Azki DE Task Technical Report

This document presents a detailed technical review of the azki-de project, with a particular emphasis on data quality, validation, and architectural decisions. The review is based on a comprehensive inspection of the repository, including Python ingestion logic, ClickHouse schema definitions, Spark batch transformations, Docker infrastructure, and the data quality checks embedded across the pipeline.

The most compelling aspect of this project is its explicit prioritisation of data validation. Unlike many similar projects where data quality is often an afterthought, I treated it as a first-class concern from the outset. This design philosophy ensures that the pipeline enforces schema discipline, integrity checks, and type safety before data is consumed downstream, thereby preventing the propagation of errors and making the system more robust.

Certain production-grade components such as CDC, continuous integration/deployment pipelines, monitoring, and retry mechanisms for Kafka consumers are intentionally not included. These omissions were conscious trade-offs due to time constraints and scope considerations. While they limit the pipeline's readiness for direct production deployment, they do not detract from the project's core value: demonstrating deliberate, thoughtful decision-making in data engineering, architectural design, and validation strategies.

1. Technologies Used

- **Python** – Core scripting, ingestion logic, Spark interactions
- **Apache Kafka** – Event streaming and decoupling of producers/consumers
- **ClickHouse** – Analytical storage, materialised views, and denormalisation
- **Apache Spark** – Batch transformations and large-scale data processing
- **Docker & Docker Compose** – Local infrastructure orchestration
- **SQL** – Table creation, schema design, and analytical queries
- **Git** – Version control and source management

2. Component Breakdown

Component	Purpose
`ingestion/`	Kafka producers and consumers for event-driven pipelines

`quality/`	Dedicated data validation and integrity checks
`clickhouse/`	Table DDLs, materialised views, access control, partitioning logic
`spark/`	Batch transformations and denormalisation routines
`utils/`	Shared helpers for connections, environment configuration, and reusable functions
`scripts/`	Seeding scripts and operational utilities
`docker/`	Local infrastructure setup with Docker Compose

3. Data Quality & Validation

3.1 Explicit Quality Layer

Data quality in this project is not embedded as ad-hoc logic within ingestion pipelines. Instead, validation checks are implemented as a separate, explicit layer, independent of transport or storage. This separation provides several benefits:

- **Maintainability:** Clear boundaries between ingestion and validation make the pipeline easier to reason about.
- **Evolvability:** Quality rules can be extended or modified without impacting the core ingestion logic.
- **Reliability:** Errors and anomalies are caught early, reducing the risk of corrupting downstream analytical tables.

3.2 Validation Philosophy

The validation strategy is guided by the following principles:

- **Schema conformity** – Ensures data matches the expected table structures.
- **Field requirements** – Differentiates between required and optional fields.
- **Type safety** – Prevents type mismatches that could lead to runtime errors.

- **Null handling** – Ensures missing values are either defaulted or flagged appropriately.

These checks are particularly important given the combination of ClickHouse's minimal enforcement at ingestion and the event-driven nature of Kafka pipelines, where bad data can propagate quickly.

4. ClickHouse Design

4.1 Schema Design

ClickHouse schemas follow best practices for analytical workloads:

- **Time-based partitioning** – Facilitates efficient pruning of historical data.
- **Explicit ordering keys** – Optimises query performance for common access patterns.
- **Materialised views for denormalisation** – Provides pre-aggregated, read-optimised tables for analytical queries.

4.2 Analytical Readiness

The design differentiates raw ingestion tables from analytical tables:

- Raw tables store unmodified event data, enabling reprocessing and auditing.
- Denormalised views provide query-ready structures for business intelligence and analytics.

This clear separation ensures the system can handle high-volume ingestion while remaining performant and reliable for analytical queries.

5. Kafka & Ingestion Logic

5.1 Design Characteristics

Kafka is leveraged as a decoupling layer between event producers and downstream consumers. The consumer implementation is explicit and deterministic, with clearly defined ingestion steps and minimal side effects. This simplicity helps ensure reliability and makes debugging more straightforward.

5.2 Limitations

The current ingestion setup does not include:

- Retry and exponential backoff mechanisms
- Dead-letter queues for failed messages
- Documented offset management or checkpointing strategies

These omissions are intentional and out-of-scope for this project. In a production environment, these components would be essential to ensure fault tolerance and resilience.

6. CDC: Not Implemented

6.1 Impact

CDC:

- Incremental updates from OLTP systems are not captured.
- Late-arriving updates or corrections do not propagate downstream.
- The pipeline is therefore primarily suitable for append-only or event-style datasets.

6.2 Scope Considerations

The absence of CDC is documented and intentional, reflecting the project’s scope and time constraints. For production systems handling transactional data, CDC would be necessary to maintain consistency and support updates or deletes efficiently.

---

7. Observations and Reflections

- **Trade-offs are explicit:** Every design decision such as avoiding CDC, retry logic, or CI/CD pipeline is documented and justified.
- **Focus on correctness over completeness:** The project prioritises data quality and architectural clarity over production readiness.
- **Reusability and extensibility:** Components like validation logic and ClickHouse schema design are modular and can be extended with minimal friction.