KAFKA

INTRODUCTION TO KAFKA

APACHE KAFKA is a distributed event streaming platform capable of handling trillions of events a day. It was originally developed by LinkedIn and later open-sourced as part of the Apache Software Foundation. Kafka is used for building real-time data pipelines and streaming applications. It is horizontally scalable, fault-tolerant, and extremely fast.

KEY COMPONENTS OF KAFKA:

- **PRODUCER:** Sends records to a Kafka topic.
- **CONSUMER:** Reads records from a Kafka topic.
- **BROKER:** A Kafka server that stores data and serves clients.
- **TOPIC:** A category or feed name to which records are sent.
- **PARTITION:** A topic is divided into partitions for parallelism.
- **ZOOKEEPER:** Manages and coordinates Kafka brokers.

WORKING WITH KAFKA USING SPARK

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python, and R, and an optimized engine that supports general execution graphs.

SPARK STREAMING is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.

To work with Kafka using Spark, you typically use the **spark-streaming-kafka** library.

SPARK STREAMING ARCHITECTURE

Spark Streaming follows a micro-batch architecture. It divides the incoming data stream into small batches, which are then processed by the Spark engine to generate the final stream of results in batches.

KEY COMPONENTS:

- **INPUT DSTREAM:** Represents the stream of input data received from a source.
- **RECEIVER:** Receives data from a source and stores it in Spark's memory.
- **DRIVER:** The main program that runs the Spark Streaming application.
- **EXECUTOR:** Runs tasks that make up the application and returns results to the driver.

SPARK STREAMING APIS

Spark Streaming provides a high-level abstraction called DStream (Discretized Stream), which represents a continuous stream of data.

EXAMPLE OF CREATING A DSTREAM FROM KAFKA:

```
import org.apache.spark.streaming._
import org.apache.spark.streaming.kafka._
val ssc = new StreamingContext(sc, Seconds(1))
val kafkaStream = KafkaUtils.createStream(ssc, zkQuorum, groupId, topics)
val lines = kafkaStream.map(_._2)
```

BUILDING STREAM PROCESSING APPLICATION WITH SPARK

To build a stream processing application with Spark, follow these steps:

1. **SET UP THE ENVIRONMENT:** Install Spark and Kafka.
2. **CREATE A SPARK STREAMING CONTEXT:** Define the batch interval.
3. **DEFINE THE INPUT SOURCES:** Use Kafka as the input source.
4. **PROCESS THE DATA:** Apply transformations and actions on the DStream.
5. **OUTPUT THE RESULTS:** Store the results in a database or file system.

SETTING UP KAFKA PRODUCER AND CONSUMER

KAFKA PRODUCER EXAMPLE IN JAVA:

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
KafkaProducer<String, String> producer = new KafkaProducer<>(props);
ProducerRecord<String, String> record = new ProducerRecord<>("my-topic", "key", "value");
producer.send(record);
producer.close();
```

KAFKA CONSUMER EXAMPLE IN JAVA:

```
Properties props = new Properties();
props.put("bootstrap.servers", "localhost:9092");
props.put("group.id", "test-group");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
```

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("my-topic"));
while (true) {
   ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
   for (ConsumerRecord<String, String> record : records) {
      System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());
   }
}
```

KAFKA CONNECT API

KAFKA CONNECT is a framework for connecting Kafka with external systems such as databases, key-value stores, search indexes, and file systems. It makes it simple to define connectors that move large collections of data into and out of Kafka.

EXAMPLE OF A KAFKA CONNECT CONFIGURATION FOR A JDBC SOURCE:

```
{
  "name": "jdbc-source",
  "config": {
    "connector.class": "io.confluent.connect.jdbc.JdbcSourceConnector",
    "tasks.max": "1",
    "connection.url": "jdbc:mysql://localhost:3306/mydb",
    "connection.user": "user",
    "connection.password": "password",
    "table.whitelist": "mytable",
    "mode": "incrementing",
    "incrementing.column.name": "id",
    "topic.prefix": "jdbc-"
  }
}
```

This configuration will stream data from the **mytable** table in a MySQL database to a Kafka topic prefixed with **jdbc.**