Weather Data Pipeline Implementation

Carlos Peralta

Introduction

This document details the implementation of a weather data processing pipeline with focus on GNSS meteorology and operational forecasting. The primary goal of this pipeline is to ingest raw GRIB2 forecast data from multiple sources, process it into a structured and efficient format, and produce actionable visualizations.

A key feature of the current implementation is its use of **Zarr** for cloud-native, chunked data storage, which is highly efficient for the time-series and multi-dimensional data typical of weather models.

Parameter Selection for GNSS Meteorology

The pipeline is configured to process specific atmospheric variables that are crucial for **Global Navigation Satellite System (GNSS) meteorology** and general operational awareness.

What is GNSS Meteorology?

GNSS meteorology is a technique that uses the signals from satellites (like GPS, Galileo, etc.) to measure the amount of water vapor in the atmosphere. When GNSS signals travel from a satellite to a ground-based receiver, they are delayed by the atmosphere. This delay, known as the Zenith Tropospheric Delay (ZTD), has two main components: a "dry" part caused by all atmospheric gases and a "wet" part caused almost entirely by water vapor.

By precisely measuring the total delay and subtracting the well-understood dry component (which can be calculated from surface pressure), scientists can isolate the wet delay. This wet delay is directly proportional to the total amount of water vapor in the column of atmosphere above the receiver. This quantity is called **Precipitable Water Vapor (PWV)**.

Choice of Variables

The variables selected for this pipeline are essential for both direct use in GNSS meteorology and for providing operational context:

- **`precipitable_water` (pwat)**: This is the primary variable of interest. Weather models like GFS provide a forecast of this value, which can be compared against the real-time measurements from a GNSS receiver for model validation or data assimilation.
- **`surface_pressure` (sp)**: Surface pressure is required to calculate the "dry" part of the signal delay. Accurate pressure forecasts are vital for deriving accurate PWV from GNSS measurements.
- **`temperature` (t2m)**: Temperature influences the atmospheric density and the constants used in the PWV calculation.
- **`wind` (u10, v10, u100, v100)**: Wind fields at 10m and 100m are crucial for understanding the transport of water vapor, for renewable energy applications (wind turbines), and for aviation safety.
- **`wind_gust` (gust)**: This is a direct model output critical for safety-related decisions in aviation, event management, and energy infrastructure.
- **`precipitation` (tp)** and **`cloud_cover` (tcc)**: These variables provide operational context. They help meteorologists understand whether the atmospheric water vapor is likely to result in rain or cloud formation, which is essential for decision-making.

System Architecture

The pipeline is designed as a modular, orchestrated system that handles data from ingestion to visualization.

```
graph TD
    A[run_pipeline.sh] --> B{Orchestration};
    B --> C[GFS Downloader];
    B --> D[MET Nordic Downloader];
    C --> E{Data Processing};
    D --> E;
    E --> F[Zarr Storage];
    F --> G{Visualization};
    G --> H[Static Maps];
    G --> I[Interactive Dashboard];
```

Orchestration

The entire pipeline is managed by the `scr/run_pipeline.sh` script, which calls `orchestration/pipeline_scheduler.py`. This scheduler executes the pipeline steps in

a predefined order for a given forecast cycle (`--date` and `--cycle`).

The sequence of operations is: 1. Download GFS Data (`data_ingestion/gfs_downloader.py`) 2. Download MET Nordic Data (`data_ingestion/met_downloader.py`) 3. Process GFS Data (`data_processing/process_data.py`) and save to Zarr. 4. Create GFS Visualizations (`visualization/create_visualizations.py`). 5. Process MET Nordic Data (`data_processing/process_met_data.py`). 6. Create MET Visualizations (`visualization/create_met_visualizations.py`).

Data Ingestion

The pipeline ingests data from two sources as required: - **Global Model**: The **Global Forecast System (GFS)** is downloaded via `gfs_downloader.py`. - **Regional Model**: The **MET Nordic** forecast covering Northern Europe is downloaded via `met_downloader.py`.

Data Processing and Storage

The core processing logic resides in `data_processing/process_data.py` (for GFS) and `process_met_data.py` (for MET).

1. **File Iteration**: The functions iterate through the raw GRIB2 files for a given date and cycle.
2. **Data Loading**: For each file, `xarray` and the `cfgrib` engine are used to open datasets for the relevant atmospheric variables.
3. **Data Merging & Calculation**: The individual datasets are merged into a single `xarray.Dataset`. New variables, such as `wind_speed_10m`, are calculated from the `u` and `v` wind components.
4. **Storage**: The final, processed `xarray.Dataset` for the entire forecast cycle is written to a cycle-specific Zarr store (e.g., `data/processed/gfs_{date}_{cycle}.zarr`). Zarr was chosen for its efficient handling of chunked, multi-dimensional data, which allows for fast access to data subsets (e.g., a specific variable over a specific time range) without loading the entire file.

Wind Gust Calculation

The project includes a dedicated script, `data_processing/calculate_wind_gust.py`, to explore different methods of deriving wind gusts from model outputs. This script is not part of the main pipeline but serves to answer the technical question posed in the project requirements. It implements three common methods:

1. **Multiplicative Factor**: A simple approach where the sustained 10m wind speed is multiplied by a constant factor (e.g., 1.5).

2. **Friction Velocity**: A more physical approach using momentum fluxes (`u_flux`, `v_flux`) to calculate the friction velocity (`u*`), which is then added to the sustained wind speed. `Gust = U10m + alpha * u*`.
3. **Turbulent Kinetic Energy (TKE)**: Uses the TKE field from the model, which represents the intensity of turbulence. `Gust = U10m + beta * sqrt(TKE)`.

While the main pipeline uses the direct `gust` variable from the model output, this script demonstrates the ability to derive this critical parameter if it were not available.

Visualization

The pipeline produces two types of visualizations from the processed Zarr data:

1. **Static Maps**: The `visualization/create_visualizations.py` script generates a series of `.png` map images for each variable and for each time step in the forecast. These maps are saved in the `visualization/plots` directory. They are useful for quick, static views of the forecast evolution.
2. **Interactive Dashboard**: The `visualization/run_dashboard.py` script launches a web-based interactive dashboard built with Plotly and Dash. This dashboard allows a user to:
    - Select the forecast model (GFS or MET).
    - Choose a specific date and cycle.
    - Select the atmospheric variable to display.
    - Use a time-slider to smoothly animate the forecast over time.
    - Pan and zoom the map for detailed regional views.

This interactive tool is the primary interface for decision-support, as it provides a dynamic and user-friendly way to explore the forecast data.

Usage

Configuration

The main configuration, such as the map boundaries (`EUROPE_BOUNDS`), is located in `config.py`.

Running the Pipeline

The primary way to run the pipeline is using the `scr/run_pipeline.sh` script.

The script has four modes:

1. **`default`**: Automatically determines the latest available GFS cycle and runs the pipeline for it. This is the most common use case for automated runs. `bash ./scr/run_pipeline.sh default`
2. **`manual`**: Allows you to run the pipeline for a specific date and cycle. This is useful for reprocessing historical data or debugging. `bash ./scr/run_pipeline.sh manual --date 20231027 --cycle 12`
3. **`scheduler`**: Runs the pipeline in a continuous loop, waiting 6 hours between each run. This is intended for a 24/7 operational deployment. `bash ./scr/run_pipeline.sh scheduler`
4. **`dashboard`**: Starts the interactive web dashboard. `bash ./scr/run_pipeline.sh dashboard`

Dependencies

The key Python libraries used in this pipeline are:

- **`xarray`**: For handling multi-dimensional labeled data.
- **`cfgrib`**: The engine used by xarray to read GRIB files.
- **`zarr`**: For chunked, compressed, N-dimensional array storage.
- **`pandas`**: Used for time-series manipulation.
- **`cartopy`**: For creating the static map projections.
- **`plotly` & `dash`**: For the interactive dashboard.

All required dependencies are listed in the `requirements.txt` file.