**Custom Partitioner Example**

Let's assume that we have a retail site that consumers can use to order products anywhere in the world. Based on usage, we know that most consumers are in either the United States or India. We want to partition our application to send orders from the US or India to their own respective consumers, while orders from anywhere else will go to a third consumer.

To start, we'll create a CountryPartitioner that implements the org.apache.kafka.clients.producer.Partitioner interface. We must implement the following methods:

1. Kafka will call **configure()** when we initialize the Partitioner class, with a Map of configuration properties. This method initializes functions specific to the application's business logic, such as connecting to a database. In this case we want a fairly generic partitioner that takes countryName as a property. We can then use configProperties.put("partitions.0","USA") to map the flow of messages to partitions. In the future we can use this format to change which countries get their own partition.
2. The Producer API calls **partition()** once for every message. In this case we'll use it to read the message and parse the name of the country from the message. If the name of the country is in the countryToPartitionMap, it will return partitionId stored in the Map. If not, it will hash the value of the country and use it to calculate which partition it should go to.
3. We call **close()** to shut down the partitioner. Using this method ensures that any resources acquired during initialization are cleaned up during shutdown.

Note that when Kafka calls configure(), the Kafka producer will pass all the properties that we've configured for the producer to the Partitioner class. It is essential that we read only those properties that start with partitions., parse them to get the partitionId, and store the ID in countryToPartitionMap.

Below is our custom implementation of the Partitioner interface.

```
package com.jpmc;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;
```

```
package com.jpmc;

import java.util.HashMap;
import java.util.Map;

import org.apache.kafka.clients.producer.Partitioner;
import org.apache.kafka.common.Cluster;
```

```
public class CountryPartitioner implements Partitioner {
   private static Map<String,Integer> countryToPartitionMap;

   // This method will gets called at the start, you should use it to do one time startup activity
   public void configure(Map<String, ?> configs) {
      System.out.println("Inside CountryPartitioner.configure " + configs);
      countryToPartitionMap = new HashMap<String, Integer>();
      for(Map.Entry<String,?> entry: configs.entrySet()){
         if(entry.getKey().startsWith("partitions.")){
            String keyName = entry.getKey();
            String value = (String)entry.getValue();
            System.out.println( keyName.substring(11));
            int paritionId = Integer.parseInt(keyName.substring(11));
            countryToPartitionMap.put(value,paritionId);
         }
      }
   }

   //This method will get called once for each message
   public int partition(String topic, Object key, byte[] keyBytes, Object value, byte[]
valueBytes,
                  Cluster cluster) {
      List partitions = cluster.availablePartitionsForTopic(topic);
      String valueStr = (String)value;
      String countryName = ((String) value).split(":")[0];
      if(countryToPartitionMap.containsKey(countryName)){
         //If the country is mapped to particular partition return it
         return countryToPartitionMap.get(countryName);
      }else {
         //If no country is mapped to particular partition distribute between remaining
partitions
         int noOfPartitions = cluster.topics().size();
         return  value.hashCode()%noOfPartitions + countryToPartitionMap.size() ;
      }
   }

   // This method will get called at the end and gives your partitioner class chance to cleanup
   public void close() {}
}
```

```
package com.jpmc;

import java.util.Properties;
```

```java
import java.util.Scanner;

import org.apache.kafka.clients.producer.Callback;
import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;

public class SimpleProducer {

        private static Scanner in;
   public static void main(String[] argv)throws Exception {
      if (argv.length != 1) {
         System.err.println("Please specify 1 parameters ");
         System.exit(-1);
      }
      String topicName = argv[0];
      in = new Scanner(System.in);
      System.out.println("Enter message(type exit to quit)");

      //Configure the Producer
      Properties configProperties = new Properties();
      configProperties.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,"localhost:9093");

configProperties.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.co
mmon.serialization.ByteArraySerializer");

configProperties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,"org.apache.kafka.
common.serialization.StringSerializer");


configProperties.put(ProducerConfig.PARTITIONER_CLASS_CONFIG,CountryPartitioner.class
.getCanonicalName());
      configProperties.put("partitions.0","USA");
      configProperties.put("partitions.1","India");

      org.apache.kafka.clients.producer.Producer          producer          =          new
KafkaProducer(configProperties);
      String line = in.nextLine();
      while(!line.equals("exit")) {
         ProducerRecord<String, String> rec = new ProducerRecord<String, String>(topicName,
line);

         producer.send(rec, new Callback() {
            public void onCompletion(RecordMetadata metadata, Exception exception) {
                System.out.println("Message sent to topic ->" + metadata.topic()+ " ,parition->" +
metadata.partition() +" stored at offset->" + metadata.offset());
              }
          });
          line = in.nextLine();
      }
      in.close();
      producer.close();
}

}


package com.jpmc;

import java.util.Arrays;
import java.util.Collection;
import java.util.Properties;
import java.util.Scanner;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRebalanceListener;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.TopicPartition;

public class SimpleConsumer {

          private static Scanner in;
   private static boolean stop = false;

   public static void main(String[] argv) throws Exception {
      if (argv.length != 2) {
         System.err.printf("Usage: %s <topicName> <groupId>\n",
                  SimpleConsumer.class.getSimpleName());
         System.exit(-1);
      }
      in = new Scanner(System.in);
      String topicName = argv[0];
      String groupId = argv[1];

      ConsumerThread consumerThread = new ConsumerThread(topicName, groupId);
      consumerThread.start();
      String line = "";
      while (!line.equals("exit")) {
         line = in.next();
```

```
        }
        consumerThread.getKafkaConsumer().wakeup();
        System.out.println("Stopping consumer .....");
        consumerThread.join();
    }

    private static class ConsumerThread extends Thread {
        private String topicName;
        private String groupId;
        private KafkaConsumer<String, String> kafkaConsumer;

        public ConsumerThread(String topicName, String groupId) {
            this.topicName = topicName;
            this.groupId = groupId;
        }

        public void run() {
            Properties configProperties = new Properties();
            configProperties.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
"localhost:9093");
            configProperties.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
            configProperties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
"org.apache.kafka.common.serialization.StringDeserializer");
            configProperties.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);

            //Figure out where to start processing messages from
            kafkaConsumer = new KafkaConsumer<String, String>(configProperties);
            kafkaConsumer.subscribe(Arrays.asList(topicName), new
ConsumerRebalanceListener() {
                public void onPartitionsRevoked(Collection<TopicPartition> partitions) {
                    System.out.printf("%s topic-partitions are revoked from this consumer\n",
Arrays.toString(partitions.toArray()));
                }
                public void onPartitionsAssigned(Collection<TopicPartition> partitions) {
                    System.out.printf("%s topic-partitions are assigned to this consumer\n",
Arrays.toString(partitions.toArray()));
                }
            });
            //Start processing messages
            try {
                while (true) {
                    ConsumerRecords<String, String> records = kafkaConsumer.poll(100);
                    for (ConsumerRecord<String, String> record : records)
                        System.out.println(record.value());
                }
            } catch (WakeupException ex) {
```

```
            System.out.println("Exception caught " + ex.getMessage());
        } finally {
            kafkaConsumer.close();
            System.out.println("After closing KafkaConsumer");
        }
    }

    public KafkaConsumer<String, String> getKafkaConsumer() {
        return this.kafkaConsumer;
    }
}

}
```

Create a new topic for this example

1. kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic part-demo
2. java -cp Kafka2-0.0.1-SNAPSHOT-jar-with-dependencies.jar com.jpmc.SimpleProducer part-demo
3. Create 4 instances of the SingleConsumer and check how reassignment of partitions is done.

java -cp Kafka2-0.0.1-SNAPSHOT-jar-with-dependencies.jar com.jpmc.SimpleConsumer part-demo group1

4. Then type the messages and see how it goes to the respective partitions.

```
USA: First order
India: First order
USA: Second order
France: First order
```