Consumer Group Example

In this Lab

Objective: Learn more about this particular techonology.

Successful Outcome: Simply following along with the step written below and complete each before moving to the next.

Lab Files: `xx`

Steps

Consumer group is a multi-threaded or multi-machine consumption from Kafka topics.

☑

Consumer Group

- Consumers can join a group by using the same "group.id".
- The maximum parallelism of a group is that the number of consumers in the group \<= no of partitions.
- Kafka assigns the partitions of a topic to the consumer in a group, so that each partition is consumed by exactly one consumer in the group.
- Kafka guarantees that a message is only ever read by a single consumer in the group.
- Consumers can see the message in the order they were stored in the log.

☑

Re-balancing of a Consumer

Adding more processes/threads will cause Kafka to re-balance. If any consumer or broker fails to send heartbeat to ZooKeeper, then it can be re-configured via the Kafka cluster. During this re-balance, Kafka will assign available partitions to the available threads, possibly moving a partition to another process.

```
import  java.util.Properties;
import  java.util.Arrays;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import  org.apache.kafka.clients.consumer.ConsumerRecord;
public class ConsumerGroup {
        public  static  void  main(String[]  args)  throws  Exception  {
                 if(args.length  <  2)
                 {
                         System.out.println("Usage:  consumer  <topic
<groupname>"); return;
                 }
                 String topic = args[0].toString();
                 String group = args[1].toString();
                 Properties props = new Properties();
                 props.put("bootstrap.servers", "localhost:9092");
                 props.put("group.id", group);
                 props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
                 props.put("session.timeout.ms", "30000");
                 props.put("key.deserializer", "org.apache.kafka.common.serializa-
tion.StringDeserializer");
                 props.put("value.deserializer",
"org.apache.kafka.common.serializa- tion.StringDeserializer");
                 KafkaConsumer<String, String> consumer = new
KafkaConsumer<String, String>(props);
                 consumer.subscribe(Arrays.asList(topic));

                 System.out.println("Subscribed to topic " + topic);
                 int i = 0;
                 while (true) {
                          ConsumerRecords<String, String> records =
consumer.poll(100);
                          for (ConsumerRecord<String, String> record : records)
                                   System.out.printf("offset = %d, key = %s,
value = %s\n", record.offset(), record.key(), record.value());
                  }
         }
}
```

☑

Compilation

```
javac -cp “/path/to/kafka/kafka_2.11-0.9.0.0/libs/*" ConsumerGroup.java
```

☑

Execution

```
>> java -cp “/path/to/kafka/kafka_2.11-0.9.0.0/libs/*":. ConsumerGroup
<topic-name> my-group
>> java -cp "/home/bala/Workspace/kafka/kafka_2.11-0.9.0.0/libs/*":.
ConsumerGroup <topic-name> my-group
```

Here we have created a sample group name as "my-group" with two consumers. Similarly, you can create your group and number of consumers in the group.

☑

Input

Open producer CLI and send some messages like –

```
Test consumer group 01
Test consumer group 02
```

✅

Output of the First Process

```
Subscribed to topic Hello-kafka
offset = 3, key = null, value = Test consumer group 01
```

Now hopefully you would have understood SimpleConsumer and ConsumeGroup by using the Java client demo. Now you have an idea about how to send and receive messages using a Java client. Let us continue Kafka integration with big data technologies in the next chapter.

Results

You are finished! Great job!