Big data Hadoop- Assignment8_kafka2 and Spark Streaming

Task 5:

Create a java program MyKafkaProducer.java that takes a file name and delimiter as input arguments.

It should read the content of file line by line.

Fields in the file are in following order

1. Kafka Topic Name

2. Key

3. value

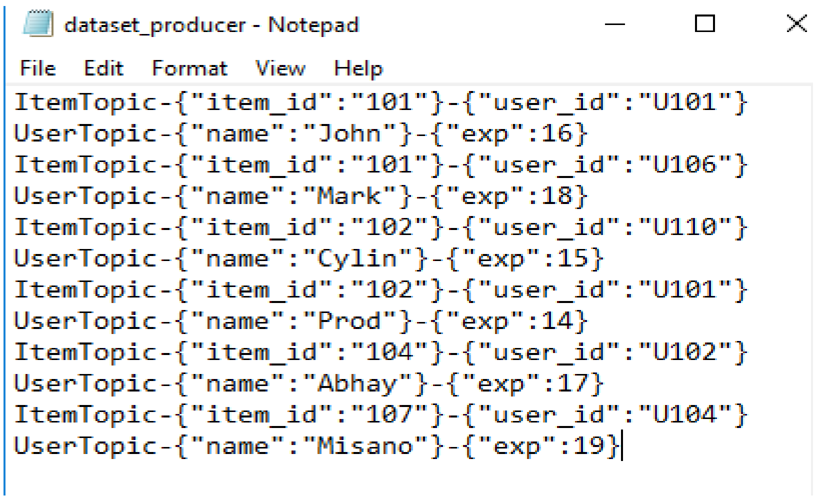
For every line, insert the key and value to the respective Kafka broker in a fire and forget mode.

After record is sent, it should print appropriate message on screen.

Pass dataset_producer.txt as the input file and -as delimiter.

LINK: https://drive.google.com/file/d/0B_Qjau8wv1KoSnR5eHpKOF9rTFU/view?usp=sharing

Dataset Used for this assignment is as below



```
ItemTopic-{"item_id":"101"}-{"user_id":"U101"}
UserTopic-{"name":"John"}-{"exp":16}
ItemTopic-{"item_id":"101"}-{"user_id":"U106"}
UserTopic-{"name":"Mark"}-{"exp":18}
ItemTopic-{"item_id":"102"}-{"user_id":"U110"}
UserTopic-{"name":"Cylin"}-{"exp":15}
ItemTopic-{"item_id":"102"}-{"user_id":"U101"}
UserTopic-{"name":"Prod"}-{"exp":14}
ItemTopic-{"item_id":"104"}-{"user_id":"U102"}
UserTopic-{"name":"Abhay"}-{"exp":17}
ItemTopic-{"item_id":"107"}-{"user_id":"U104"}
UserTopic-{"name":"Misano"}-{"exp":19}
```

Topics Creation from Command Line

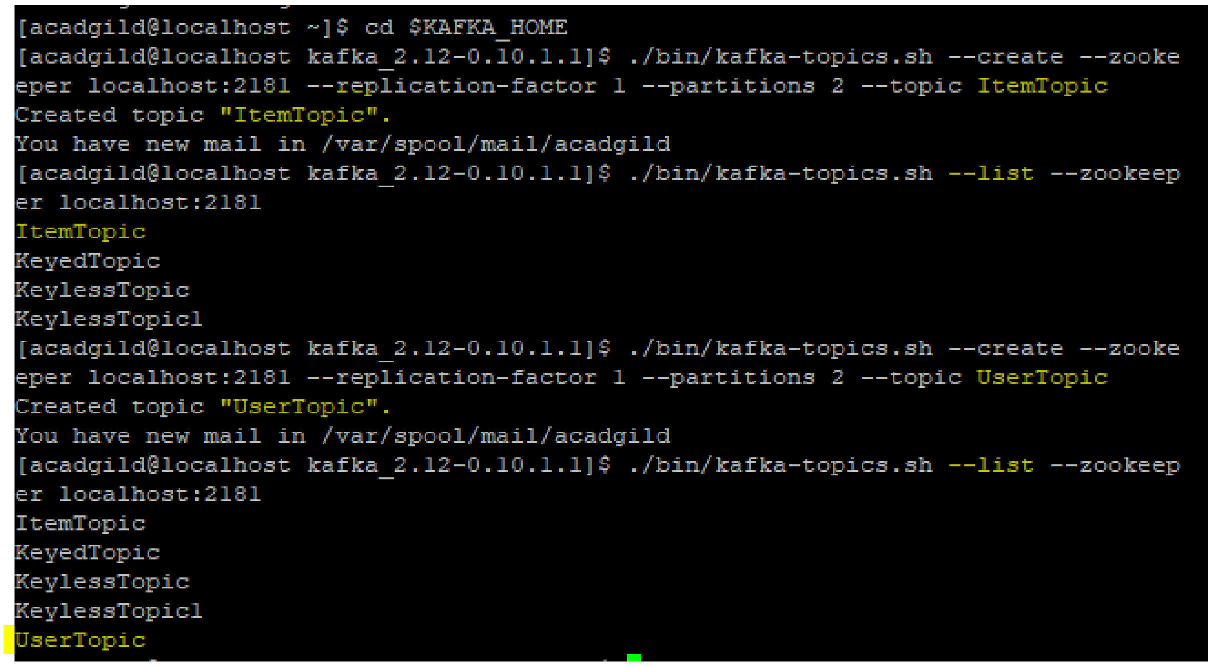
It has two topics. These topics needs to be created from command line as below

- Created topic named as **ItemTopic**

"bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic ItemTopic "

- Created topic named as **UserTopic**

"bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic UserTopic "



```
[acadgild@localhost ~]$ cd $KAFKA_HOME
[acadgild@localhost kafka_2.12-0.10.1.1]$ ./bin/kafka-topics.sh --create --zooke
eper localhost:2181 --replication-factor 1 --partitions 2 --topic ItemTopic
Created topic "ItemTopic".
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost kafka_2.12-0.10.1.1]$ ./bin/kafka-topics.sh --list --zookeep
er localhost:2181
ItemTopic
KeyedTopic
KeylessTopic
KeylessTopic1
[acadgild@localhost kafka_2.12-0.10.1.1]$ ./bin/kafka-topics.sh --create --zooke
eper localhost:2181 --replication-factor 1 --partitions 2 --topic UserTopic
Created topic "UserTopic".
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost kafka_2.12-0.10.1.1]$ ./bin/kafka-topics.sh --list --zookeep
er localhost:2181
ItemTopic
KeyedTopic
KeylessTopic
KeylessTopic1
UserTopic
```

Solution Approach

After creating the Java class imported required kafka jars

```
import org.apache.kafka.clients.producer.KafkaProducer;

 import org.apache.kafka.clients.producer.ProducerRecord;

 import java.io.BufferedReader;

 import java.io.FileReader;

 import java.io.IOException;

 import java.util.Properties;
```

As per requirement need to check if two arguments are passed

```
//We must pass file name and delimiter as input while execution

  if (args.length != 2) {

  System.out.println("Please provide command line arguments as file name as delimiter");
System.exit(-1);

    }
```

Set all the properties for kafka producer

```
//Configuring the properties for Kafka Producers

    Properties props = new Properties();

   props.put("bootstrap.servers", "localhost:9092");
```

```
  props.put("acks", "all");
```

- We then instantiate the KafkaProducer class called producer, we have mentioned string in <> because both key and value are String.
- We add the properties instance (props)to KafkaProducer instance.
- We also instantiate ProducerRecord as producerRecord

```
    props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
```

- Now we take the data provided in the command line i.e. file name and delimiter and save them in the array of string variables called filename and delimiter

```
//Read the file (file name is passed through command line argument with delimiter)

 String fileName = args[0];        String delimiter = args[1];
```

- We read the contents of the input file, and save their contents arrays in different variables: o We save the topic name i.e. first part of array(0th index elements) in String variable topic and similarly we save key and value variables too.
- Now, we pass the variables topic,key and value to producer record.
- We also print appropriate message which shows the topics,key and value contents.
- We inally, close the producer

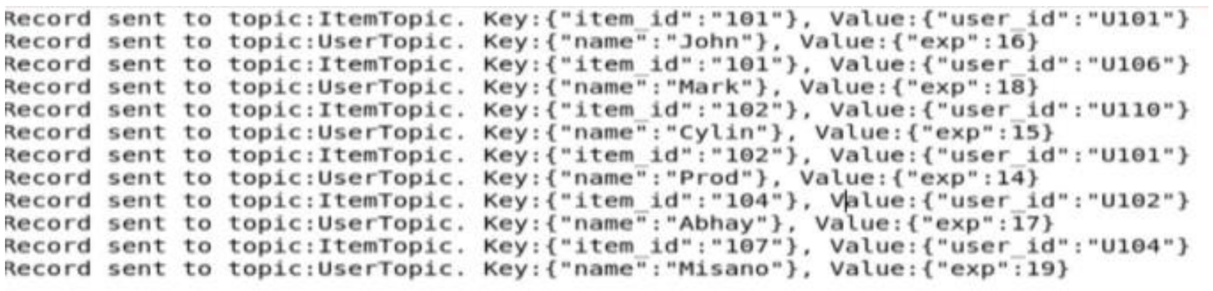
<u>Complete Code</u>

```
 package com.kafkaTest;

import org.apache.kafka.clients.producer.KafkaProducer;

import org.apache.kafka.clients.producer.ProducerRecord;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Properties;
```

Big data Hadoop- Assignment8_kafka2 and Spark Streaming

```
public class MykafkaProducer{

            public static void main(String[] args) throws IOException {

                        if (args.length != 2){

                            System.out.println("Please provide appropriate command line arguments");

                            System.exit(-1)

                           }

                           Properties props = new Properties();

                           props.put("bootstrap.servers", "localhost:9092");

                                   props.put("key.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

                          props.put("value.serializer",
"org.apache.kafka.common.serialization.StringSerializer");

                           KafkaProducer<String, String> producer = new KafkaProducer<>(props);

                                   ProducerRecord<String, String> producerRecord = null;

                           String fileName = args[0];

                           String delimiter = args[1];

                           try(BufferedReader br = new BufferedReader(new FileReader(fileName))){

                                   for(String line; (line = br.readLine()) != null; ){

                                 String[] tempArray = line.split(delimiter);

                                 String topic = tempArray[0];

                                 String key = tempArray[1];

                                 String value = tempArray[2];

                                 producerRecord = new ProducerRecord<String, String>(topic, key, value);

                                 producer.send(producerRecord);

                                 System.out.printf("Record sent to topic:%s. Key:%s, Value:%s\n", topic, key,
value)

                             }

                           }

                           producer.close();
            }

}
```

OUTPUT:



```
Record sent to topic:ItemTopic. Key:{"item_id":"101"}, Value:{"user_id":"U101"}
Record sent to topic:UserTopic. Key:{"name":"John"}, Value:{"exp":16}
Record sent to topic:ItemTopic. Key:{"item_id":"101"}, Value:{"user_id":"U106"}
Record sent to topic:UserTopic. Key:{"name":"Mark"}, Value:{"exp":18}
Record sent to topic:ItemTopic. Key:{"item_id":"102"}, Value:{"user_id":"U110"}
Record sent to topic:UserTopic. Key:{"name":"Cylin"}, Value:{"exp":15}
Record sent to topic:ItemTopic. Key:{"item_id":"102"}, Value:{"user_id":"U101"}
Record sent to topic:UserTopic. Key:{"name":"Prod"}, Value:{"exp":14}
Record sent to topic:ItemTopic. Key:{"item_id":"104"}, Value:{"user_id":"U102"}
Record sent to topic:UserTopic. Key:{"name":"Abhay"}, Value:{"exp":17}
Record sent to topic:ItemTopic. Key:{"item_id":"107"}, Value:{"user_id":"U104"}
Record sent to topic:UserTopic. Key:{"name":"Misano"}, Value:{"exp":19}
```

Reading the topics from kafka Console Consumers

./bin/kafka-console-consumer.sh --topic ItemTopic --from-beginning --zookeeper localhost:2181 -property print.key=true

Complete Code

```
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 KafkaConsumer_Ass8{

public static void main(String[] args) throws Exception {

  //we have to read two topics which should be passed as arguments

  if(args.length !=2){

  System.out.println("Enter 2 topic names");

   return;

   }

  String topicName1 = args[0].toString();

String topicName2 = args[1].toString();

Properties props = new Properties();
```

```
 props.put("bootstrap.servers", "localhost:9092");

  props.put("group.id","ItemTopic");

    props.put("session.timeut.ms", "300000");

   props.put("key.deserializer",   "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer",   "org.apache.kafka.common.serialization.StringDeserializer");
KafkaConsumer<String, String> consumer = new KafkaConsumer  <String, String>(props);
//Kafka Consumer subscribes list of topics here.
consumer.subscribe(Arrays.asList(topicName1));

  //print the topic name

  System.out.println("Subscribed to topic " + topicName1);

      consumer.poll(0);

 consumer.seekToBeginning(consumer.assignment());

  while(true)          {

    ConsumerRecords<String, String> records = consumer.poll(1);

                //String str = records.toString();

  //ystem.out.printf(str);

     if(records.isEmpty())

    {

   System.out.printf("Empty");

           }

      else

   {

   for (ConsumerRecord<String, String> consumerRecord : records) {

     System.out.printf("Key= %s \n", consumerRecord.key(), consumerRecord.value());

     }         }       }

               } }
```

Big data Hadoop- Assignment8_kafka2 and Spark Streaming

Task 6:

Modify the previous program **MyKafkaProducer.java** and create a new Java program **KafkaProducerWithAck.java** This should perform the same task as of **KafkaProducer.java** with some modification. When passing any data to a topic, it should wait for acknowledgement. After acknowledgement is received from the broker, it should print the key and value which has been written to a specified topic. The application should attempt for 3 retries before giving any exception.

Solution Approach –

The entire code will remain as above only we need to add two more properties for creating the kafka producer as below

We configure the properties for KafkaProducer:

- Acks "all"- this means that the producer will receive a success response from the broker once all in-sync replicas received the message.
- Retries 3- When the producer receives an error message from the server, the error could be transient (e.g., a lack of leader for a partition). In this case, the value of the retries parameter will control how many times the producer will retry sending the message before giving up and notifying the client of an issue.

```
 props.put("acks", "all");   props.put("retries", 3);
```

Complete Code

```
import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.Properties;

 import java.util.concurrent.ExecutionException;

import org.apache.kafka.clients.producer.KafkaProducer;

 import org.apache.kafka.clients.producer.ProducerRecord;

public class MyKafkaProducerAck {  public static void main(String[] args) throws IOException {
//We must pass file name and delimiter as input while execution

     if (args.length != 2) {

   System.out.println("Please provide command line arguments as file name as delimiter");
System.exit(-1);

  }
```

```
            //Configuring the properties for Kafka Producers

     Properties props = new Properties();

     props.put("bootstrap.servers", "localhost:9092");

   props.put("acks", "all");

   props.put("retries", 3);

     props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

       KafkaProducer<String, String> producer = new KafkaProducer<>(props);
ProducerRecord<String, String> producerRecord = null;

       //Read the file (file name is passed through command line argument with delimiter)

    String fileName = args[0];

    String delimiter = args[1];

       try (BufferedReader br = new BufferedReader(new FileReader(fileName))) {

      for (String line; (line = br.readLine()) != null; ) {

     String[] tempArray = line.split(delimiter);

         String topic = tempArray[0];

    String key = tempArray[1];

        String value = tempArray[2];

              producerRecord = new ProducerRecord<String, String>(topic, key, value);
producer.send(producerRecord).get();

     System.out.printf("Record sent to topic & Acknowledged as well:%s. Key:%s, Value:%s\n", topic,
key, value);

   }

     } catch (InterruptedException e) {    // TODO Auto-generated catch block

  e.printStackTrace();   }

catch (ExecutionException e) {    // TODO Auto-generated catch block

   e.printStackTrace();   }

      producer.close();

 }
```

Big data Hadoop- Assignment8_kafka2 and Spark Streaming

Task 7

Read a stream of Strings, fetch the words which can be converted to numbers. Filter out the rows, where the sum of numbers in that line is odd. Provide the sum of all the remaining numbers in that batch

Solution Approach:

1. We have to create after listening from port 9999 which has tuple for word and number associated with that

2. As per the requirement to have to check if line has any numbers then add those and print the line if sum is ODD. This can be done while data is streaming

3. However we have to get the sum of all the remaining lines which has even sum. To achieve this we have to declare an accumulator which will hold the sum of numbers belonging to lines having even sum

To have live streaming from socket connections we will be using net cat utility, computer networking utility for reading from and writing to network connections using TCP or UDP.

1. We have downloaded the net cat utility for Windows.

2. Opened the command prompt as administrator and navigated to path of net cat utility placed in file system

3. Ran command nc64.exe –l –p -9999 (64 indicates 64 bit machine, - I for local)

Complete code:

```
import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.streaming.{Seconds, StreamingContext}

object Even_Number_Line{

  def main(args: Array[String]): Unit = {

    def  Get_Lines_Sum(input : String) : Double ={

      val line = input.split("")

      var number : Double = 0.0

      for (x <- line)

      {

        try{
```

```
      val value = x.toDouble

      number = number + value

    }

    catch

      {

        case ex : Exception => {}

      }

  }

  return number

}


println("This is the task7 of assignment of session 8")

val conf = new SparkConf().setMaster("local[2]").setAppName("EvenLines")

val sc = new SparkContext(conf)

sc.setLogLevel("WARN”)

println("Spark Context Created")

val ssc = new StreamingContext(sc, Seconds(20))

println("Spark Streaming Context Created ")

val lines = ssc.socketTextStream("localhost", 9999)

val lines_filter = lines.filter(x => Get_Lines_Sum(x)%2 == 0)

val lines_sum = lines_filter.map(x => Get_Lines_Sum(x))

println("Lines with even sum:")

lines_filter.print()

println("Sum of the numbers in even lines:")

lines_sum.reduce(_+_).print()

ssc.start()

ssc.awaitTermination()
```

```
  }
}
```

Task 8

Read two streams

1. List of strings input by user

2. Real-time set of offensive words Find the word count of the offensive words inputted by the user as per the real-time set of offensive words

Solution Approach :

As we have to read the two stream we can either read from Textfile stream or having another socket streaming with different portal.

```
import org.apache.spark.{SparkConf, SparkContext}

import org.apache.spark.streaming.{Seconds, StreamingContext}

object  Offensive_Words_Count {

  def main(args: Array[String]): Unit = {

    println("This is the task8 of assignment of session 8")

    val conf = new SparkConf().setMaster("local[2]").setAppName("SparkSteamingExample")

    val sc = new SparkContext(conf)

    sc.setLogLevel("WARN")

    println("Spark Context Created")

    val offensive_word_list: Set[String] = Set("idiot", "fool", "bad","nonsense")

    println(s"$offensive_word_list")

    val ssc = new StreamingContext(sc, Seconds(20))

    println("Spark Streaming Context Created !")

    val lines = ssc.socketTextStream("localhost", 9999)

    val words = lines.flatMap(_.split(" ")).map(x => x)

    val Offensive_Word_Count = words.filter(x => offensive_word_list.contains(x)).map(x => (x, 1)).reduceByKey(_ + _)

    Offensive_Word_Count.print()

    ssc.start()
    ssc.awaitTermination()
  }
}
```