**POC - 1:**

**STREAMING API DATA using KAFKA**

**Introduction:**

The goal of this Proof of Concept (PoC) is to demonstrate the end-to-end process of collecting, processing, and analyzing financial data from Yahoo Finance, streaming it to a Kafka topic, and finally storing it in Delta tables for further analysis. The PoC utilizes PySpark for data processing, Kafka for real-time data streaming, and Databricks for visualization.

**Workflow :**

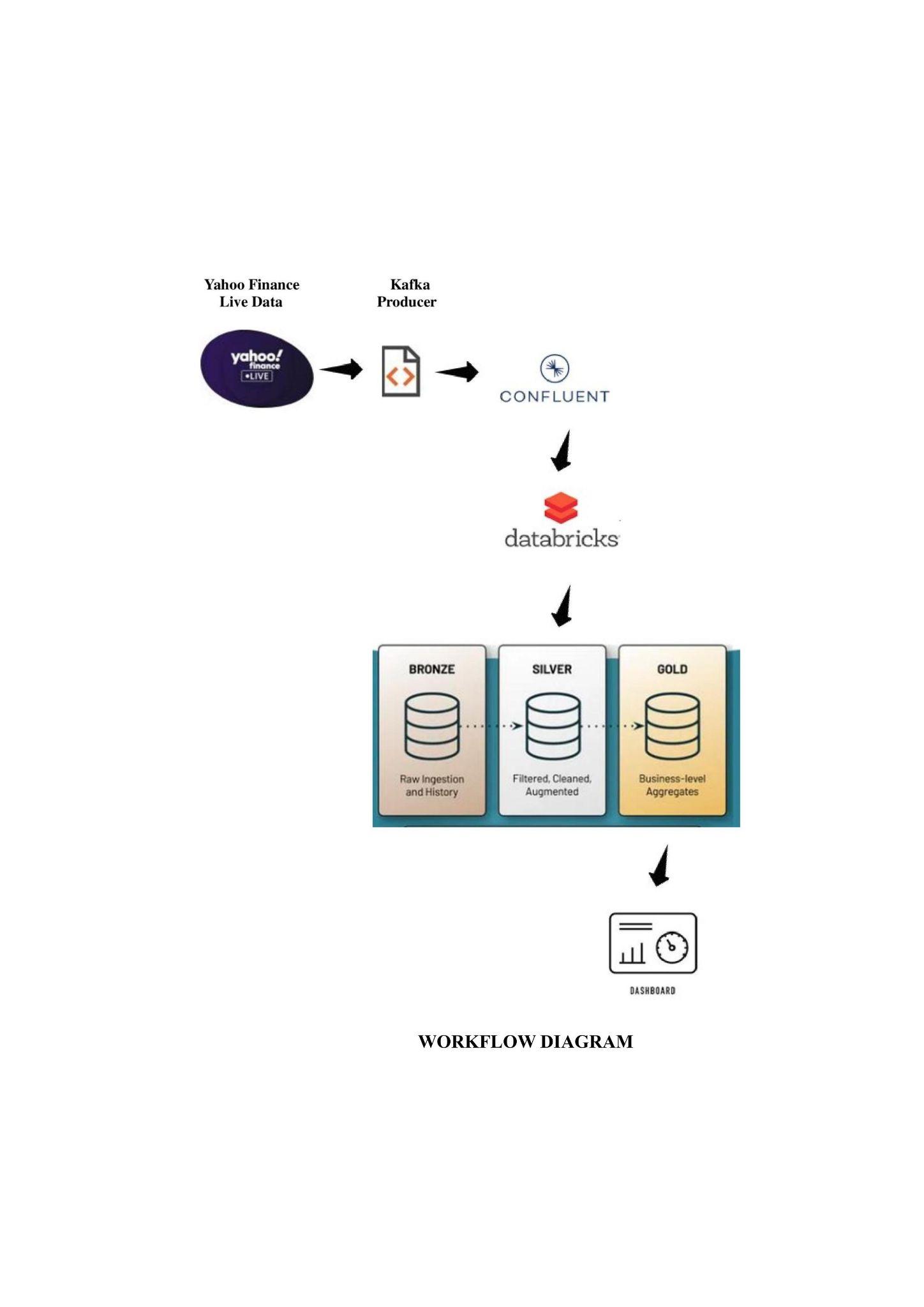
1.Web Scraping and Kafka Producer (Bronze Table)

2.Streaming and Storing in Bronze Delta Table

3.Transforming and Storing in Silver Delta Table

4.Windowed Aggregation and Storing in Gold Delta Table

**Workflow Diagram:**



**1.Web Scraping and Kafka Producer (Bronze Table)**

web scraping techniques like selenium with Kafka messaging to periodically fetch and publish financial data for a set of stocks to a Kafka topic. The Kafka topic can then be consumed by databricks for further processing or analysis.

**A. Web Scraping Function (`scraped\_data\_india`):**

1.A function that takes a stock symbol (e.g., "SBIN.NS") as input and uses Selenium with Firefox WebDriver to navigate to the Yahoo Finance page of the specified stock.

2.Extracts information such as the stock name, Indian market price, change, and change percentage from the corresponding elements on the webpage.

3.Processes and converts the extracted data into a dictionary (`result\_data`), including the stock name, Indian market price, and market details.

**B.Kafka Configuration:**

1. Configuration settings for the Kafka producer, including bootstrap servers, security protocol, SASL mechanisms, and credentials.

**C. Kafka Producer Initialization:**

1. Initializes a Kafka producer (`Producer`) with the provided Kafka configuration.

**D.Generating messages:**

1.We generating and sending messages containing stock data to a Kafka topic.

2.The process involves iterating through different stock URLs, scraping relevant data, and transmitting it as messages. Additionally, print statements aid in tracking the progress and content of the generated messages.

**2.Streaming and Storing in Bronze Delta Table:**

We reads data from a Kafka topic, processes it, displays the streaming DataFrame, and writes the data to a Delta table, allowing SQL queries to be performed on the stored data.

**A.Importing Spark Libraries:**

We imports necessary functions and types from the PySpark library for working with Spark DataFrames.

**B. Setting Widgets:**

Widgets are set up using `dbutils.widgets.text` and `dbutils.widgets.dropdown` to take input parameters like API key, API secret, Kafka bootstrap server, checkpoint location, target table, and target database.

**C. Constructing Kafka JAAS Configuration:**

It creates a Kafka JAAS (Java Authentication and Authorization Service) configuration string using the provided API key and API secret.

**D. Reading from Kafka:**

1.Utilizes the `spark.readStream.format("kafka")` API to create a streaming DataFrame (`bronze\_DF`) by reading from a Kafka topic ("topic\_2").

2. Specifies Kafka configurations like bootstrap servers, security protocol, JAAS configuration, SSL settings, starting offsets, and subscription to the Kafka topic.

**E. Transforming DataFrame:**

1. It selects the "value" column, casts it as a string, and renames the "timestamp" column to "confluent\_timestamp".

**F. Displaying the DataFrame:**

1.It uses the `display` function to show the streaming DataFrame `bronze\_DF` in the Databricks notebook.

**G. Writing to Delta Table:**

1.It writes the streaming DataFrame to a Delta table (`path`) using `bronze\_DF.writeStream.format("delta")`.

2.It specifies output mode as "append" and provides the checkpoint location for fault-tolerance.

**H. SQL Query:**

It executes a SQL query using `%sql` to display the contents of the Delta table (`default.bronzeindia`).

**3.Transforming and Storing in Silver Delta Table :**

It reads streaming data from a Delta table, performs transformations on the data, displays the results, and writes the transformed data to another Delta table. The final SQL query showcases the contents of the target Delta table.

**A. Widget Inputs**

It uses Databricks widgets to take user inputs for source and target tables, source and target databases, and checkpoint location.

**B.Constructing Paths:**

It constructs source and target paths using the specified source and target databases and tables.

**C. Reading from Delta Table:**

It reads streaming data from a Delta table (`source\_path`) using `spark.readStream.format("delta").table(source\_path)`.

**D.Defining Schema and Parsing JSON:**

It defines a schema (`df\_schema`) for the streaming data and parses the JSON data using `from\_json` and `regexp\_extract` functions.

**E. Transforming DataFrame:**

It selects specific columns from the parsed DataFrame and performs operations like extracting company names and symbols.

**F. Exploding and Renaming Columns:**

It uses the `explode` function to split the array column 'market\_details' and renames the resulting columns appropriately.

**G. Adding Date and Time Columns:**

It derives new columns 'Date' and 'Time' from the 'confluent\_timestamp' column using the `date\_format` function.

**H. Displaying the DataFrame:**

It displays the transformed DataFrame (`silver`) in the Databricks notebook using the `display` function.

**I. Writing to Delta Table:**

It writes the transformed streaming DataFrame (`silver`) to a Delta table (`target\_path`) with options like output mode, checkpoint location, and table path.

**J. SQL Query:**

It executes a SQL query using `%sql` to display the contents of the Delta table (`default.silverindia`).

**4.Windowed Aggregation and Storing in Gold Delta Table:**

It demonstrates windowed aggregation on stock market data, writes the aggregated data to a Delta table, visualizes the results through SQL queries, and provides separate dashboards for individual stock symbols.

**A. Displaying Global DataFrame:**

It retrieves and displays the contents of the global temporary view `gblsilver\_india` using the `spark.sql` query.

**B. Windowed Aggregation:**

1. It performs windowed aggregation on the `gblsilver\_india` DataFrame, grouping by 'Symbol' and a 1-minute window on the 'confluent\_timestamp'.

2. Computes the maximum value of 'Indian\_Market\_Price' within each window.

**C. Writing Windowed Aggregation to Delta Table:**

1.It writes the windowed aggregation DataFrame (`streamingCountsDF`) to a Delta table named `default.window\_Aggregation\_table\_india`.

2. It specifies the output mode as "complete" and sets the checkpoint location for fault tolerance.

**D. Displaying Final Windowed Aggregation DataFrame:**

It displays the final streaming DataFrame (`final\_streamingCountsDF`) resulting from the windowed aggregation in the Databricks notebook.

**E. SQL Queries for Visualization:**

1. It executes SQL queries using `%sql` to visualize the aggregated data.

2. The first query selects and orders the maximum prices for each symbol in 1-minute windows.

3. The second query further aggregates the data by counting symbols and ordering the results by time and maximum price.

**F.Individual Stock Dashboards:**

It filters the global DataFrame (`globaldf`) for specific stock symbols (e.g., SBIN.NS, HDFCBANK.NS) and displays dashboards for each individual stock using the `display` function.

**Conclusion:**

The Proof of Concept successfully demonstrates the end-to-end workflow for collecting real-time financial data, streaming it into Delta tables, performing transformations, aggregating the data, and storing the results. The use of web scraping, Kafka messaging, and Delta tables in PySpark provides a robust solution for handling and processing financial data in a scalable and fault-tolerant manner. This POC serves as a foundation for building a more comprehensive financial data processing pipeline.