

Smart City Real-Time Monitoring System

Overview

The **Smart City Real-Time Monitoring System** is a complete end-to-end data pipeline designed to monitor and analyze **urban traffic and air quality** across multiple Egyptian cities in real-time. The system simulates IoT sensor data, processes it through **Apache Kafka** and **Apache Spark**, stores it in **PostgreSQL**, and visualizes key insights using an interactive **Streamlit dashboard**.

It includes automated **KPI calculations**, **email alerts** for critical thresholds, and full orchestration using **Docker Compose**.

The architecture is fully scalable, fault-tolerant, and ready for production, built to handle high-velocity IoT data streams.

Currently, the system monitors **five major Egyptian cities** *Cairo, Giza, Alexandria, Mansoura, and Aswan* generating synthetic sensor data on **traffic congestion, vehicle flow, and air quality metrics** such as PM2.5, PM10, NO₂, CO, O₃, and AQI.

System Architecture

1. Data Generation (Sensor Simulation)

- **Purpose:** Simulates IoT sensors producing live traffic and air quality data.
- **Implementation:** `data_producer.py` written in Python using Faker and numpy to generate realistic data.
- **Kafka Topics:**
 - `traffic_data` → timestamp, city, congestion_level (%), avg_speed_kmh, vehicle_count
 - `air_quality_data` → timestamp, city, pm25, pm10, no2, co, o3, air_quality_index
- **Output:** JSON messages published every second to Kafka (~3,600/hour per topic).

2. Kafka Producer

- Publishes generated data to Apache Kafka (`kafka:9092`) for distributed streaming.
- Uses **JSON serialization**, **acks=all** for durability, and **retry with backoff** for reliability.

3. Spark Structured Streaming Consumer

- **Consumes** live Kafka data, **cleans and transforms** it, and **writes results to PostgreSQL**.
- Implemented via `spark_consumer.py` using **PySpark Structured Streaming**.
- Features:
 - Real-time parsing & aggregation.
 - Continuous trigger for low-latency.
 - Checkpointing for fault tolerance (`/tmp/checkpoints`).
- **Dependencies:** `spark-sql-kafka-0-10_2.12:3.5.0`, `postgresql:42.7.3`.

4. PostgreSQL Database

- Central storage for both raw and aggregated data.
- **Schema:**
 - `traffic_data`, `air_quality_data`, `kpis_summary`
- Indexed on timestamp and city for fast analytics.

5. KPI Calculator (Spark Batch Job)

- PySpark script (`kpi_calculator.py`) computing advanced metrics:
 - **Air Quality KPIs:** Average AQI, compliance rate, pollution peaks.
 - **Traffic KPIs:** Congestion %, avg speed, vehicle density, emission estimates.
 - **Correlation Metrics:** Relationship between traffic and air pollution.
 - **Anomaly Detection:** Detects sudden congestion spikes.
- Writes results to `kpis_summary`.
- Scheduled hourly (via cron or future Airflow DAG).

6. Streamlit Dashboard

- Web-based dashboard (`app.py`) for real-time monitoring and visualization.
- Features:
 - KPI metrics and visual alerts.
 - Charts (bar, line, pie) and live data tables.

- Auto-refresh every 30 seconds using `@st.cache_data(ttl=30)`.
- **Libraries:** Streamlit, Pandas, Plotly, psycopg2.
- Available on localhost:8501.

7. Email Alerts & Automation

- **Goal:** Send notifications when KPIs exceed thresholds (e.g., AQI >100 or congestion >70%).
- **Implementation:** Python smtplib integrated into Streamlit or standalone script.
- **Planned Enhancements:** Automate alert scheduling via **Airflow** or **cron jobs**.

8. Docker Compose Orchestration

- All services containerized and linked for easy deployment.
 - **Services:**
 - zookeeper, kafka, spark-master, spark-worker, postgres, data-producer, spark-consumer, kpi-job, streamlit.
 - **Commands:**
 - `docker-compose up -d` → Start all services.
 - `docker-compose run kpi-job` → Trigger KPI batch job.
 - Includes volumes (`pgdata`, `spark_checkpoints`) and preconfigured networks.
-

Deployment & Usage

1. Clone the repository.
 2. Run `docker-compose up -d`.
 3. Wait for data flow (≈5–10 minutes).
 4. Access dashboard at `http://localhost:8501`.
 5. Monitor Spark UI (`http://localhost:8080`) for streaming status.
 6. Run KPI job manually: `docker-compose run kpi-job`.
-

Conclusion

The Smart City Real-Time Monitoring System demonstrates a fully integrated **real-time analytics pipeline** that can scale for smart city infrastructure.

It enables **data-driven decision-making** for urban planning, offering live insights into traffic flow, pollution trends, and environmental risks.