# I. Architecture Overview

## A. Data Storage

Dataset A is known to be a rather large dataset and hence, due to the streaming nature of this dataset, the storage solution needs to be one that is capable of handling high-throughput and low-latency writes.

**Solution: Apache Kafka**

Used to build real-time streaming data pipelines and real-time streaming applications. Suitable for large scale data processing applications due to being more robust, reliable and fault tolerant.

Dataset B is a static dataset of a smaller size than Dataset A, where it is used as a reference table. Hence, it can be stored in a distributed database for fast read access.

**Solution: Apache Cassandra**

An open-source NoSQL data storage system that leverages a distributed architecture to enable high availability and reliability, managed by the Apache non-profit organization. It offers low-latency read and write operations, making it suitable for real-time applications where low latency is essential. This ensures that dashboard visualization tools can access join results quickly, providing a responsive user experience. It is designed to be highly scalable, allowing for seamless horizontal scaling as data volume grows. This scalability is essential for accommodating the high volume of events generated by Dataset A. Furthermore, Cassandra can be easily integrated with Apache Kafka, enabling seamless data pipelines from Kafka topics to Cassandra tables. This integration simplifies data ingestion and processing workflows, facilitating the real-time processing and visualization of join results.

## B. Data Processing

When joining Dataset A and B, due to having the join results of the dataset being available upon publishing Dataset A, we would require a processing framework that is capable of real-time stream processing and efficient batch processing for Dataset B to achieve this.

**Solution: Apache Flink**

Apache Flink is a framework and distributed processing engine for stateful computations over unbounded and bounded data streams. It offers real-time processing, a llowing it to process incoming events from Dataset A in real-time. This capability ensures that join operations with Dataset B can be performed promptly as

events are published, meeting the requirement for timely availability of join results. known for its high performance and low-latency stream processing capabilities. This allows it to efficiently process large volumes of streaming data with minimal processing delays, enabling timely generation of join results for visualization on the dashboard. It offers seamless integration with Apache Kafka, allowing it to consume data from Kafka topics directly. This simplifies the data ingestion process and facilitates the integration of Dataset A with the stream processing pipeline, enabling end-to-end real-time data processing.

## C. Real-time Processing

For this section, we would require a solution that can allow us to perform real-time processing and join operations on streams of data.

**Solution: Apache Kafka Streams**

Kafka Streams is a client library for building applications and microservices, where the input and output data are stored in Kafka clusters. Kafka Streams pairs the ease of utilizing standard Java and Scala application code on the client end with the strength of Kafka's robust server-side cluster architecture. Kafka Streams is a natural fit for environments where Kafka is already being used as the messaging system, which is the case here. Kafka Streams is chosen for its seamless integration with Kafka, exactly-once semantics, stateful stream processing capabilities, scalability, fault tolerance, developer-friendly APIs, and low latency, making it well-suited for building real-time stream processing applications like the one described in the architecture.

## D. Duplicate Handling

To handle the duplicate events that occur in Dataset A, we could rely n Kafka to implement deduplication mechanisms. This is due to the exactly-once processing semantics offered by Kafka Streams, ensuring that each event is processed and delivered exactly once, even in the presence of failures or retries. This guarantees data consistency and eliminates the risk of duplicate events, which is crucial for maintaining the integrity of the joined dataset.

## E. Dashboard Visualisation

To build a dashboard, we could make use of visualization tools such as **Apache Superset or Tableau** as these tools can connect to the data sources stored in the tools mentioned above. These tools have a high visualization functionality and is known to have a robust and reliable overall performance. They operate efficiently on big data and will be suitable for large datasets like Dataset A. These tools can be

utilized efficiently for real-time data analysis, which would be useful in joining and displaying the results quickly on the publishing of Dataset A.

## II. Considerations and Questions

This section outlines the considerations and questions that should be taken to help ensure that the architecture is tailored to the specific needs and constraints of the project, leading to a well-designed, efficient, and secure system that meets the end-users' requirements effectively.

### 1. Data Volume and Velocity

Understanding the volume and velocity of data streams is crucial for sizing Kafka and Cassandra clusters appropriately. Oversized clusters can lead to unnecessary infrastructure costs, while undersized clusters may result in performance bottlenecks or data loss. By knowing the volume and velocity of data streams, the architecture can be designed to handle the expected load efficiently, ensuring scalability and optimal resource utilization.

### 2. Data Retention

Determining the retention policy for Kafka topics and data in Cassandra is essential for managing storage costs and ensuring compliance with data retention regulations. Different types of data may have varying retention requirements based on business needs or regulatory requirements. Establishing a clear retention policy helps in optimizing storage usage, managing data lifecycle, and ensuring data availability when needed.

### 3. Schema Evolution

Handling changes in schema over time is critical for maintaining compatibility and data integrity as Dataset A and B evolve. Without proper schema evolution strategies, changes in data schema could lead to compatibility issues, data corruption, or processing errors. By understanding how the schema may evolve over time, the architecture can incorporate flexibility and resilience to accommodate schema changes seamlessly without disrupting data processing or visualization.

### 4. Security

Ensuring data security and access control mechanisms are in place is essential to protect sensitive information and prevent unauthorized access or data breaches. Questions related to security help identify potential vulnerabilities, establish

authentication and authorization mechanisms, encrypt data in transit and at rest, and implement auditing and monitoring capabilities. By addressing security concerns proactively, the architecture can mitigate security risks and safeguard the confidentiality, integrity, and availability of data.

## 5. Dashboard Requirements

Understanding the specific visualization requirements for the dashboard ensures that the visualization tools chosen meet the end-users' needs effectively. Questions about dashboard requirements help identify the types of visualizations needed, the level of interactivity required, the frequency of updates, and the performance expectations. By aligning the architecture with the dashboard requirements, it becomes possible to design a dashboard that provides actionable insights, enhances user experience, and supports decision-making processes effectively.