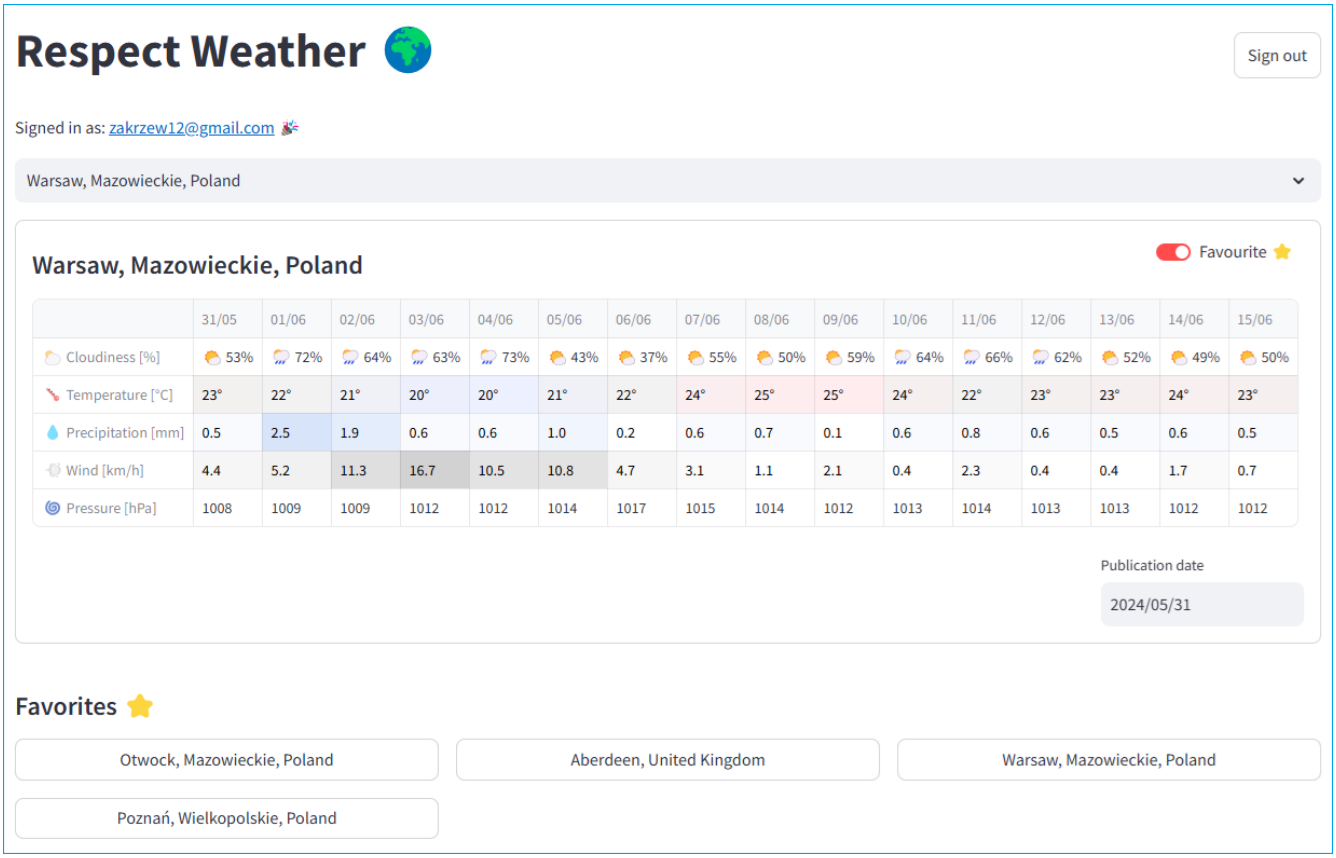

Figure 3: Appearance of the application after logging in.

After proper authentication, the layout of the application changes slightly (Figure 3). Firstly, in the upper left corner of the central box of the application, a toggle button appears, allowing users to add or remove the currently displayed location from their favorites list. At the bottom of the app, the favorites list appears in the form of buttons displayed in three columns. These buttons serve as convenient shortcuts to favorite locations.

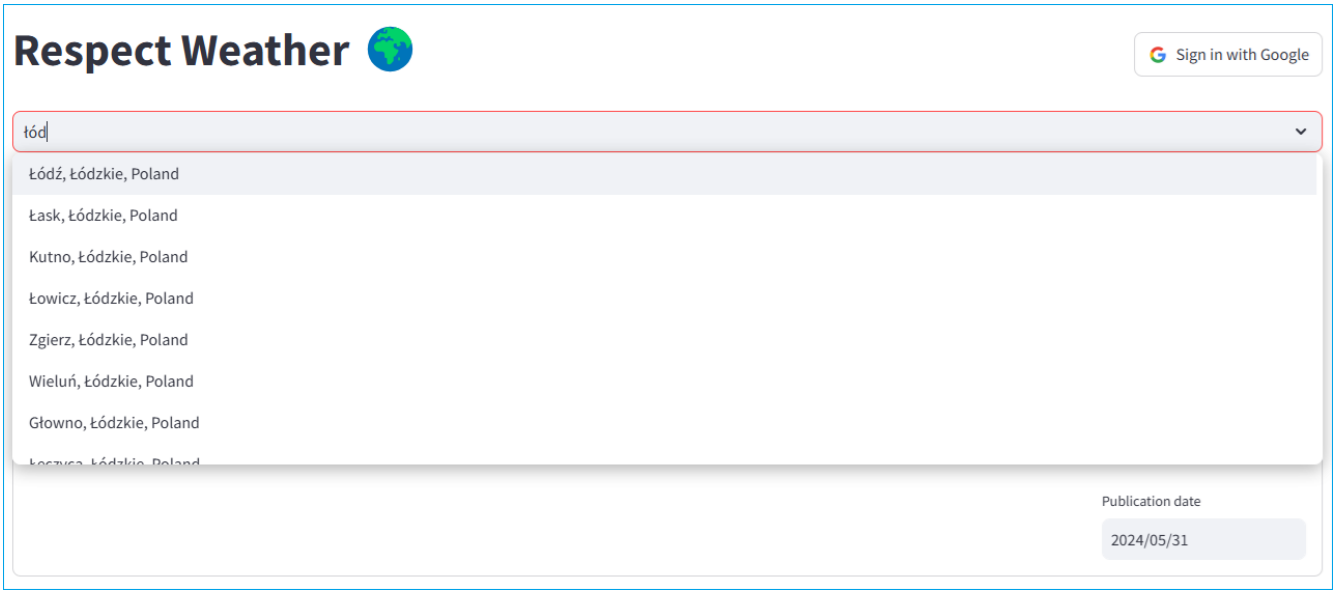

Figure 4: View of an expanded list for selecting a location.

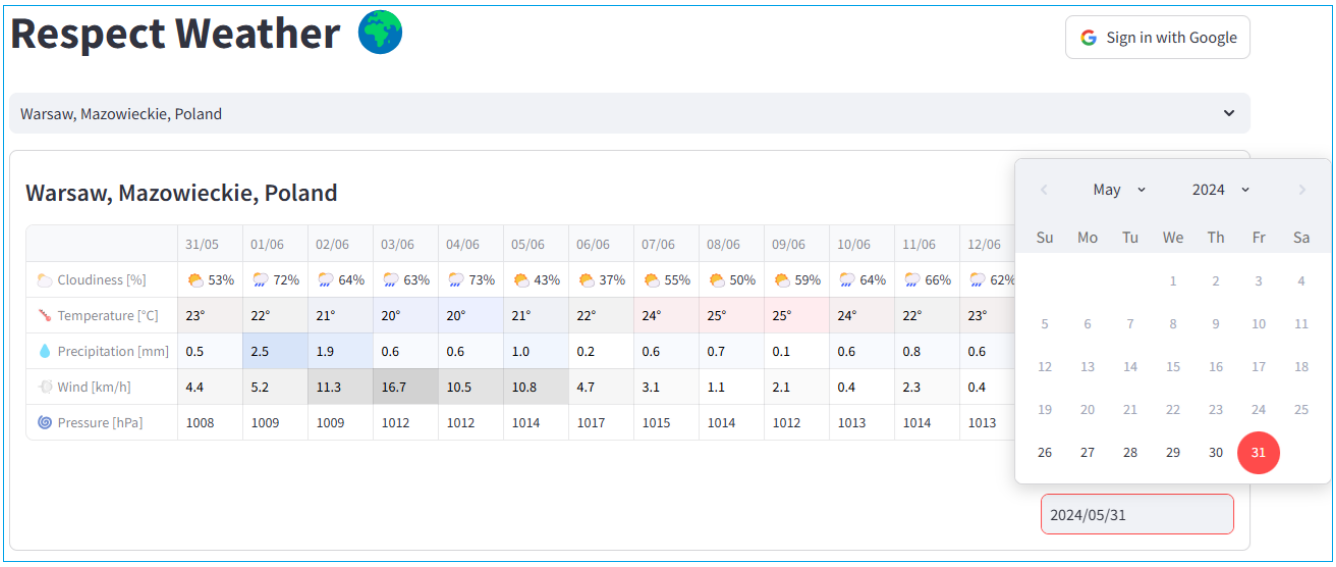

Figure 5: View of an expanded date picker for selecting a publication date.

Additional screenshots presenting expanded list for selecting a location or expanded date picker for selecting a publication date are visible in Figure 4 and Figure 5.

6 Large scale assumptions

Although for now our weather app is closer to a proof-of-concept rather than a full-enterprise solution, we took care to maintain large-scale assumptions. We can summarize that in the following key points:

- The cluster on which Dataproc Workflows and Jobs are launched could be easily scaled up. Additionally, our ETL process is designed as a PySpark Job and with some modifications could be parallelized to achieve faster data processing and transfer.

- BigQuery with a partitioned table is a perfect solution for handling extremely large volumes of meteorological data.
- The AlloyDB cluster can also be scaled up to meet necessary requirements. Google claims that it is a *compatible database service for your most demanding enterprise workloads*, so it should handle an increasing number of users.
- Cloud Run services, on which the API and Streamlit App are based, could also be scaled up. The minimum number of serving instances that these resources should have could be increased, and there should be no problems with high request demand.

7 SLA / SLO / SLI

7.1 SLA: Service Level Agreement

The SLA establishes the foundation of trust and reliability between the weather app service and its users, detailing expected service standards, including system uptime, performance metrics, and overall service quality.

7.2 SLO: Service Level Objectives

Specific targets set within the SLA to ensure high service quality:

- **Availability:** Aim for 99.9% system uptime monthly.
- **Response Rate:** Achieve a 99.5% success rate in responding to user requests.
- **Processing Time:** Ensure 95% of requests are processed within 1 second.
- **Data Freshness:** Update weather forecasts in the app within 25 minutes of their release.

7.3 SLI: Service Level Indicators

Metrics to assess compliance with the SLOs:

- Monitoring system uptime.
- Tracking the percentage of successfully handled requests.
- Measuring the average time taken to process requests.
- Observing the timeliness of weather data updates in the app.

A Meteorological data

The Global Ensemble Forecast System (GEFS) is a weather forecasting model composed of 21 separate forecasts (*ensemble members*). The National Centers for Environmental Prediction (NCEP) launched the GEFS model to address the inherent "uncertainty" in weather observations. The GEFS model attempts to quantify this forecast uncertainty by generating a set of multiple forecasts, each slightly different or perturbed from the original observations.

The GEFS has a global reach, with new forecasts published four times daily (every six hours starting at midnight). The forecast itself covers a 16-day period from the time of publication, with forecasts made every three hours for the first 10 days and every six hours for days 10 to 16. This means that one publication contains predictions for dozens of weather variables (meteorological parameters) separately for each of the 21 bundles, for each latitude and longitude with an accuracy of half a degree, for every future point distanced from the date of publication by 0, 3, 6, ..., 237, 240, 246, ..., 378, 384 hours.

Forecasts from the GEFS model are published as a set of files in the GRIB2 format. The GRIB2 file format is a popular format for storing meteorological data. Opening such a file requires the use of dedicated software.

GEFS model forecasts are stored and publicly available through an Amazon S3 bucket. Using a web-based explorer facilitates familiarisation with the file naming and hierarchy. The address of a sample GEFS forecast file is presented as follows:

`https://noaa-gefs-pds.s3.amazonaws.com/gefs.20231201/00/atmos/pgrb2ap5/geavg.t00z.pgrb2a.0p50.f00`

With some patience, this address can be deciphered:

- `https://noaa-gefs-pds.s3.amazonaws.com/gefs`
  the base address where the Amazon S3 bucket is located;
- `.20231201/`
  the date in the format `YYYYMMDD`;
- `/00/`
  indicates the time of publication; there are four publications per day: `00`, `06`, `12`, and `18`;
- `/atmos/`
  indicates atmospheric readings;
- `/pgrb2ap5/`
  indicates group `a`; group `a` contains several dozen of the most commonly used variables, including variables defining the wind speed of interest; a full list of variables can be found here; there is also group `b` of several hundred "less popular" variables;
- `/geavg.`
  is the ensemble number; there are 21 regular bundles from `gep01` to `gep21`, the control bundle `gec00` (model forecast without disturbances) and the bundle `geavg` being the average of all bundles;
- `.t00z.`
  again indicates the time of publication;
- `.pgrb2ap5.`
  again indicates the variable group;

- `.0p50.`
  an irrelevant part of the address;
- `.f000`
  the future point to which the predictions refer, from `f000` to `f384` (16 days) every 3 hours or every 6 hours.

Each bundle results in 105 GRIB2 forecast files. The sum of 23 bundles gives 2415 files per publication, or 9660 files per day. Each file is about 10 MB, resulting in a considerable volume of 100 GB of raw data per day. Given the project's focus, we're directing our attention specifically towards data for Europe and selected cities, aligning with where our initial customer base is located. This tailored approach allows us to efficiently utilise our resources while keeping the system adaptable for future expansions.

Table 1 contains the first few rows from a table created based on the same file. This example only illustrates two variables of interest, namely the u and v components of wind speed measured 10 meters above the ground.

latitude	longitude	time	step	valid_time	u10	v10
45.0	5.0	2023-12-01	7 days	2023-12-08	2.43	0.66
45.0	5.5	2023-12-01	7 days	2023-12-08	2.44	0.64
45.0	6.0	2023-12-01	7 days	2023-12-08	2.44	0.62
45.0	6.5	2023-12-01	7 days	2023-12-08	2.45	0.59
45.0	7.0	2023-12-01	7 days	2023-12-08	2.45	0.57

Table 1: Example file with GEFS model forecast transformed into tabular format

Dimensions and variables that we will extract from raw data to use in our weather panel are presented in list below:

- `time` - time of forecast publication (two times a day 00z, 12z)
- `valid_time` - time of forecast (every 6 hours up to 16 days)
- grid 1.0 degree (latitude / longitude)
- variables:
  - u10 (wind u-component 10 meters above surface)
  - v10 (wind v-component 10 meters above surface)
  - tp (total precipitation)
  - tcc (total cloud cover)
  - t2m (temperature 2 meters above surface)
  - prmls (pressure)