I have a dataset which is a csv file having header and records

|  |
| --- |
| id,name,actor  1,pink,amitabh  2,black,amitabh  3,blue,akshay  4,pk,aamir  5,fan,sharukh  6,sultan,salman  7,piku,amitabh  8,rustam,akshay  9,bang bang,hrithik  10,force,john  11,dishoom,john  12,bajrangi bhaijan,salman  13,fanna,aamir  14,happy new year,sharukh  15,shivay,ajay  16,chennai express,sharukh  17,golman,ajay  18,dhoom,aamir  19,drishyam,ajay |

|  |
| --- |
| put file in hdfs  hdfs dfs -put /home/cloudera/user/training/movies.csv /user/training/ |
| Open spark-shell  val contentRDD = sc.textFile("hdfs://quickstart.cloudera:8020/user/training/movies.csv") |
| remove/ignore header(first) record  val filterRDD = contentRDD.filter(x=> !x.startsWith("id,")) |
| split by ,  val splitRDD = filterRDD.map( line => line.split(",") ) |
| Put only actor in RDD  val actorRDD = splitRDD.map(\_(2)) |
| Count actor  actorRDD.countByValue() |
| Output  countRDD: scala.collection.Map[String,Long] = Map(john -> 2, sharukh -> 3, hrithik -> 1, ajay -> 3, salman -> 2, aamir -> 3, amitabh -> 3, akshay -> 2) |

Now we are going to put movies.csv file in kafka using java api and later will do count of actor using spark API in scala

Program that read properties from kafka\_simulation.properties file and put records on kafka

|  |
| --- |
| package com.cybage.simulator;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileFilter;  import java.io.FileInputStream;  import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.IOException;  import java.util.Properties;  import org.apache.log4j.Logger;  import org.apache.kafka.clients.producer.KafkaProducer;  import org.apache.kafka.clients.producer.Producer;  import org.apache.kafka.clients.producer.ProducerRecord;  public class FileKafkaSimulator implements FileFilter{    boolean runOnce = false;  private Logger logger = Logger.getLogger(FileKafkaSimulator.class);  private long sleepTime = 60 \* 1000; // 1 minute.    private String configfile="kafka\_simulation.properties";  private Properties configuration = new Properties();  private static FileKafkaSimulator \_instance = new FileKafkaSimulator();    protected File inpupt\_directory;  protected File archive\_directory;    protected File error\_directory;  private String fileNamePattern;  private String topicName;    Producer<String, String> producer = null;    public static FileKafkaSimulator getInstance() {  return \_instance;  }  public static void main(String[] args) {  FileKafkaSimulator.getInstance().start(args);  }    private void start(String[] args) {    boolean success = configure(args);  if(!success) {  logger.fatal("Configuration failed");  logger.info("Exiting Simulator");  return;  }  try {  processData();  } catch (IOException e) {  logger.error(" exception occurs " ,e);  }      }    private boolean configure(String args[]) {    //Read command line parameters  if (args.length > 0 && args[0].equals("runOnce")) {  runOnce = true;  }    logger.info("Loading configuration.");  //Load configuration from the config file  try {  File file = new File(configfile);  FileInputStream fis = new FileInputStream(file);  configuration.load(fis);  fis.close();    String input\_directory\_path = configuration.getProperty("input\_directory").trim();  String archive\_directory\_path = configuration.getProperty("archive\_directory").trim();  String error\_directory\_path = configuration.getProperty("error\_directory").trim();  if (input\_directory\_path == null || input\_directory\_path.equals("") ||  archive\_directory\_path == null || archive\_directory\_path.equals("") ||  error\_directory\_path == null || error\_directory\_path.equals("")) {  logger.error("Directory parameters missing");  return false;  }  inpupt\_directory = new File(input\_directory\_path);  archive\_directory = new File(archive\_directory\_path);  error\_directory = new File(error\_directory\_path);    fileNamePattern = configuration.getProperty("file\_name\_pattern",".\*");  topicName = configuration.getProperty("topic\_name");  if (topicName == null) {  logger.error(" topic name parameter is missing");  return false;  }    producer = new KafkaProducer<>(configuration);    } catch (FileNotFoundException e) {  logger.fatal(configfile+" cannot be found", e);  return false;  } catch (IOException e) {  logger.fatal("LoadProcess configuration file cannot be read - exiting.", e);  return false;  }  //Parse configuration parameters sleepTime and to\_addresses  try {  sleepTime \*= Integer.parseInt(configuration.getProperty("sleepTime", "1").trim());  } catch (Exception e) {  logger.fatal("sleepTime entry in configuration file could not be parsed.", e);  return false;  }    logger.info("Configuration loaded successfully.");  return true;  }  private void processData () throws IOException{  while (true) {  File[] files = inpupt\_directory.listFiles(this);  for (File file : files) {  readFile (file);  moveFile (archive\_directory,file);  }  if (runOnce){  System.exit(0);;  }  else {  try {Thread.sleep(sleepTime);}  catch (InterruptedException e) {  logger.fatal("Thread Interrupted", e);  }  }  }  }    private void readFile(File fin) throws IOException {  // Construct BufferedReader from FileReader  BufferedReader br = new BufferedReader(new FileReader(fin));    String line = null;  try {  while ((line = br.readLine()) != null) {  sendRecordToKafka(line);  }  } finally {  //producer.flush();  //lproducer.close ();  br.close();  }  }    private void moveFile (File destination,File file) {  if (destination != null) {  destination = new File(destination, file.getName() );  if (destination.exists())  destination.delete();  if (!file.renameTo(destination)) {  logger.error("File rename failed for file '" + file.getPath() + "' to '" + destination.getPath() + "'");  } else {  logger.info(file.getName()+" file has been moved to " +destination.getAbsolutePath());  }  }  }    private void sendRecordToKafka (String record) {  producer.send(new ProducerRecord<String, String>(topicName,record));  }  @Override  public boolean accept(File file) {    return file.getName().matches(fileNamePattern);  }  } |

Kafka-simulation.properties file , put this file into your project root directory

|  |
| --- |
| input\_directory = /home/cloudera/user/training/healthcare-claim/Data\_files/input  archive\_directory= /home/cloudera/user/training/healthcare-claim/Data\_files/archive  error\_directory=/home/cloudera/user/training/healthcare-claim/Data\_files/error  #file\_name\_pattern = npidata\_\\d{8}-\\d{8}.(CSV|csv)  bootstrap.servers=localhost:9092  key.serializer=org.apache.kafka.common.serialization.StringSerializer  value.serializer=org.apache.kafka.common.serialization.StringSerializer  topic\_name=test1 |

log4j.propertis file put this under source folder (need to be present in classpath)

|  |
| --- |
| ; ~~~ Log4j.properties configuration file  ; ~~~ Set default threshold level for all loggers and configure two appenders.  log4j.rootLogger=ALL, myFileAppender, myConsoleAppender  ; ~~~ CONSOLE APPENDER  log4j.appender.myConsoleAppender=org.apache.log4j.ConsoleAppender  ; ~~~ Threshold = INFO.  log4j.appender.myConsoleAppender.Threshold=INFO  ; ~~~ Pattern layout for Log4J output to console.  log4j.appender.myConsoleAppender.layout=org.apache.log4j.PatternLayout  log4j.appender.myConsoleAppender.layout.ConversionPattern=%d{dd MMM yyyy HH:mm:ss} [%t]%-5p %c - %m%n  ; ~~~ FILE APPENDER  log4j.appender.myFileAppender=org.apache.log4j.DailyRollingFileAppender  ; ~~~ Threshold = INFO.  log4j.appender.myFileAppender.Threshold=INFO  ; ~~~ Log file to store Log4J output.  log4j.appender.myFileAppender.File=/home/cloudera/kafka-simulator/logs/simulation.log  ; ~~~ Format for filenames.  log4j.appender.myFileAppender.DatePattern=.yyyy-MM-dd  ; ~~~ Append to existing file.  log4j.appender.myFileAppender.Append=TRUE  ; ~~~ Pattern layout for Log4J output to log file.  log4j.appender.myFileAppender.layout=org.apache.log4j.PatternLayout  log4j.appender.myFileAppender.layout.ConversionPattern=%d{dd MMM yyyy HH:mm:ss} [%t]%-5p %c - %m%n |

Spark streaming program that consume data from kafka and do transformation and action on 2 sec batch interval

|  |
| --- |
| package com.cybage.simulator  import java.util.HashMap  import org.apache.spark.streaming.\_  import org.apache.spark.streaming.kafka.\_  import org.apache.spark.SparkConf  //import com.databricks.spark.csv.\_  //import org.apache.spark.sql.SQLContext  object SparkStreamingUsingKafka {  def main(args: Array[String]) {  if (args.length < 4) {  System.err.println("Usage: SparkStreamingUsingKafkal <zkQuorum> <group> <topics> <numThreads>")  System.exit(1)  }  // StreamingExamples.setStreamingLogLevels()  val Array(zkQuorum, group, topics, numThreads) = args  val sparkConf = new SparkConf().setAppName("SparkStreamingUsingKafka ").setMaster("local[2]")  val ssc = new StreamingContext(sparkConf, Seconds(2))  //checkpoint is directory created to track the progress  ssc.checkpoint("checkpoint")    //split topics name by "," and create a map  val topicMap = topics.split(",").map((\_, numThreads.toInt)).toMap  // data written on topics are in the form of key and value , value is the line  // and key is the offset so here we are creating a stream of lines by value \_2 is for value \_1 is for key (offest)  val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(\_.\_2)    //val filterRDD = lines.filter(x=> !x.startsWith("id,"))    val splitRDD = lines.map( line => line.split(",") )    val actorRDD = splitRDD.map(\_(2))    val countRDD = actorRDD.countByValue()    // we can also do the same by using Window period  //val countRDD = actorRDD.countByValueAndWindow(Minutes(5), Seconds(2), 1)      countRDD.print();  ssc.start()  ssc.awaitTermination()  }  } |

Command line argument - **localhost:2181** **testgroup** **test1 1**

* **localhost:2181** – zookeeper server address
* **testgroup** – name of group , you can use anything
* **test1**- name of topics , you can specify more than one topic separated by comma
* **1** – number of threads

|  |
| --- |
| You can also use group by key command to group the data  val groupedRDD = splitRDD.map(x=>(x(2),x(1))).groupByKey()  groupedRDD.collect()  output  Array[(String, Iterable[String])] = Array((amitabh,CompactBuffer(pink, black, piku)), (sharukh,CompactBuffer(fan, happy new year, chennai express)), (aamir,CompactBuffer(pk, fanna, dhoom)), (ajay,CompactBuffer(shivay, golman, drishyam)), (salman,CompactBuffer(sultan, bajrangi bhaijan)), (hrithik,CompactBuffer(bang bang)), (john,CompactBuffer(force, dishoom)), (akshay,CompactBuffer(blue, rustam))) |

Ignore this

|  |
| --- |
| 5. cr eate new structure of file having npi and statecode ( this is just for R&D purpose no need to do this)  val npiRDD = splitRDD.map(arr => {arr(0) +","+ arr(31) })  6. save back to hdfs  npiRDD.saveAsTextFile("hdfs://quickstart.cloudera:8020/user/training/npi1");  // below command also save the file in hdfs only and create respective folders  npiRDD.saveAsTextFile("/home/cloudera/user/training/healthcare-claim /npi1"); |