Data Architecture Design for Real-Time Item Detection Dashboard

1. Problem Overview & Key Requirements

- High-Throughput Streaming: ~10,000 events/second ingested from edge video cameras (Dataset A).
- Static Reference Table: Dataset B contains fixed metadata like location names.
- Real-Time Join & Visualization: Users expect joined results available with minimal latency.
- Deduplication Required: Duplicate events may occur due to retries or upstream issues.
- Dashboard Ready Output: Result must support fast querying (e.g., for Power BI or Grafana).

---

2. Architecture Pattern: Kappa Architecture (vs Lambda)

We adopt Kappa Architecture, as this use case focuses on real-time streaming with no explicit batch processing requirement.

Kappa Architecture simplifies by treating batch and stream processing uniformly, using a single stream-processing pipeline for both real-time and reprocessing historical data (if necessary).

If future batch analysis is needed, we can extend the system toward Lambda by integrating a scheduled batch job layer in Databricks.

---

3. Proposed Architecture (Azure Stack)

Component	Purpose
Azure Event Hub	Ingests high-frequency real-time events from edge devices (Dataset A)
Azure Databricks (Structured Streaming)	Real-time data cleaning, deduplication, enrichment (join with Dataset B)
Azure Data Lake + Delta Lake	Stores output table with ACID guarantees and time-travel queries
Power BI / Grafana	Visualizes the joined results with sub-second latency
Azure Key Vault	Secures credentials and secrets
Azure Monitor / Log Analytics	Tracks job health, latency, error rates

---

4. Data Flow & Processing Logic

Step-by-Step Flow:

1. Ingestion:
    - Events (Dataset A) are ingested from edge devices into Azure Event Hub.

2. Streaming Processing (Azure Databricks):
   - A Structured Streaming job reads events from Event Hub.
   - Deduplication can be performed using:

     ```
     streamDF.dropDuplicates("event_id")
     ```

     If event_id is not available or timestamps are unreliable, we can fallback to UUID-based deduplication or watermarking.
   - Static Dataset B (location mappings) is broadcast-joined for fast enrichment.
3. Output & Querying:
   - The enriched, deduplicated stream is written to a Delta Lake table in Azure Data Lake.
   - Delta Lake allows:
     - Schema evolution
     - ACID updates
     - Time travel queries (for rollback or trend analysis)
4. Dashboard Access:
   - Power BI or Grafana queries the Delta table directly via SQL Analytics endpoint for live dashboards.

---

5. Security Considerations

Security Aspect	Implementation
Access Control	Role-based access (RBAC) and row-level security in Delta Lake
Encryption	Data is encrypted in transit (TLS) and at rest (Azure-managed keys or BYOK)
Audit Logging	Access logs tracked via Azure Monitor and Log Analytics

---

6. Scalability & Fault Tolerance

- Auto-Scaling: Use Databricks autoscaling clusters to handle spikes (e.g., peak traffic events).
- Checkpointing: Use checkpoint locations in Structured Streaming to ensure exactly-once processing.
- Backpressure Handling: Configure Event Hub consumer groups and trigger rates to avoid overload.

---

7. Key Questions for PM / Stakeholders

Data Retention Policy (e.g., 7 days, 30 days?)

Why It Matters:

- Determines **storage costs** and the best strategy for **hot, warm, or cold storage tiers**.
- Ensures **compliance** with **data governance and privacy regulations**.
- Impacts whether **historical trends** can be analyzed for **audit, debugging, or predictions**.

Key Considerations:

- **Raw data** may be needed for **reprocessing or forensic analysis** but is costly to store long-term.
- **Processed data** drives the dashboard but might not need permanent storage.
- Decision impacts whether to use **Delta Lake (versioned storage) or Azure Blob Archive (cold storage)**.

Expected End-to-End Latency for Dashboard Visibility?

Why It Matters:

- Defines **processing speed requirements** (e.g., sub-second, near real-time, batch processing).
- Guides **architecture choice** (e.g., **structured streaming vs micro-batch Spark**).
- Influences **resource allocation** for **low-latency performance**.

Key Considerations:

- Determines whether **Spark Structured Streaming** or **batch processing** is the right approach.
- Affects **checkpoint intervals**, **auto-scaling cluster sizing**, and **processing guarantees** (e.g., **at-least-once vs exactly-once delivery**).
- Helps define **Service-Level Agreements (SLAs)** for **monitoring & tuning performance**.

Expected Data Growth Rate, Peak Volume, and Query Concurrency?

Why It Matters:

- Ensures **scalability** to handle peak traffic efficiently.
- Defines **auto-scaling policies**, **partitioning strategies**, and **load balancing**.
- Prevents **performance bottlenecks** in query processing—especially **high-concurrency workloads**.

Key Considerations:

- How many **users will query simultaneously**?
- What is the **estimated increase in event volume** over time?

- Should indexing strategies be **adaptive** to load changes?

Do Dashboards Require Raw Events, Aggregates, or Both?

Why It Matters:

- Determines the **data processing strategy** for the dashboard (raw logs vs. summary stats).
- Guides **SQL indexing, caching strategies**, and **ETL pipeline optimization**.
- Affects **database schema design** & **storage formats** (e.g., **Delta Lake vs Azure SQL**).

Key Considerations:

- If **aggregates** are used, **precomputing summaries** reduces query load.
- If **raw events** are queried, **indexed partitioning** is essential for fast access.
- If **both** are needed, caching strategies must balance **speed vs. storage**.

Should Deduplication Rely on Event ID, Timestamp, or Both?

Why It Matters:

- Ensures **accurate deduplication** while **retaining valid events**.
- Helps prevent **false positives** in duplicate detection.
- Impacts **data validation steps**, especially when **timestamps are unreliable** (due to **clock drift**).

Key Considerations:

- If **event IDs are unique**, **dropDuplicates("event_id")** simplifies deduplication.
- If **timestamps are used**, ensure **clock synchronization across devices**.
- If both are combined, build a **windowing strategy to prevent data loss**.

Schema Evolution: Can New Fields Be Added Over Time?

Why It Matters:

- Determines whether the **storage format** can **evolve dynamically**.
- Ensures **backward compatibility** when upstream systems **modify event schemas**.
- Reduces system downtime or **manual intervention** for field additions.

Key Considerations:

- **Delta Lake supports schema evolution**, making it ideal for changing formats.
- If the schema **changes frequently**, prefer **schema-on-read solutions**.
- Ensure **dashboards remain compatible** with schema updates.

Query Patterns: Will Users Query Historical Trends or Only Live Data?

Why It Matters:

- Impacts **storage format** & **partitioning strategy**.
- Defines **indexing optimizations** for fast retrieval.

Key Considerations:

- If **only live data** is queried, optimize for **fast ingestion & recent data access**.
- If **historical trends are needed**, implement **efficient partitioning & time-travel queries**.
- Helps decide between **real-time stores (Azure SQL, Delta Lake) vs long-term warehousing (Azure Synapse)**.

Criticality: What's the Business Impact of Delays or Missed Events?

Why It Matters:

- **Affects fault tolerance design** and system resilience.
- Helps **justify investment** in **high-availability architecture, replication, and redundancy**.
- Guides **error handling, monitoring, and incident response policies**.

Key Considerations:

- If **business impact is critical**, guarantee **exactly-once delivery** with strong durability.
- If **impact is moderate**, leverage **best-effort processing with failure recovery**.
- Helps optimize **high-availability zones & disaster recovery planning**.

---

8. Why This Tech Stack?

Tech	Reason
Azure Event Hub	Handles massive streaming ingestion at low cost and high reliability
Databricks Structured Streaming	Built-in deduplication, fault tolerance, seamless Spark integration
Delta Lake	Combines the power of data lakes and data warehouses; ACID + fast reads
Power BI	Native integration with Azure stack, rich visualization
Kappa Architecture	Simple, scalable, ideal for streaming-first systems