Table of Contents

NOTE : PLEASE UPDATE THE CONFLUENT PATH TO YOUR PATH AS AND WHERE NEEDED

NOTE: win-scripts folder needs to be copied in every new project that is imported. Make the changes as and when mentioned for a particular project.

ALSO MAKE SURE TO STOP THE SERVER, BROKER, CONSUMERS BEFORE MOVING ON TO NEXT PROJECT

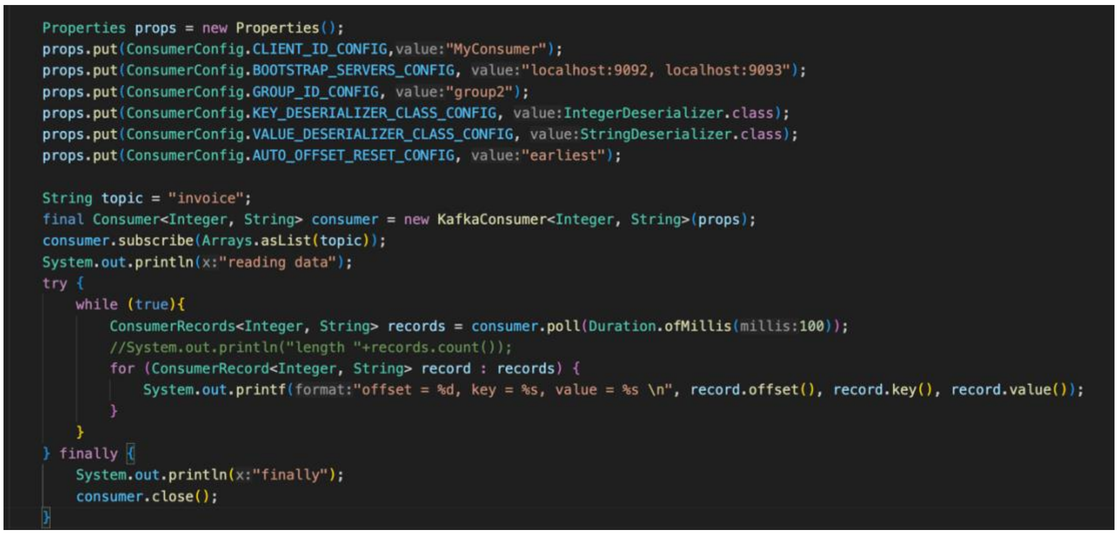
1. CONSUMER DEMO

1.1. Create a java maven quickstart project ConsumerDemo

1.2. Add kafka client dependency

```
<dependency>
        <groupId>org.apache.kafka</groupId>
         <artifactId>kafka-clients</artifactId>
        <version>3.5.1</version>
 </dependency>
```

1.3. Create a class AppConsumer as follows:



```
Properties props = new Properties();
props.put(ConsumerConfig.CLIENT_ID_CONFIG,value:"MyConsumer");
props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, value:"localhost:9092, localhost:9093");
props.put(ConsumerConfig.GROUP_ID_CONFIG, value:"group2");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, value:IntegerDeserializer.class);
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, value:StringDeserializer.class);
props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, value:"earliest");

String topic = "invoice";
final Consumer<Integer, String> consumer = new KafkaConsumer<Integer, String>(props);
consumer.subscribe(Arrays.asList(topic));
System.out.println(x:"reading data");
try {
    while (true){
        ConsumerRecords<Integer, String> records = consumer.poll(Duration.ofMillis(millis:100));
        //System.out.println("length "+records.count());
        for (ConsumerRecord<Integer, String> record : records) {
            System.out.printf(format:"offset = %d, key = %s, value = %s \n", record.offset(), record.key(), record.value());
        }
    }
} finally {
    System.out.println(x:"finally");
    consumer.close();
}
```

1.4. Open the FirstKafkaDemo created yesterday. Start the zookeeper, broker, create invoice topic and run the producer to produce 10 records. Modify the for loop to produce only 10 records

1.5. Now run this consumer class to be able to consume the data produced by the producer. This consumer will be running infinitely and will continue reading data if producer produces more data.

1.6. To set number of records to read at a time

```
props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, "10");
```

2. Consumer Read Messages

2.1. In the last section, we assigned a group id of group2 to the consumer.
2.2. Stop the consumer application, and re-run it. You will notice there are no messages displayed in the console. This is due to the fact that Kafka keeps track of consumer reads (consumer groups offsets) and therefore the consumer will not encounter the same messages twice.
2.3. If you change the group id to group1 and rerun the application it will display all the messages

3. Consumer Rebalance:

3.1. Moving partition ownership from one consumer to another is called a rebalance. Rebalances are important because they provide the consumer group with high availability and scalability.

3.2. Consumer rebalances happen for the following events:
3.2.1. Number of partitions change for any of the subscribed topics
3.2.2. A subscribed topic is created or deleted
3.2.3. An existing member of the consumer group is shutdown or fails
3.2.4. A new member is added to the consumer group

3.3. To see rebalancing in effect create below classes in the ConsumerDemo application:

3.3.1. Create a class as follows:

```
public class ConsumerRebalanceListenerImpl implements ConsumerRebalanceListener {

  private KafkaConsumer<String, String> consumer;
  private Map<TopicPartition, OffsetAndMetadata> currentOffsets = new HashMap<>();

  public ConsumerRebalanceListenerImpl(KafkaConsumer<String, String> consumer) {
    this.consumer = consumer;
  }

  public void addOffsetToTrack(String topic, int partition, long offset){
    currentOffsets.put(
        new TopicPartition(topic, partition),
        new OffsetAndMetadata(offset + 1, null));
  }

  @Override
  public void onPartitionsRevoked(Collection<TopicPartition> partitions) {
    System.out.println("onPartitionsRevoked callback triggered");
    System.out.println("Committing offsets: " + currentOffsets);

    consumer.commitSync(currentOffsets);
  }

  @Override
  public void onPartitionsAssigned(Collection<TopicPartition> partitions) {
    System.out.println("onPartitionsAssigned callback triggered ");
  }
```

```
    // this is used when we shut down our consumer gracefully
    public Map<TopicPartition, OffsetAndMetadata> getCurrentOffsets() {
        return currentOffsets;
    }
}
```

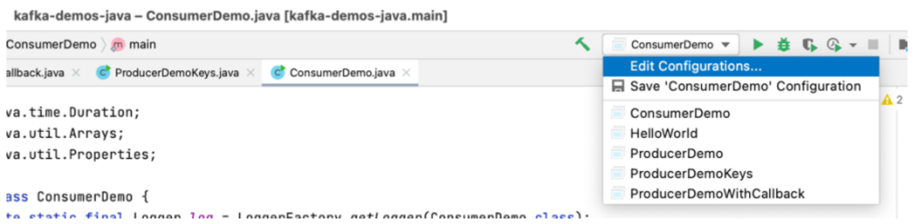
3.3.2. Create class ConsumerDemoRebalanceListener as follows:

```
public class ConsumerDemoRebalanceListener {

    public static void main(String[] args) {
        System.out.println("I am a Kafka Consumer with a Rebalance");

        String bootstrapServers = "127.0.0.1:9092";
        String groupId = "my-fifth-application";
        String topic = "invoice";

        // create consumer configs
        Properties properties = new Properties();
        properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        properties.setProperty(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

        // we disable Auto Commit of offsets
        properties.setProperty(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false");

        // create consumer
        KafkaConsumer<String, String> consumer = new KafkaConsumer<>(properties);

        ConsumerRebalanceListenerImpl listener = new ConsumerRebalanceListenerImpl(consumer);

        // get a reference to the current thread
        final Thread mainThread = Thread.currentThread();

        // adding the shutdown hook
        Runtime.getRuntime().addShutdownHook(new Thread() {
            public void run() {
                System.out.println("Detected a shutdown, let's exit by calling consumer.wakeup()...");
                consumer.wakeup();

                // join the main thread to allow the execution of the code in the main thread
                try {
                    mainThread.join();
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });

        try {
            // subscribe consumer to our topic(s)
            consumer.subscribe(Arrays.asList(topic), listener);

            // poll for new data
            while (true) {
                ConsumerRecords<String, String> records =
                        consumer.poll(Duration.ofMillis(100));

                for (ConsumerRecord<String, String> record : records) {
                    System.out.println("Key: " + record.key() + ", Value: " + record.value());
                    System.out.println("Partition: " + record.partition() + ", Offset:" + record.offset());

                    // we track the offset we have been committed in the listener
                    listener.addOffsetToTrack(record.topic(), record.partition(), record.offset());
                }

                // We commitAsync as we have processed all data and we don't want to block until the next .poll() call
                consumer.commitAsync();
            }
        } catch (WakeupException e) {
            System.out.println("Wake up exception!");
```

```
                // we ignore this as this is an expected exception when closing a consumer
            } catch (Exception e) {
                System.out.println("Unexpected exception " +e);
            } finally {
                try {
                    consumer.commitSync(listener.getCurrentOffsets()); // we must commit the offsets synchronously here
                } finally {
                    consumer.close();
                    System.out.println("The consumer is now gracefully closed.");
                }
            }
        }
    }
```

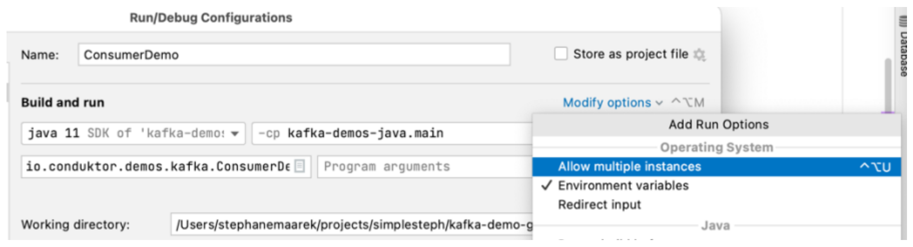
3.3.3. To see rebalances in action, launch another consumer by running the same consumer application.

3.3.3.1. Make sure you have configured to run multiple instances of the application in project setting:



```
va.time.Duration;
va.util.Arrays;
va.util.Properties;

ass ConsumerDemo {
```

3.3.3.2. Then, edit your configuration to allow for Multiple Instances



Name: ConsumerDemo

java 11 SDK of 'kafka-demos
-cp kafka-demos-java.main
io.conduktor.demos.kafka.ConsumerDe
Program arguments

Working directory: /Users/stephanemaarek/projects/simplesteph/kafka-demo-g

✓ Environment variables

3.3.3.3. Then apply and okay to save your changes. Now you can run your consumer multiples times.

4. Assign and Seek

4.1. In case you are looking to read specific messages from specific partitions, the .seek() and .assign() API may help you.

4.2. These APIs are also helpful to replay data from a specific offset.

4.3. To use these API, make the following changes:
   4.3.1. Remove the group.id from the consumer properties (we don't use consumer groups anymore)
   4.3.2. Remove the subscription to the topic
   4.3.3. Use consumer assign() and seek() APIs

4.4. Make sure the partition offset of partition 0 of the topic demo_java is at least 7. Produce a number of messages to the topic to achieve that.

4.5. Create a class and add below code in the main method as follows:

```
Logger log = LoggerFactory.getLogger(ConsumerDemoAssignSeek.class.getName());

String bootstrapServers = "127.0.0.1:9092";
String topic = "invoice";

// create consumer configs
Properties properties = new Properties();
properties.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
properties.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
properties.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
properties.setProperty(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
```

```
        // create consumer
        KafkaConsumer<String, String> consumer = new KafkaConsumer<String, String>(properties);

        // assign and seek are mostly used to replay data or fetch a specific message

        // assign
        TopicPartition partitionToReadFrom = new TopicPartition(topic, 0);
        long offsetToReadFrom = 7L;
        consumer.assign(Arrays.asList(partitionToReadFrom));

        // seek
        consumer.seek(partitionToReadFrom, offsetToReadFrom);

        int numberOfMessagesToRead = 5;
        boolean keepOnReading = true;
        int numberOfMessagesReadSoFar = 0;

        // poll for new data
        while(keepOnReading){
            ConsumerRecords<String, String> records =
                    consumer.poll(Duration.ofMillis(100));

            for (ConsumerRecord<String, String> record : records){
                numberOfMessagesReadSoFar += 1;
                log.info("Key: " + record.key() + ", Value: " + record.value());
                log.info("Partition: " + record.partition() + ", Offset:" + record.offset());
                if (numberOfMessagesReadSoFar >= numberOfMessagesToRead){
                    keepOnReading = false; // to exit the while loop
                    break; // to exit the for loop
                }
            }
        }

        log.info("Exiting the application");
    }
```

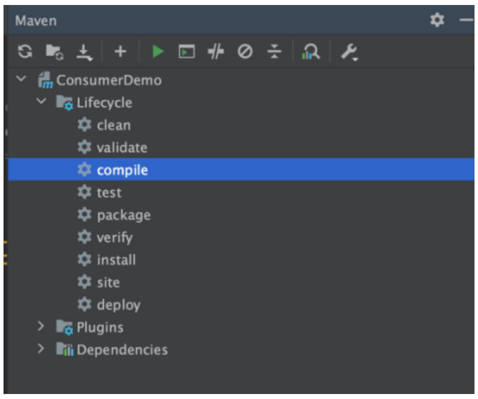
5. JSON and AVRO Consumer

NOTE: AVRO will not work for windows

5.1. Copy the schema and data for json and avro schema from the JsonAvroToPOJO project within the ConsumerDemo project

5.2. Add the avro and json dependencies and plugins from the JsonAvroToPOJO project

5.3. Expand the Maven tab and run the lifecycle compile phase . It will generate respective POJOS from the schemas. Check avro and types folder for respective generated POJOS



5.4. Create win-scripts folder.

5.5. Create valid-user-topic-create.cmd with below lines within win-scripts folder

```
set KAFKA_HOME=<path to confluent>/confluent-7.5.0
        %KAFKA_HOME%\bin\windows\kafka-topics .bat --create --bootstrap-server localhost:9092 --replication-factor 3 --
partitions 3 --topic valid-user
```

5.6. Create invalid-user-topic-create..cmd with below lines within win-scripts folder

```
set KAFKA_HOME=<path to confluent>/confluent-7.5.0
%KAFKA_HOME%\bin\windows\kafka-topics .bat --create --bootstrap-server localhost:9092 --topic invalid-user --partitions 5 --
replication-factor 3
```

5.7. Create invalid-user-consumer.cmd with below lines within win-scripts folder

```
set KAFKA_HOME=<path to confluent>/confluent-7.5.0
        %KAFKA_HOME%\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --from-beginning --
topic invalid-user
```

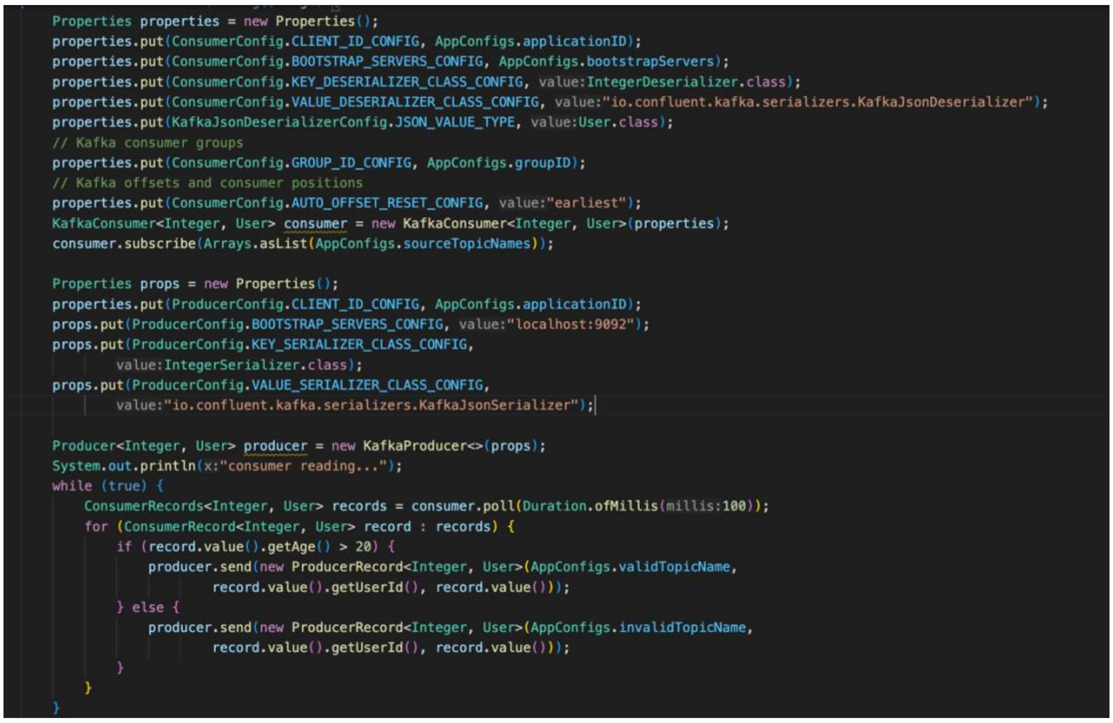
5.8. Create valid-user-consumer.cmd with below lines within win-scripts folder

```
set KAFKA_HOME=<path to confluent>/confluent-7.5.0
        %KAFKA_HOME%\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --from-beginning --
topic valid-user
```

5.9. Create AppCOnfigs as follows:

```
public final static String applicationID = "UserValidator";
public final static String bootstrapServers = "localhost:9092,localhost:9093";
public final static String groupID = "UserValidatorGroup";
public final static String[] sourceTopicNames = {"user-topic"};
public final static String validTopicName = "valid-user";
public final static String invalidTopicName = "invalid-user";
```

5.10. Create a UserValidator class to consume data from user-topic created in previous project JsonAvroToPOJO. Validate the user for valid and invalid users and create producers accordingly as shown in the below pic



```
Properties properties = new Properties();
properties.put(ConsumerConfig.CLIENT_ID_CONFIG, AppConfigs.applicationID);
properties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, AppConfigs.bootstrapServers);
properties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, value:IntegerDeserializer.class);
properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, value:"io.confluent.kafka.serializers.KafkaJsonDeserializer");
properties.put(KafkaJsonDeserializerConfig.JSON_VALUE_TYPE, value:User.class);
// Kafka consumer groups
properties.put(ConsumerConfig.GROUP_ID_CONFIG, AppConfigs.groupID);
// Kafka offsets and consumer positions
properties.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, value:"earliest");
KafkaConsumer<Integer, User> consumer = new KafkaConsumer<Integer, User>(properties);
consumer.subscribe(Arrays.asList(AppConfigs.sourceTopicNames));

Properties props = new Properties();
properties.put(ProducerConfig.CLIENT_ID_CONFIG, AppConfigs.applicationID);
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, value:"localhost:9092");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
        value:IntegerSerializer.class);
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
        value:"io.confluent.kafka.serializers.KafkaJsonSerializer");

Producer<Integer, User> producer = new KafkaProducer<>(props);
System.out.println(x:"consumer reading...");
while (true) {
    ConsumerRecords<Integer, User> records = consumer.poll(Duration.ofMillis(millis:100));
    for (ConsumerRecord<Integer, User> record : records) {
        if (record.value().getAge() > 20) {
            producer.send(new ProducerRecord<Integer, User>(AppConfigs.validTopicName,
                    record.value().getUserId(), record.value()));
        } else {
            producer.send(new ProducerRecord<Integer, User>(AppConfigs.invalidTopicName,
                    record.value().getUserId(), record.value()));
        }
    }
}
```

5.11. Run valid and invalid consumers cmd file to check whether data was produced on the respective output.

5.12. Make sure to run the zookeeper, kafka brokers and produce user data by running the JsonAvroToPOJO project

5.13. Then run the UserValidator class to consume the messages and send to respective topics after processing

5.14. For AVRO below is the code:

```
  Properties props = new Properties();

props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
props.put(ConsumerConfig.GROUP_ID_CONFIG, "avrogroup");
props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");
props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, "io.confluent.kafka.serializers.KafkaAvroDeserializer");
props.put("schema.registry.url", "http://localhost:8081");
props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

String topic = "emp-topic";
final Consumer<String, GenericRecord> consumer = new KafkaConsumer<String, GenericRecord>(props);
consumer.subscribe(Arrays.asList(topic));
try {
  while (true) {
    ConsumerRecords<String, GenericRecord> records = consumer.poll(100);
    for (ConsumerRecord<String, GenericRecord> record : records) {
      System.out.printf("offset = %d, key = %s, value = %s \n", record.offset(), record.key(), record.value());
    }
  }
} finally {
  consumer.close();
}
```

6. REFERENCES

https://www.conduktor.io/kafka/complete-kafka-consumer-with-java/

https://www.conduktor.io/kafka/kafka-consumer-groups-and-consumer-offsets/

https://www.conduktor.io/kafka/delivery-semantics-for-kafka-consumers/

https://www.conduktor.io/kafka/java-consumer-rebalance-listener/#Consumer-Rebalance-Listeners-Example-1

https://kafka.apache.org/30/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html

https://www.oreilly.com/library/view/kafka-the-definitive/9781491936153/ch04.html

https://kafka.apache.org/24/javadoc/index.html?org/apache/kafka/clients/consumer/ConsumerRebalanceListener.html

https://kafka.apache.org/30/javadoc/org/apache/kafka/clients/consumer/KafkaConsumer.html#commitSync()

https://medium.com/@rramiz.rraza/kafka-programming-different-ways-to-commit-offsets-7bcd179b225a