Real-Time Weather Data Streaming Pipeline Using Kafka, Flink, and PostgreSQL

Shanmukha Manoj Kakani
Department of Software Engineering
San Jose State University
San Jose, USA
shanmukhamanoj.kakani@sjsu.edu

***Abstract*—This paper presents a real-time data streaming pipeline designed for weather data aggregation. The pipeline integrates Apache Kafka, Apache Flink, and PostgreSQL to enable seamless data flow, processing, and storage. A Python-based producer simulates weather data, which is streamed to Kafka for ingestion. Apache Flink processes the data using a tumbling window of one minute to calculate city-level average temperatures, and the results are stored in PostgreSQL for analysis. By leveraging Docker for containerization, the pipeline achieves modularity, scalability, and ease of deployment. This project highlights the application of modern streaming technologies for low-latency, high-throughput data processing.**

***Index Terms*—Real-Time Processing, Apache Kafka, Apache Flink, PostgreSQL, Docker, Weather Data, Stream Processing.**

I. Introduction

With the proliferation of IoT devices, social media platforms, and real-time monitoring systems, the demand for low-latency data processing has surged. Traditional batch processing systems fall short in meeting the immediate insights required by modern applications. To address these challenges, this project implements a real-time weather data streaming pipeline that processes data continuously and efficiently.

This pipeline simulates weather data generation, processes it in real time using Apache Flink, and stores the results in PostgreSQL for further analysis. The integration of Apache Kafka ensures fault-tolerant data streaming, while Docker facilitates consistent deployment across environments. The system exemplifies the potential of streaming technologies to meet the demands of modern data-driven applications.

II. Pipeline Architecture

The pipeline consists of three key layers: data generation, stream processing, and data storage. Figure 1 illustrates the architecture and interactions between these components.

A. Data Generation

The Python-based producer generates synthetic weather data for multiple cities using the Faker library. Each data point includes the city name, temperature, and a timestamp, formatted as a JSON record. These records are published to a Kafka topic at a rate of one record per second, mimicking real-world data streams.

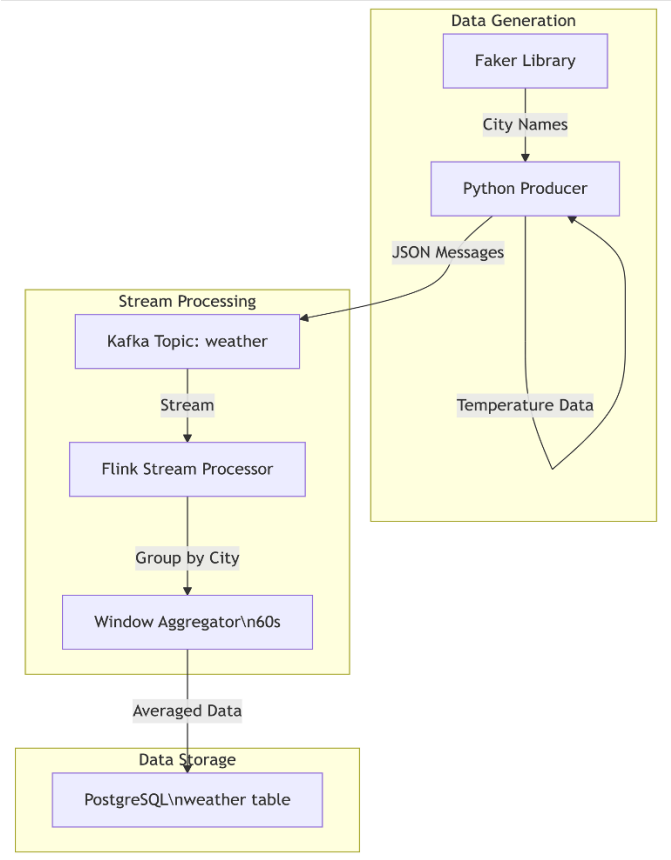

Fig. 1. Architecture of the real-time weather data streaming pipeline.

B. Stream Processing

Apache Flink processes the incoming data in real-time. By employing a tumbling window of one minute, Flink aggregates temperature readings for each city. The computed metrics, such as average temperatures, are then forwarded to the storage layer for further analysis.

C. Data Storage

The processed data is stored in a PostgreSQL database. Each record in the database contains the city name, average temperature, and timestamp corresponding to the aggregation

window. PostgreSQL ensures efficient querying and supports long-term data analysis.

III. Implementation Details

The system integrates multiple technologies to create a seamless and scalable pipeline. Key implementation steps are detailed below:

A. Data Flow

The data flow follows these steps:

1) The Python producer generates JSON records containing city names, temperatures, and timestamps.
2) Kafka brokers receive the data and make it available for Apache Flink.
3) Flink processes the data in real time, applying a one-minute tumbling window to compute city-level aggregates.
4) The processed data is written to PostgreSQL for storage and analysis.

B. System Components

- **Producer:** A Python script simulates weather data generation and publishes it to Kafka topics.
- **Processing Layer:** Apache Flink ingests and processes data from Kafka, applying transformations and aggregations.
- **Storage Layer:** PostgreSQL stores the aggregated results for further analysis.

IV. Dockerized Deployment

To ensure consistent environments and ease of deployment, the entire system is containerized using Docker. The components include:

- **Kafka and Zookeeper:** These containers handle the messaging system, enabling reliable data streaming.
- **Flink Processor:** This container executes the Flink job, consuming and processing data in real time.
- **PostgreSQL:** A containerized PostgreSQL instance stores the aggregated data.
- **Python Producer:** This container generates weather data and streams it to Kafka.

Docker Compose is used to orchestrate the containers, enabling seamless setup and teardown of the pipeline.

V. Key Features

- **Real-Time Processing:** Processes continuous data streams with minimal latency.
- **Scalability:** Modular architecture supports independent scaling of components.
- **Fault Tolerance:** Kafka ensures reliable messaging, while Flink provides stateful processing with recovery capabilities.
- **Persistence:** PostgreSQL stores aggregated results, enabling efficient querying and analysis.
- **Ease of Deployment:** Docker simplifies deployment and ensures consistent environments.

VI. Conclusion

This project demonstrates the implementation of a real-time weather data streaming pipeline that leverages Apache Kafka, Flink, and PostgreSQL. The system's modular architecture, combined with Docker-based deployment, ensures scalability and reliability. Future enhancements could include integrating real-time alerts for temperature anomalies and developing visualization dashboards for data exploration.

Acknowledgments

The author acknowledges the seamless integration of modern streaming technologies that enable efficient real-time data processing solutions.

References

[1] N. Narkhede, G. Shapira, and T. Palino, "Kafka: The Definitive Guide," O'Reilly Media, 2017. Available: https://kafka.apache.org/documentation/
[2] S. Ewen, F. Hueske, and V. Kalavri, "Stream Processing with Apache Flink," O'Reilly Media, 2019. Available: https://flink.apache.org/
[3] PostgreSQL Global Development Group, "PostgreSQL: The World's Most Advanced Open Source Relational Database," Available: https://www.postgresql.org/
[4] D. Merkel, "Docker: Lightweight Linux Containers for Consistent Development and Deployment," Linux Journal, vol. 2014, no. 239, p. 2, Mar. 2014. Available: https://www.docker.com/
[5] M. Kleppmann, "Designing Data-Intensive Applications," O'Reilly Media, 2017. Available: https://dataintensive.net/