Lab 7-2. Playing with Avro using Schema Registry

Open new terminal and go to following directory:

```
cd /headless/kafka-advanced/confluent-6.1.1
```

Make sure confluent services are running and let's send some Avro data to a Kafka topic. Although you would normally do this from one of your applications, we'll use a utility provided with Kafka to send the data without having to write any code. We direct it at our local Kafka cluster, tell it to write to the topic test, read each line of input as an Avro message, validate the schema against the Schema Registry at the specified URL, and finally indicate the format of the data.

```
./bin/kafka-avro-console-producer \
             --broker-list localhost:9092 --topic test \
             --property value.schema='{"type":"record","name":"myrecord","fields":
[{"name":"f1","type":"string"}]}'
```

Once started, the process will wait for you to enter messages, one per line, and will send them immediately when you hit the Enter key. Try entering a couple of messages:

```
{"f1": "value1"}
{"f1": "value2"}
{"f1": "value3"}
```

When you're done, use Ctrl+C to shut down the process.

Now we can check that the data was produced by using Kafka's console consumer process to read data from the topic. We point it at the same test topic, our ZooKeeper instance, tell it to decode each message using Avro using the same Schema Registry URL to look up schemas, and finally tell it to start from the beginning of the topic (by default the consumer only reads messages published after it starts).

```
./bin/kafka-avro-console-consumer --topic test \
             --zookeeper localhost:2181 \
             --from-beginning
```

You should see all the messages you created in the previous step written to the console in the same format.

The consumer does not exit after reading all the messages so it can listen for and process new messages as they are published. Try keeping the consumer running and repeating step 5 – you will see messages delivered to the consumer immediately after you hit Enter for each message in the producer.

When you're done, shut down the consumer with Ctrl+C.

Now let's try to produce data to the same topic using an incompatible schema. We'll run the producer with nearly the same command, but change the schema to expect plain integers.

```
./bin/kafka-avro-console-producer \
             --broker-list localhost:9092 --topic test \
             --property value.schema='{"type":"int"}'
```

Now if you enter an integer and hit enter, you should see an exception:

```
org.apache.kafka.common.errors.SerializationException: Error registering Avro schema:
"int"
Caused by:
io.confluent.kafka.schemaregistry.client.rest.exceptions.RestClientException: Schema
being registered is incompatible with the latest schema; error code: 409
       at
io.confluent.kafka.schemaregistry.client.rest.utils.RestUtils.httpRequest(RestUtils.java

       at
io.confluent.kafka.schemaregistry.client.rest.utils.RestUtils.registerSchema(RestUtils.j

       at
io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient.registerAndGetId(Cac

       at
io.confluent.kafka.schemaregistry.client.CachedSchemaRegistryClient.register(CachedSchem

       at
io.confluent.kafka.serializers.AbstractKafkaAvroSerializer.serializeImpl(AbstractKafkaAv

       at
io.confluent.kafka.formatter.AvroMessageReader.readMessage(AvroMessageReader.java:155)
       at kafka.tools.ConsoleProducer$.main(ConsoleProducer.scala:94)
       at kafka.tools.ConsoleProducer.main(ConsoleProducer.scala)
When the producer tried to send a message, it checked the schema with the Schema
Registry, which returned an error indicating the schema was invalid because it does
not preserve backwards compatibility (the default Schema Registry setting). The
console producer simply reports this error and exits, but your own applications could
handle the problem more gracefully. Most importantly, we've guaranteed no incompatible
data was published to Kafka.
```