First create a scala program that will stream through kafka

This program counts the number of occurrences of the word and act as a consumer to kafka topic

This is old program prior to spark 1.3 now we don’t need to use ZKquorum we can directly talk with kafka brokder

|  |
| --- |
| import java.util.HashMap  import org.apache.spark.streaming.\_  import org.apache.spark.streaming.kafka.\_  import org.apache.spark.SparkConf  /\*\*  \* Consumes messages from one or more topics in Kafka and does wordcount.  \* Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>  \* <zkQuorum> is a list of one or more zookeeper servers that make quorum  \* <group> is the name of kafka consumer group  \* <topics> is a list of one or more kafka topics to consume from  \* <numThreads> is the number of threads the kafka consumer should use  \*  \* Example:  \* `$ bin/run-example \  \* org.apache.spark.examples.streaming.KafkaWordCount zoo01,zoo02,zoo03 \  \* my-consumer-group topic1,topic2 1`  \*/  object KafkaWordCount {  def main(args: Array[String]) {  if (args.length < 4) {  System.err.println("Usage: KafkaWordCount <zkQuorum> <group> <topics> <numThreads>")  System.exit(1)  }  StreamingExamples.setStreamingLogLevels()  val Array(zkQuorum, group, topics, numThreads) = args  val sparkConf = new SparkConf().setAppName("KafkaWordCount").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  // kafka stream contains (topic,message) pair  // data written on topics are in the form of key and value, value is actual the message  // so to access value we are using \_2 basically it is a tuple of tuple like ((topic list),(key,value)) so to access value we need to use \_.\_2  val lines = KafkaUtils.createStream(ssc, zkQuorum, group, topicMap).map(\_.\_2)  // split line by space and get the words  val words = lines.flatMap(\_.split(" "))  // calculating words  // in reduceByKeyAndWindow first column give us word count  // second column is inversion funciton , which give us initial values not the previous values  // interval in minute (interval on which Dstream to be traversed)  // seconds are slider interval  val wordCounts = words.map(x => (x, 1L))  .reduceByKeyAndWindow(\_ + \_, \_ - \_, Minutes(10), Seconds(2), 1)  wordCounts.print()  ssc.start()  ssc.awaitTermination()  }  } |

Before start this program we need to start zookeeper service, Kafka service and Kafka producer

In cloudera manage quick vm zookeeper service already started at port 2181 so no need to start zookeeper service



1. Move to kafka home directory

**cd kafka\_2.11-0.10.0.1**

2. Start kafka service

**bin/kafka-server-start.sh config/server.properties**

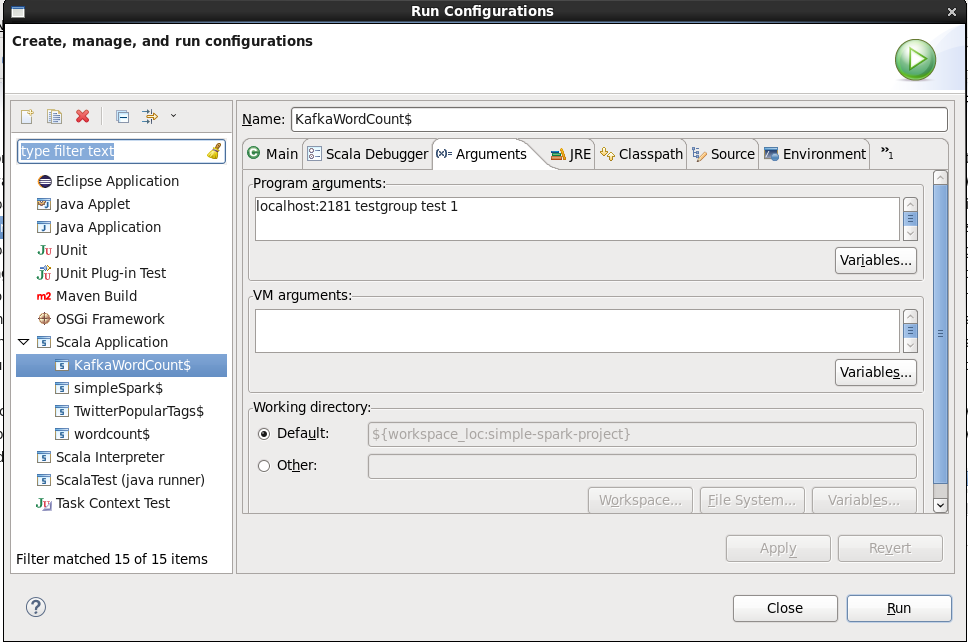
3. Create a topic if not already created

**bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test**

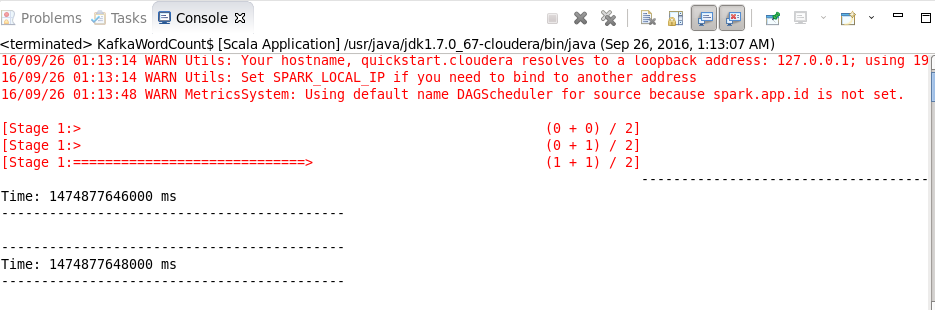
4. start a producer to produce the messages

**bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test**

Run scala program by providing right arguments



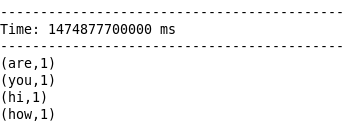
* **localhost:2181** – zookeeper server address
* **testgroup** – name of group , you can use anything
* **test**- name of topics , you can specify more than one topic separated by comma
* **1** – number of threads



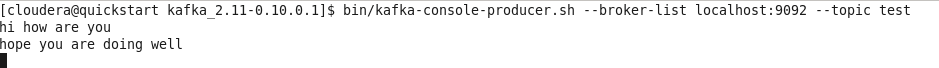
Now send some message to test topic by producer



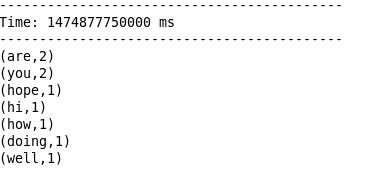
Now check the console in eclipse



Again send some different messages



Now check console again



So that is how we can do the stream of data in spark using kafka

Direct read from kafka broker instead of zookeeper

|  |
| --- |
| **package** org.apache.spark.examples.streaming  **import** kafka.serializer.StringDecoder  **import** org.apache.spark.streaming.\_ **import** org.apache.spark.streaming.kafka.\_ **import** org.apache.spark.SparkConf  */\*\*  \* Consumes messages from one or more topics in Kafka and does wordcount.  \* Usage: DirectKafkaWordCount* <**brokers**> <**topics**>  *\** <**brokers**> *is a list of one or more Kafka brokers  \** <**topics**> *is a list of one or more kafka topics to consume from  \*  \* Example:  \* $ bin/run-example streaming.DirectKafkaWordCount broker1-host:port,broker2-host:port \  \* topic1,topic2  \*/* **object** DirectKafkaWordCount {  **def** main(args: Array[String]) {  **if** (args.length < 2) {  System.err.println(**s"""  |Usage: DirectKafkaWordCount <brokers> <topics>  | <brokers> is a list of one or more Kafka brokers  | <topics> is a list of one or more kafka topics to consume from  |  """**.stripMargin)  System.exit(1)  }   *//StreamingExamples.setStreamingLogLevels()* **val** *Array*(brokers, topics) = args   *// Create context with 2 second batch interval* **val** sparkConf = **new** SparkConf().setAppName(**"DirectKafkaWordCount"**).setMaster(**"local[2]"**)  **val** ssc = **new** StreamingContext(sparkConf, Seconds(2))   ssc.checkpoint(**"checkpoint"**)  *// Create direct kafka stream with brokers and topics* **val** topicsSet = topics.split(**","**).toSet  **val** kafkaParams = Map[String, String](**"metadata.broker.list"** -> brokers)  **val** messages = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](  ssc, kafkaParams, topicsSet)   *// Get the lines, split them into words, count the words and print* **val** lines = messages.map(\_.\_2)  **val** words = lines.flatMap(\_.split(**" "**))  *//val wordCounts = words.map(x => (x, 1L)).reduceByKey(\_ + \_)* **val** wordCounts = words.map(x => (x, 1L))  .reduceByKeyAndWindow(\_ + \_, \_ - \_, Minutes(10), Seconds(2), 1)   wordCounts.print()   *// Start the computation* ssc.start()  ssc.awaitTermination()  } } *// scalastyle:on println* |