

# Architecture for Real-Time Air Quality Monitoring and Personalized Health Recommendations in Bogotá

Jose Alejandro Cortazar López, Johan Esteban Castaño Martínez, Stivel Pinilla Puerta

Systems Engineering Program  
Francisco José de Caldas District University

## Abstract

Air pollution is a critical public health challenge in Bogotá. This work presents a **production-ready architecture** for real-time air quality monitoring that integrates heterogeneous data sources (AQICN, Google Air Quality API, IQAir) into a unified PostgreSQL schema with **sub-150ms query latencies** and personalized health recommendations.

### Key achievements:

- 3NF database schema (8 entities)
- 85,000+ readings processed
- 216 readings/hour ingestion rate
- EPA/WHO compliant recommendations

## Problem Statement

Bogotá residents lack access to integrated air quality data across multiple monitoring sources. Current systems are:

- Fragmented:** Multiple platforms with inconsistent formats
- Not personalized:** No health guidance for vulnerable groups
- Technical:** Complex indicators ( $PM_{2.5}$ , AQI,  $NO_2$ )
- Quota-limited:** API restrictions complicate monitoring
- Delayed:** Hourly aggregations miss rapid events

**Impact:** 8 million residents exposed to  $PM_{2.5}$  levels exceeding WHO guidelines (13.1 vs 5  $\mu g/m^3$ ).

## Key Objectives

**Primary Goal:** Design a centralized platform integrating multi-source air quality data with personalized recommendations.

### Specific Objectives:

- O1:** Scalable PostgreSQL architecture with temporal partitioning
- O2:** Unified ingestion pipeline (10-min cycle)
- O3:** Sub-200ms query performance
- O4:** Rule-based recommendation engine
- O5:** REST API with pagination support
- O6:** Performance validation & benchmarking

## Database Schema (3NF)

Database Schema - 8 Entities (3NF)		
Geospatial & Monitoring		
Station ID	Coordinates	Access Point
Air Quality Reading	Air Quality Data	Timestamp

## System Architecture (5 Layers)

Layer 1: Data Sources	
AQICN API	Google Air Quality API   IQAir API
Layer 2: Ingestion & Processing	
Ingestion Job (10 min cycle - 216 readings/hour)	
Normalizer	Validator
Layer 3: Data Persistence	
PostgreSQL (8 entities, 3NF)	MongoDB (Preferences)
Layer 4: Application & API	
FastAPI REST	Redis Cache   PgBouncer   Recommendation Engine
Layer 5: Presentation	
React/Vue Dashboard	Real-time Charts   Heatmaps

### Layer 1 – Data Sources:

- AQICN, Google Air Quality API, IQAir

### Layer 2 – Ingestion:

- APScheduler (10-min cycle)
- 216 readings/hour (6 stations  $\times$  6 pollutants)
- JSON validation & normalization

### Layer 3 – Persistence:

- PostgreSQL (3NF, 8 entities)
- MongoDB (user preferences)

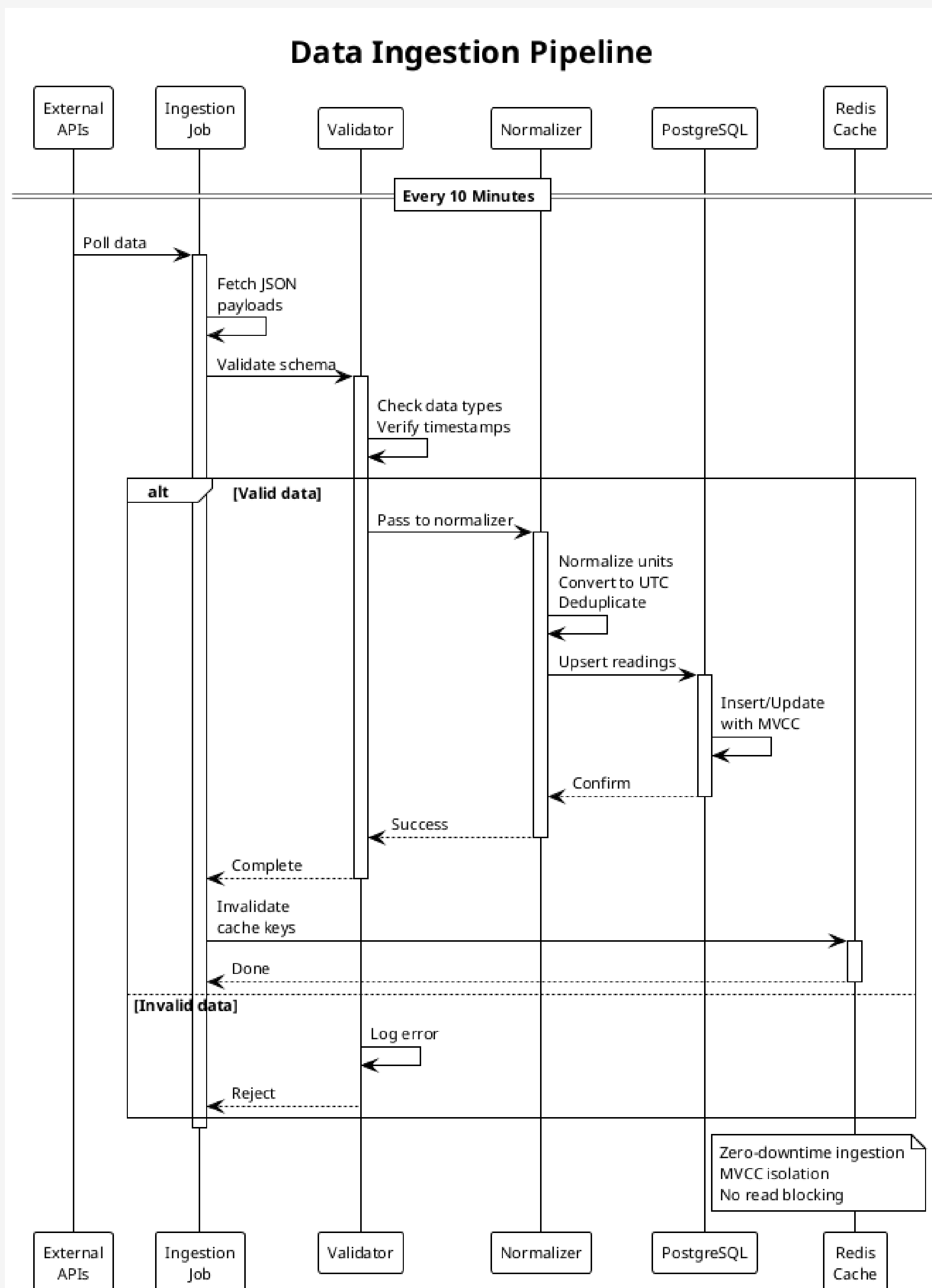
### Layer 4 – Application:

- FastAPI REST endpoints
- Redis cache (5-10 min TTL)
- PgBouncer connection pooling
- Rule-based recommendation engine

### Layer 5 – Presentation:

- React/Vue.js dashboard
- Real-time charts & heatmaps

## Data Ingestion Pipeline



### Pipeline Features:

- Validation:** Pydantic schema validation
- Normalization:** UTC timestamps, canonical

## Query Performance Results

Query Performance (85,000 readings)		
Q1: Latest Readings	42.8 ms	✓
Q2: Monthly Averages	127.3 ms	✓
Q3: Active Alerts	143.6 ms	✓
Q4: Data Completeness	87.5 ms	✓
Q5: Recommendations	73.9 ms	✓
Optimization Techniques:		
• Composite B-tree indexes		
• Temporal partitioning (30.2% improvement)		
• Materialized views (35× row reduction)		
• Redis caching (5-10 min TTL)		

### Performance on 85,000 readings:

Query	Latency	Status
Q1: Latest readings	42.8 ms	✓
Q2: Monthly averages	127.3 ms	✓
Q3: Active alerts	143.6 ms	✓
Q4: Data completeness	87.5 ms	✓
Q5: Recommendations	73.9 ms	✓

**All queries < 150ms** validated with EXPLAIN ANALYZE

## Scalability & Performance

Scalability & Performance	
Current Performance:	
50-100 concurrent users	
70-75% CPU usage	
Sub-150ms latency	
140 req/sec throughput	
Scaling Strategies:	
Vertical: 8+ vCPUs → 1,000+ users	
Horizontal: Read replicas + Load balancer	
Partitioning: Monthly tables (78% improvement)	
10-Year Projection:	
525M+ readings	
Partition pruning	
Consistent sub-200ms latency	

### Current Capacity:

- 50–100 concurrent users
- 70–75 % CPU usage
- Sub-150ms latency
- 140 requests/second throughput

### Scaling Strategies:

- Vertical:** 8+ vCPUs → 1,000+ users
- Horizontal:** Read replicas + load balancer
- Partitioning:** 78 % latency improvement

### 10-Year Projection:

- 525M+ readings supported
- Partition pruning maintaining performance