Kafka Word Count Project for GitHub

This document contains all the necessary files and instructions to set up a complete Kafka Streams Word Count project. You can use this to create the project structure on your local machine and push it to a new GitHub repository.

1. Project Directory and File Structure

First, create the following directory and file structure on your local machine.

```
kafka-word-count/
├── src/
│   └── main/
│       └── java/
│           └── com/
│               └── example/
│                   └── WordCount.java
├── pom.xml
└── README.md
```

2. File Contents

Copy and paste the following content into the corresponding files in the directory structure you just created.

pom.xml

This file handles the project dependencies and includes the maven-assembly-plugin to create a single, runnable "fat" JAR.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.example</groupId>
    <artifactId>kafka-word-count</artifactId>
    <version>1.0-SNAPSHOT</version>
```

```
<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <kafka.version>4.0.0</kafka.version>
</properties>

<dependencies>
  <!-- Kafka Streams dependency -->
  <dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-streams</artifactId>
    <version>${kafka.version}</version>
  </dependency>

  <!-- Kafka Clients dependency -->
  <dependency>
    <groupId>org.apache.kafka</groupId>
    <artifactId>kafka-clients</artifactId>
    <version>${kafka.version}</version>
  </dependency>

  <!-- Added SLF4J simple logger to fix the SLF4J warning -->
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.36</version>
  </dependency>
</dependencies>

<build>
  <plugins>
    <!-- Maven Compiler Plugin -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.11.0</version>
      <configuration>
        <source>${maven.compiler.source}</source>
```

```
            <target>${maven.compiler.target}</target>
        </configuration>
      </plugin>
      <!-- Maven Assembly Plugin to create an executable fat JAR -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-assembly-plugin</artifactId>
        <version>3.6.0</version>
        <configuration>
          <descriptorRefs>
            <descriptorRef>jar-with-dependencies</descriptorRef>
          </descriptorRefs>
          <archive>
            <manifest>
              <mainClass>com.example.WordCount</mainClass>
            </manifest>
          </archive>
        </configuration>
        <executions>
          <execution>
            <id>make-assembly</id>
            <phase>package</phase>
            <goals>
              <goal>single</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

WordCount.java

This is the main application code that implements the Kafka Streams topology for word counting. The BOOTSTRAP_SERVERS_CONFIG and STATE_DIR_CONFIG are set explicitly to resolve previous errors. **Remember to update kafka2:9092,kafka3:9092 with your actual hostnames if they are different from the example.**

```
package com.example;

import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;
import org.apache.kafka.streams.kstream.KStream;
import org.apache.kafka.streams.kstream.Materialized;
import org.apache.kafka.streams.kstream.Produced;

import java.util.Arrays;
import java.util.Properties;

/**
 * A simple Kafka Streams application that reads lines of text from an input topic,
 * counts the occurrences of each word, and writes the word counts to an output
topic.
 */
public class WordCount {

    public static void main(String[] args) {
        // Step 1: Configure the Kafka Streams application
        Properties props = new Properties();
        // The unique application ID. This is important for state management.
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-wordcount-app");
        // The bootstrap servers for the 3-node cluster.
        // Using 'kafka-199' as the hostname for the first node.
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG,
"kafka-199:9092,kafka-200:9092,kafka-201:9092");
        // Default serializers and deserializers for keys and values
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG,
Serdes.String().getClass());
        // CRITICAL FIX: Explicitly set the state directory to avoid permissions issues
        props.put(StreamsConfig.STATE_DIR_CONFIG, "./kafka-streams-state");

        // Step 2: Define the stream processing topology
```

```
        StreamsBuilder builder = new StreamsBuilder();

        // Read the input stream from the source topic
        KStream<String, String> source = builder.stream("streams-plaintext-input");

        // Process the stream:
        // 1. Convert all text to lowercase
        // 2. Split lines into individual words
        // 3. Group by the new word key
        // 4. Count the occurrences of each word
        source.flatMapValues(value -> Arrays.asList(value.toLowerCase().split("\\W+")))
              .groupBy((key, word) -> word)
              .count(Materialized.as("counts-store"))
              .toStream()
              .to("streams-wordcount-output", Produced.with(Serdes.String(),
Serdes.Long()));

        // Step 3: Build the topology and start the Kafka Streams application
        Topology topology = builder.build();
        KafkaStreams streams = new KafkaStreams(topology, props);

        // Add a shutdown hook to gracefully close the application
        Runtime.getRuntime().addShutdownHook(new Thread(streams::close));

        System.out.println("Starting the Kafka Streams WordCount application...");
        streams.start();
        System.out.println("Application started successfully.");
    }
}
```

README.md

This file provides a clean, user-friendly guide for anyone who clones your repository. It summarizes the necessary steps and commands.

```
# Kafka Streams Word Count Example

This is a simple Java application that uses the Kafka Streams library to count words from a source topic and write the results to a destination topic.
```

Prerequisites

- Java 17 or later
- Maven
- A running Apache Kafka cluster (version 4.0.0 or later)

How to Run

Step 1: Create Kafka Topics

First, you need to create the input and output topics on your Kafka cluster. Run these commands from one of your Kafka nodes. Remember to replace `kafka-199:9092` with the correct bootstrap server address if yours is different.

```bash
# Create the input topic
sudo -u kafka /usr/local/kafka/bin/kafka-topics.sh --create \
  --topic streams-plaintext-input \
  --partitions 3 \
  --replication-factor 3 \
  --bootstrap-server kafka-199:9092

# Create the output topic
sudo -u kafka /usr/local/kafka/bin/kafka-topics.sh --create \
  --topic streams-wordcount-output \
  --partitions 3 \
  --replication-factor 3 \
  --bootstrap-server kafka-199:9092
```

Step 2: Build the Application

Navigate to the project root directory and use Maven to compile and package the application into a single executable JAR.

mvn clean package

This will create a JAR file named

kafka-word-count-1.0-SNAPSHOT-jar-with-dependencies.jar in the target/ directory.

Step 3: Run the Application

Execute the JAR file to start the Kafka Streams application.

```
java -jar target/kafka-word-count-1.0-SNAPSHOT-jar-with-dependencies.jar
```

Step 4: Produce Messages (Input)

In a new terminal, use the console producer to send lines of text to the input topic.

```
sudo -u kafka /usr/local/kafka/bin/kafka-console-producer.sh \
  --topic streams-plaintext-input \
  --bootstrap-server kafka-199:9092
```

Step 5: Consume Results (Output)

In another terminal, use the console consumer to view the word counts being produced by the application.

```
sudo -u kafka /usr/local/kafka/bin/kafka-console-consumer.sh \
  --topic streams-wordcount-output \
  --bootstrap-server kafka-199:9092 \
  --from-beginning \
  --property print.key=true \
  --property
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer \
  --property
value.deserializer=org.apache.kafka.common.serialization.LongDeserializer
```