Data Architecture Design: Real-Time Video Event Dashboard

Overview

We need to handle video camera event data (about 10,000 per second), clean up duplicates, join it with a static reference table, and make the results available in a dashboard with low latency. The goal is to make sure the dashboard always shows the latest information without duplicates, and that the system is reliable enough for production.

Questions

- What exactly should the dashboard show — counts per geo, top-X items, anomaly alerts?
- How fresh does the data need to be — seconds, under a minute, or is a small delay fine?
- How long do we need to keep the raw data vs aggregated results?
- What defines a duplicate? Do we just drop them or keep a history?
- What's the acceptable latency from ingestion to display?
- Do we need alerts if data processing falls behind or fails?

Assumptions

- Each event has a unique ID (or at least a combination like geoId + deviceId + timestamp).
- Reference table (geoId → geoName) is small and rarely changes.
- Dashboard should be close to real-time (<1 min delay).
- Raw events are kept for ~30 days, aggregates for ~90 days.
- Queries on the dashboard are mostly read-heavy (counts, top-X).
- It's okay to lose strict ordering if we meet the latency requirement.

Proposed Flow

1. Ingestion

Use **Kafka** as the main entry point. All camera events go into Kafka. I

- fast, reliable, and can handle spikes.
- decouples producers from the rest of the system.

2. Stream Processing

Use **Flink** to read from Kafka. Flink handles stateful operations well, so it's a good fit for **deduplication** (using keyed state on eventId). That way retries from the edge don't end up as duplicate events in the pipeline.

Once deduplicated, Flink can either push clean events back into Kafka or directly feed Spark.

3. Aggregation + Join

Use **Spark Structured Streaming** (or even micro-batch Spark jobs) to:

- Join events with the reference table (Dataset B). Since Dataset B is small, we can use a **broadcast join** for speed.
- Aggregate per geo id
- Handle skew if needed (salting)

4. Storage

- **Raw events** → Stored in **S3 with Delta Lake**.

  Delta Lake adds ACID transactions and schema enforcement on top of S3
- **Reference table** → Since it doesn't change much, stored in **Parquet** (or also Delta Lake)
- **Aggregated results** → Stored in **ClickHouse** for fast OLAP queries. This gives the dashboard sub-second queries even on large datasets.

5. Dashboard

Grafana to visualize results from ClickHouse.

- Dashboard queries will be low-latency, and refresh rate can be selected (e.g., every 30s or 1 min).

Considerations

- **Deduplication**: Flink ensures retries from upstream don't inflate counts.
- **Latency**: Kafka + Flink + Spark should keep total end-to-end latency under 1 minute.
- **Scalability**: Kafka and Flink handle throughput at the stream layer, Spark handles aggregation at scale, and ClickHouse makes queries fast.
- **Monitoring**: Add alerts for Kafka consumer lag, Flink checkpoint failures, Spark job delays, or ClickHouse query errors.