Introduction

In this exercise, you will run a Java client application that produces messages to and consumes messages from an Apache Kafka® cluster.

Prerequisites

This guide assumes that you already have:

Gradle installed

Java 11 installed and configured as the current java version for the environment

Later in this exercise you will set up a new Kafka cluster or connect to an existing one.

Create Project

Create a new directory anywhere you'd like for this project:

```
mkdir kafka-java-getting-started && cd kafka-java-getting-started
```

Create the following Gradle build file for the project, named `build.gradle`:

```
buildscript {

    repositories {

        mavenCentral()

    }

    dependencies {

        classpath "com.github.jengelman.gradle.plugins:shadow:4.0.3"

    }

}



plugins {
    id "java"

    id "idea"

    id "eclipse"

}


sourceCompatibility = "1.11"

targetCompatibility = "1.11"

version = "0.0.1"


repositories {

    mavenCentral()


    maven {

        url "https://packages.confluent.io/maven"

    }

}


apply plugin: "com.github.johnrengelman.shadow"


dependencies {

    implementation group: 'org.slf4j', name: 'slf4j-nop', version: '1.7.36'

    implementation group: 'org.apache.kafka', name: 'kafka-clients', version:
'3.1.0'

}


jar {

    manifest {
        attributes('Main-Class': 'examples.ProducerExample')

    }
}
```

Configuration

Paste the following configuration data into a file at `getting-started.properties`

The below configuration includes the required settings for a connection to bootstrap servers.

```
bootstrap.servers= localhost:9092


# Required for correctness in Apache Kafka clients prior to 2.6

client.dns.lookup=use_all_dns_ips


# Best practice for Kafka producer to prevent data loss

acks=all

key.serializer=org.apache.kafka.common.serialization.StringSerializer

value.serializer=org.apache.kafka.common.serialization.StringSerializer

key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
```

Create Topic

Events in Kafka are organized and durably stored in named topics. Topics have parameters that determine the performance and durability guarantees of the events that flow through them.

Create a new topic, `purchases`, which we will use to produce and consume events.

```
bin/kafka-topics.sh --create --bootstrap-server localhost:9092 --replication-factor 1 --
partitions 1--topic purchases
```

Build Producer

Create a directory for the Java files in this project:

```
mkdir -p src/main/java/examples
```

Paste the following Java code into a file located at `src/main/java/examples/ProducerExample.java`

```
package examples;


import org.apache.kafka.clients.producer.*;

import java.io.*;

import java.nio.file.*;

import java.util.*;


public class ProducerExample {


    public static void main(final String[] args) throws IOException {

        if (args.length != 1) {

            System.out.println("Please provide the configuration file path as
a command line argument");

            System.exit(1);

        }


        // Load producer configuration settings from a local file

        final Properties props = loadConfig(args[0]);

        final String topic = "purchases";


        String[] users = { "eabara", "jsmith", "sgarcia", "jbernard",
"htanaka", "awalther" };
```

```
        String[] items = { "book", "alarm clock", "t-shirts", "gift card",
"batteries" };

        Producer<String, String> producer = new KafkaProducer<>(props);


        final Long numMessages = 10L;

        for (Long i = 0L; i < numMessages; i++) {

            Random rnd = new Random();

            String user = users[rnd.nextInt(users.length)];

            String item = items[rnd.nextInt(items.length)];


            producer.send(

              new ProducerRecord<>(topic, user, item),

              (event, ex) -> {

                  if (ex != null)

                      ex.printStackTrace();

                  else

                      System.out.printf("Produced event to topic %s: key = %-
10s value = %s%n", topic, user, item);

              });

        }


        producer.flush();

        System.out.printf("%s events were produced to topic %s%n",
numMessages, topic);

        producer.close();

    }
    /**
     * We'll reuse this function to load properties from the Consumer as well
     */
    public static Properties loadConfig(final String configFile) throws
IOException {
        if (!Files.exists(Paths.get(configFile))) {
            throw new IOException(configFile + " not found.");
        }
        final Properties cfg = new Properties();
        try (InputStream inputStream = new FileInputStream(configFile)) {
            cfg.load(inputStream);
        }
        return cfg;
    }
}
```

You can test the code before preceding by compiling with:

```
gradle build
```

And you should see:

```
BUILD SUCCESSFUL
```

Build Consumer

Paste the following Java code into a file located at `src/main/java/examples/ConsumerExample.java`

```
package examples;


import org.apache.kafka.clients.consumer.*;
import java.time.Duration;

import java.util.Arrays;

import java.util.Properties;


public class ConsumerExample {


    public static void main(final String[] args) throws Exception {

        if (args.length != 1) {

            System.out.println("Please provide the configuration file path as
a command line argument");

            System.exit(1);

        }


        final String topic = "purchases";


        // Load consumer configuration settings from a local file

        final Properties props = ProducerExample.loadConfig(args[0]);


        // Add additional properties.

        props.put(ConsumerConfig.GROUP_ID_CONFIG, "kafka-java-getting-
started");

        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");


        // Add additional required properties for this consumer app

        final Consumer<String, String> consumer = new KafkaConsumer<>(props);

        consumer.subscribe(Arrays.asList(topic));


        try {
            while (true) {

                ConsumerRecords<String, String> records =
consumer.poll(Duration.ofMillis(100));

                for (ConsumerRecord<String, String> record : records) {

                    String key = record.key();

                    String value = record.value();

                    System.out.println(

                        String.format("Consumed event from topic %s: key = %-
10s value = %s", topic, key, value));

                }

            }

        } finally {

            consumer.close();

        }

    }
}
```

Once again, you can compile the code before preceding by with:

```
gradle build
```

And you should see:

```
BUILD SUCCESSFUL
```

Produce Events

To build a JAR that we can run from the command line, first run:

```
gradle shadowJar
```

And you should see:

```
BUILD SUCCESSFUL
```

Run the following command to build and execute the producer application, which will produce some random data events to the `purchases` topic.

```
java -cp “build/libs/kafka-java-getting-started-0.0.1.jar;
/path/to/kafka/kafka_2.12-3.1.0/libs/*” examples.ProducerExample getting-
started.properties
```

You should see output that resembles:

```
Produced event to topic purchases: key = awalther   value = t-shirts

Produced event to topic purchases: key = htanaka    value = t-shirts

Produced event to topic purchases: key = htanaka    value = batteries

Produced event to topic purchases: key = eabara     value = t-shirts

Produced event to topic purchases: key = htanaka    value = t-shirts

Produced event to topic purchases: key = jsmith     value = book

Produced event to topic purchases: key = awalther   value = t-shirts

Produced event to topic purchases: key = jsmith     value = batteries

Produced event to topic purchases: key = jsmith     value = gift card

Produced event to topic purchases: key = eabara     value = t-shirts
10 events were produced to topic purchases
```

Consume Events

From another terminal, run the following command to run the consumer application which will read the events from the `purchases` topic and write the information to the terminal.

Re-run the producer to see more events, or feel free to modify the code as necessary to create more or different events.

```
java -cp “build/libs/kafka-java-getting-started-0.0.1.jar;
/path/to/kafka/kafka_2.12-3.1.0/libs/*” examples.ConsumerExample getting-
started.properties
```

The consumer application will start and print any events it has not yet consumed and then wait for more events to arrive. On startup of the consumer, you should see output that resembles the below. Once you are done with the consumer, press `ctrl-c` to terminate the consumer application.

```
Consumed event from topic purchases: key = awalther   value = t-shirts

Consumed event from topic purchases: key = htanaka    value = t-shirts
Consumed event from topic purchases: key = htanaka    value = batteries

Consumed event from topic purchases: key = eabara     value = t-shirts

Consumed event from topic purchases: key = htanaka    value = t-shirts

Consumed event from topic purchases: key = jsmith     value = book

Consumed event from topic purchases: key = awalther   value = t-shirts

Consumed event from topic purchases: key = jsmith     value = batteries

Consumed event from topic purchases: key = jsmith     value = gift card
Consumed event from topic purchases: key = eabara     value = t-shirts
```