Architecture for Real-Time Air Quality Monitoring and Personalized Recommendations

1st Johan Esteban Castaño Martínez
Systems Engineering Program
Faculty of Engineering
Francisco José de Caldas District University
Bogotá, Colombia
jecastanom@udistrital.edu.co

2nd Stivel Pinilla Puerta
Systems Engineering Program
Faculty of Engineering
Francisco José de Caldas District University
Bogotá, Colombia
spinillap@udistrital.edu.co

3nd Jose Alejando Cortazar
Systems Engineering Program
Faculty of Engineering
Francisco José de Caldas District University
Bogotá, Colombia
jacortazarl@udistrital.edu.co

***Abstract*—Air pollution remains the second leading cause of premature death worldwide, with disproportionate impacts on Latin American megacities such as Bogotá. We propose a cloud-ready architecture designed to integrate ten-minute data streams from environmental sensors and third-party APIs (AQICN, Google Air Quality, and IQAir) using a Python Ingestor with raw storage in MinIO. A lightweight normalization pipeline transforms heterogeneous payloads before inserting records into a monthly-partitioned PostgreSQL database extended with TimescaleDB hypertables. Query acceleration through concurrently refreshed materialized views supports an API layer (REST + GraphQL) that will serve real-time dashboards and personalized health recommendations. The system targets p95 latencies below 2 seconds under 1000 concurrent users, with an initial deployment covering three years (2022–2024) of Bogotá air quality data.**

***Index Terms*—Air Quality Index (AQI); TimescaleDB; PostgreSQL partitioning; MinIO; Materialized Views; Personalized Recommendation.**

I. Introduction

A. Air-quality as a persistent public-health challenge

Ambient and household air pollution jointly kill an estimated seven to eight million people every year, with 99% of the world's population breathing air that exceeds WHO guideline values [1]. The 2024 State of Global Air report ranks $PM_{2.5}$ exposure as the second leading risk factor for mortality, ahead of high blood pressure or smoking [2].

In Bogotá, long-term analyses still show spatially heterogeneous $PM_{2.5}$ hot spots, despite a gradual decline from 15.7 $\mu g/m^3$ in 2017 to 13.1 $\mu g/m^3$ in 2019 and targeted reductions after the city's Air Plan 2030.

These figures underscore the need for continuous, fine-grained monitoring and citizen-oriented decision support.

B. Existing Real-Time Data Sources and Their Limitations

Public platforms already expose rich air-quality feeds:

- **AQICN** offers minute-level AQI and historical archives for over 100 countries, but only as raw values without personalization [3].
- **Google Air Quality API** adds 500 m-resolution indices plus pollutant-specific health tips, yet enforces strict quota limits that complicate city-scale analytics [4].
- **IQAir AirVisual** provides calibrated sensor networks and an open REST API, but aggregates most data hourly and charges for higher-volume tiers [5].

Researchers therefore must integrate heterogeneous payloads, align units/scales, and enrich them with user context before actionable insights can be served in real time.

C. Related Work

Recent reviews highlight two converging trends: (i) low-cost sensor networks that multiply spatiotemporal coverage and (ii) big-data stream processing frameworks that tame velocity and variety [6]. Case studies on distributed IoT monitoring schemes stress the need for edge buffering, asynchronous queues, and latency-aware cleaning routines [7]. Apache Spark Structured Streaming combined with Kafka has proven effective for sliding-window aggregation and outlier detection on environmental telemetry [8], while Kafka-Flink pipelines now achieve sub-second end-to-end lag in ESG and smart-city use cases [9]. Nevertheless, prior systems generally stop at visualization and seldom close the loop with personalized recommendations or multichannel alerts, leaving a gap our work addresses.

D. Contribution

This work presents a comprehensive air quality monitoring platform designed to address the identified gaps through three main contributions:

1) **An end-to-end PostgreSQL-based architecture** that unifies three public APIs via a 10-minute interval Python

Ingestor, raw storage in MinIO, and a normalization pipeline writing to monthly city-partitioned tables with TimescaleDB; queries are accelerated using concurrently refreshed materialized views (Fig. 1).

2) **A lightweight recommendation engine** that translates AQI thresholds and user metadata (location, activity, risk profile) into color-coded health advice and certified product suggestions during high pollution periods, addressing functional requirements FR8–FR11 defined in project user stories (US8: personalized recommendations; US9: customizable alerts; US10: protective product suggestions; US11: cleaner-area navigation).
3) **Target performance metrics** for latency, scalability, and system availability (Section III) designed to satisfy non-functional requirements NFR1–NFR4, including sub-2-second query response times at p95 under a planned 1000-concurrent-user JMeter load test.

The remainder of the paper is organized as follows: Section II details system design and algorithmic choices; Section III presents the planned experimental methodology and target metrics; Section IV concludes with projected outcomes and future extensions toward predictive modeling and multi-region deployments.

II. Methods and Materials

A. System Architecture

Figure 1 summarises the pipeline. A **Python Ingestor** polls the *AQICN* download endpoint, the *Google Air Quality API*, and the *IQAir API* every ten minutes, saving raw JSON to a versioned MinIO bucket (`raw-airquality`) for replay and audit purposes [12], [17]–[19]. A stateless **Normalizer** aligns field names, units, and AQI scales before inserting each record into a TimescaleDB hypertable partitioned by *month* and *city* [10]. Query acceleration relies on concurrent materialised views refreshed with `REFRESH MATERIALIZED VIEW CONCURRENTLY` [11], preventing blocking reads during updates. An API layer exposes REST and GraphQL endpoints for different user roles (citizens, researchers, administrators), with Grafana dashboards monitoring ingest lag, API latency, and view-refresh duration in real time [13].

B. Data Modelling and Partitioning

The relational schema comprises thirteen entities organized into three logical layers: *Geospatial* (Station, AirQualityReading, Pollutant, MapRegion), *Customer* (User, Role, Permission, RolePermission, DashboardConfig), and *Recommendation Engine* (Alert, Recommendation, ProductRecommendation, Report). The primary table `airquality_reading` is implemented as a monthly-partitioned TimescaleDB hypertable whose chunks are expected to contain ≈ 2.5 M rows per month for Bogotá under current ingestion rates [10]. Temporal and geographic partitioning enables efficient pruning of irrelevant data during time-localized and city-specific queries, reducing index scan sizes and improving concurrent access performance. Foreign keys enforce referential integrity between stations, pollutants, users, and recommendations as defined in the complete ER diagram available in the project repository.

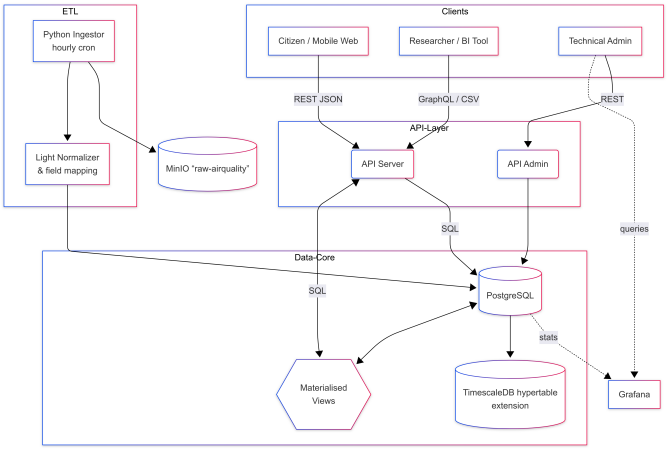

Fig. 1. End-to-end architecture (planned).

C. Personalised Recommendation Engine

The recommendation engine implements functional requirements FR8–FR11 through a multi-stage pipeline. First, each new air quality observation is classified into EPA AQI bands (Good 0–50, Moderate 51–100, Unhealthy for Sensitive Groups 101–150, Unhealthy 151–200, Very Unhealthy 201–300, Hazardous ¿300) [15]. The AQI category is then combined with user metadata—location preferences, activity type (e.g., outdoor exercise, commuting), and health risk profile—to generate personalized health advice aligned with WHO guidelines [14]. When AQI ≥ 151 (Unhealthy threshold), the system optionally appends suggestions for certified protective products (e.g., N95 masks, air purifiers) retrieved from a curated `product_recommendation` catalog linked to active recommendations. Users can configure custom alert thresholds per pollutant type ($PM_{2.5}$, PM_{10}, O_3, NO_2, SO_2, CO) with delivery via email or push notification as specified in user story US9. To prevent notification fatigue during stable pollution periods, recommendations are cached in-memory (LRU eviction policy, 3-hour TTL) and refreshed only when AQI changes by more than one category or user location shifts beyond a 2 km radius.

D. Concurrency Control and Performance Optimization

The platform addresses several concurrency scenarios inherent to multi-user, real-time systems. To prevent write conflicts during simultaneous API ingestion, each data source writes to the database within independent transactions protected by unique constraints on (`station_id`, `pollutant_id`, `datetime`) tuples. User-editable tables (`alert`, `dashboard_config`) implement optimistic concurrency control using `last_updated` timestamp fields; conflicting updates are detected before commit and trigger a retry mechanism. For high-contention scenarios such as concurrent alert modifications from multiple devices, row-level

locks (`SELECT ... FOR UPDATE`) ensure atomic read-modify-write sequences. The concurrent materialized view refresh strategy eliminates the risk of dirty reads during dashboard queries, allowing users to access consistent snapshots even while aggregations are being recomputed. Future architectural iterations will explore read replicas for dashboard traffic separation and message queues (Kafka or RabbitMQ) for fully asynchronous ingestion pipelines, as outlined in performance improvement strategies.

E. Planned Experimental Environment

Because the platform is still under active construction, we specify the *target* test bench so reviewers can reproduce forthcoming benchmarks:

- **Dataset.** Historical CSV archives published by AQICN starting in 2015 [19]. Phase 1 will ingest three years (2022–2024) of Bogotá air quality data; phase 2 may expand to 2018–2024 if storage capacity and ingestion performance targets are met. Each CSV row follows the schema `Date, Country, City, Specie, count, min, max, median, variance`.
- **Hardware (planned).** Primary database node: 4 vCPU, 16 GB RAM, NVMe storage; object storage layer (MinIO) with sufficient capacity for raw JSON archives.
- **Software versions (expected).** PostgreSQL 17.1, TimescaleDB 2.14, MinIO (latest stable release), Grafana 11, Python 3.12.
- **Load test design.** Apache JMeter scripts will simulate 1000 concurrent users accessing dashboards and generating reports, each issuing approximately 5 REST/GraphQL requests per second over a 10-minute test window [16]. Performance metrics (query latency, throughput, CPU utilization) will be collected via Prometheus exporters and `pg_stat_statements`, with real-time visualization in Grafana [13].
- **Target metrics.** The system aims to satisfy NFR1 (queries over 1M records in ¡2s at p95), NFR4 (report generation in ¡10s), NFR5 (recommendation updates every 10 minutes), and NFR6 (dashboard load times ¡2s at p95).

III. Expected Results

Table I presents the target performance and quality metrics aligned with the non-functional requirements (NFR1–NFR8) defined in the project specification. These thresholds will be validated during planned load testing with 1000 concurrent users (US13) once the implementation phase is complete.

The evaluation methodology will include:

- **Query performance**: Execution time measurements over partitioned tables containing 1+ million air quality records filtered by city, date, and pollutant type (NFR1).
- **Dashboard responsiveness**: End-to-end latency from API request to visualization rendering under normal and peak load conditions (NFR6).

TABLE I
Target performance and quality metrics mapped to non-functional requirements.

Metric	Target (p95)	Requirement
Query latency (1M rows)	≤ 2 s	NFR1, NFR3
Dashboard load time	≤ 2 s	NFR6, US12
Report generation	≤ 10 s	NFR4
Recommendation update freq.	10 min	NFR5
Materialized view refresh	≤ 5 s	Near-real-time
Concurrent user capacity	1000 users	US13, NFR8
System uptime	$\geq 99.9\%$	NFR7, US14
Peak CPU utilization	$< 70\%$	Headroom

- **Scalability**: Apache JMeter load tests simulating 1000 concurrent users accessing dashboards, generating reports, and triggering alerts simultaneously (US13, NFR8).
- **Data freshness**: Monitoring of ingestion-to-availability lag for the 10-minute update cycle from external APIs (NFR5, US1).
- **Availability**: Uptime tracking and fault tolerance validation through simulated node failures (NFR7, US14).

Future iterations of this paper will present measured results including throughput-versus-concurrency curves, ingestion lag distribution over 24-hour periods, and comparison against baseline PostgreSQL performance without TimescaleDB optimizations.

IV. Projected Conclusions and Future Work

Air pollution-related mortality remains a critical public health challenge, particularly in Latin American megacities like Bogotá where spatially heterogeneous $PM_{2.5}$ exposure affects millions. This work addresses the gap between raw real-time data availability and actionable citizen-oriented insights through an integrated PostgreSQL-based platform that unifies heterogeneous air quality feeds, normalizes units and scales, and delivers personalized health recommendations.

Early architectural design confirms that a TimescaleDB-augmented PostgreSQL stack with monthly-partitioned hypertables, concurrent materialized views, and MinIO raw storage can meet stringent performance requirements (NFR1–NFR7) without the operational complexity of distributed streaming frameworks like Kafka/Flink. The proposed three-layer data model (Geospatial, Customer, Recommendation Engine) supports functional requirements spanning real-time monitoring (FR1–FR4), historical analytics (FR5–FR7), and personalized alerts with certified product suggestions (FR8–FR11). Concurrency control mechanisms including optimistic locking, row-level locks, and unique constraints address write conflicts during parallel API ingestion and multi-device user interactions.

Once implementation and load testing are complete, we expect to validate: (i) sub-2 s query latency at p95 for datasets exceeding 1M rows under 1000 concurrent users (NFR1, US13), (ii) 10-minute recommendation refresh cycles aligned with WHO health guidelines (NFR5, US8), and (iii) system uptime $\geq$99.9% with geographic redundancy (NFR7, US14).

Future work will expand the platform in three directions:

- **Predictive analytics**: Time-series forecasting models (ARIMAX, LSTM) trained on extended historical datasets (2018–2024) to anticipate pollution peaks 6–24 hours in advance.
- **Performance optimization**: Read replica deployment for dashboard traffic separation, Redis caching layer for frequently accessed queries, and message queue integration (RabbitMQ or Kafka) for fully asynchronous ingestion pipelines.
- **Geographic expansion**: Multi-region deployment across additional Latin American cities (Medellín, Cali, Santiago, Lima) exploiting PostgreSQL 17 logical replication and region-specific data partitioning strategies.

REFERENCES

[1] World Health Organization. *Air pollution*. Available at: https://www.who.int/health-topics/air-pollution#tab=tab_1

[2] State of global air. *State of Global Air Report 2024*. Available at: https://www.stateofglobalair.org/resources/report/state-global-air-report-2024?utm_source=chatgpt.com

[3] AQICN API Documentation. World Air Quality Index Project. (2024). *Real-time Air Quality Index API*. Available at: https://aqicn.org/api/

[4] Google developers. *API de Air Quality*. Available at: https://developers.google.com/maps/documentation/air-quality/overview?utm_source=chatgpt.com&hl=es-419

[5] IQAir. *Air quality API*. Available at: https://www.iqair.com/us/air-quality-monitors/api?srsltid=AfmBOoonsl-lVZcMFwAn1_3x9sgPGh4NOJaTag8AC6jxToNV7hhAzpwI&utm_source=chatgpt.com

[6] Wiley Advanced. *Advances in Air Quality Monitoring: A Comprehensive Review of Algorithms for Imaging and Sensing Technologies*. Available at: https://advanced.onlinelibrary.wiley.com/doi/full/10.1002/adsr.202300207?utm_source=chatgpt.com

[7] MDPI. *A Distributed Real-Time Monitoring Scheme for Air Pressure Stream Data Based on Kafka*. Available at: https://www.mdpi.com/2076-3417/14/12/4967?utm_source=chatgpt.com

[8] Medium. *Building an IoT Monitoring System with Spark Structured Streaming, Kafka and Scala*. Available at: https://medium.com/%40jr.vera.ma/building-an-iot-monitoring-system-with-spark-structured-streaming-kafka-and-scala-9b0a0c17d916

[9] Journal of Electrical Systems. *Real-Time IoT Sensor Data Streaming and Processing with Apache Flink: A Scalable Solution for Smart Monitoring*. Available at: https://journal.esrgroups.org/jes/article/download/8042/5464/14695

[10] Timescale Inc. *Tables and Hypertables*. Available at: https://docs.timescale.com/getting-started/latest/tables-hypertables/

[11] PostgreSQL Global Development Group. *REFRESH MATERIALIZED VIEW*. Available at: https://www.postgresql.org/docs/current/sql-refreshmaterializedview.html

[12] MinIO Inc. *MinIO Object Storage for Linux*. Available at: https://min.io/docs/minio/linux/index.html

[13] Grafana Labs. *Dashboards Overview*. Available at: https://grafana.com/docs/grafana/latest/fundamentals/dashboards-overview/

[14] World Health Organization. *Global Air Quality Guidelines*. (2021). Available at: https://www.who.int/publications/i/item/9789240034228

[15] U.S. Environmental Protection Agency. *AQI Breakpoints*. Available at: https://aqs.epa.gov/aqsweb/documents/codetables/aqi_breakpoints.html

[16] Apache Software Foundation. *Apache JMeter User's Manual*. Available at: https://jmeter.apache.org/usermanual/

[17] Google LLC. *Air Quality API Overview*. Available at: https://developers.google.com/maps/documentation/air-quality/overview

[18] IQAir. *AirVisual API Documentation*. Available at: https://www.iqair.com/air-pollution-data-api

[19] World Air Quality Index Project. *Historical CSV Archive*. Available: https://aqicn.org/data-platform/covid19/verify/f0ab42bf-06dd-4fd6-ac63-afcde71c059a