Azki Data Engineering Hiring Task — Technical Report

Overview

This project implements a scalable data engineering pipeline for Azki. The pipeline supports:

- **Streaming ingestion** of user events enriched with user attributes
- **Denormalized purchase analytics**, combining events with order + financial data
- **Real-time and batch aggregations** for dashboards and BI
- **Data quality checks, monitoring, and backfill workflows**

Technologies Used

- **Kafka** – streaming backbone
- **Kafka Connect + Debezium** – CDC from MySQL
- **Schema Registry** – schema evolution and compatibility
- **MySQL** – source OLTP system
- **ClickHouse** – analytical warehouse
- **Redpanda Console** – Kafka UI
- **Docker Compose** – local cluster orchestration
- **Spark** – batch ETL and historical backfill

Architecture

The pipeline follows a **Streaming ELT** pattern:

Users and Orders (CDC Path)

- User and Orders data is stored in **MySQL**.
- **Debezium CDC** streams inserts/updates into a Kafka topic.
- ClickHouse consumes this with the **Kafka Engine**, maintaining an up-to-date users dimension using **ReplacingMergeTree**.

User Events (Streaming Path)

- A Python producer streams user_events from CSV into Kafka.
- ClickHouse consumes this stream into a **MergeTree** table.
- Events are enriched or aggregated via Materialized Views.

Aggregations & Denormalization

ClickHouse Materialized Views (MVs) are used for:

- **Real-time daily aggregation**.
- **Denormalizing purchase events**.

Batch & Backfill with Spark

Spark is used for:

- Batch daily aggregations
- Recomputing corrupted partitions
- Handling late-arriving data
- Providing deterministic, idempotent results

ClickHouse vs Spark for Aggregation

Instead of computing aggregates in Spark directly from Kafka/MySQL, the pipeline first loads **all raw data into ClickHouse**, then aggregates.

Why ClickHouse-Centric?

- Optimized columnar storage for analytics
- Very fast for SUM, COUNT, GROUP BY
- Materialized Views allow **ingestion-time aggregation**
- Reduces Spark operational overhead
- Simplifies architecture
- Spark is reserved for **batch recomputation**, not real-time streaming

Real-Time Aggregation (ClickHouse)

SummingMergeTree merges rows with the same primary key by summing numeric fields, perfect for streaming.

The MV:

1. Listens to inserts into user_events
2. Joins with users to enrich events with city
3. Aggregates by event_date, city, channel
4. Writes into daily_premium_by_city_channel

Why Keep a Batch Aggregation Path?

Real-time MVs are fast but vulnerable to:

- Duplicate ingestion
- CDC delay
- Late-arriving data
- Kafka replays

Spark complements the real-time path by providing:

- **Batch aggregates** (deterministic)
- **Backfill for corrupted or incomplete data**
- **Historical recomputation** across date ranges
- Ability to **replace real-time MVs entirely**

Part 2 – Denormalization & Query Performance

Part 2 focuses on building a denormalized model combining user events, product-specific order tables, and financial data.

The challenge:

- Order data is spread across **four** product tables
- Financial data is in a separate table
- User events arrive separately via Kafka

Solution:

Create a **unified order table**, then use a Materialized View to join everything into a flattened analytical fact table.

Unified Orders Layer (orders_union)

Each product table:

- Has a slightly different schema
- Gets standardized into orders_union
- Uses **ReplacingMergeTree** to handle CDC-like updates
- Has one MV per product type for ingestion

This allows downstream models to join against **one** consistent table.

Denormalized Purchase Table (user_purchase_events)

This table merges:

- User event metadata
- Product/order attributes
- Financial pricing info

Partitioning by event_time and sorting by (event_time, product_type, user_id) support efficient queries.

6.3 Materialized View for Purchase Enrichment

The MV:

- Listens to **user_events**
- Filters to event_type = 'purchase'
- Joins orders_union and financial_order
- Writes into user_purchase_events

This approach:

- Performs expensive joins **once at ingestion**, not at query time
- Makes the analytical table **directly BI-ready**
- Greatly improves query performance

Query Performance Optimizations

- **Denormalization** removes runtime joins
- **Partitioning by event_time** enables pruning
- **LowCardinality columns** reduce storage and speed up GROUP BY
- **ReplacingMergeTree** handles CDC for order + financial tables
- **MergeTree ORDER BY** optimizes time-series queries

6.5 Data Governance

- **RBAC** for analysts vs engineers
- Masking sensitive fields in views
- Query and access logs from ClickHouse system tables provide auditing capabilities.

Part 3 – Data Quality & Monitoring

Part 3 defines a comprehensive data-quality and monitoring strategy for the pipeline.
Because the system spans **Kafka**, **Debezium**, **ClickHouse**, and **Spark**, the goal is to ensure:

- **Freshness** – data arrives on time
- **Completeness** – no missing or dropped events
- **Correctness** – no duplicates or inconsistent joins
- **Schema stability** – safe evolution of upstream schemas
- **Recoverability** – ability to backfill or repair data

Freshness / Delay Monitoring

To track real-time ingestion latency, an ingestion_time column is added to raw ClickHouse tables.
Latency is computed as:

- latency = ingestion_time - event_time
- Monitor **p95 latency** and alert if it exceeds thresholds (e.g., > 90 seconds).
- Detect ingestion stalls when **no new events appear for X minutes**.

These checks ensure real-time dashboards and aggregates reflect **recent and timely data**.

Completeness & Consistency Checks

Completeness checks detect missing events, CDC gaps, or misaligned records between tables.

Missing Data Detection

- **Event gaps:** ensure continuous user event flow (events per minute/hour/day).
- **CDC gaps:** compare MySQL row count vs ClickHouse users table.
- **Missing financial/order data:** purchases without corresponding rows in orders_union or financial_order.

Volume Anomaly Detection

- Detect sudden spikes or drops in:
  - Events per hour
  - Orders per product type
  - Financial records

These checks highlight upstream outages, producer bugs, or incorrect message routing.

Schema Drift & Validation

Schema evolution is controlled through **Schema Registry**, ensuring safe, versioned ingestion.

Schema Drift Handling

- Enforce **backward/forward compatibility** for Kafka producers.
- Breaking schema changes are rejected early.
- Validate message schemas during ingestion into ClickHouse.
- Version-aware ingestion allows ClickHouse to evolve gracefully as schemas change.

This ensures ClickHouse always receives structurally valid, schema-consistent data.

Pipeline & Infrastructure Monitoring

A monitoring layer tracks the health of the entire pipeline.

Kafka Monitoring

- **Consumer lag** for ClickHouse Kafka Engine and Spark
- Partition health
- In/out throughput
- Producer error rates

ClickHouse Monitoring

- Merge queue size (indicates heavy ingestion pressure)
- Disk usage per partition
- Replication/merge delays (if replicated tables are used)

Spark Monitoring

- Batch job success/failure
- Runtime duration spikes
- Data volume processed
- Backfill job integrity

Alerting Conditions

Alerts trigger when:

- Kafka consumer lag exceeds threshold
- No new events (ingestion stall)
- Missing partitions in aggregated tables
- Sudden drop in event volume
- ClickHouse merge backlog grows abnormally

These alerts provide real-time visibility into ingestion issues and downstream failures.

Backfill & Recovery (Spark)

Spark provides a robust mechanism for correcting data or recomputing partitions.

Backfill Workflow

1. **Detect corrupted or incomplete partitions**
    - Missing events
    - Incorrect aggregates
    - Duplicate purchase rows
2. **Drop affected partitions in ClickHouse**
   Ensures a clean state.
3. **Recompute using Spark**

When Backfills Are Used

- Debezium CDC outage
- Kafka topic corruption or replay
- Incorrect materialized view logic requiring historical recalculation
- Business rule changes (pricing, classification, funnel logic)

The backfill capability guarantees long-term accuracy of analytical data.