CASE STUDY 5 **Spark Streaming**

In this case, study, we create two spark applications:

First we create Spark Application which streams data from a local directory on our machine and will do a word count.

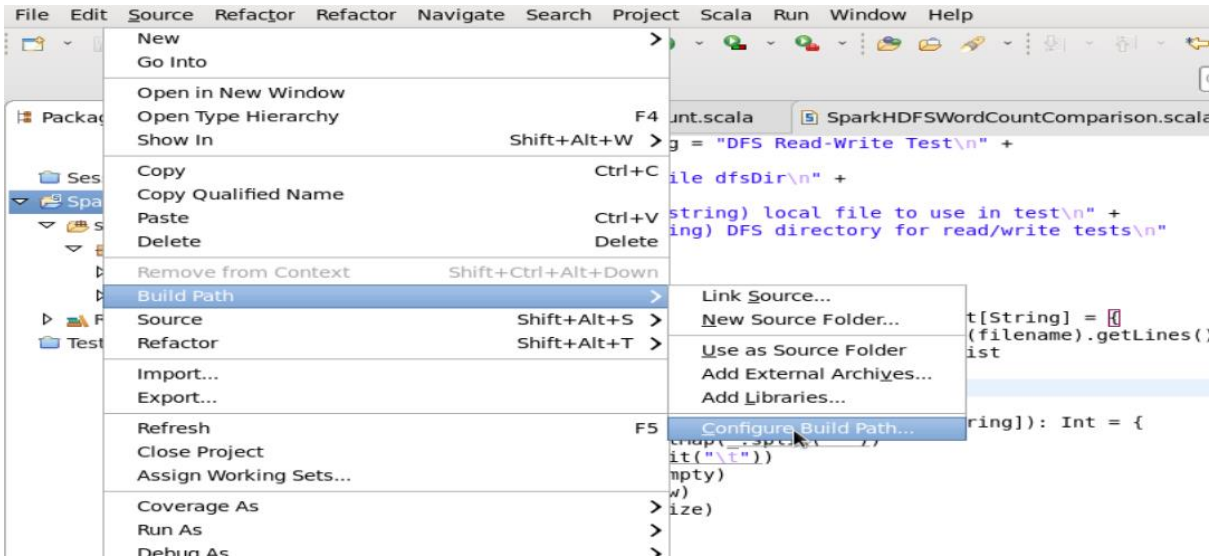
The word count should done by spark application in such a way that as soon as we drop a file in that local directory, our spark-application should immediately do the word count of that file.

Second, we create Spark Application that pick up a file from local directory and do the word count, and then put the same file on HDFS, then same application will word count from this copied file on HDFS.

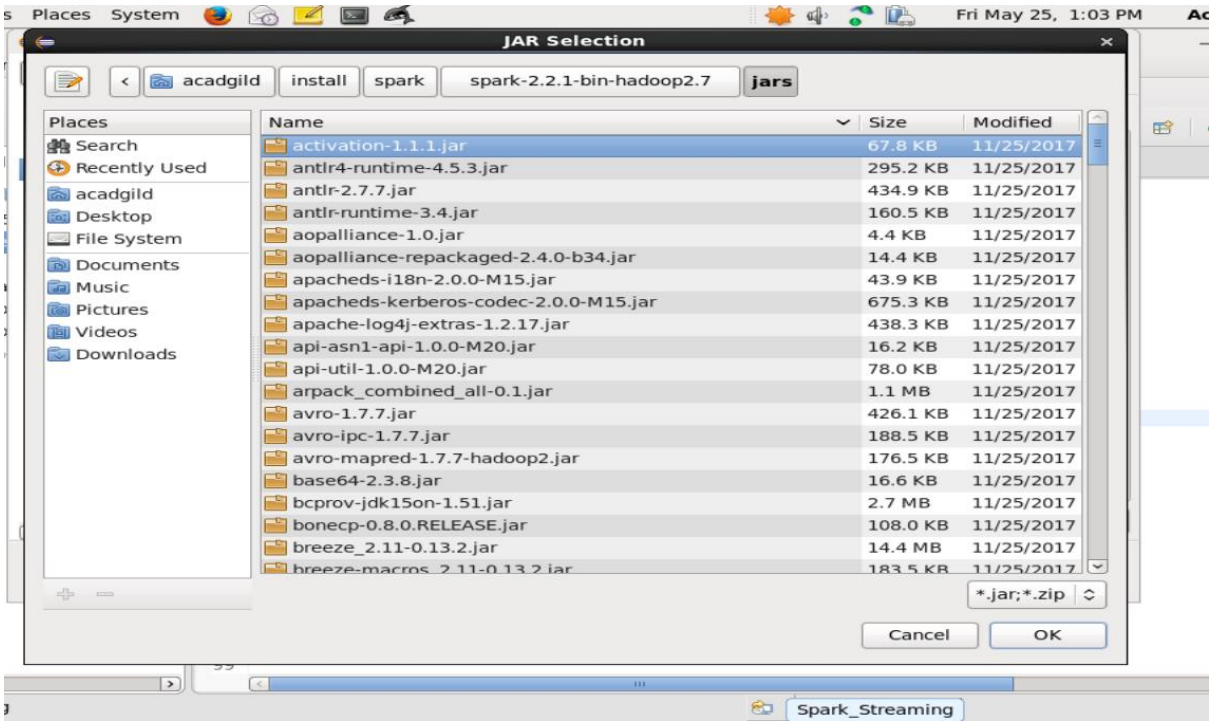
Lastly compare the word count results of first and second step, both should match otherwise throws an error.

We used scala eclipse-oxygen to create spark application. To write spark application first we need to import spark jars files by using configuring build path.

Below screenshot shows the method to add the required libraries in the spark application



Adding the jar files from spark installed directory to referenced libraries.



Below is the spark program for first case

```
import org.apache.spark.{SparkConf, SparkContext}
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.log4j.{Level,Logger}


object SparkFileStreamingWordCount

    {

       def main(args: Array[String]): Unit = {
       println("hey Spark Streaming")

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

     val sc = new SparkContext(conf)

     val rootLogger =Logger.getRootLogger()

     rootLogger.setLevel(Level.ERROR)

   // Create Streaming context  to set batch duration 5 seconds
     val ssc = new StreamingContext(sc, Seconds(5))

//Create RDD for text file streaming by

     val lines = ssc.textFileStream("/home/acadgild/Desktop/Spark_Streaming")
```

```
//Split each line into words
    val words = lines.flatMap(_.split(" "))

//Count each word in each batch
  val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)

    wordCounts.print()

//Start the computation
  ssc.start()

//wait for the computation to terminate
    ssc.awaitTermination()

    }

}
```

Let us create a directory **Spark_Streaming** in the Desktop folder of home acadgild directory by using following command:

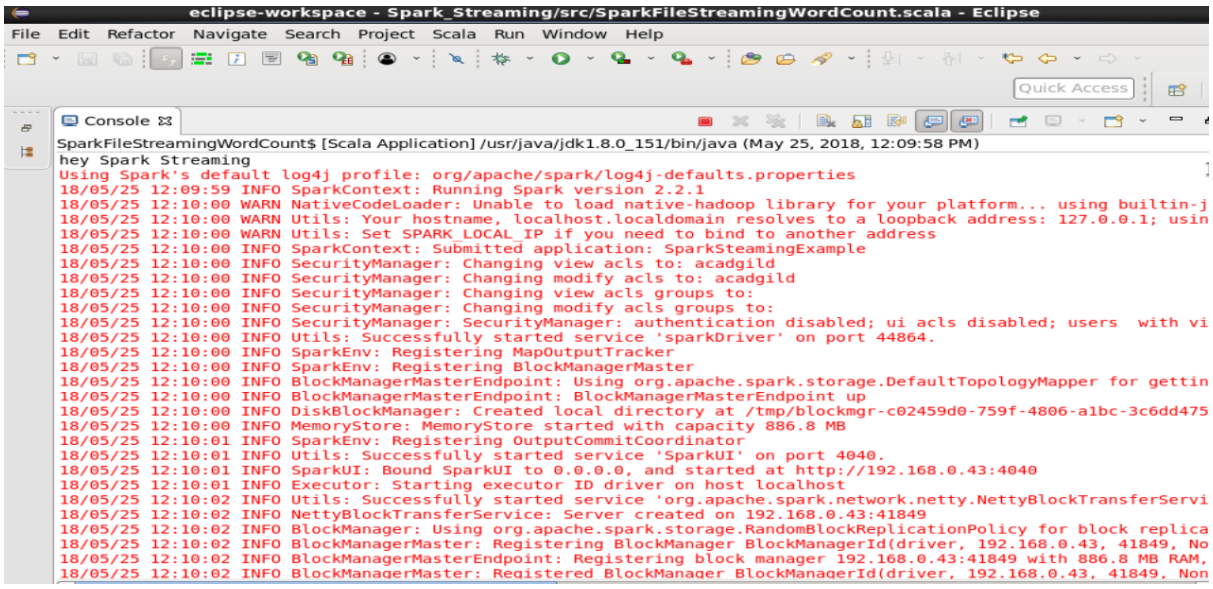
cd Desktop
mkdir Spark_Streaming

<u>Below screenshot shows the required directory in Desktop</u>

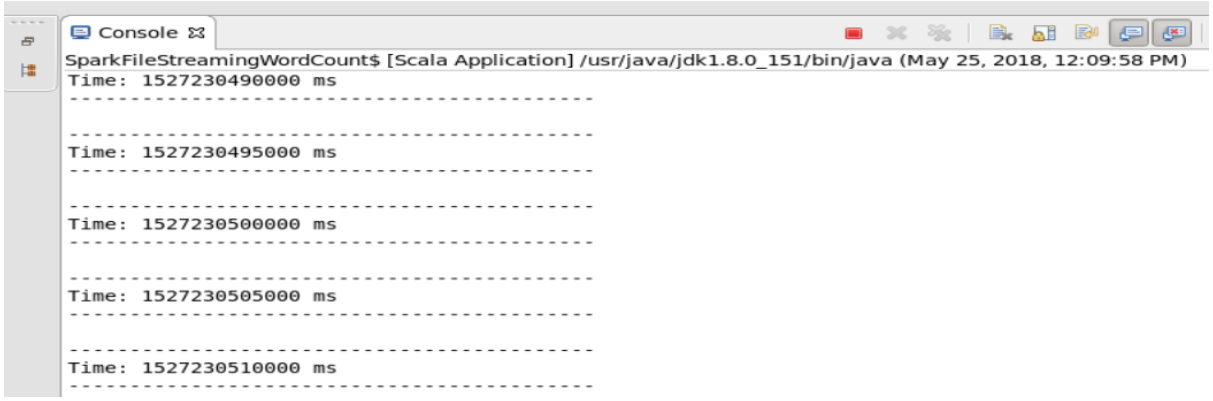
```
acadgild@localhost:~/Desktop                    acadgild@localhost:~
[acadgild@localhost ~]$ cd Desktop
[acadgild@localhost Desktop]$ mkdir Spark_Streaming
[acadgild@localhost Desktop]$ ls -lh
total 12K
-rw-rw-r--. 1 acadgild acadgild 1.3K Dec 29 15:15 bashrc.txt~
drwxrwxr-x. 2 acadgild acadgild 4.0K Feb  2 12:45 README
drwxrwxr-x. 2 acadgild acadgild 4.0K May 25 12:03 Spark_Streaming
You have new mail in /var/spool/mail/acadgild
[acadgild@localhost Desktop]$
```

```
SparkFileStreamingWordCount.scala    SparkHDFSWordCountComparison.scala
1  import org.apache.spark.{SparkConf, SparkContext}
2  import org.apache.spark.streaming.{Seconds, StreamingContext}
3  import org.apache.log4j.{Level,Logger}
4
5
6  object SparkFileStreamingWordCount {
7
8    def main(args: Array[String]): Unit = {
9      println("hey Spark Streaming")
10
11     val conf = new SparkConf().setMaster("local[2]").setAppName("SparkSteamingExample")
12     val sc = new SparkContext(conf)
13 val rootLogger =Logger.getRootLogger()
14    rootLogger.setLevel(Level.ERROR)
15     val ssc = new StreamingContext(sc, Seconds(5))
16     val lines = ssc.textFileStream("/home/acadgild/Desktop/Spark_Streaming")
17
18     val words = lines.flatMap(_.split(" "))
19     val wordCounts = words.map(x => (x, 1)).reduceByKey(_ + _)
20     wordCounts.print()
21     ssc.start()
22     ssc.awaitTermination()
23
24
25   }
26
27 }
28
29
```

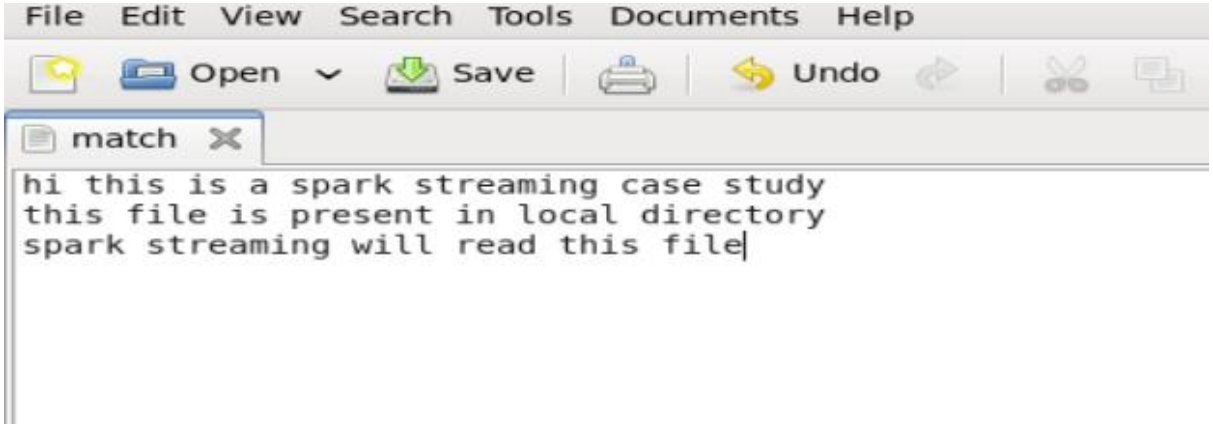
Run the spark application:



In below screenshot we are able to see that spark streaming is running every 5 seconds



Now we put a blank text file 'match' **and then putting some words in this file**



Output of word count in console:

```
Console
<terminated> SparkFileStreamingWordCount$ [Scala Application] /usr/java/jdk1.8.0_151/bi
-------------------------------------------
Time: 1527241505000 ms
-------------------------------------------

-------------------------------------------
Time: 1527241510000 ms
-------------------------------------------

-------------------------------------------
Time: 1527241515000 ms
-------------------------------------------

-------------------------------------------
Time: 1527241520000 ms
-------------------------------------------
(this,3)
(is,2)
(will,1)
(read,1)
(streaming,2)
(case,1)
(file,2)
(spark,2)
(directory,1)
(a,1)
```

Above Screenshot shows the word count of each words in text file from local file directory

Below is the spark programming for second case

```
import java.io.File
import org.apache.spark.{SparkConf, SparkContext}
import scala.io.Source._
import org.apache.log4j.{Level,Logger}



object SparkHDFSWordCountComparison

 {

// defining the local file directory
   private var localFilePath: File = new File("/home/acadgild/Desktop/Spark_Streaming/text")

//defining the directory in hdfs path
   private var dfsDirPath: String = "hdfs://localhost:8020/user"
   private val NPARAMS = 2


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

 //parseArgs(args)
```

```
    println("SparkHDFSWordCountComparison : Main Called Successfully")

    println("Performing local word count")

//read the file which is present in local directory  and convert into string
    val fileContents = readFile(localFilePath.toString())

    println("Performing local word count - File Content ->>"+fileContents)

    val localWordCount = runLocalWordCount(fileContents)

    println("SparkHDFSWordCountComparison : Main Called Successfully -> Local Word Count is -
>>" +localWordCount)

    println("Performing local word count Completed !!")

    println("Creating Spark Context")

//Create spark context

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

    val sc = new SparkContext(conf)

// Setting log level to [WARN] for streaming executions  and to override add a custom log4j.properties to the
//classpath

   val rootLogger =Logger.getRootLogger()
    rootLogger.setLevel(Level.ERROR)


    println("Spark Context Created")

    println("Writing local file to DFS")

   val dfsFilename = dfsDirPath + "/dfs_read_write_test"

    val fileRDD = sc.parallelize(fileContents)

   fileRDD.saveAsTextFile(dfsFilename)

    println("Writing local file to DFS Completed")

    println("Reading file from DFS and running Word Count")

    val readFileRDD = sc.textFile(dfsFilename)

    val dfsWordCount = readFileRDD
      .flatMap(_.split(" "))
      .flatMap(_.split("\t"))
      .filter(_.nonEmpty)
      .map(w => (w, 1))
      .countByKey()
      .values
      .sum

    sc.stop()
```

```
//apply if condition to check word count result from both the directories

  if (localWordCount == dfsWordCount)
    {
        println(s"Success! Local Word Count ($localWordCount) " +
       s"and DFS Word Count ($dfsWordCount) agree.")
       }
     else {
          println(s"Failure! Local Word Count ($localWordCount) " +
             s"and DFS Word Count ($dfsWordCount) disagree.")
     }

 }

 /***private def parseArgs(args: Array[String]): Unit = {
  if (args.length != NPARAMS) {
    printUsage()
    System.exit(1)
  }
 }***/

 private def printUsage(): Unit = {

val usage: String = "DFS Read-Write Test\n" +
    "\n" +
    "Usage: localFile dfsDir\n" +
    "\n" +
    "localFile - (string) local file to use in test\n" +
    "dfsDir - (string) DFS directory for read/write tests\n"

  println(usage)
 }



 private def readFile(filename: String): List[String] = {

  val lineIter: Iterator[String] = fromFile(filename).getLines()
  val lineList: List[String] = lineIter.toList
  lineList
 }

 def runLocalWordCount(fileContents: List[String]): Int = {
  fileContents.flatMap(_.split(" "))
    .flatMap(_.split("\t"))
    .filter(_.nonEmpty)
    .groupBy(w => w)
    .mapValues(_.size)
    .values
    .sum
 }
```

}

SparkFileStreamingWordCount.scala | SparkHDFSWordCountComparison.scala

```scala
import java.io.File

import org.apache.spark.{SparkConf, SparkContext}
import scala.io.Source._
import org.apache.log4j.{Level,Logger}


object SparkHDFSWordCountComparison {

  private var localFilePath: File = new File("/home/acadgild/Desktop/Spark_Streaming/match")
  private var dfsDirPath: String = "hdfs://localhost:8020/user"
  private val NPARAMS = 2


  def main(args: Array[String]): Unit = {
    //parseArgs(args)
    println("SparkHDFSWordCountComparison : Main Called Successfully")

    println("Performing local word count")
    val fileContents = readFile(localFilePath.toString())

    println("Performing local word count - File Content ->>"+fileContents)
    val localWordCount = runLocalWordCount(fileContents)

    println("SparkHDFSWordCountComparison : Main Called Successfully -> Local Word Count is ->>"+localWordCount)

    println("Performing local word count Completed !!")

    println("Creating Spark Context")
```

SparkFileStreamingWordCount.scala | SparkHDFSWordCountComparison.scala

```scala
    val conf = new SparkConf().setMaster("local[1]").setAppName("SparkHDFSWordCountComparisonApp")
    val sc = new SparkContext(conf)
    val rootLogger =Logger.getRootLogger()
    rootLogger.setLevel(Level.ERROR)


    println("Spark Context Created")

    println("Writing local file to DFS")
    val dfsFilename = dfsDirPath + "/dfs_read_write_test"
    val fileRDD = sc.parallelize(fileContents)
    fileRDD.saveAsTextFile(dfsFilename)
    println("Writing local file to DFS Completed")

    println("Reading file from DFS and running Word Count")
    val readFileRDD = sc.textFile(dfsFilename)

    val dfsWordCount = readFileRDD
      .flatMap(_.split(" "))
      .flatMap(_.split("\t"))
      .filter(_.nonEmpty)
      .map(w => (w, 1))
      .countByKey()
      .values
      .sum

    sc.stop()

    if (localWordCount == dfsWordCount) {
```

SparkFileStreamingWordCount.scala | SparkHDFSWordCountComparison.scala

```scala
    sc.stop()

    if (localWordCount == dfsWordCount) {
      println(s"Success! Local Word Count ($localWordCount) " +
        s"and DFS Word Count ($dfsWordCount) agree.")
    } else {
      println(s"Failure! Local Word Count ($localWordCount) " +
        s"and DFS Word Count ($dfsWordCount) disagree.")
    }

  }

  private def printUsage(): Unit = {
    val usage: String = "DFS Read-Write Test\n" +
      "\n" +
      "Usage: localFile dfsDir\n" +
      "\n" +
      "localFile - (string) local file to use in test\n" +
      "dfsDir - (string) DFS directory for read/write tests\n"

    println(usage)
  }

  private def readFile(filename: String): List[String] = {
    val lineIter: Iterator[String] = fromFile(filename).getLines()
    val lineList: List[String] = lineIter.toList
    lineList
  }
```

```scala
  private def readFile(filename: String): List[String] = {
    val lineIter: Iterator[String] = fromFile(filename).getLines()
    val lineList: List[String] = lineIter.toList
    lineList
  }

  def runLocalWordCount(fileContents: List[String]): Int = {
    fileContents.flatMap(_.split(" "))
      .flatMap(_.split("\t"))
      .filter(_.nonEmpty)
      .groupBy(w => w)
      .mapValues(_.size)
      .values
      .sum
  }
}
```

Output of the second spark application:

```
Console
<terminated> SparkHDFSWordCountComparison$ [Scala Application] /usr/java/jdk1.8.0_151/bin/java (May 25, 2018, 12:29:34 PM)
SparkHDFSWordCountComparison : Main Called Successfully
Performing local word count
Performing local word count - File Content ->>List(hi this is a spark streaming case study, this file is present in
SparkHDFSWordCountComparison : Main Called Successfully -> Local Word Count is ->>21
Performing local word count Completed !!
Creating Spark Context
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
18/05/25 12:29:35 INFO SparkContext: Running Spark version 2.2.1
18/05/25 12:29:36 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-ja
18/05/25 12:29:36 WARN Utils: Your hostname, localhost.localdomain resolves to a loopback address: 127.0.0.1; using
18/05/25 12:29:36 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
18/05/25 12:29:36 INFO SparkContext: Submitted application: SparkHDFSWordCountComparisonApp
18/05/25 12:29:36 INFO SecurityManager: Changing view acls to: acadgild
18/05/25 12:29:36 INFO SecurityManager: Changing modify acls to: acadgild
18/05/25 12:29:36 INFO SecurityManager: Changing view acls groups to:
18/05/25 12:29:36 INFO SecurityManager: Changing modify acls groups to:
18/05/25 12:29:36 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users  with vie
18/05/25 12:29:36 INFO Utils: Successfully started service 'sparkDriver' on port 41329.
18/05/25 12:29:37 INFO SparkEnv: Registering MapOutputTracker
18/05/25 12:29:37 INFO SparkEnv: Registering BlockManagerMaster
18/05/25 12:29:37 INFO BlockManagerMasterEndpoint: Using org.apache.spark.storage.DefaultTopologyMapper for getting
18/05/25 12:29:37 INFO BlockManagerMasterEndpoint: BlockManagerMasterEndpoint up
18/05/25 12:29:37 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-075763db-61aa-4009-8175-8cc24a601
18/05/25 12:29:37 INFO MemoryStore: MemoryStore started with capacity 886.8 MB
18/05/25 12:29:37 INFO SparkEnv: Registering OutputCommitCoordinator
18/05/25 12:29:37 INFO Utils: Successfully started service 'SparkUI' on port 4040.
18/05/25 12:29:37 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.0.43:4040
18/05/25 12:29:37 INFO Executor: Starting executor ID driver on host localhost
18/05/25 12:29:37 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferServic
```


```
18/05/25 12:29:37 INFO SparkEnv: Registering OutputCommitCoordinator
18/05/25 12:29:37 INFO Utils: Successfully started service 'SparkUI' on port 4040.
18/05/25 12:29:37 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://192.168.0.43:4040
18/05/25 12:29:37 INFO Executor: Starting executor ID driver on host localhost
18/05/25 12:29:37 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferServic
```


```
18/05/25 12:29:37 INFO NettyBlockTransferService: Server created on 192.168.0.43:36391
18/05/25 12:29:37 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replicat
18/05/25 12:29:37 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.0.43, 36391, Non
18/05/25 12:29:38 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.0.43:36391 with 886.8 MB RAM,
18/05/25 12:29:38 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.0.43, 36391, None
18/05/25 12:29:38 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 192.168.0.43, 36391, None)
Spark Context Created
Writing local file to DFS
Writing local file to DFS Completed
Reading file from DFS and running Word Count
Success! Local Word Count (21) and DFS Word Count (21) agree.
```

Above screen shot shows that, word count of file from local file directory and HDFS directory file is same.

Output file in HDFS directory shown below in screen shot.

/user/dfs_read_write_test

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	acadgild	supergroup	0 B	3	128 MB	_SUCCESS
-rw-r--r--	acadgild	supergroup	116 B	3	128 MB	part-00000

```
File  Edit  View  Search  Terminal  Tabs  Help
acadgild@localhost:~                    acadgild@localhost:~
[acadgild@localhost ~]$ hdfs dfs -cat /user/dfs_read_write_test/part-00000
18/05/25 12:37:56 WARN util.NativeCodeLoader: Unable to load native-hadoop library
asses where applicable
hi this is a spark streaming case study
this file is present in local directory
spark streaming will read this file
[acadgild@localhost ~]$
```

Above screen, shot shows the content of the output file **part-00000** which same as the file present in the local directory.