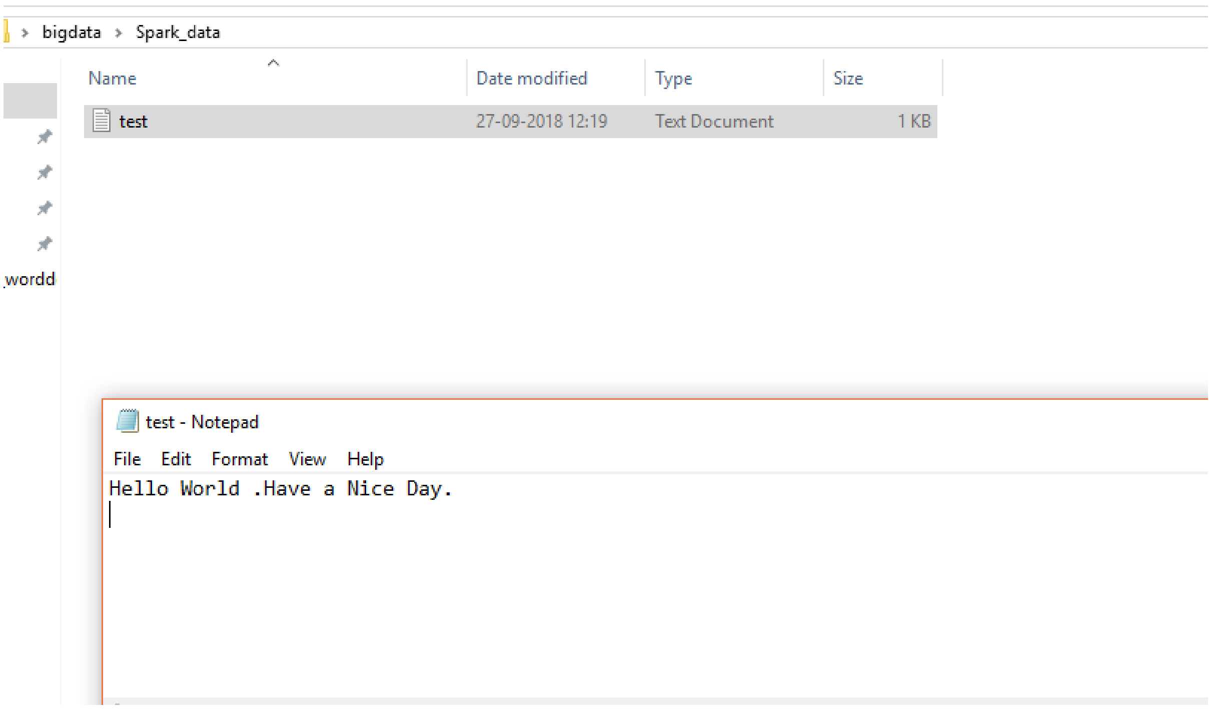
SparkStreaming Usecase

There are two parts this case study
-First Part - You have to create a Spark Application which streams data from a file on local directory on your machine and does the word count on the fly. The word should be done by the spark application in such a way that as soon as you drop the file in your local directory, your spark application should immediately do the word count for you.

First part:

DataSet: FileName :--test.txt



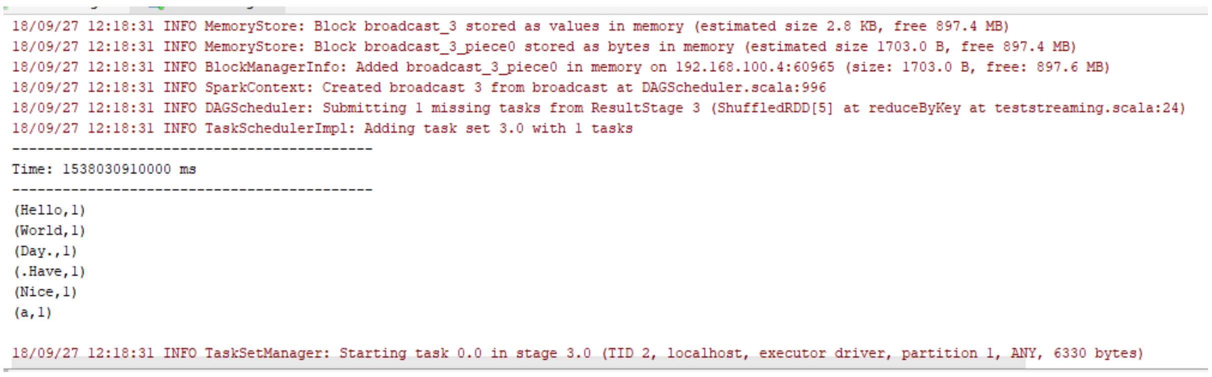
Name	Date modified	Type	Size
test	27-09-2018 12:19	Text Document	1 KB

```
Hello World .Have a Nice Day.
```

Code:

```
package Asg_26

import org.apache.spark.SparkConf
import org.apache.spark.storage.StorageLevel
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark._
//import java.util.stream.StreamSpliterators.ArrayBuffer
import scala.collection.mutable.ArrayBuffer

object Wordcount {
  def main(args: Array[String]) {

    System.setProperty("hadoop.home.dir", "c://winutil//")
    val conf = new SparkConf().setAppName("Application").setMaster("local[2]")
    //val sc = new SparkContext(conf)
    val ssc = new StreamingContext(conf,Seconds(30))
    val
input=ssc.textFileStream("file:////C:/Users/Admin/Desktop/bigdata/Spark_data/")
    val lines=input.flatMap(_.split(" "))
    val words=lines.map(word=>(word,1))
    val counts=words.reduceByKey(_+_)
    counts.print()
    val arr = new ArrayBuffer[String]();
    ssc.start()
    ssc.awaitTermination()


  }

}
```

Output:



```
18/09/27 12:18:31 INFO MemoryStore: Block broadcast_3 stored as values in memory (estimated size 2.8 KB, free 897.4 MB)
18/09/27 12:18:31 INFO MemoryStore: Block broadcast_3_piece0 stored as bytes in memory (estimated size 1703.0 B, free 897.4 MB)
18/09/27 12:18:31 INFO BlockManagerInfo: Added broadcast_3_piece0 in memory on 192.168.100.4:60965 (size: 1703.0 B, free: 897.6 MB)
18/09/27 12:18:31 INFO SparkContext: Created broadcast 3 from broadcast at DAGScheduler.scala:996
18/09/27 12:18:31 INFO DAGScheduler: Submitting 1 missing tasks from ResultStage 3 (ShuffledRDD[5] at reduceByKey at teststreaming.scala:24)
18/09/27 12:18:31 INFO TaskSchedulerImpl: Adding task set 3.0 with 1 tasks
-------------------------------------------
Time: 1538030910000 ms
-------------------------------------------
(Hello,1)
(World,1)
(Day.,1)
(.Have,1)
(Nice,1)
(a,1)

18/09/27 12:18:31 INFO TaskSetManager: Starting task 0.0 in stage 3.0 (TID 2, localhost, executor driver, partition 1, ANY, 6330 bytes)
```

-Second Part - In this part, you will have to create a Spark Application which should do the following :

1. Pick up a file from the local directory and do the word count
2. Then in the same Spark Application, write the code to put the same file on HDFS.
3. Then in same Spark Application, do the word count of the file copied on HDFS in step 2
4. Lastly, compare the word count of step 1 and 2. Both should match, other throw an error

Code:

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.{FileAlreadyExistsException, FileSystem, FileUtil, Path}
import scala.io.Source
object WordCountHDFS {
def main(args: Array[String]) {
//Create conf object
val conf = new SparkConf().setMaster("local[*]")
.setAppName("WordCount")
//create spark context object
val sc = new SparkContext(conf)
val hadoopConf = new Configuration()
//Check whether sufficient params are supplied
if (args.length < 2) {
println("Usage: ScalaWordCount<output1> <output2>")
System.exit(1)
}
//Read file and create RDD
val rawData = sc.textFile("/home/acadgild/wordcount/")
hadoopConf.addResource(new Path("/home/acadgild/install/hadoop/hadoop-
2.6.5/etc/hadoop/core-site.xml"))
hadoopConf.addResource(new Path("/home/acadgild/install/hadoop/hadoop-
2.6.5/etc/hadoop/hdfs-site.xml"))
val fs = FileSystem.get(hadoopConf);
val sourcePath = new Path("/home/acadgild/wordcount/");
val destPath = new Path("hdfs://localhost:8020/");
if(!(fs.exists(destPath)))
{
System.out.println("No Such destination exists :"+destPath);
return;
}
fs.copyFromLocalFile(sourcePath, destPath);
//convert the lines into words using flatMap operation
val words = rawData.flatMap(line => line.split(" "))
val hdfsfile = sc.textFile("hdfs://localhost:8020/wordcount/test")
val hdfswords = hdfsfile.flatMap(line => line.split(" "))
//count the individual words using map and reduceByKey operation
val wordCount = words.map(word => (word, 1)).reduceByKey(_ + _)
val hdfsWC = hdfswords.map(word => (word,1)).reduceByKey(_ + _)
//Save the result
wordCount.saveAsTextFile(args(0))
hdfsWC.saveAsTextFile(args(1))
val LFSWCfile = Source.fromFile("/home/acadgild/wordcount1/part-
00000").getLines().toArray
val hdfsWCfile= Source.fromFile("/home/acadgild/wordcount2/part-
00000").getLines().toArray
val elem = LFSWCfile.sameElements(hdfsWCfile)
if(elem == false){
println("Error!: Contents mismatch")
}else
println("Contents match!")
wordCount.collect().foreach(print)
hdfsWC.collect().foreach(print)
//stop the spark context
sc.stop
}
}
```