

Big Data

Project

Bacha Wondimu

Samsom Micheal

Luwam Weldetensae

Yosief Gebrewold



Wholeness of Project

- ❖ In this project, we implemented a data pipeline by integrating **Kafka** with **Spark Streaming** to handle real-time data ingestion. Data from Kafka topics is processed in Spark Streaming and further analyzed using **Spark SQL** for advanced querying and transformation.
- ❖ The processed data is stored in **Hive**, a scalable and fault-tolerant data warehouse system, to enable efficient storage and query execution for large datasets.
- ❖ Finally, **Tableau** is used to visualize the results, providing dynamic charts and dashboards that offer insights into the processed data, facilitating better decision-making.

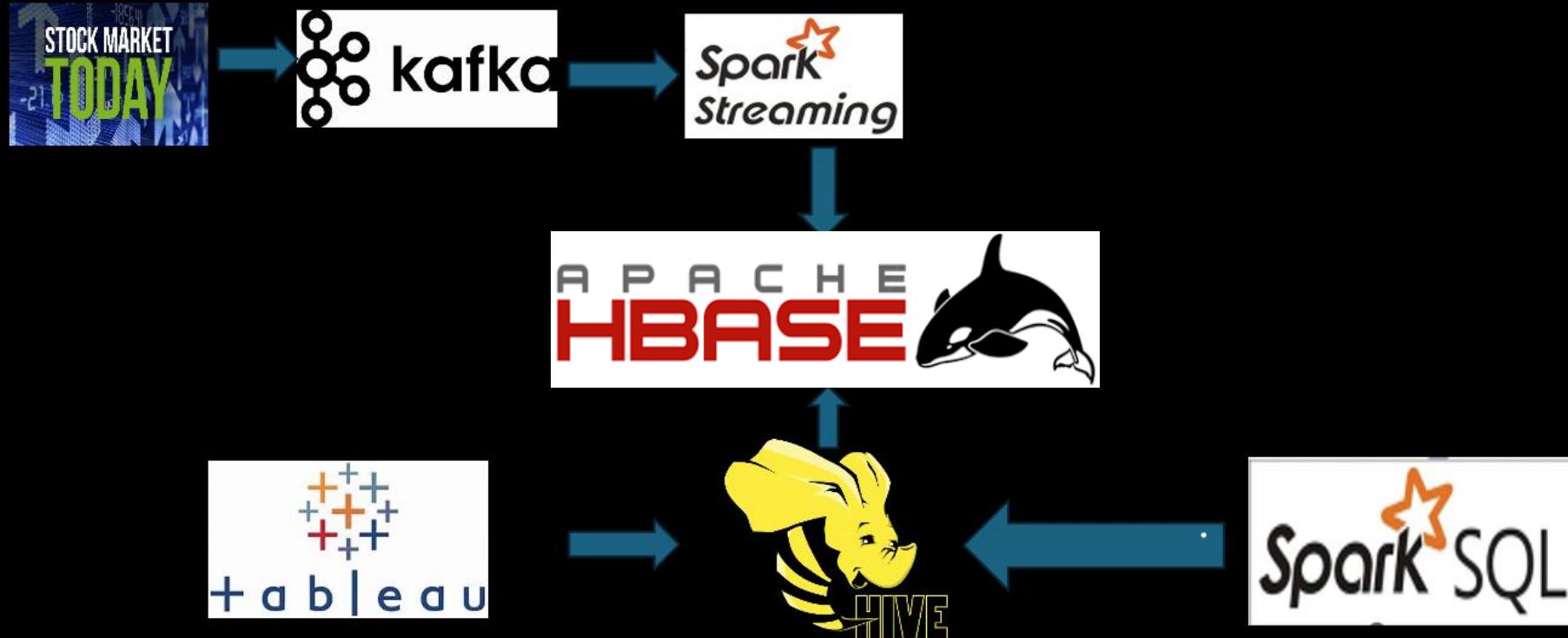
Wholeness of Project

- ❖ Exploring the Science and Technology of Consciousness:
- ❖ This project also explores the deeper aspects of knowledge and awareness, recognizing the layered nature of creation. In parallel with the technical work, we engage in **Transcendental Meditation** as a means to uncover the unified field of consciousness.
- ❖ Through regular practice, we find that the clarity and focus gained through meditation help enhance our problem-solving abilities, creativity, and overall approach to technology. This conscious alignment supports our efforts in navigating the complexities of our technical work, creating solutions that are not only efficient but also aligned with a deeper understanding of interconnectedness.

Tools used for development

- ❖ **Stock Market Alpha Vantage API:** Used for fetching real-time stock market data.
- ❖ **Kafka (Stream Processing):** Enables real-time data ingestion and processing from various data sources.
- ❖ **Spark Streaming:** Processes and analyzes streaming data in real-time.
- ❖ **Spark SQL:** Performs SQL queries and data transformations on the streaming data.
- ❖ **HBase:** A distributed, scalable, and NoSQL database used for storing large volumes of unstructured or semi-structured data in real-time.
- ❖ **Hive:** A data warehouse system for querying and managing large datasets.
- ❖ **Tableau:** Provides data visualization to generate meaningful insights from the processed data.

End-to-End Data Streaming pipeline.





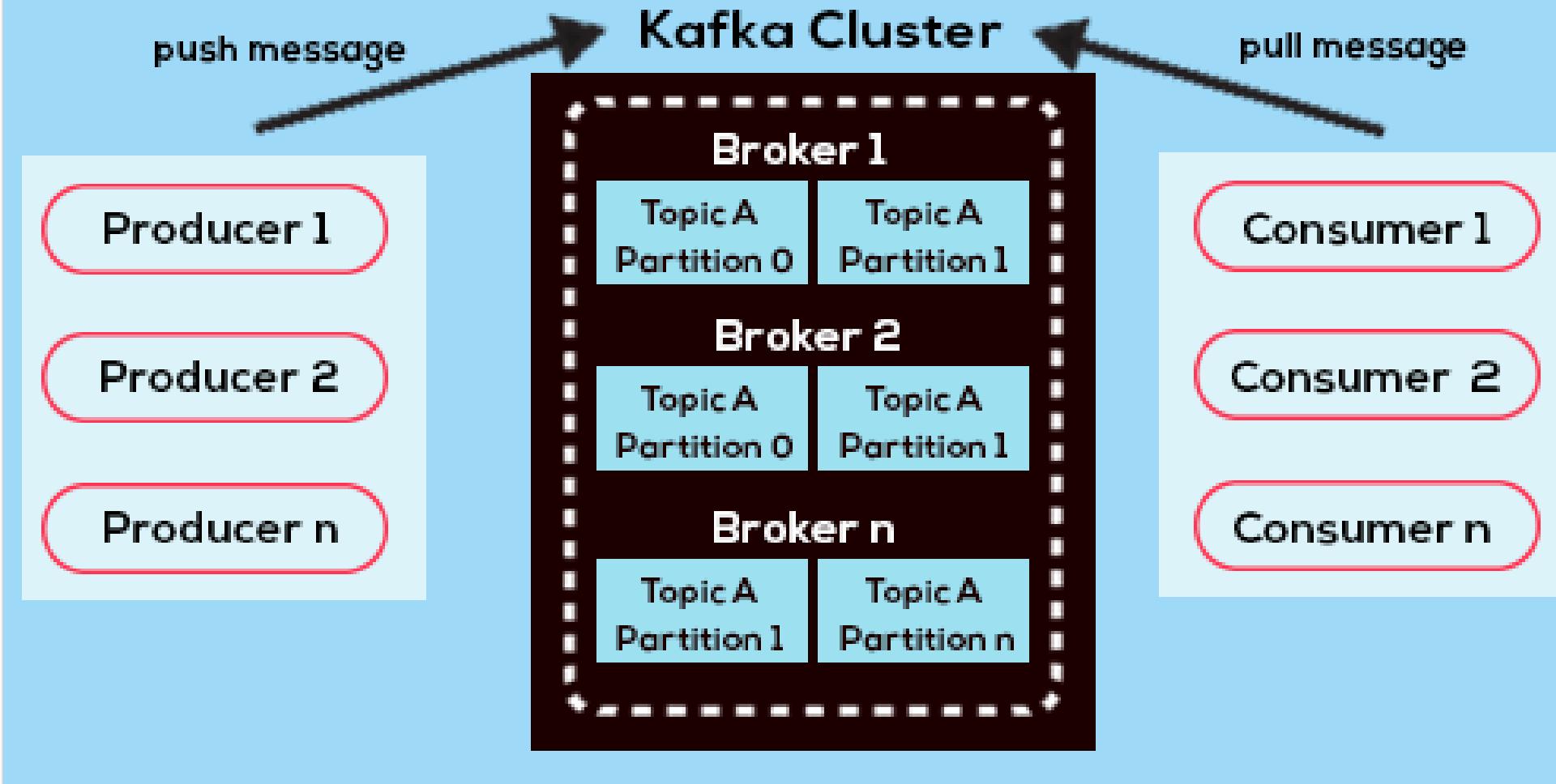
What is Apache Kafka?

- ❖ Apache Kafka is a **distributed event streaming platform**. This means it is used to send and receive large amounts of data across distributed systems in real time. Kafka allows applications to produce and consume messages in a highly scalable, fault-tolerant manner.

Kafka Architecture

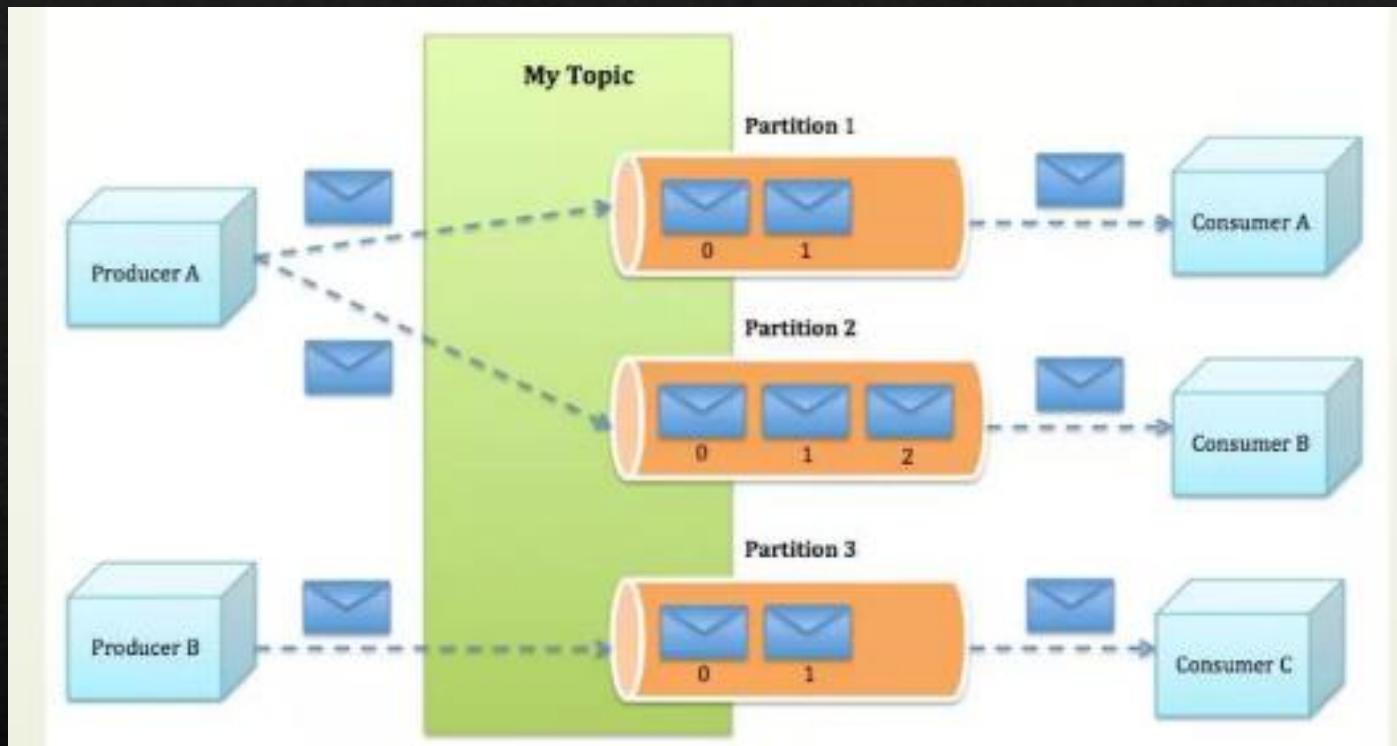
- ❖ **Producer:** Sends messages to Kafka.
- ❖ **Broker:** Kafka server that stores and manages messages.
- ❖ **Consumer:** Reads messages from Kafka.
- ❖ **Topic:** Logical channel to which producers send messages and consumers read from.
- ❖ **Partition:** Kafka topics are divided into partitions for scalability.
- ❖ **Replication:** Ensures message durability and fault tolerance

Apache Kafka Architecture



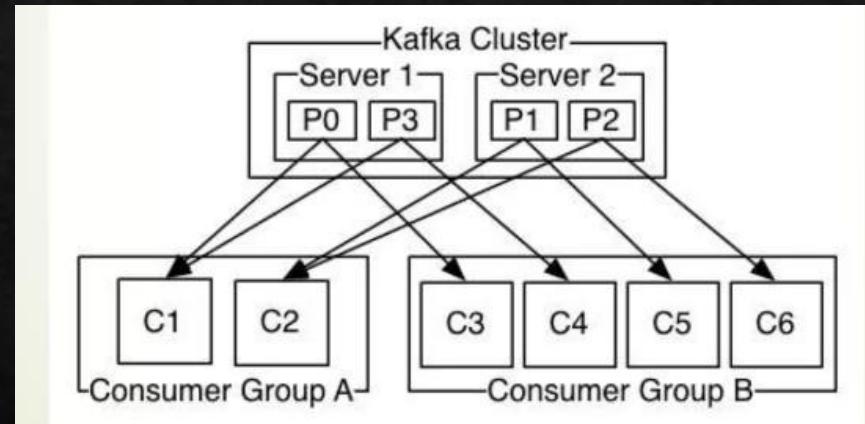
Producers

- ❖ Producers are responsible for publishing data to Kafka topics by **assigning messages to partitions** within a topic.
- ❖ Messages can be assigned to partitions either in a **round-robin fashion** to distribute the load evenly or based on a **semantic partition function** (such as using a specific key in the message to direct it to a particular partition).



Consumer Groups

- ❖ Topics in Kafka are typically subscribed to by a small number of **consumer groups**, with one group for each **logical subscriber**.
- ❖ Each consumer group consists of multiple consumer instances, allowing for **scalability** and **fault tolerance** within the system.



Retention

- ❖ The **Kafka cluster** retains all published messages, whether or not they have been consumed, for a **configurable period of time**. After this time, messages are discarded to free up space.
- ❖ Metadata is retained on a **per-consumer** basis, tracking the consumer's position in the log, known as the **offset**. This offset is **controlled by the consumer**.
- ❖ Typically, a consumer advances its offset linearly as it reads messages. However, it has the flexibility to consume messages in any order.
- ❖ Kafka consumers can **come and go** without significantly impacting the cluster or other consumers, ensuring high availability and flexibility.

Ordering Guarantees

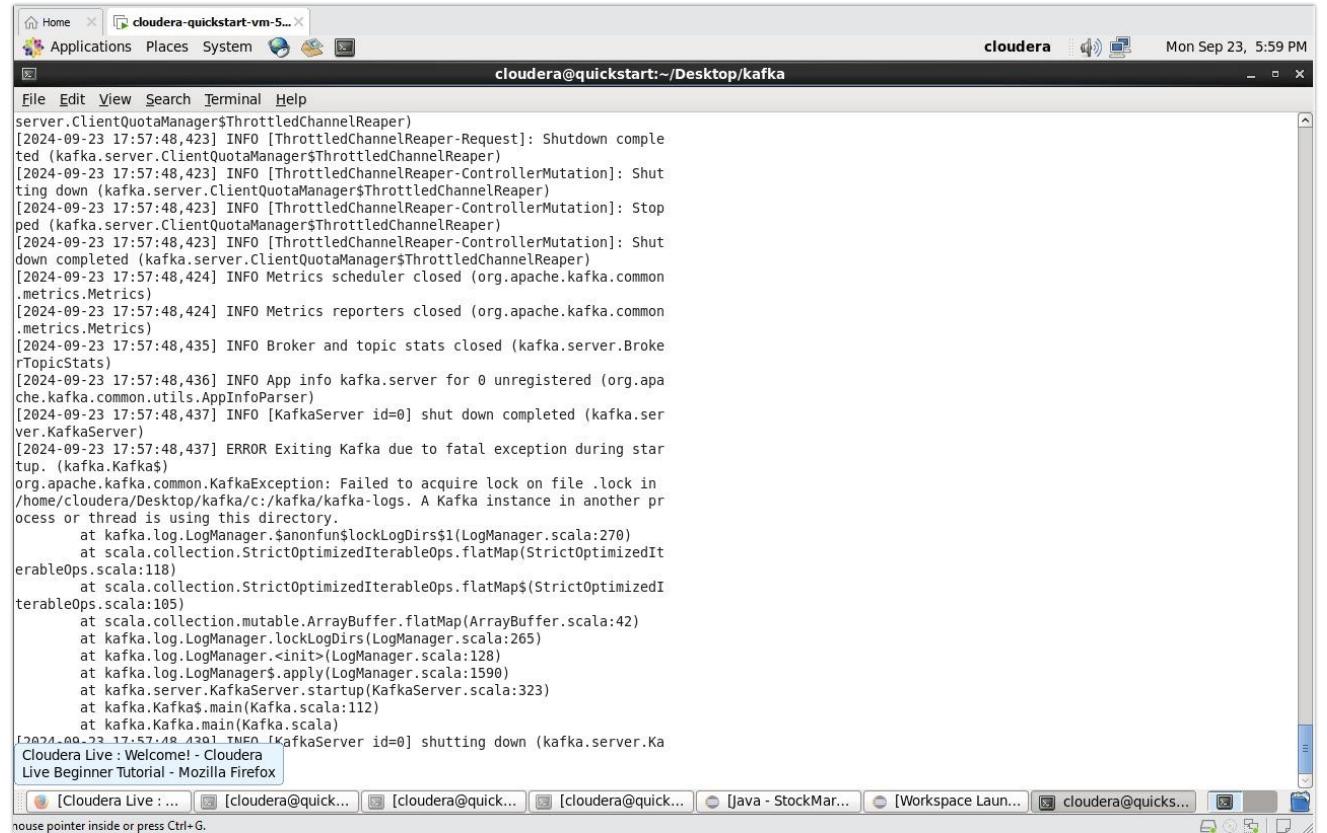
- ❖ Kafka assigns **partitions** in a topic to **consumers** in a consumer group. This ensures that each partition is consumed by exactly one consumer in the group.
- ❖ **Limitation:** There cannot be more consumer instances in a consumer group than there are partitions.
- ❖ Kafka provides a **total order** over messages **within a partition**, but not between different partitions in a topic.



Getting Practical



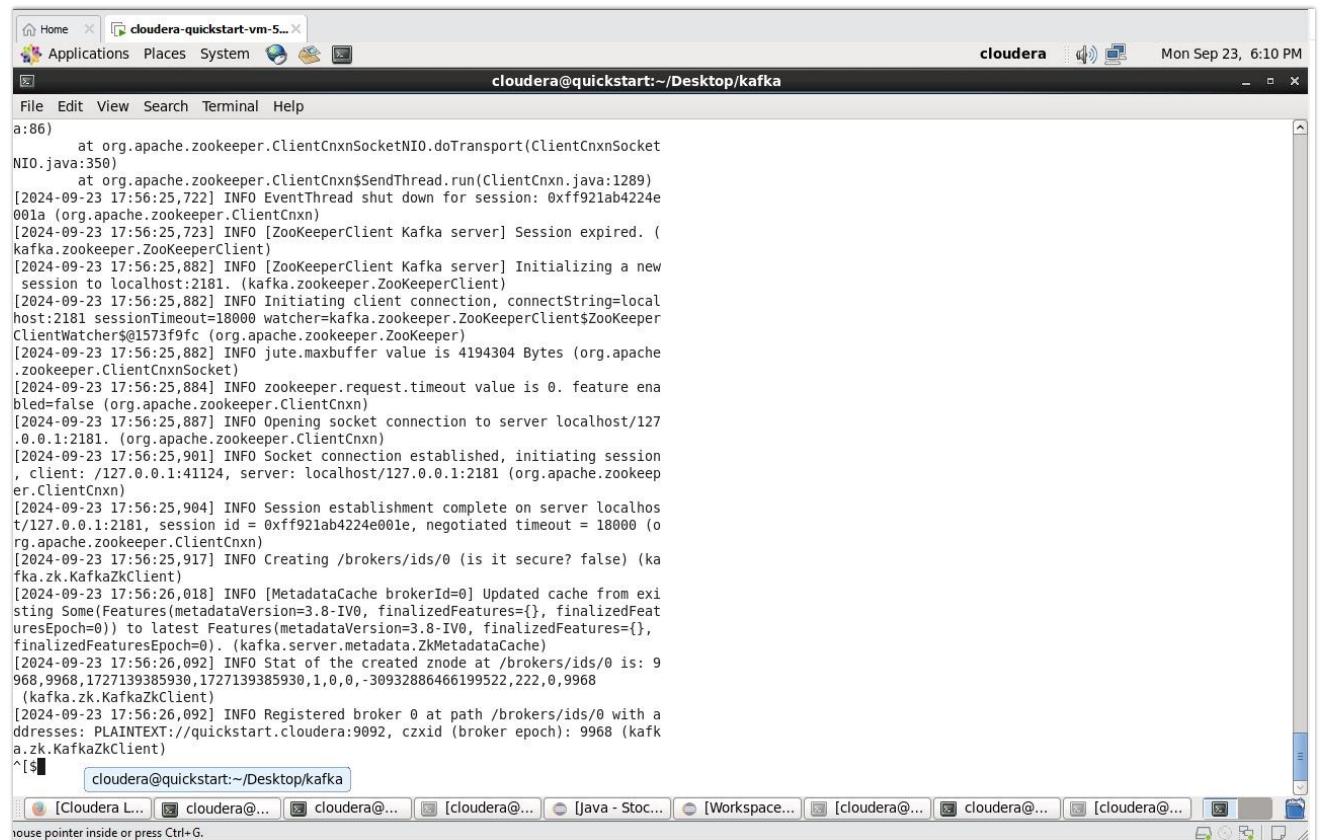
bin/kafka-server-start.sh
config/server.properties



The screenshot shows a Linux desktop environment with a terminal window open. The terminal title is "cloudera@quickstart:~/Desktop/kafka". The window contains a log of Kafka server startup errors. The log starts with shutdown messages for various reaper and mutation components, followed by metrics scheduler and reporter closure, and then a broker and topic stats closure. It ends with an error message about failing to acquire a lock on a file lock in /home/cloudera/Desktop/kafka/c:/kafka/kafka-logs, which is preventing Kafka from starting up. The error message also mentions that another process or thread is using this directory. The stack trace shows the code paths leading to this exception, including LogManager, IterableOps, and Kafka's main method.

```
server.ClientQuotaManager$ThrottledChannelReaper)
[2024-09-23 17:57:48,423] INFO [ThrottledChannelReaper-Request]: Shutdown completed (kafka.server.ClientQuotaManager$ThrottledChannelReaper)
[2024-09-23 17:57:48,423] INFO [ThrottledChannelReaper-ControllerMutation]: Shutting down (kafka.server.ClientQuotaManager$ThrottledChannelReaper)
[2024-09-23 17:57:48,423] INFO [ThrottledChannelReaper-ControllerMutation]: Stopped (kafka.server.ClientQuotaManager$ThrottledChannelReaper)
[2024-09-23 17:57:48,423] INFO [ThrottledChannelReaper-ControllerMutation]: Shutdown completed (kafka.server.ClientQuotaManager$ThrottledChannelReaper)
[2024-09-23 17:57:48,424] INFO Metrics scheduler closed (org.apache.kafka.common.metrics.Metrics)
[2024-09-23 17:57:48,424] INFO Metrics reporters closed (org.apache.kafka.common.metrics.Metrics)
[2024-09-23 17:57:48,435] INFO Broker and topic stats closed (kafka.server.BrokerTopicStats)
[2024-09-23 17:57:48,436] INFO App info kafka.server for 0 unregistered (org.apache.kafka.common.utils.AppInfoParser)
[2024-09-23 17:57:48,437] INFO [KafkaServer id=0] shut down completed (kafka.server.KafkaServer)
[2024-09-23 17:57:48,437] ERROR Exiting Kafka due to fatal exception during startup. (kafka.Kafka$)
org.apache.kafka.common.KafkaException: Failed to acquire lock on file .lock in /home/cloudera/Desktop/kafka/c:/kafka/kafka-logs. A Kafka instance in another process or thread is using this directory.
    at kafka.log.LogManager.$anonfun$lockLogDirs$1(LogManager.scala:270)
    at scala.collection.StrictOptimizedIterableOps.flatMap(StrictOptimizedIterableOps.scala:118)
    at scala.collection.StrictOptimizedIterableOps.flatMap$(StrictOptimizedIterableOps.scala:105)
    at scala.collection.mutable.ArrayBuffer.flatMap(ArrayBuffer.scala:42)
    at kafka.log.LogManager.lockLogDirs(LogManager.scala:265)
    at kafka.log.LogManager.<init>(LogManager.scala:128)
    at kafka.log.LogManager$.apply(LogManager.scala:1590)
    at kafka.server.KafkaServer.startup(KafkaServer.scala:323)
    at kafka.Kafka$.main(Kafka.scala:112)
    at kafka.Kafka.main(Kafka.scala)
[2024-09-23 17:57:48,437] INFO [KafkaServer id=0] shutting down (kafka.server.KafkaServer)
Cloudera Live : Welcome! - Cloudera Live Beginner Tutorial - Mozilla Firefox
```

bin/zookeeper-server-start.sh
config/zookeeper.properties



The screenshot shows a Linux desktop environment with a terminal window open. The terminal window title is "cloudera@quickstart:~/Desktop/kafka". The terminal content displays a log of Zookeeper server startup and configuration. The log includes messages about session expiration, client connections, socket openings, and broker registration. The log ends with a message about a registered broker at path /brokers/ids/0.

```
File Edit View Search Terminal Help
a:86)
        at org.apache.zookeeper.ClientCnxnSocketNIO.doTransport(ClientCnxnSocketNIO.java:350)
        at org.apache.zookeeper.ClientCnxn$SendThread.run(ClientCnxn.java:1289)
[2024-09-23 17:56:25,722] INFO EventThread shut down for session: 0xff921ab4224e001a (org.apache.zookeeper.ClientCnxn)
[2024-09-23 17:56:25,723] INFO [ZooKeeperClient Kafka server] Session expired. (kafka.zookeeper.ZooKeeperClient)
[2024-09-23 17:56:25,882] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2024-09-23 17:56:25,882] INFO Initiating client connection, connectString=local host:2181 sessionTimeout=18000 watcher=kafka.zookeeper.ZooKeeperClients$ZooKeeperClientWatcher@1573f9ff (org.apache.zookeeper.ZooKeeper)
[2024-09-23 17:56:25,882] INFO jute.maxbuffer value is 4194304 Bytes (org.apache.zookeeper.ClientCnxnSocket)
[2024-09-23 17:56:25,884] INFO zookeeper.request.timeout value is 0. feature enabled=false (org.apache.zookeeper.ClientCnxn)
[2024-09-23 17:56:25,887] INFO Opening socket connection to server localhost/127.0.0.1:2181. (org.apache.zookeeper.ClientCnxn)
[2024-09-23 17:56:25,901] INFO Socket connection established, initiating session, client: /127.0.0.1:41124, server: localhost/127.0.0.1:2181 (org.apache.zookeeper.ClientCnxn)
[2024-09-23 17:56:25,904] INFO Session establishment complete on server localhost/127.0.0.1:2181, session id = 0xff921ab4224e001e, negotiated timeout = 18000 (org.apache.zookeeper.ClientCnxn)
[2024-09-23 17:56:25,917] INFO Creating /brokers/ids/0 (is it secure? false) (kafka.zk.KafkaZkClient)
[2024-09-23 17:56:26,018] INFO [MetadataCache brokerId=0] Updated cache from existing Some(Features(metadataVersion=3.8-IV0, finalizedFeatures={}, finalizedFeaturesEpoch=0)) to latest Features(metadataVersion=3.8-IV0, finalizedFeatures={}, finalizedFeaturesEpoch=0). (kafka.server.metadata.ZKMetadataCache)
[2024-09-23 17:56:26,092] INFO Stat of the created znode at /brokers/ids/0 is: 9968,9968,1727139385930,1727139385930,1,0,0,-30932886466199522,222,0,9968 (kafka.zk.KafkaZkClient)
[2024-09-23 17:56:26,092] INFO Registered broker 0 at path /brokers/ids/0 with addresses: PLAINTEXT://quickstart.cloudera:9092, czxid (broker epoch): 9968 (kafka.zk.KafkaZkClient)
^[$ cloudera@quickstart:~/Desktop/kafka]
```

```
import org.apache.kafka.clients.producer.KafkaProducer;
public class Producer {
    private static final String TOPIC = "stock_market_data";
    private static final String API_KEY = "SOfw5hpGK0YP9b4aikgLpYDIyNjPInVj";
    private static final String KAFKA_BROKER = "localhost:9092";
    private static final String SYMBOL = "AAPL";
    public static void main(String[] args) {
        // Kafka properties
        Properties properties = new Properties();
        properties.put("bootstrap.servers", KAFKA_BROKER);
        properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
        // Kafka producer
        try (KafkaProducer<String, String> producer = new KafkaProducer<>(properties)) {
            CloseableHttpClient httpClient = HttpClientBuilder.create().build();
            ObjectMapper objectMapper = new ObjectMapper();
            while (true) {
                // Fetch stock data
```

```
                while (true) {
                    // Fetch stock data
                    String url = String.format("https://www.alphavantage.co/query?function=TIME_SERIES_DAILY");
                    HttpGet request = new HttpGet(url);
                    HttpResponse response = httpClient.execute(request);
                    try (BufferedReader reader = new BufferedReader(new InputStreamReader(response.getEntity().getContent()));
                         StringBuilder stringBuilder = new StringBuilder()) {
                        String line;
                        while ((line = reader.readLine()) != null) {
                            stringBuilder.append(line);
                        }
                        String stockData = stringBuilder.toString();
                        System.out.println("Fetched Stock Data: " + stockData);
                    }
                    // Send stock data to Kafka
                    ProducerRecord<String, String> record = new ProducerRecord<>(TOPIC, stockData);
                    producer.send(record);
                }
                // Wait 10 seconds before fetching again
                Thread.sleep(10000);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

What is Spark Streaming?



- ❖ Spark Streaming is an **extension** of Apache Spark that allows real-time data processing.
- ❖ It processes **live data streams** (e.g., from Kafka, Flume, or HDFS) and performs scalable and fault-tolerant stream processing.
- ❖ Used for **real-time analytics**, monitoring, and processing of continuous data flows.

Legacy Project

- ❖ Spark Streaming is a legacy project
- ❖ <https://spark.apache.org/docs/latest/streaming-programming-guide.html>
- ❖ You should use **Spark Structured Streaming** for your streaming applications and pipelines
- ❖ <https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html>

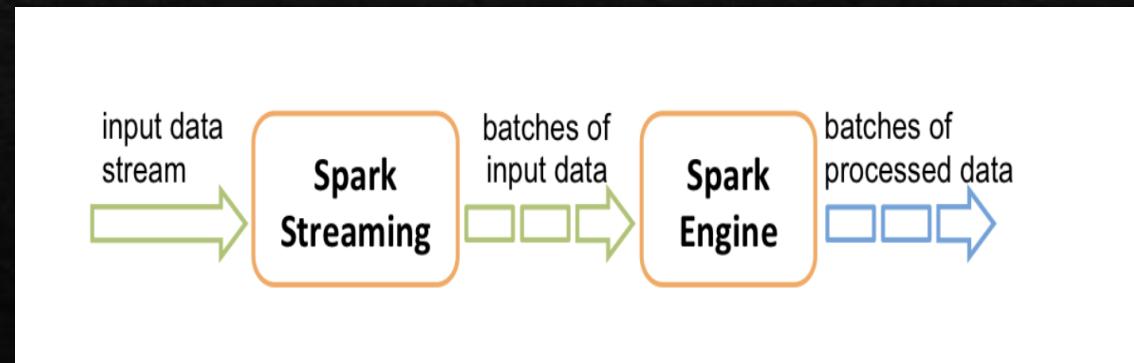


Why Spark Streaming?

- ❖ Key Benefits:
- ❖ Scalable, high-throughput, fault-tolerant stream processing of live data streams.
- ❖ Real-Time Processing: Allows analysis of data in real-time as it arrives.
- ❖ Scalable: Can process massive amounts of data with distributed systems.
- ❖ Fault-Tolerant: Automatically recovers from failures during processing.
- ❖ Integration: Easily integrates with other Apache Spark components (e.g., Spark SQL, MLLib).

How Spark Streaming Works

- ❖ Spark Streaming receives live input data streams and **divides the data into batches**, which are then processed by the Spark engine to generate the final stream of results in batches.

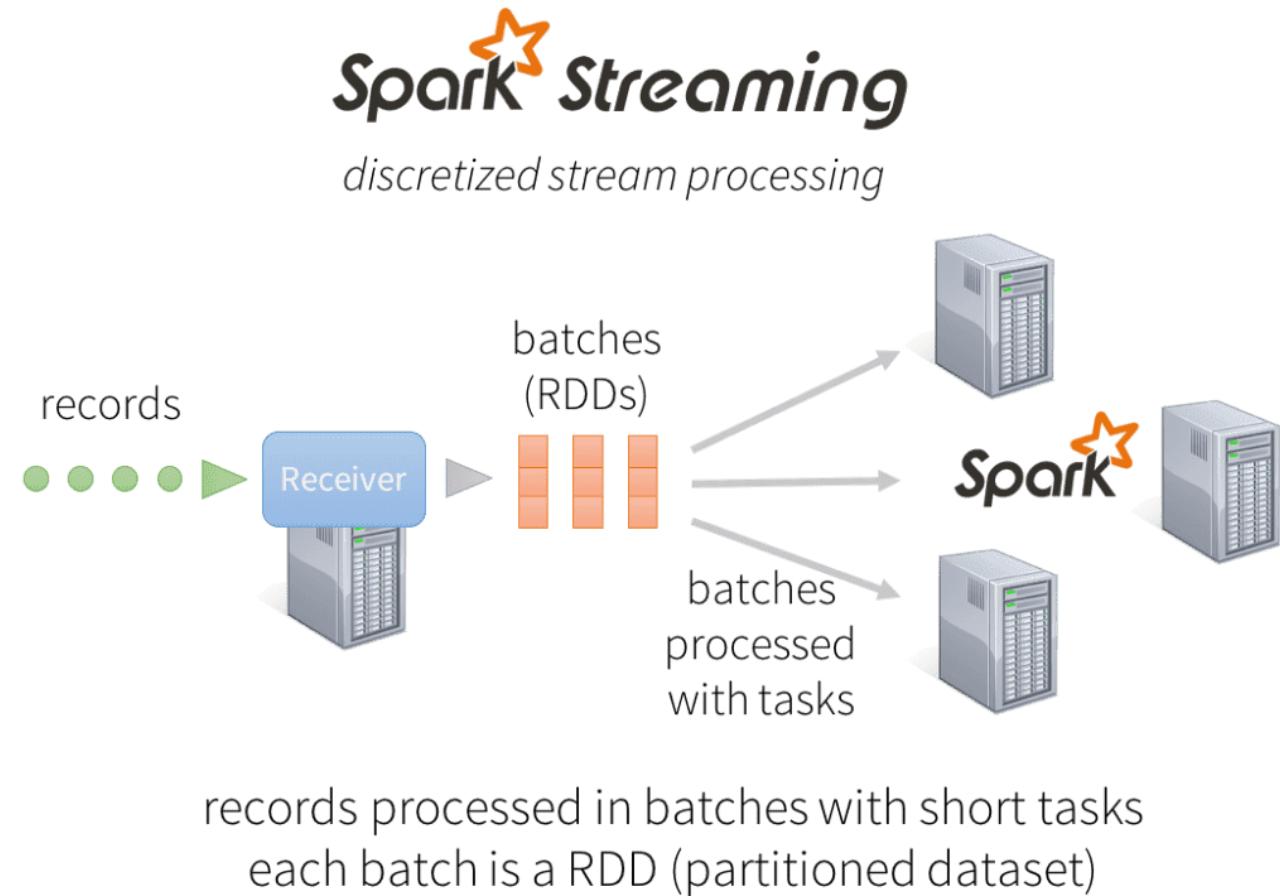


How Spark Streaming Works

- ❖ Spark Streaming provides a high-level abstraction called discretized stream or **DStream**.
- ❖ Dstream: a sequence of RDD (Resilient Distributed Dataset).
- ❖ It represents the stream of incoming data.
- ❖ **Input Sources:** Data can be ingested from a variety of sources like **Kafka**, Flume, or even a directory.
- ❖ **Transformations:** Similar to Spark, DStreams support transformations like **map**, **reduceByKey**, **filter**, etc.
- ❖ **Output Operations:** After processing, data can be written to external systems like HDFS, databases, or dashboards.

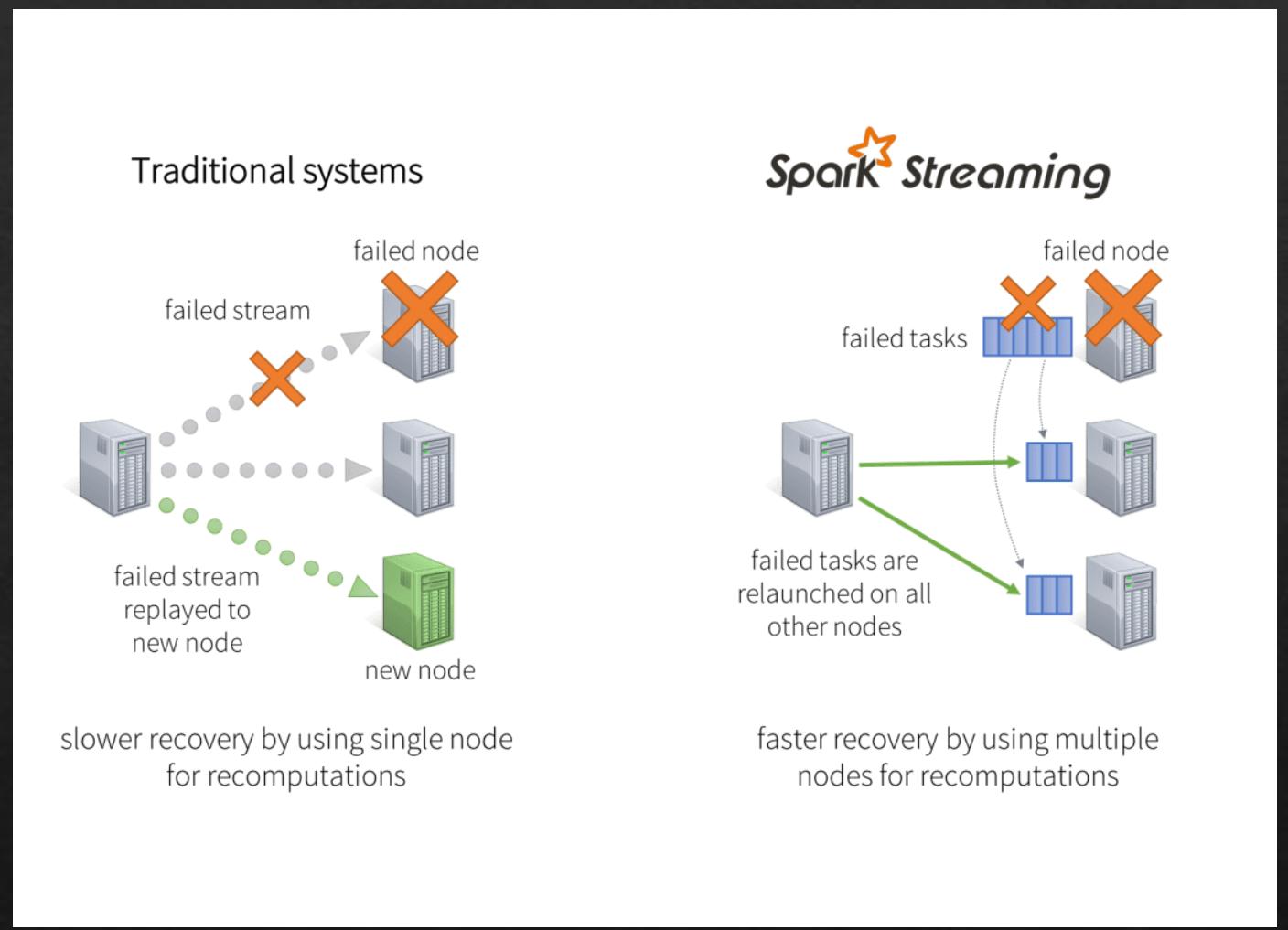
Spark Streaming Architecture

- ❖ Batch Processing of Streams:
 - ❖ Spark Streaming divides the real-time data into small batches (micro-batches).
 - ❖ Each micro-batch is processed by the Spark engine and results are returned in near real-time.
 - ❖ Data is processed with Spark's high-level operations like RDD transformations.



Spark Streaming vs. Traditional Stream Processing

- ❖ **Micro-Batch Processing:**
- ❖ Spark Streaming processes data in **micro-batches**, providing near-real-time processing.
- ❖ **Traditional Stream Processing:**
- ❖ Traditional systems (like Storm) process each record as it arrives, without batch processing.



Spark Streaming Use Cases

- ❖ Popular Use Cases:
- ❖ **Real-time Analytics:** Processing real-time logs, clickstream data, and other time-sensitive information.
- ❖ **Fraud Detection:** Continuous monitoring of transactions to detect anomalies.
- ❖ **IoT Data Processing:** Handling sensor data from connected devices in real-time.
- ❖ **Social Media Analytics:** Streaming and analyzing real-time user interactions from platforms like Twitter or Facebook.



Working with Spark Streaming & Kafka

- ❖ **Kafka as a Data Source:**
- ❖ Spark Streaming can easily consume data from Apache Kafka.
- ❖ Example: Real-time stock price streaming from Kafka topics, which is then processed by Spark Streaming for analysis.
- ❖ **Fault Tolerance:** Ensures that data is not lost even if Spark Streaming or Kafka experiences failures.

Getting Practical



Steps to create a spark streaming job

- ❖ Create a streaming context which is an entry point

```
// Initialize Spark streaming context
SparkConf sparkConf = new SparkConf()
    .setAppName("KafkaSparkConsumer")
    .setMaster("local[*]")
    .set("spark.ui.port", "4050");

JavaStreamingContext streamingContext = new JavaStreamingContext(sparkConf, new Duration(60000));
```

Create Dstream: kafkaUtils, TCP socket, local directory

```
// Create a direct stream from Kafka
JavaPairInputDStream<String, String> kafkaStream = KafkaUtils.createDirectStream(
    streamingContext,
    String.class,
    String.class,
    StringDecoder.class,
    StringDecoder.class,
    kafkaParams,
    topics
);

// Process the stream
JavaDStream<String> messages = kafkaStream.map(Tuple2::_2);
```

```
// Create a DStream that will connect to hostname:port, like localhost:9999
JavaReceiverInputDStream<String> lines = jssc.socketTextStream("localhost", 9999);
```

```
// Monitor a directory for new files
JavaDStream<String> lines = ssc.textFileStream(args[0]);
System.out.println("Monitoring directory: " + args[0]);

lines.foreachRDD(rdd ->
```

Create Hive Table using following query

```
create external table commodities(
    price_date date,
    price_1 float,
    Price_2 float
)
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
with serdeproperties(
    "separatorChar" = ",",
    "quoteChar"="\"",
    "escapeChar"="\\"
)
STORED AS TEXTFILE
location '/user/cloudera/project'
TBLPROPERTIES ("skip.header.line.count"="0");
```

Hue - Editor - Mozilla Firefox

Cloudera Live : Welcome... x Hue - Editor x Hue - Editor x +

quickstart.cloudera:8888/hue/editor?editor=21 | Search

Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started

HUE Query Search data and saved documents... Jobs cloudera

default default text ?

Tables mytable

Hive Add a name... Add a description... (1) ▼ + ↻

```
3 | price_date date,
4 | Price float
5 |
6 | row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
7 | with serdeproperties(
8 | "separatorChar" = ",",
9 | "quoteChar"="\"
10| )
```

Success.

The screenshot shows the Hue Editor interface within a Mozilla Firefox browser window. The title bar reads 'Hue - Editor - Mozilla Firefox'. The address bar shows the URL 'quickstart.cloudera:8888/hue/editor?editor=21'. The main header includes the 'HUE' logo, a 'Query' button, a search bar, and user authentication information ('Jobs' and 'cloudera'). On the left, there's a sidebar with icons for databases, tables, and a search bar. Below it, the 'Tables' section lists 'default' and 'mytable'. The main workspace is titled 'Hive' and contains a query editor with the following code:

```
3 | price_date date,
4 | Price float
5 |
6 | row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
7 | with serdeproperties(
8 | "separatorChar" = ",",
9 | "quoteChar"="\"
10| )
```

Below the code, a message box indicates 'Success.'.

HUE

Query ▾ Search data and saved documents...

Jobs ⚡ cloudera

default

Tables (2) ↴ + ↵

commodities

mytable

✓ Success.

Query History Q X Saved Queries Q C

a few seconds ago ➔

```
create external table commodities(
    Com_Type string, Price_Date date, Price float
)
row format serde
'org.apache.hadoop.hive.serde2.OpenCSVSerde'
with serdeproperties( "separatorChar" = ",",
"quoteChar"="\\" )
location '/user/cloudera/project'
```

drop table commodities;

quickstart.cloudera:8888/hue/editor?editor=22

Search

Cloudera Hue Hadoop HBase Impala Spark Solr Oozie Cloudera Manager Getting Started

HUE

Query Search data and saved documents... Jobs cloudera

default

Tables (2) ▾ +

commodities mytable

Hive Add a name... Add a description... 0s default text

1| Select * FROM commodities Limit 10;

Success.

Result

The screenshot shows the Hue web interface for Cloudera, specifically the Hive editor. The URL in the browser is `quickstart.cloudera:8888/hue/editor?editor=61`. The top navigation bar includes links for Cloudera, Hue, Hadoop, HBase, Impala, Spark, Solr, Oozie, Cloudera Manager, and Getting Started. The main area is titled "HUE" and shows a "Query" tab selected. A search bar at the top right says "Search data and saved documents...". On the left, there's a sidebar for "Hive" with "Databases" (1) and "default" selected. The main pane displays a query editor with the following code:

```
1|SELECT * FROM commodities LIMIT 10;
```

Below the query, there are buttons for "Query History", "Saved Queries", and "Results (10)". The results table has two columns: "commodities.price_date" and "commodities.price_1". The data is as follows:

	commodities.price_date	commodities.price_1
1	2023-12-01	210.310476190476
2	2023-11-01	197.433181818182
3	2023-10-01	183.951363636364
4	2023-09-01	183.589047619048
5	2023-08-01	186.346956521739

```
// Initialize SQLContext or HiveContext depending on the use case
val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

// Show databases
hiveContext.sql("SHOW DATABASES").show()

// Use a specific database (optional)
hiveContext.sql("USE DEFAULT")

// Show tables in the selected database
hiveContext.sql("SHOW TABLES").show()

// Print schema
df.printSchema()

// Query a specific table
val df = hiveContext.sql("SELECT * FROM commodities LIMIT 10")

// Show the results
df.show()
```

Getting Practical



What is Tableau

- ❖ Tableau is one of the fastest data visualization tools currently being used
- ❖ Visualization allows us visual access to huge amounts of data in easily digestible formats
- ❖ It is the best way to change or transform raw data set in to an easily understandable format.

Tableau

The screenshot shows the Tableau Connect interface. On the left, there's a sidebar with options like 'Search for Data', 'Tableau Server', 'To a File' (Microsoft Excel, Text file, JSON file), 'Microsoft Access', 'PDF file', 'Spatial file', 'Statistical file', 'More...', 'To a Server' (Vertica, Web Data Connector (deprecated), Other Databases (JDBC), Other Databases (ODBC)), 'Saved Data Sources' (Sample - Superstore, World Indicators), and a 'More...' button. The 'Other Databases (ODBC)' option under 'To a Server' is highlighted.

The main area displays the 'Other Databases (ODBC)' connection dialog. It has two tabs: 'Connect Using' and 'Connection Attributes'. In 'Connect Using', the 'Driver' is selected as 'Cloudera ODBC Driver for Apache Hive'. In 'Connection Attributes', the 'Server' is set to '192.168.227.128', 'Port' to '10000', 'Database' to 'HIVE', 'Username' to 'cloudera', and 'Password' is masked. A 'String Extras' section contains the following configuration string:

```
AllowHostNameCNMismatch=0;AllowInvalidCACert=0;AllowSelfSignedServerCert=0;ApplySSWPWithQueries=1;AsyncExecPollInterval=100;Auth_Flow=1;AuthMech=2;AutoReconnect=1;BinaryColumnLength=32767;CheckCertRevocation=1;DecimalColumnScale=10;DefaultStringColumnLength=255;DelegateKrbCreds=0;DelegationLUID=DESCRIPTION=;DriverConfigTakePrecedence=0;EnableAsynchExec=0;EnableTempTable=0;EnableTranslationForCTAS=1;EnableUniqueColumnName=1;FastSQLPrepare=0;FixUnquotedDefaultSchemaNameInQuery=1;GetTablesWithQuery=0;HDFSTemporaryTableDir=/tmp/simba;HDFSUser=hdfs;HiveServerType=2;Host=
```

At the bottom right of the dialog is a 'Sign In' button. To the right of the dialog, a list of other database connectors is visible, including Essbase, Greenplum Database, eSQL, Progress OpenEdge, Presto (deprecated), Oracle, Oracle Data Cloud, ANA, JetWeaver Business Warehouse, MySQL ASE, MySQL IQ, eNow ITSM (deprecated), Point Lists (deprecated), Store, Table, SQL, Data, Data OLAP Connector, Data Virtualization, and Data Connector (deprecated). The list is sorted by 'Name (a-z)'.

Hbase

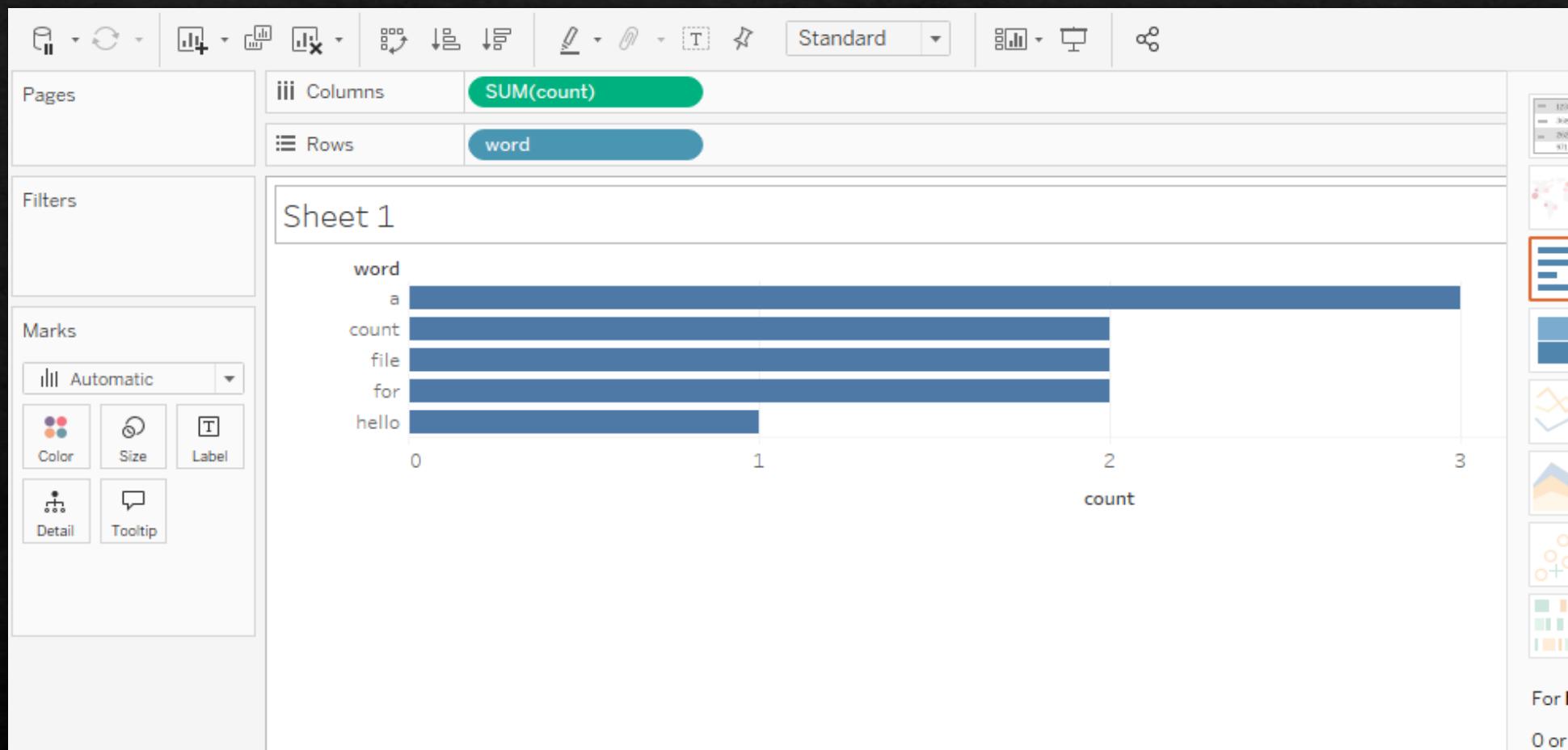
The screenshot shows the Hue web interface for managing Hadoop data. The top navigation bar includes links for Cloudera, Hue, Hadoop, HBase, Impala, Spark, Solr, Oozie, Cloudera Manager, and Getting Started. The main area is titled "HUE" with a "Query" dropdown and a search bar. On the left, a sidebar shows the "default" database with tables: chicagoemp (no columns), records, and wordcount_hbase1 (with columns count and word). The central pane is titled "Impala" and displays the following query:

```
1|SELECT word, count FROM wordcount_hbase1 LIMIT 5;
```

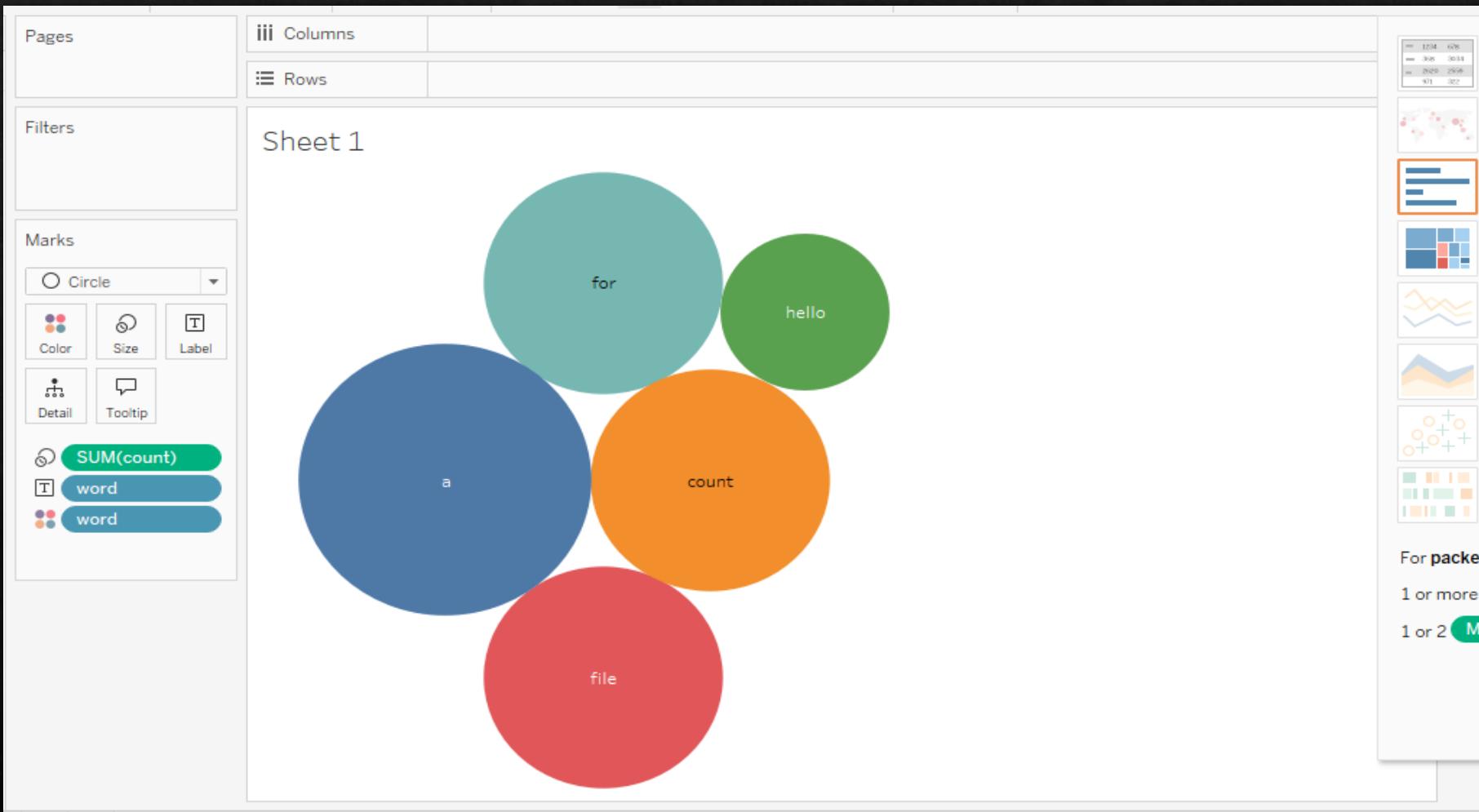
Below the query, a "Results (5)" section shows the following data:

	word	count
1	a	3
2	count	2
3	file	2
4	for	2
5	hello	1

Tableau



Tableau



Tableau

Connect

Search for Data

Tableau Server

To a File

- Microsoft Excel
- Text file
- JSON file
- Microsoft Access

PDF

Processing Request

Span

Stat

Executing query.

More... >

Saved Data Sources

- Sample - Superstore
- World Indicators

Installed Connectors (70)

Google BigQuery

Sort by Name (a-z)

Other Databases (ODBC)

Connect Using

Generic ODBC requires additional configuration for publishing to succeed. Select DSN (data source name) for cross-platform portability. A DSN with the same name must be configured on Tableau Server.

DSN: []

the Hive

Port: []

Azure Synap

Box

Cloudera Hado

Databricks

Denodo (dep

Dremio

Dropbox

Esri

Exasol

Firebird 3

Google Analy

Database:

Username:

Password:

String Extras:

Oracle Essbase

PostgreSQL

Presto

Progress OpenEdge

Subole Presto (deprecated)

Salesforce

Salesforce Data Cloud

AP HANA

AP NetWeaver Business Warehouse

AP Sybase ASE

AP Sybase IQ

ServiceNow ITSM (deprecated)

SharePoint Lists (deprecated)

SingleStore

Snowflake

Spark SQL

Splunk

Steradata

Steradata OLAP Connector

UBCO Data Virtualization

Vertica

Web Data Connector (deprecated)

12:21 AM 61°F 9/24/2024

Thank you!