**Final Group Project - ETL Pipeline with Spark Streaming**

**Overview**

This report outlines the implementation of a real-time ETL (Extract, Transform, Load) pipeline that processes news articles using Apache Kafka, Spark Streaming, and Hive. The system fetches data from the NewsAPI, processes it in real time, and stores it in Hive for further analysis. This project demonstrates the integration of modern big data tools to create an efficient and scalable data pipeline.

**Pipeline Overview**

The ETL pipeline is divided into three main stages:

Producer: Fetches news articles from NewsAPI and streams them into a Kafka topic.

First Spark Session: Processes the Kafka stream, structures the data, and saves it as JSON files.

Second Spark Session: Reads JSON files, cleans and selects relevant fields, and stores the data in Hive.

**Here First we are first setting and starting the Zookeeper:**

A screenshot of a computer

Description automatically generated

**Then we are starting and setting the Kafka:**

A screen shot of a computer code

Description automatically generated

**Then we are creating a new Kafka Topic and named it as “finalProject”:**

A black screen with many colorful lights

Description automatically generated with medium confidence

**Then we are setting the kafka Producer:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

The Producer is a Python script that collects articles from NewsAPI and sends them to a Kafka topic. It uses an API key to connect to NewsAPI and fetches articles related to the keyword "technology." The articles are organized in JSON format to match the pipeline structure. They are then streamed to the Kafka topic named *finalProject* on the server *localhost:9092*. A small delay is added between sending each article to ensure smooth processing.

**Then we are just simply starting the Producer:**

A screenshot of a computer

Description automatically generated

**First Spark session**

**After that we creating a new session in which we are starting a spark:**

The first Spark session processes real-time data from Kafka and saves it as JSON files for later use. It defines a schema with fields like author, title, content, description, publishedAt, source details, URL, and image URL. The session reads data from the Kafka topic *finalProject*, converts the binary data to JSON using the schema, and stores it in the *raw\_json* directory. A checkpoint directory ensures progress tracking and fault tolerance.

A screenshot of a computer

Description automatically generated

**Getting Message:**

A computer screen with white text

Description automatically generated

**Parse Kafka data as JSON:**

A computer screen with white text

Description automatically generated

**Write raw data to JSON Sink:**

A black screen with white text

Description automatically generated

**Second Sprak Session**

This session reads the JSON files created in the first session, cleans the data, and saves it in Hive. It reads the files from the *raw\_json* directory using the same schema. Relevant fields like author, content, description, publishedAt, source details, title, URL, and image URL are selected. Field names are standardized, such as renaming *source.id* to *source\_id*.

The cleaned data is saved in a Hive table called *news\_articles* within the *news\_database.db* database. It is stored in Parquet format to enable efficient storage and querying. A checkpoint system is used to prevent data duplication or loss.

**Define a Schema for New Article:**

A screenshot of a computer

Description automatically generated

**Creating a Hive Environment:**

A computer screen shot of a computer program

Description automatically generated

**Read a JSON Data which was created by first session:**

A screenshot of a computer program

Description automatically generated

**Selecting Importance Column:**

A screenshot of a computer

Description automatically generated

**Writing Data to Hive:**

A screenshot of a computer program

Description automatically generated

**Starting HIVE for Further Analysis:**

A computer screen with white text

Description automatically generated

**Using Default Database:**

A black background with white text

Description automatically generated

**Creating Table:**

A computer screen with white text

Description automatically generated

**Finally at the end we are running queries to do some basic analysis on the live data.**

**1 .Retrieving first 5 rows of table Using SELECT query:**

A screenshot of a computer

Description automatically generated

**2 .Get the Articles with the Most Popular Keywords in the Description:**

SELECT description, COUNT(\*) AS articles\_with\_keyword

FROM news1

WHERE description LIKE '%Tesla%'

GROUP BY description

ORDER BY articles\_with\_keyword DESC

LIMIT 5;

A screenshot of a computer

Description automatically generated

**3. Count the Number of Articles:**

SELECT COUNT(\*) AS total\_articles FROM news1;

A screenshot of a computer program

Description automatically generated

**In the example above, I ran the same query twice to demonstrate that data is being updated in real time. The query SELECT COUNT(\*) AS total\_articles FROM news1; counts the total number of articles in the table. On the first run, the result was 764, and on the second run, it increased to 779. This shows that new articles are being added to the table continuously, confirming that the data is live.**

4. **Total Articles with the Most Popular Authors:**

SELECT author, COUNT(\*) AS articles\_by\_author

FROM news1

GROUP BY author

ORDER BY articles\_by\_author DESC

LIMIT 5;

A screenshot of a computer

Description automatically generated

**5. Longest Article Descriptions**

SELECT source\_name, description, LENGTH(descritption) AS description\_length

FROM news1

ORDER BY description\_length DESC

LIMIT 10;

A computer screen with text

Description automatically generated

**Conclusion**

This ETL pipeline successfully showcases:

* Seamless integration of Kafka, Spark Streaming, and Hive for real-time data processing.
* Efficient storage and management of large datasets using Parquet format and Hive.
* A fault-tolerant and scalable architecture capable of handling continuous data streams effectively.