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.

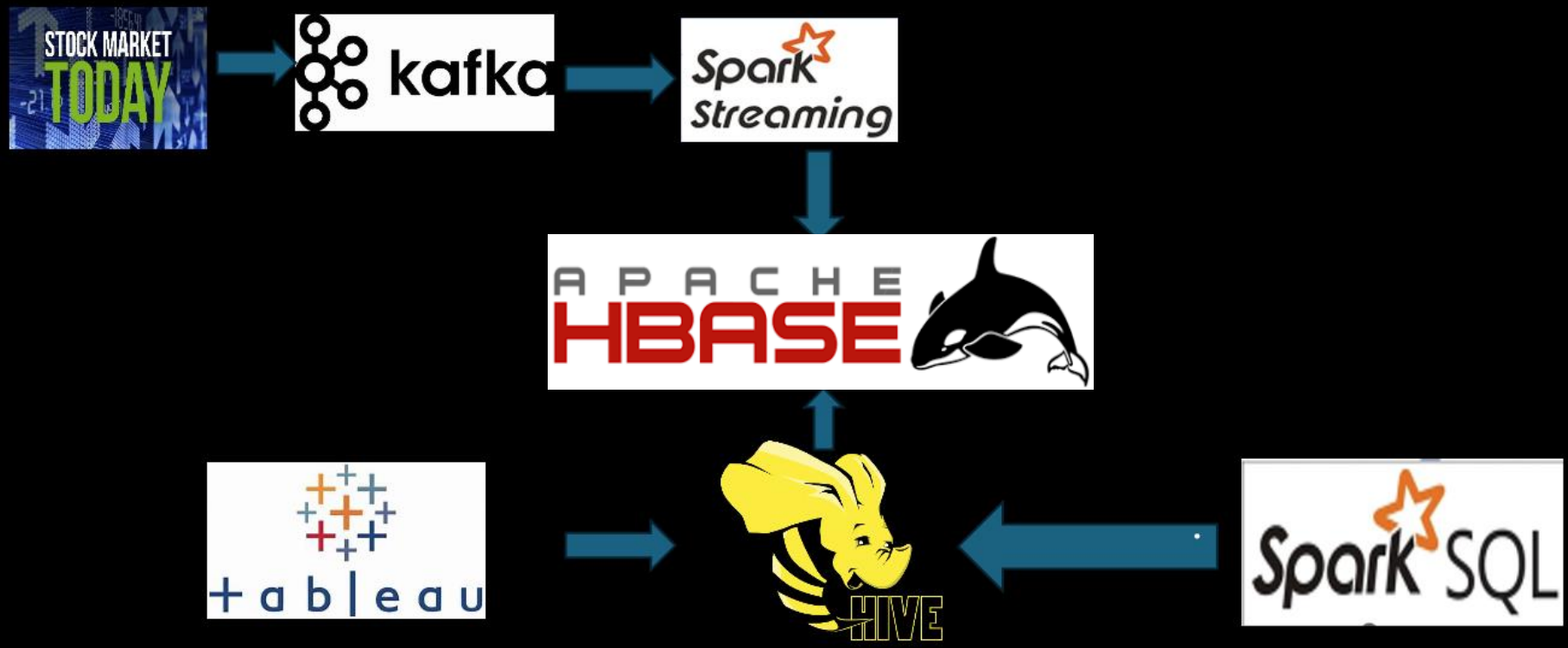
Wholeness of Project

- **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.

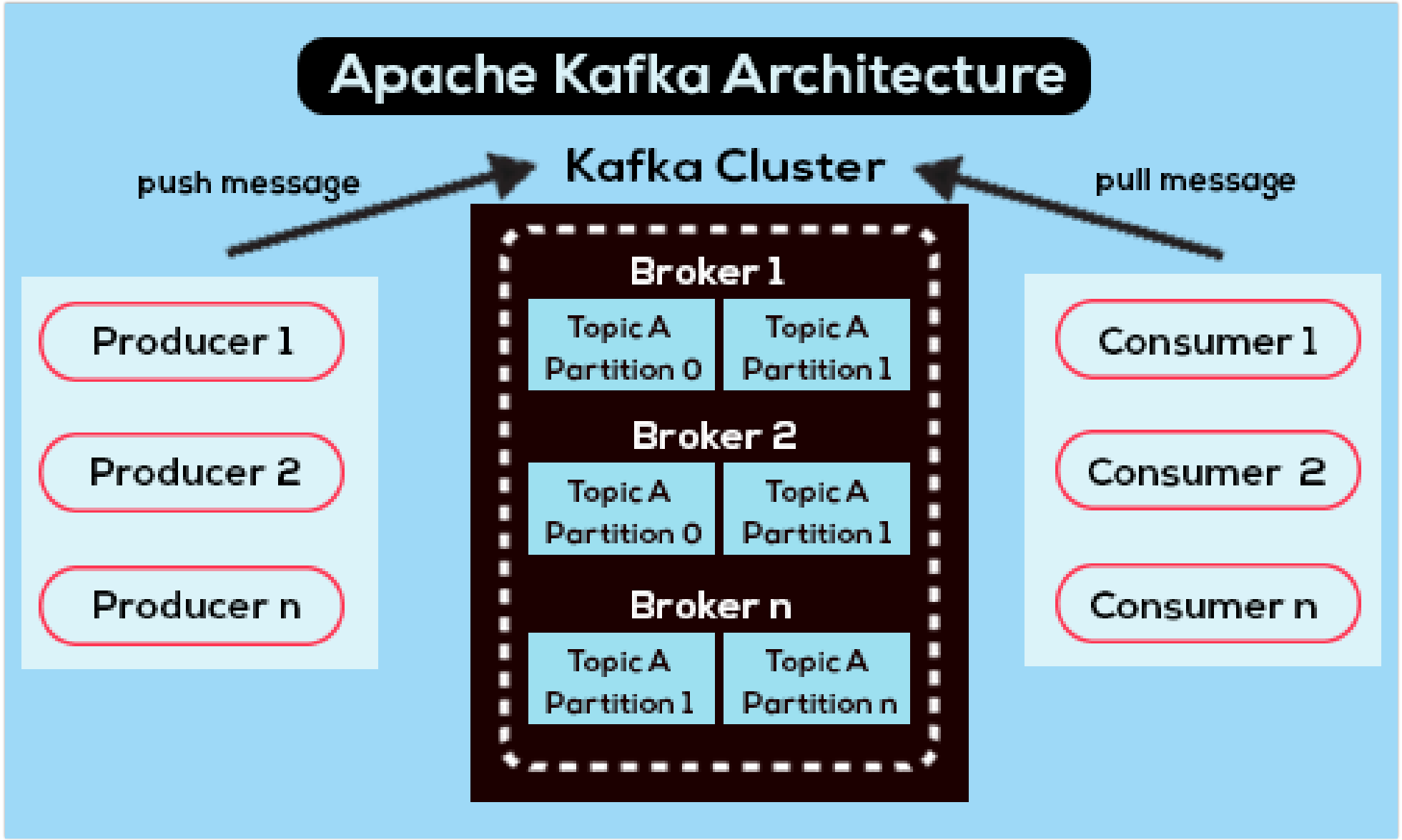


What is Apache Kafka?

- 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.

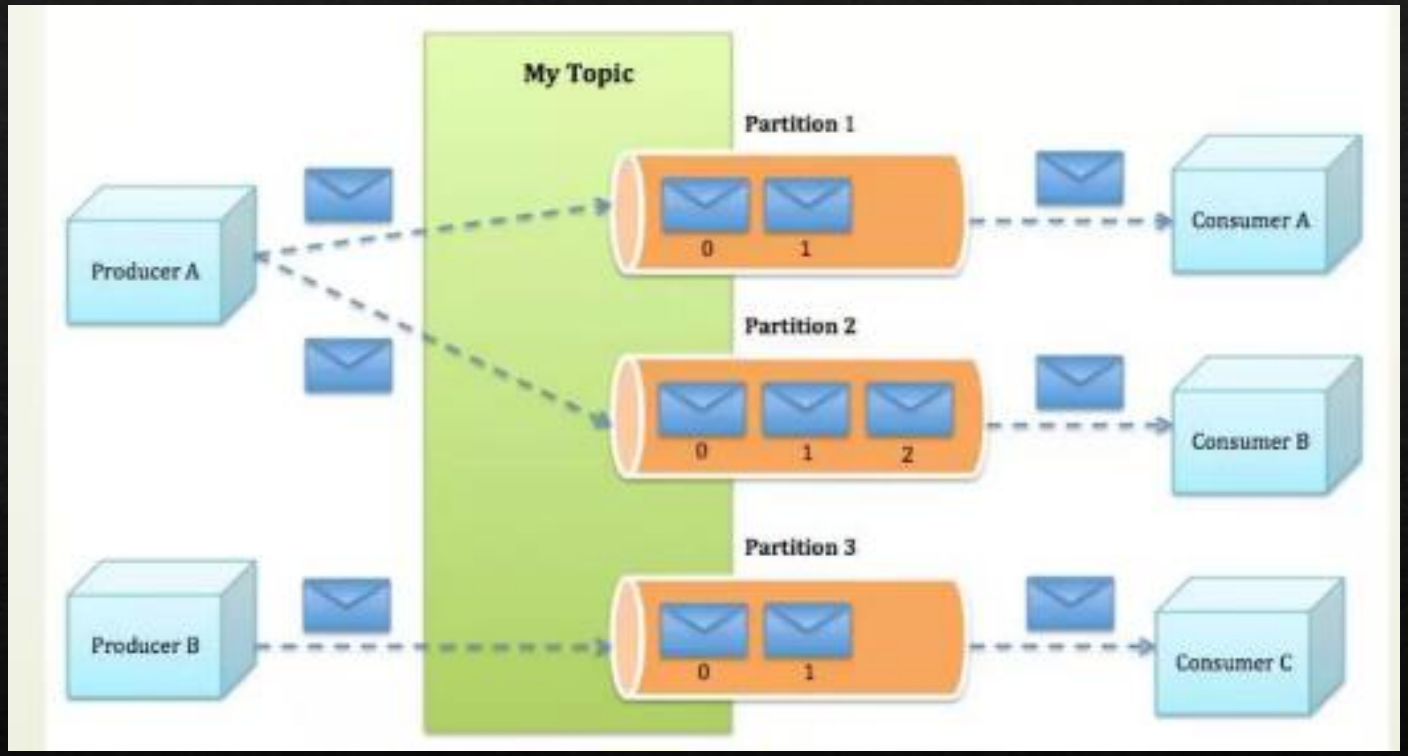
Kafka Architecture

- **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
push message
Kafka Cluster
pull message
Producer 1
Producer 2
Producer n
Broker 1
Topic A Partition 0
Topic A Partition 1
Broker 2
Topic A Partition 0
Topic A Partition 1
Broker n
Topic A Partition 1
Topic A Partition n
Consumer 1
Consumer 2
Consumer n

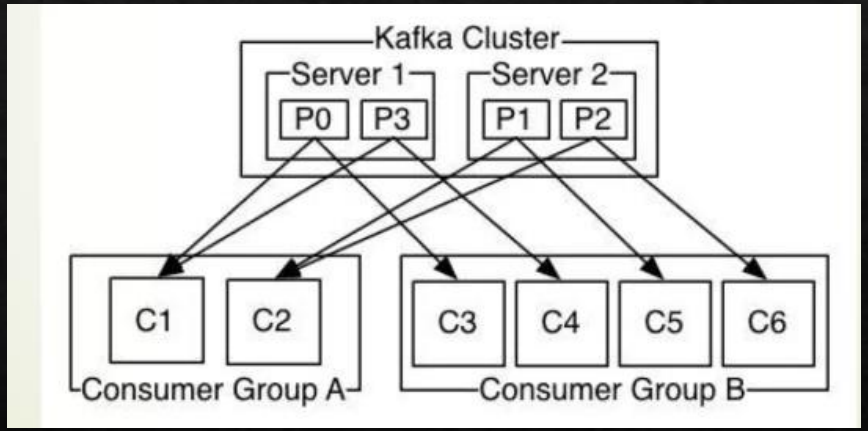
Producers

- **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

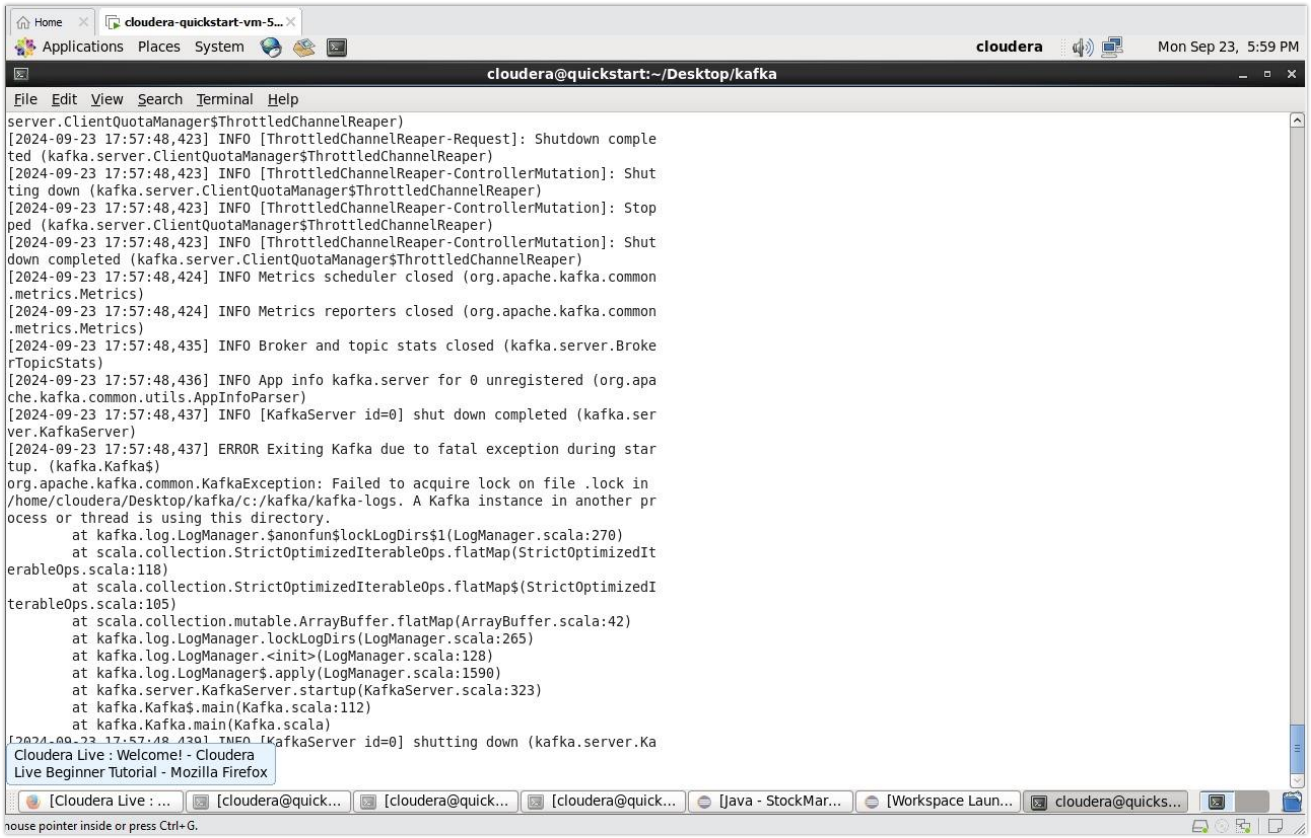
- The **Kafka cluster** retains all published messages, whether or not they have been consumed, for a **configurable period of time**. After this time, messages are discarded to free up space.
- Metadata is retained on a **per-consumer** basis, tracking the consumer's position in the log, known as the **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

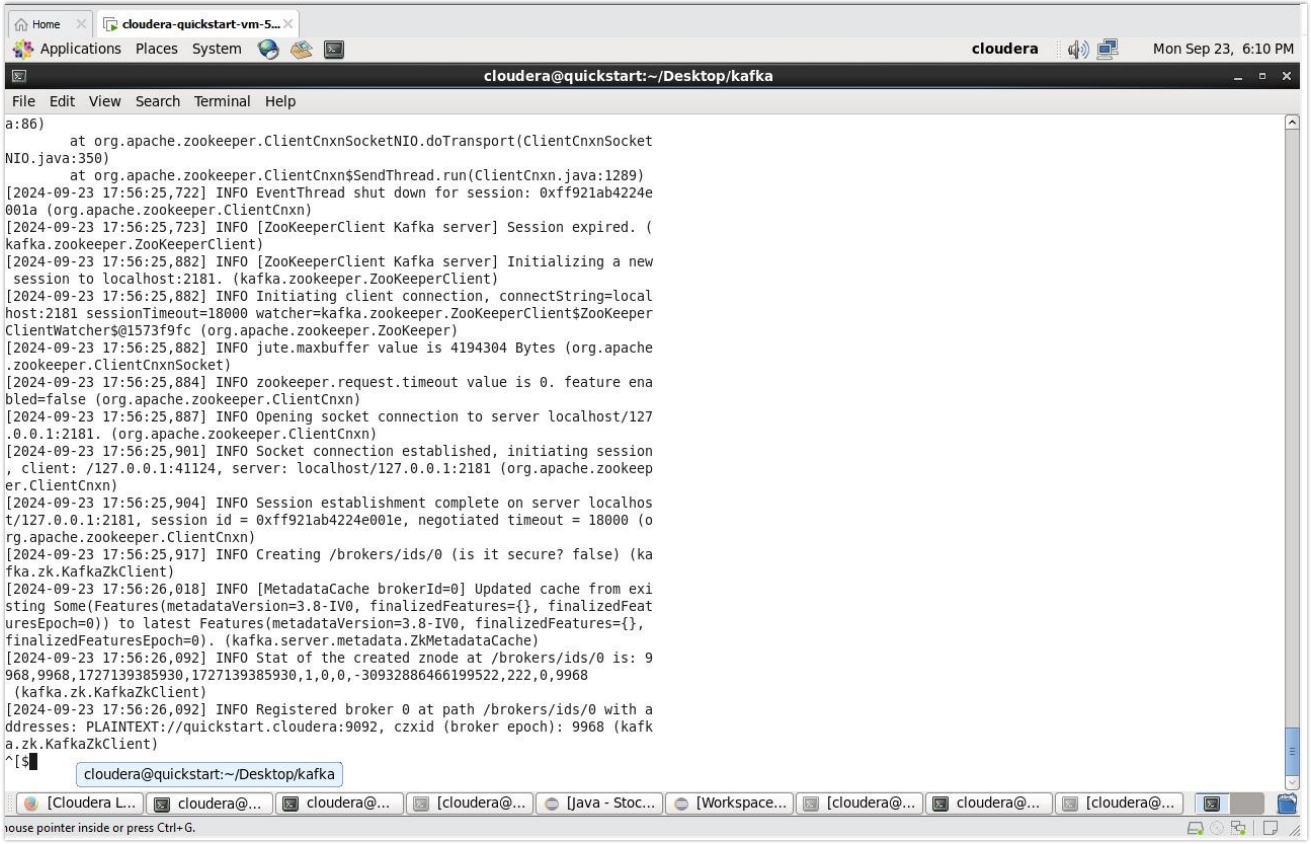
- Kafka assigns **partitions** in a topic to **consumers** in a consumer group. This ensures that each partition is consumed by exactly one consumer in the group.
- **Limitation**: There cannot be more consumer instances in a consumer group than there are partitions.
- Kafka provides a **total order**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

Getting Practical

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



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



```java
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.serializatio
        properties.put("value.serializer", "org.apache.kafka.common.serializat

        // Kafka producer
        try (KafkaProducer<String, String> producer = new KafkaProducer<>(prop
             CloseableHttpClient httpClient = HttpClients.createDefault()) {

            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_SERIE
                    HttpGet request = new HttpGet(url);
                    HttpResponse response = httpClient.execute(request);

                    try (BufferedReader reader = new BufferedReader(new InputStreamReader(response.ge
                        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

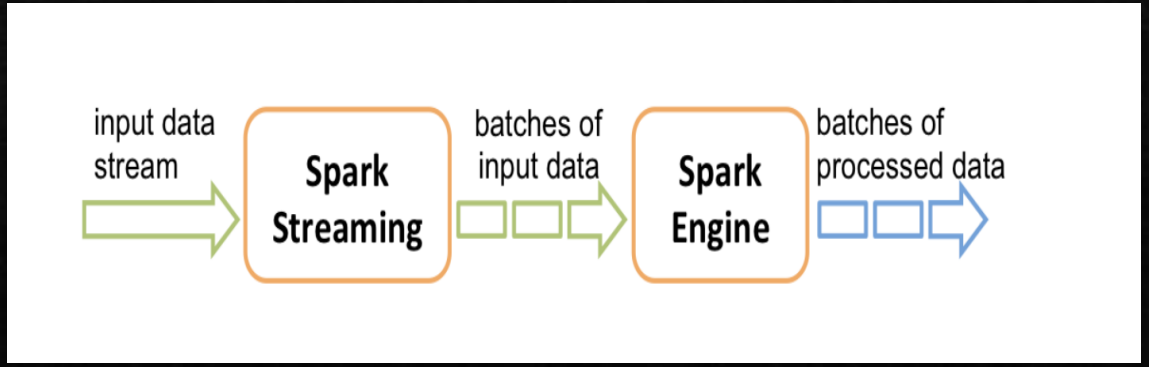
- Spark Streaming is a legacy project
- https://spark.apache.org/docs/latest/streaming-programming-guide.html
- You should use Spark Structured Streaming for your streaming applications and pipelines
- https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html

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.

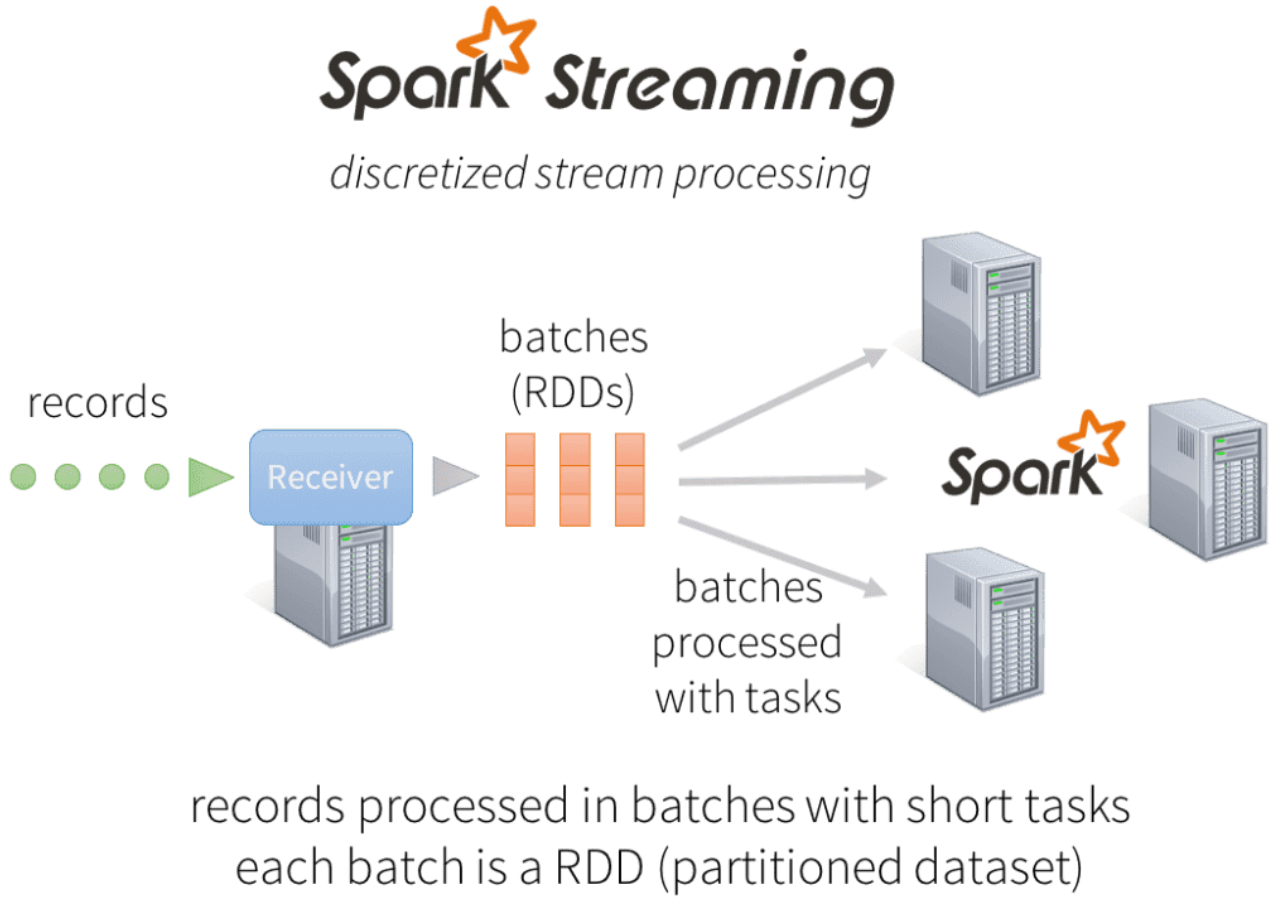


How Spark Streaming Works

- 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.

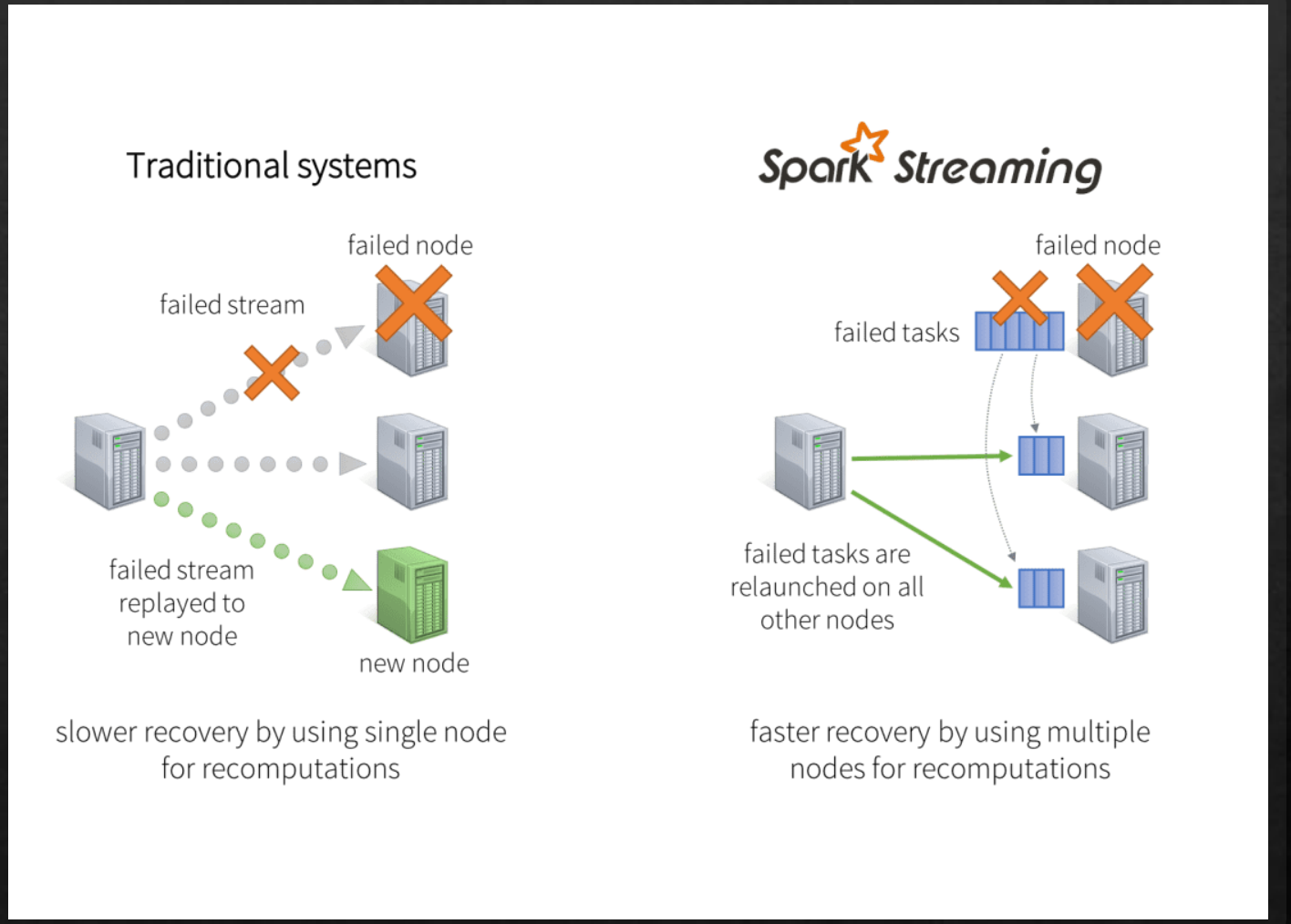
Spark Streaming Architecture

- **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.



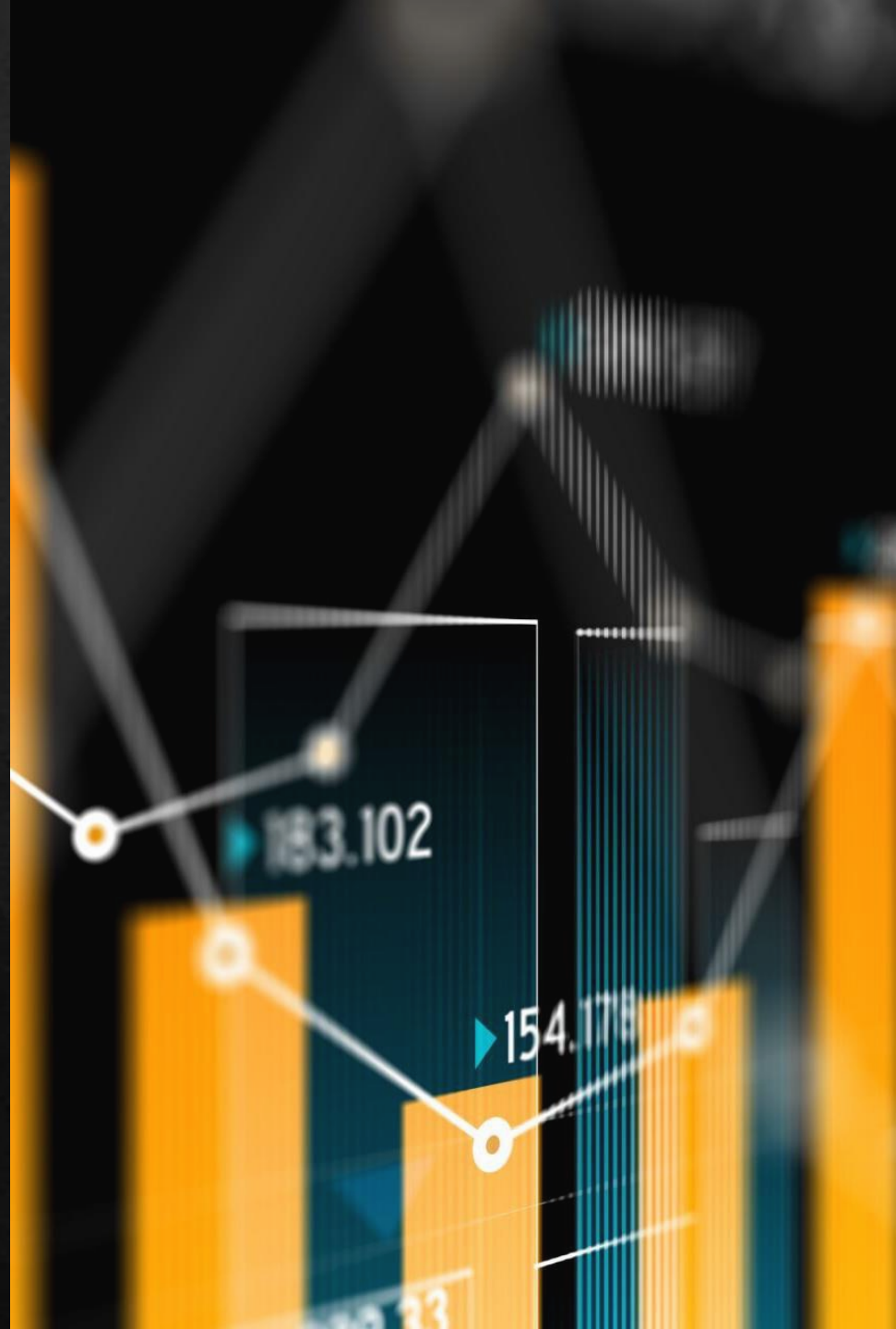
Spark Streaming vs. Traditional Stream Processing

- **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

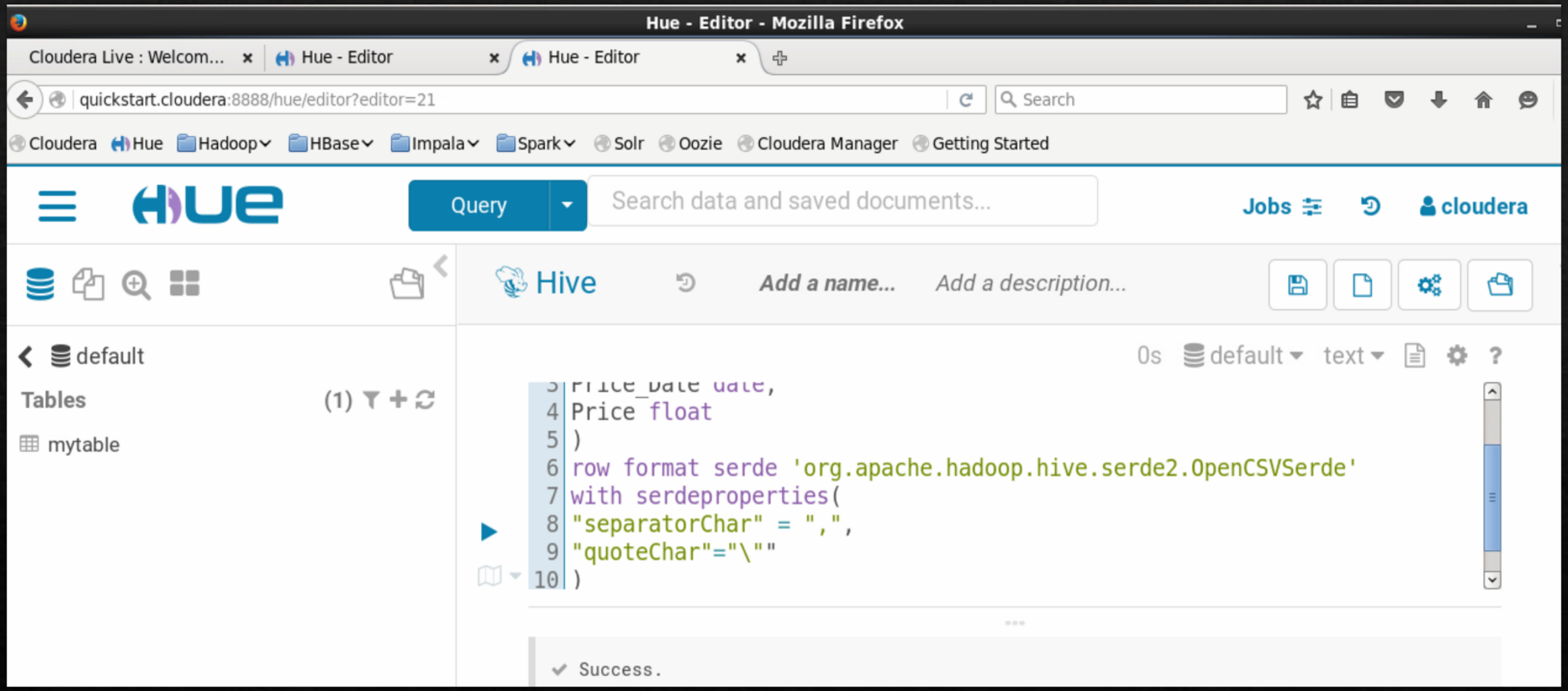
```
// Create a direct stream from Kafka
JavaPairInputDStream<String, String> kafkaStream = KafkaUtils.createDirectStream(
        streamingContext,
        String.class,
        String.class,
        StringDecoder.class,
        StringDecoder.class,
        kafkaParams,
        topics
);

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

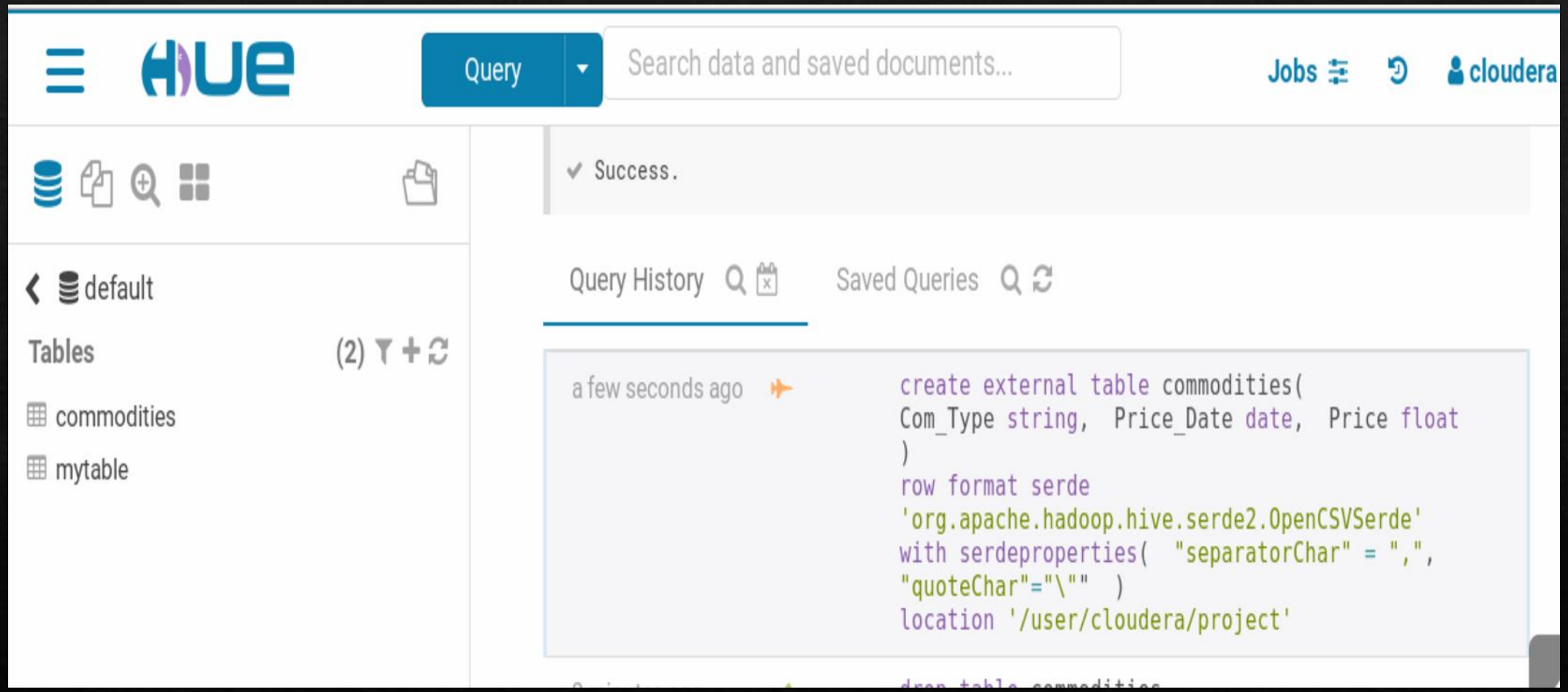
```
// Create a DStream that will connect to hostname:port, like localhost:9999
JavaReceiverInputDStream<String> lines = jssc.socketTextStream("localhost", 9999);
```

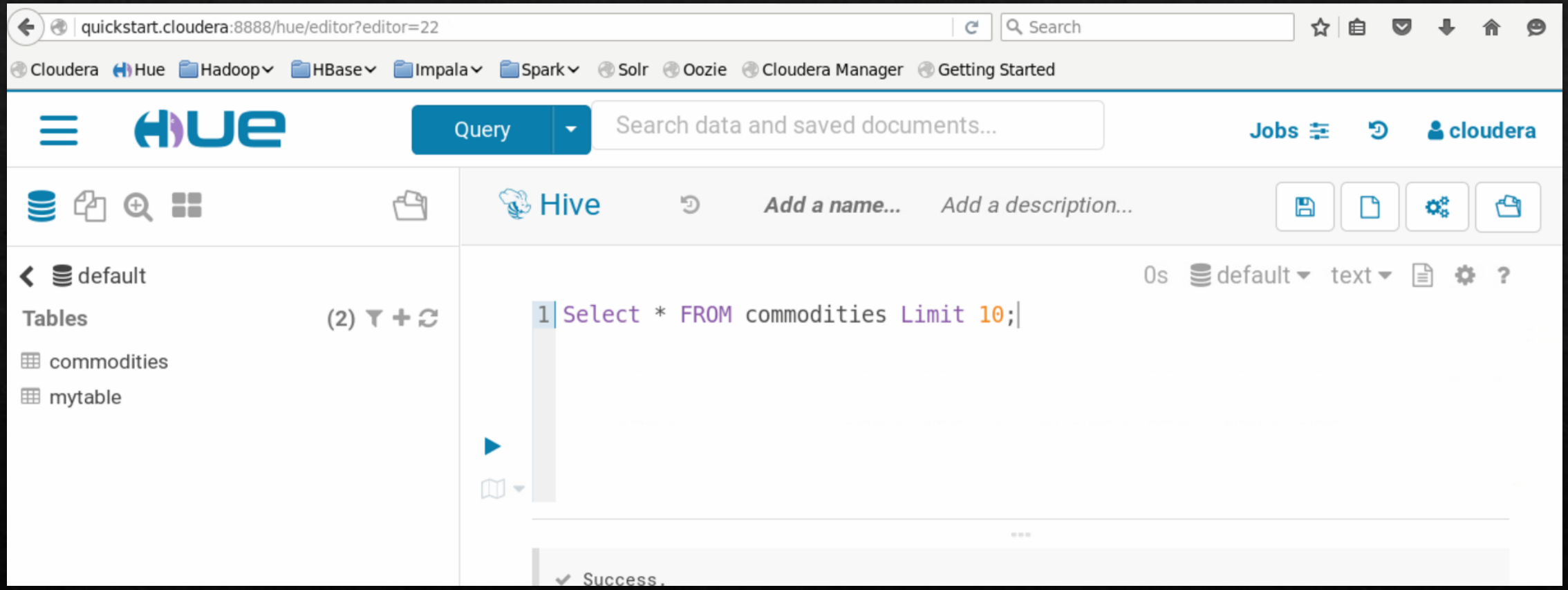
```
// Monitor a directory for new files
JavaDStream<String> lines = ssc.textFileStream(args[0]);
System.out.println("Monitoring directory: " + args[0]);
```

Create Hive Table using following query

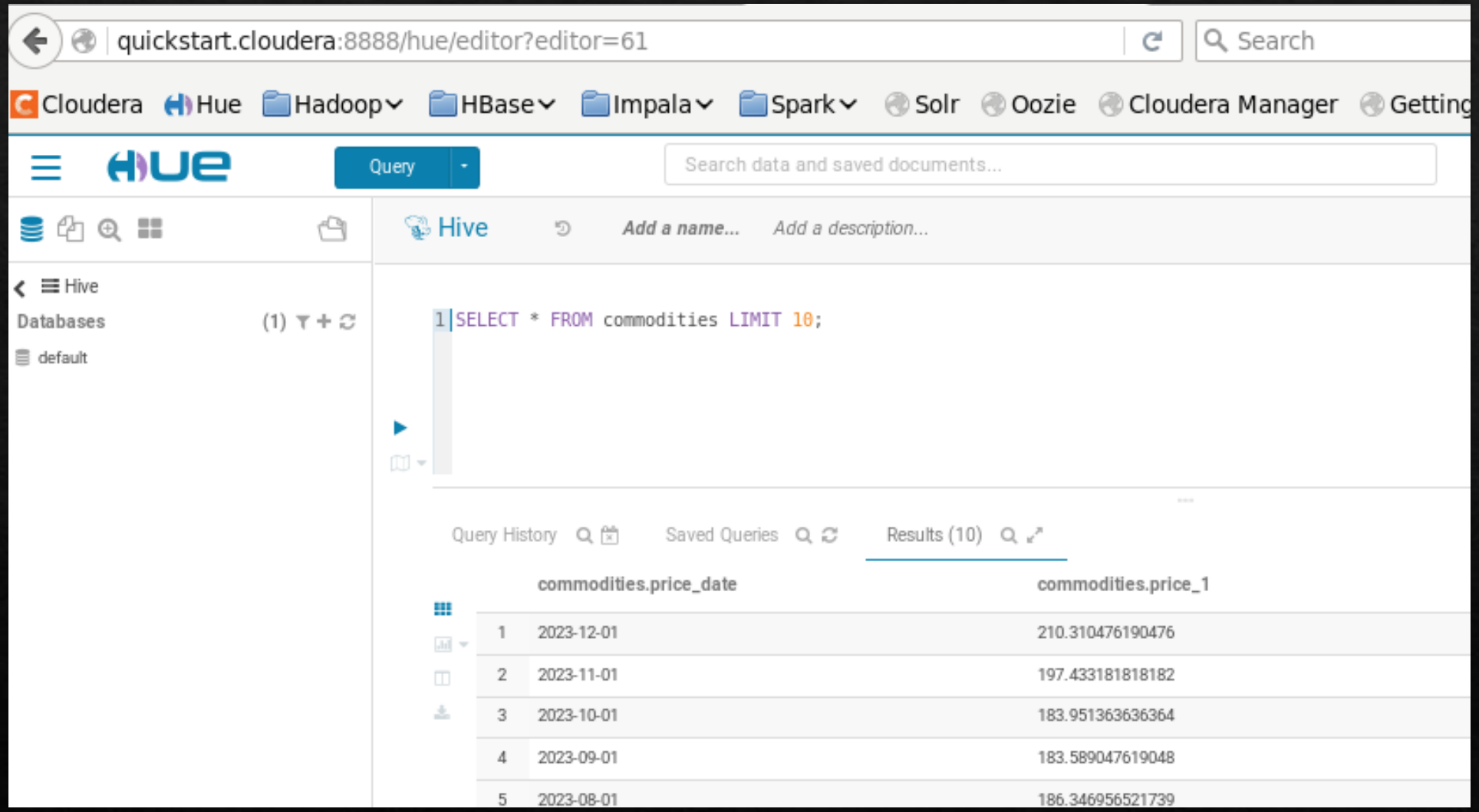
```
create external table commodities(
price_date date,
price_1 float,
Price_2 float
)
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
with serdeproperties(
"separatorChar" = ",",
"quoteChar"="\"",
"escapseChar"="\\"
)
STORED AS TEXTFILE
location '/user/cloudera/project'
TBLPROPERTIES ("skip.header.line.count"="0");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


Hue - Editor - Mozilla Firefox
Cloudera Live : Welcom...
Hue - Editor
Hue - Editor
quickstart.cloudera:8888/hue/editor?editor=21
Search
Cloudera
Hue
Hadoop
HBase
Impala
Spark
Solr
Oozie
Cloudera Manager
Getting Started
Query
Search data and saved documents...
Jobs
cloudera
Hive
Add a name...
Add a description...
default
Tables (1)
mytable
0s
default
text
Price_date date,
Price float
)
row format serde 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
with serdeproperties(
"separatorChar" = ",",
"quoteChar"="\""
)
Success.


HUE
Query
Search data and saved documents...
Jobs
cloudera
Success.
default
Tables (2)
commodities
mytable
Query History
Saved Queries
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'


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
Hive
Add a name...
Add a description...
default
Tables
(2)
commodities
mytable
0s
default
text
1 Select * FROM commodities Limit 10;
Success.

Result


quickstart.cloudera:8888/hue/editor?editor=61
Search
Cloudera
Hue
Hadoop
HBase
Impala
Spark
Solr
Oozie
Cloudera Manager
Getting
HUE
Query
Search data and saved documents...
Hive
Add a name...
Add a description...
Hive
Databases (1)
default
1 SELECT * FROM commodities LIMIT 10;
Query History
Saved Queries
Results (10)
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

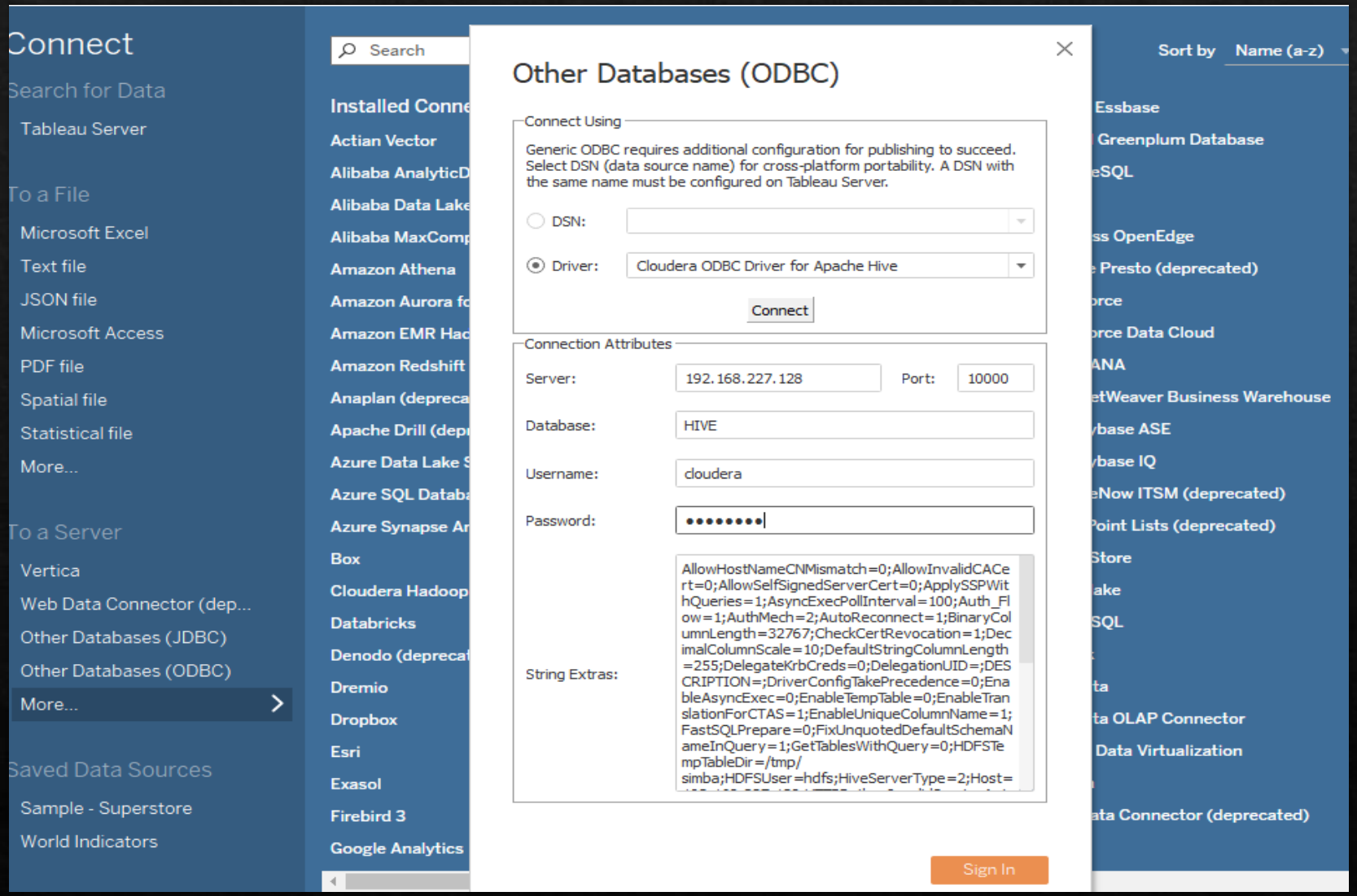
```
// Initialize SQLContext or HiveContext depending on the use case
val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc)

// 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

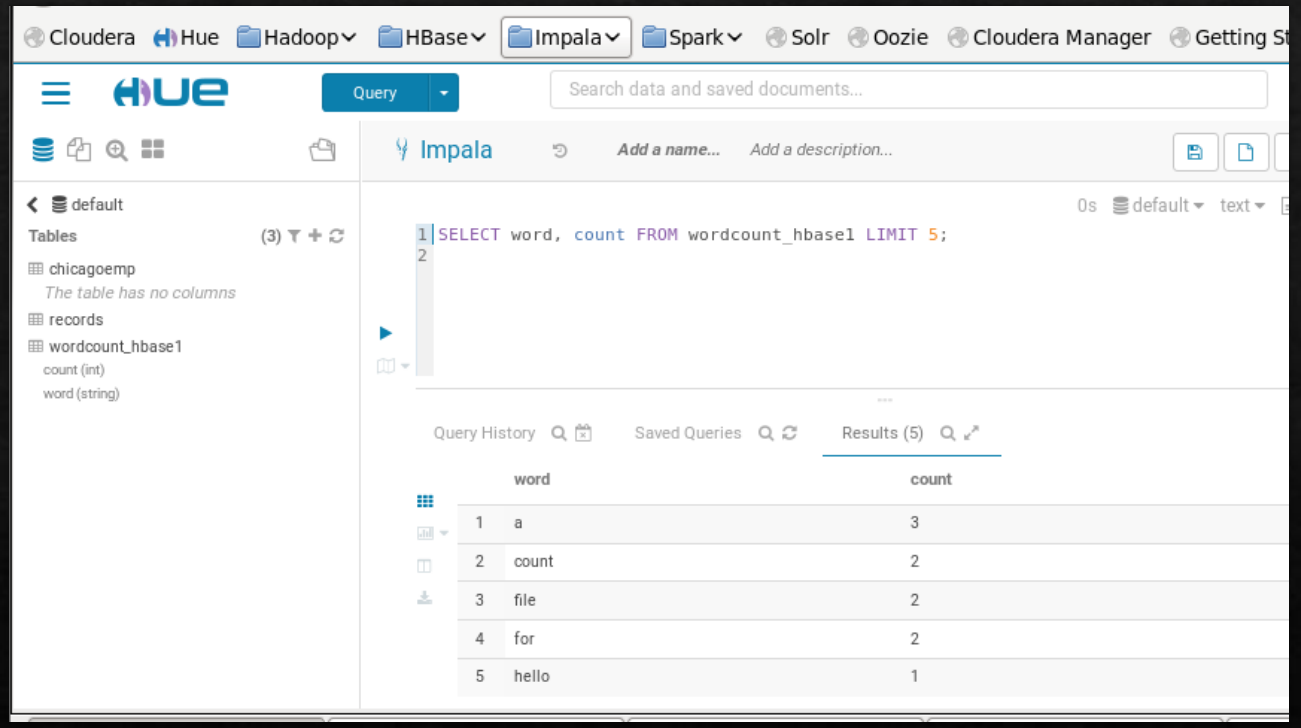
What is Tableau

- 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.

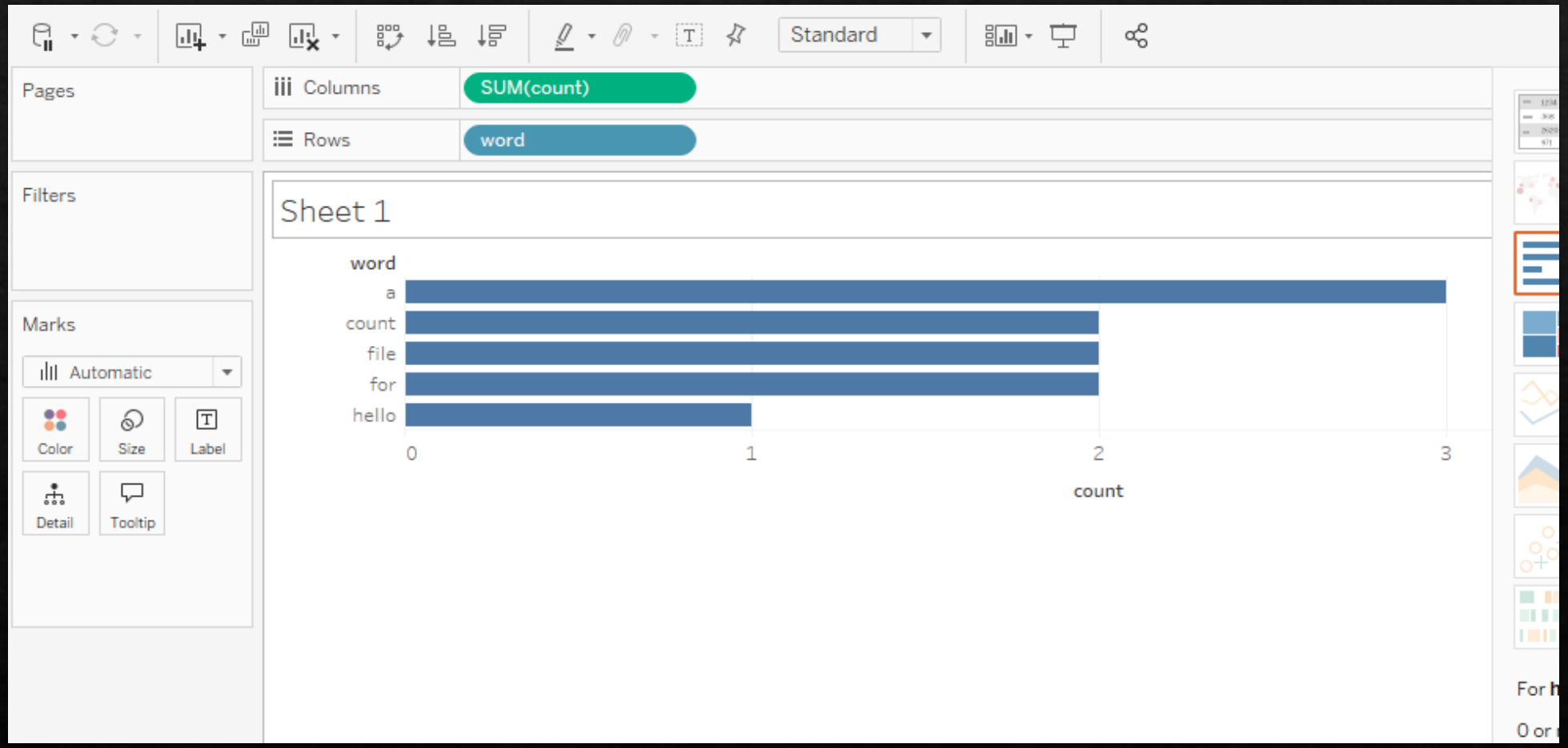
Tableau


Connect
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 (dep...
Other Databases (JDBC)
Other Databases (ODBC)
More...
Saved Data Sources
Sample - Superstore
World Indicators
Search
Installed Conne
Actian Vector
Alibaba AnalyticD
Alibaba Data Lake
Alibaba MaxComp
Amazon Athena
Amazon Aurora fo
Amazon EMR Hac
Amazon Redshift
Anaplan (depreca
Apache Drill (depr
Azure Data Lake S
Azure SQL Databa
Azure Synapse An
Box
Cloudera Hadoop
Databricks
Denodo (deprecat
Dremio
Dropbox
Esri
Exasol
Firebird 3
Google Analytics
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:
Driver:
Cloudera ODBC Driver for Apache Hive
Connect
Connection Attributes
Server:
192.168.227.128
Port:
10000
Database:
HIVE
Username:
cloudera
Password:
String Extras:
AllowHostNameCNMismatch=0;AllowInvalidCACert=0;AllowSelfSignedServerCert=0;ApplySSPWithQueries=1;AsyncExecPollInterval=100;Auth_Flow=1;AuthMech=2;AutoReconnect=1;BinaryColumnLength=32767;CheckCertRevocation=1;DecimalColumnScale=10;DefaultStringColumnLength=255;DelegateKrbCreds=0;DelegationUID=;DESCRIPTION=;DriverConfigTakePrecedence=0;EnableAsyncExec=0;EnableTempTable=0;EnableTranslationForCTAS=1;EnableUniqueColumnName=1;FastSQLPrepare=0;FixUnquotedDefaultSchemaNameInQuery=1;GetTablesWithQuery=0;HDFSTempTableDir=/tmp/
simba;HDFSUser=hdfs;HiveServerType=2;Host=
Sign In
Sort by
Name (a-z)
Essbase
Greenplum Database
eSQL
ss OpenEdge
Presto (deprecated)
orce
orce Data Cloud
ANA
etWeaver Business Warehouse
ybase ASE
ybase IQ
eNow ITSM (deprecated)
Point Lists (deprecated)
Store
ake
SQL
ta
ta OLAP Connector
Data Virtualization
ata Connector (deprecated)

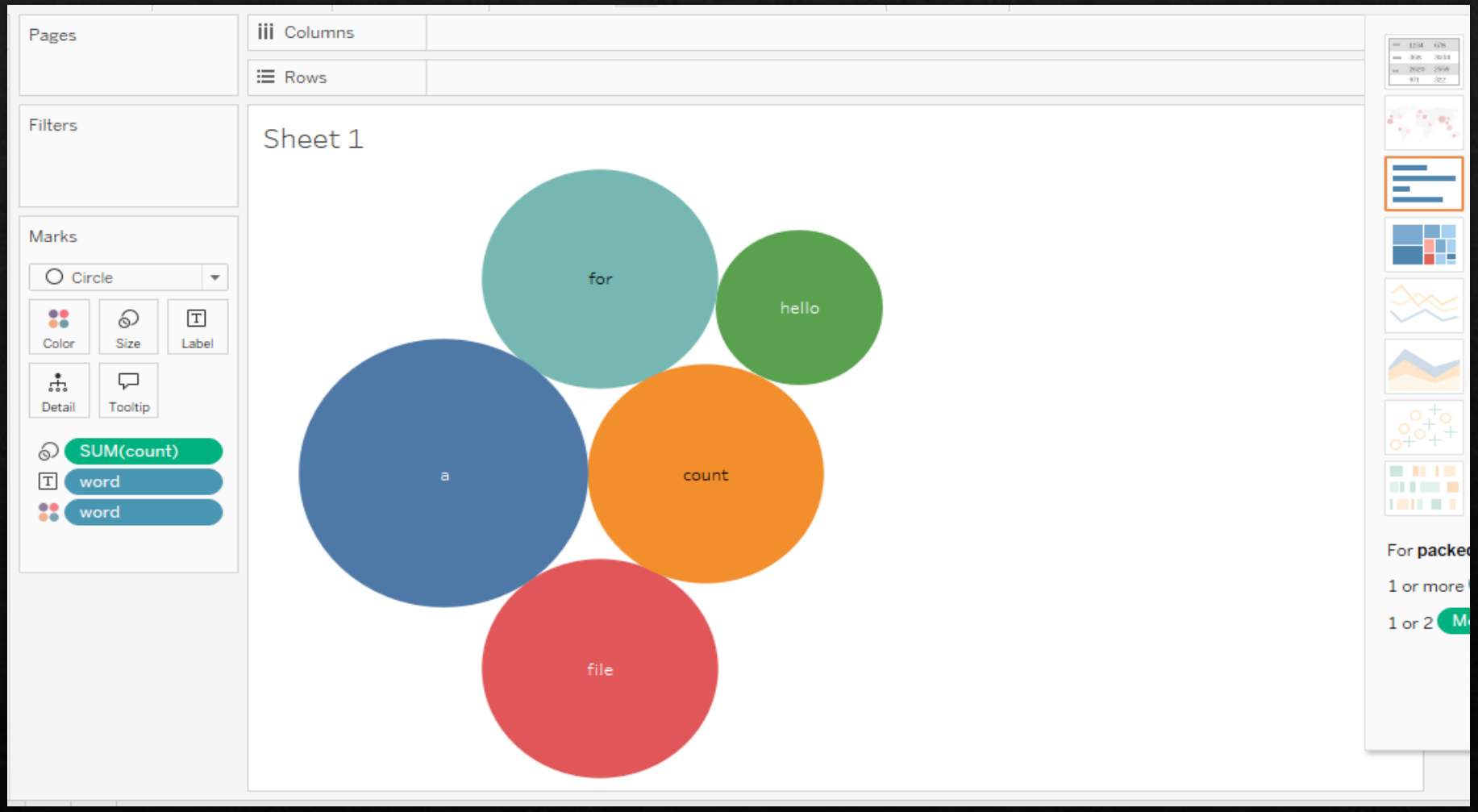
Hbase


Cloudera
Hue
Hadoop
HBase
Impala
Spark
Solr
Oozie
Cloudera Manager
Getting St
Query
Search data and saved documents...
Impala
Add a name...
Add a description...
default
Tables (3)
chicagoemp
The table has no columns
records
wordcount_hbase1
count (int)
word (string)
0s
default
text
1 SELECT word, count FROM wordcount_hbase1 LIMIT 5;
2
Query History
Saved Queries
Results (5)
word
count
1 a 3
2 count 2
3 file 2
4 for 2
5 hello 1

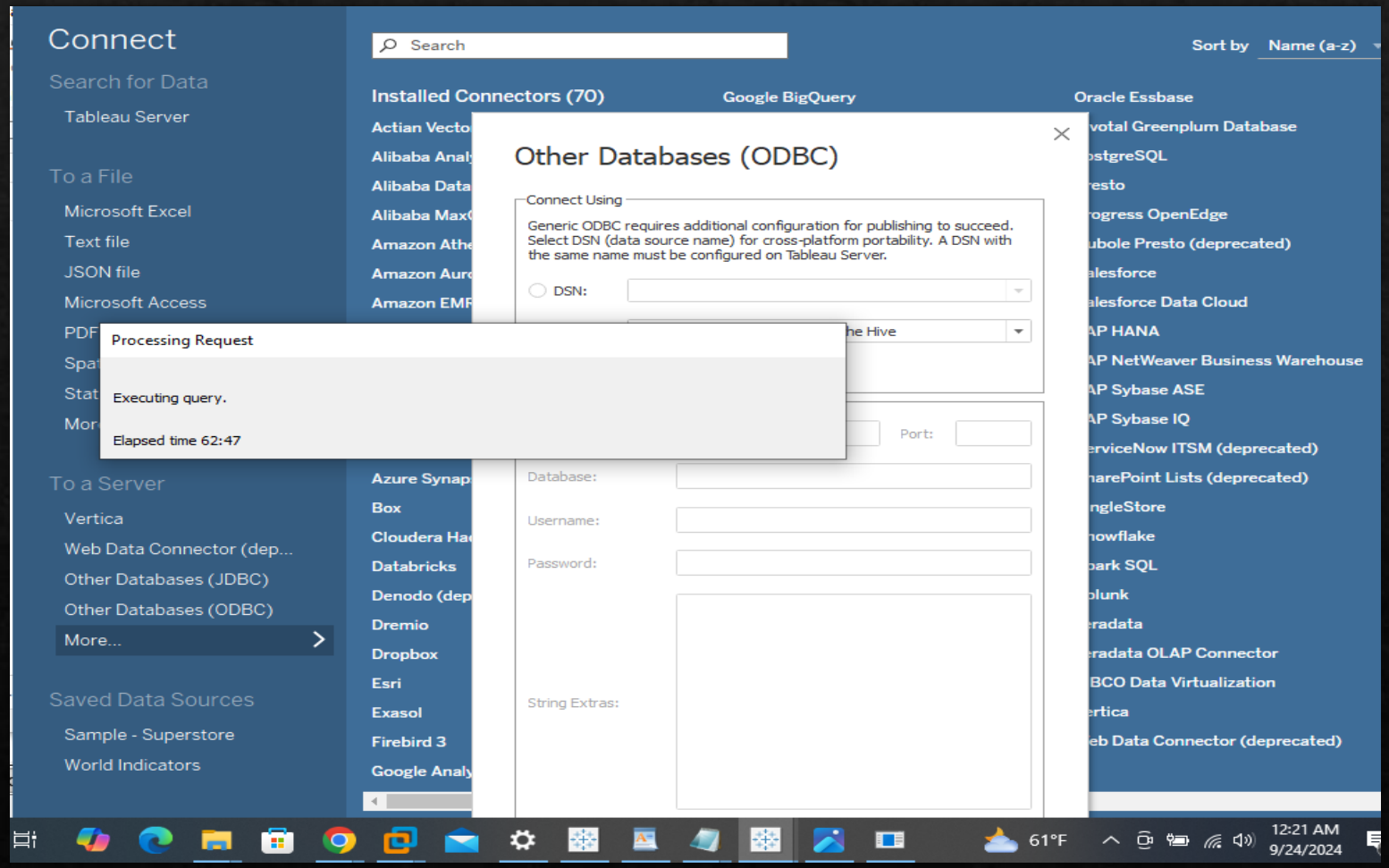
Tableau


Standard
Pages
Columns
SUM(count)
Rows
word
Filters
Sheet 1
word
a
count
file
for
hello
0
1
2
3
count
Marks
Automatic
Color
Size
Label
Detail
Tooltip

Tableau


Pages
Columns
Rows
Filters
Sheet 1
Marks
Circle
Color
Size
Label
Detail
Tooltip
SUM(count)
word
word
for
hello
a
count
file

Tableau


Connect
Search for Data
Tableau Server
To a File
Microsoft Excel
Text file
JSON file
Microsoft Access
To a Server
Vertica
Web Data Connector (dep...
Other Databases (JDBC)
Other Databases (ODBC)
More...
Saved Data Sources
Sample - Superstore
World Indicators
Search
Sort by
Name (a-z)
Installed Connectors (70)
Google BigQuery
Oracle Essbase
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:
Port:
Database:
Username:
Password:
String Extras:
Processing Request
Executing query.
Elapsed time 62:47
Box
Databricks
Dremio
Dropbox
Esri
Exasol
Firebird 3
61°F
12:21 AM
9/24/2024

Thank you!