Introduction

In this topic, you will run a Node.js client application that produces messages to and consumes messages from an Apache Kafka cluster.

Prerequisites

This exercise assumes that you have node.js version 16 or later installed. The exercise was last tested with Node version `20.11.1`.

Create Project

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

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

Then install the Apache Kafka NodeJS client library:

```
npm install @confluentinc/kafka-javascript
```

As well as dotenv:

```
npm install dotenv
```

This exercise was last tested with version `0.3.0` of `confluent-kafka-javascript`.

Create Topic

A topic is an immutable, append-only log of events. Usually, a topic is comprised of the same kind of events, e.g., in this guide we create a topic for retail purchases.

Create a new topic, `purchases`, which you 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

Let's create the Node.js producer application by pasting the following code into a file `producer.js`.

```
const Kafka = require('@confluentinc/kafka-javascript');

require("dotenv").config();


function createProducer(config, onDeliveryReport) {
  const producer = new Kafka.Producer(config);


  return new Promise((resolve, reject) => {
    producer
      .on('ready', () => resolve(producer))
      .on('delivery-report', onDeliveryReport)
      .on('event.error', (err) => {
        console.warn('event.error', err);
        reject(err);
      });
    producer.connect();
  });
}


async function produceExample() {
  const config = {
```

```
    // User-specific properties that you must set
    'bootstrap.servers': `localhost:${process.env.PLAINTEXT_PORTS}`,

    // Fixed properties
    'acks': 'all',

    // Needed for delivery callback to be invoked
    'dr_msg_cb': true
  }

  let topic = "purchases";

  let users = [ "eabara", "jsmith", "sgarcia", "jbernard", "htanaka",
"awalther" ];
  let items = [ "book", "alarm clock", "t-shirts", "gift card", "batteries" ];

  const producer = await createProducer(config, (err, report) => {
    if (err) {
      console.warn('Error producing', err)
    } else {
      const {topic, key, value} = report;
      let k = key.toString().padEnd(10, ' ');
      console.log(`Produced event to topic ${topic}: key = ${k} value =
${value}`);
    }
  });

  let numEvents = 10;
```

```
  for (let idx = 0; idx < numEvents; ++idx) {
    const key = users[Math.floor(Math.random() * users.length)];
    const value = Buffer.from(items[Math.floor(Math.random() * items.length)]);

    producer.produce(topic, -1, value, key);
  }

  producer.flush(10000, () => {
    producer.disconnect();
  });
}

produceExample()
  .catch((err) => {
    console.error(`Something went wrong:\n${err}`);
    process.exit(1);
  });
```

Create a `.env` file containing the appropriate configuration needed to connect to Kafka.

Refer to the `sample.env` file below to see which environment variables need to be set based on how Kafka is deployed.

```
# for local Kafka started via the Confluent CLI
PLAINTEXT_PORTS=value_here

# for Confluent Cloud with basic authentication
BOOTSTRAP_SERVERS=value_here
CLUSTER_API_KEY=value_here
CLUSTER_API_SECRET=value_here

# for Confluent Cloud with OAuth 2.0 authentication
BOOTSTRAP_SERVERS=value_here
OAUTH2_CLIENT_ID=value_here
OAUTH2_CLIENT_SECRET=value_here
OAUTH2_TOKEN_ENDPOINT_URL=value_here
OAUTH2_SCOPE=value_here
LOGICAL_CLUSTER_ID=value_here
IDENTITY_POOL_ID=value_here
```

```
# for an existing Kafka cluster
BOOTSTRAP_SERVERS=value_here
```

Build Consumer

Next, create the Node.js consumer application by pasting the following code into a file `consumer.js`.

```
const Kafka = require('@confluentinc/kafka-javascript');

require("dotenv").config();


function createConsumer(config, onData) {
  const consumer = new Kafka.KafkaConsumer(config, {'auto.offset.reset':
'earliest'});

  return new Promise((resolve, reject) => {
    consumer
     .on('ready', () => resolve(consumer))
     .on('data', onData);

    consumer.connect();
  });
};



async function consumerExample() {
  const config = {
    // User-specific properties that you must set
    'bootstrap.servers': `localhost:${process.env.PLAINTEXT_PORTS}`,
```

```
    // Fixed properties
    'group.id':          'kafka-nodejs-getting-started'
  }

  let topic = "purchases";

  const consumer = await createConsumer(config, ({key, value}) => {
    let k = key.toString().padEnd(10, ' ');
    console.log(`Consumed event from topic ${topic}: key = ${k} value =
${value}`);
  });

  consumer.subscribe([topic]);
  consumer.consume();

  process.on('SIGINT', () => {
    console.log('\nDisconnecting consumer ...');
    consumer.disconnect();
  });
}

consumerExample()
  .catch((err) => {
    console.error(`Something went wrong:\n${err}`);
    process.exit(1);
  });
```

Produce Events

Run the producer with the following command:

```
node producer.js
```

You should see output resembling this:

```
Produced event to topic purchases: key = jsmith     value = alarm clock
Produced event to topic purchases: key = htanaka    value = book
Produced event to topic purchases: key = eabara     value = batteries
Produced event to topic purchases: key = htanaka    value = t-shirts
Produced event to topic purchases: key = htanaka    value = t-shirts
Produced event to topic purchases: key = htanaka    value = gift card
Produced event to topic purchases: key = sgarcia    value = gift card
Produced event to topic purchases: key = jbernard   value = gift card
Produced event to topic purchases: key = awalther   value = alarm clock
Produced event to topic purchases: key = htanaka    value = book
```

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.

```
node consumer.js
```

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 resembling this:

```
Consumed event from topic purchases: key = jsmith     value = alarm clock
Consumed event from topic purchases: key = htanaka    value = book
```

```
Consumed event from topic purchases: key = eabara     value = batteries
Consumed event from topic purchases: key = htanaka    value = t-shirts
Consumed event from topic purchases: key = htanaka    value = t-shirts
Consumed event from topic purchases: key = htanaka    value = gift card
Consumed event from topic purchases: key = sgarcia    value = gift card
Consumed event from topic purchases: key = jbernard   value = gift card
Consumed event from topic purchases: key = awalther   value = alarm clock
Consumed event from topic purchases: key = htanaka    value = book
```

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

Once you are done with the consumer, enter `Ctrl-C` to terminate the consumer application.

Shut down Kafka when you are done with it:

```
confluent local kafka stop
```