- Gaurav Thorat
- July 2025

Global Partners – Weather Data Microservice

Executive Summary

Business Problem

- ❖ Need for real-time weather data processing
- ❖ Geographic-specific weather insights
- ❖ Scalable weather analytics platform

Solution Overview

- ❖ **FastAPI-based microservice** for weather data processing
- ❖ **NWS API** integration for reliable data
- ❖ **Advanced data enrichment** with geospatial analysis
- ❖ **PostgreSQL persistence** with relationship modelling`

Key Benefits

- ❖ **Real-time** processing of weather data
- ❖ **Enhanced analytics** with temperature ratios and wind patterns
- ❖ **Geographic precision** with distance calculations
- ❖ **Production-ready** containerized deployment

Technical Stack

Core Technologies

Component	Technology	Purpose
API Framework	**FastAPI + Uvicorn**	High-performance REST API
Database	**PostgreSQL 15**	Relational data storage
Caching	**-**	Performance optimization
ORM	SQLAlchemy	Database abstraction
Containerization	**Docker + Docker Compose**	Deployment & scaling
Testing	**PyTest**	Quality assurance

External Integrations

- **NWS API**
- **Geopy library for geospatial calculations**
- **HTTpie for API testing**

System Architecture

- ***Layered Architecture Design***
  - Presentation Layer
    - REST API endpoints(/weather, /healthz)
    - Input validation and error handling
- 2. Business Logic Layer
    - Weather data processing engine
    - Data enrichment algorithms
    - External API integration
- 3. Data Access Layer
    - SQLAlchemy ORM models
    - Connection pooling & caching
    - Transaction management
- 4. Data Storage Layer
    - Normalized PostgreSQL schema
    - JSON raw data preservation
    - Relationship modeling

Data Flow Architecture

- End-to-End Processing Pipeline

Client Request → FastAPI → NWS API → Data Enrichment → PostgreSQL → Response

- Step-by-Step Process:
  - **Input Validation** - Validate coordinates and user data
  - **Gridpoint Lookup** - Get NWS forecast URLs from coordinates
  - **Forecast Retrieval** - Fetch current and hourly weather data
  - **Grid Center Calculation** - Parse polygon geometry for precise location
  - **Data Enrichment** - Calculate temperature ratios, wind analysis, distances
  - **Database Persistence** - Store normalized data with relationships
  - **Response Generation** - Return processed results to client

Data Flow Architecture

- End-to-End Processing Pipeline

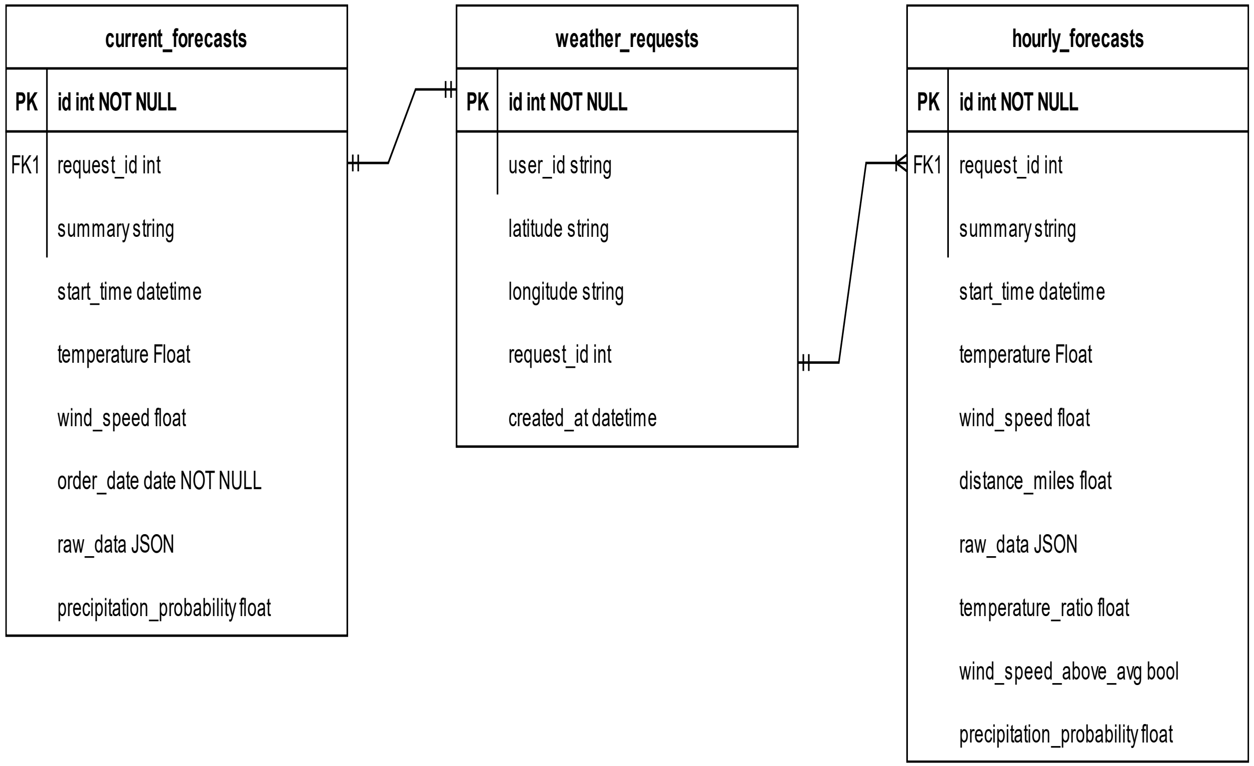
Client Request → FastAPI → NWS API → Data Enrichment → PostgreSQL → Response

- Step-by-Step Process:
  - **Input Validation** - Validate coordinates and user data
  - **Gridpoint Lookup** - Get NWS forecast URLs from coordinates
  - **Forecast Retrieval** - Fetch current and hourly weather data
  - **Grid Center Calculation** - Parse polygon geometry for precise location
  - **Data Enrichment** - Calculate temperature ratios, wind analysis, distances
  - **Database Persistence** - Store normalized data with relationships
  - **Response Generation** - Return processed results to client

Database Schema Design

- **Key Principles:**
  - **3NF Normalization for data integrity**
  - **Foreign key constraints** for referential integrity
  - **JSON storage** for raw data preservation
  - **Strategic indexing** for query performance

- **Normalized Relational Model**



Data Enrichment

Advanced Analytics Feature Calculations

1. Temperature Analysis

temperature_ratio = hourly_temp / daily_average_temp

Values above 1.0 would indicate above average temperature

2. Wind Pattern Detection

wind_above_avg = current_wind > daily_average_wind

Boolean indicator for wind conditions

3. Geospatial Calculation

wind_above_avg = current_wind > daily_average_wind

Boolean indicator for wind conditions

4. Precipitation Trends

Raw precipitation probabilities are preserved in JSON

Infrastructure & Deployment

Container-First Architecture

Docker Containeriztion

- Application container with FastAPI
- PostgreSQL container with persistence
- Docker Compose orchestration

Configuration Management

- Environment-based configuration (.env)
- Multi-environment support (dev/staging/prod)

Monitoring & Observability

- Structured JSON logging
- Health check endpoints

Quality Assurance

Comprehensive Testing Strategy

Test Coverage

- **Unit Tests** - Data enrichment algorithms
- **Integration Tests** - Full API workflow
- **Service Tests** - External API mocking

Development Workflow

docker-compose up –build

docker-compose run --rm app pytest

Design Choices & Justification - Architecture

Microservice Architecture

- Choice: Single-responsibility weather service
- Rationale: Easier to scale, deploy, and maintain independently
- Alternative: Monolithic application
- Trade-off: Slightly more complex deployment vs. better scalability

2. FastAPI Framework

- Choice: FastAPI over Flask/Django
- Rationale:
  - Built-in async support for better performance
  - Type hints and Pydantic validation
  - Modern Python 3.6+ features
- Trade-off: Newer ecosystem vs. proven performance

3. PostgreSQL Database

- Choice: PostgreSQL over NoSQL
- Rationale:
  - ACID compliant
  - Stong JSON support for raw data storage
  - Excellent geospatial capabilities (PostGIS ready)
- Trade-off: Structure schema vs document flexibilities

Design Choices & Justification – Data Design

1. Normalized Schema Design
    - Choice: 3 NF with separate tables
    - Rationale:
        - Eliminates data redundancy and ensures referential integrity
        - Enables complex queries and analytics
    - Alternative: Denormalized/flat structure
    - Trade-off: Query complexity vs. data consistency

2. Hybrid Storage Approach
    - Choice: Structured fields + JSON raw data storage
    - Rationale:
        - Fast queries on structured data
        - Preserves complete original data for future analysis
        - Flexibility for schema evolution
    - Trade-off: Storage overhead vs. data preservation

Optimization Strategies – Future Improvements

1. Performance Features
   - Connection pooling for db efficiency
   - Redis caching for frequently accessed data
   - Async processing with FastAPIOptimized SQL queries with proper indexing

2. Scalability Approach
   - Horizontal scaling with load balancers
   - Database read replicas for query distribution
   - Microservice architecture for independent scaling
   - Container orchestration for auto-scaling

Future Improvements - General

- OAuth2 / API key-based authentication
- CI/CD setup for automated testing & deploy
- Metrics via Prometheus + Grafana
- Scheduled batch ingestion from NWS