1) Create a new Scala Project, Package and Class like the first time.

2) The code looks like this :-

```
package com.evenkat

import org.apache.spark.SparkConf
import org.apache.spark.streaming.StreamingContext
import org.apache.spark.streaming._
import org.apache.spark.storage.StorageLevel._
import org.apache.spark.rdd.RDD
import org.apache.spark.streaming.dstream.DStream
import org.apache.spark.streaming.dstream.ForEachDStream

  object FirstStreaming {

  def main(args: Array[String]){

   println("Creating Spark Configuration")
    //Create an Object of Spark Configuration

   val conf = new SparkConf()
    //Set the logical and user defined Name of this Application
    conf.setAppName("My First Spark Streaming Application")

    println("Retreiving Streaming Context from Spark Conf")

    //Retrieving Streaming Context from SparkConf Object.
    //Second parameter is the time interval at which //streaming data will be
divided into batches

    val streamCtx = new StreamingContext(conf, Seconds(60))

    //Define the type of Stream. Here we are using TCP //Socket as text stream,
    //It will keep watching for the incoming data from a //specific machine
(localhost) and port (9087)
    //Once the data is retrieved it will be saved in the //memory and in case
memory
    //is not sufficient, then it will store it on the Disk
    //It will further read the Data and convert it into DStream

   val lines = streamCtx.socketTextStream("localhost", 9087,
MEMORY_AND_DISK_SER_2)

    //Apply the Split() function to all elements of DStream
    //which will further generate multiple new records from //each record in
Source Stream
    //And then use flatmap to consolidate all records and //create a new DStream.

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

    //Now, we will count these words by applying a using map()
    //map() helps in applying a given function to each //element in an RDD.

   val pairs = words.map(word => (word, 1))

    //Further we will aggregate the value of each key by //using/applying the
given function.
```

```
    val wordCounts = pairs.reduceByKey(_ + _)

     //Lastly we will print all Values
     //wordCounts.print(20)

     printValues(wordCounts,streamCtx)

     //Most important statement which will initiate the //Streaming Context

     streamCtx.start();

    //Wait till the execution is completed.

    streamCtx.awaitTermination();

    }

    def printValues(stream:DStream[(String,Int)],streamCtx: StreamingContext){

      stream.foreachRDD(foreachFunc)

      def foreachFunc = (rdd: RDD[(String,Int)]) => {
       val array = rdd.collect()
       println("---------Start Printing Results----------")
       for(res<-array){
         println(res)
       }
       println("---------Finished Printing Results----------")
      }
    }

}
```

==========➔

The client application will allow the user to type messages on the console and capture data. Once data is captured, it will immediately send it to the specific port number (socket) where our streaming job is waiting for the data.

1) Create a new java class called ClientApp inside the package in the current project.

2) The code looks like this:

```
package com.evenkat;

import java.net.*;
import java.io.*;

public class ClientApp {

       public static void main(String[] args) {
              try{
                  System.out.println("Defining new Socket");
                  ServerSocket soc = new ServerSocket(9087);
                  System.out.println("Waiting for Incoming Connection");
                  Socket clientSocket = soc.accept();

                  System.out.println("Connection Received");
```

```
                OutputStream outputStream = clientSocket.getOutputStream();
                //Keep Reading the data in a Infinite loop and send it //over to
the Socket.
                while(true){
                    PrintWriter out = new PrintWriter(outputStream, true);
                    BufferedReader read = new BufferedReader(new
InputStreamReader(System.in));
                    System.out.println("Waiting for user to input some data");
                    String data = read.readLine();
                    System.out.println("Data received and now writing it to
Socket");
                    out.println(data);
                }

            }catch(Exception e ){
                e.printStackTrace();
            }

      }

}
```

The packaging and deployment of Spark Streaming jobs on the Spark cluster is very similar to the process followed for deploying standard Spark batch jobs but with one additional step, in which we need to ensure that our data stream providing the data is up and running before our job is deployed.

2) Create a new jar file and move it to the SPARK_HOME like before

3) Execute the Jar file with the –classpath command like this:

java -classpath StreamingFirst.jar com.evenkat.ClientApp

```
notroot@ubuntu:~/lab/software/spark-1.6.0-bin-hadoop2.6$ ls
bin          data      lib         NOTICE   README.md   ScalaFirst.jar
CHANGES.txt  ec2       LICENSE     python   RELEASE     StreamingFirst.jar
conf         examples  licenses    R        sbin
notroot@ubuntu:~/lab/software/spark-1.6.0-bin-hadoop2.6$ java -classpath Streami
ngFirst.jar com.evenkat.ClientApp
Defining new Socket
Waiting for Incoming Connection
```

4) Open a new console and navigate to the SPARK_HOME\bin and execute like this:

./spark-submit --class com.evenkat.FirstStreaming --master local[2] ../StreamingFirst.jar

```
notroot@ubuntu:~/lab/software/spark-1.6.0-bin-hadoop2.6/bin$ ./spark-submit --class com.eve
nkat.FirstStreaming --master local[2] ../StreamingFirst.jar
```

6) Type in some lines in the ClientApp Window

==========➔

1) Create a new class called FirstSteaming_Java inside the package.

2) The code looks like this

```
package com.evenkat;
```

```
import java.util.Arrays;

import org.apache.spark.*;
import org.apache.spark.api.java.function.*;
import org.apache.spark.storage.StorageLevel;
import org.apache.spark.streaming.*;
import org.apache.spark.streaming.api.java.*;

import scala.Tuple2;

public class FirstStreaming_Java {

	public static void main(String[] args) {

		System.out.println("Creating Spark Configuration");
	    //Create an Object of Spark Configuration

		SparkConf conf = new SparkConf();
	    //Set the logical and user defined Name of this Application

		conf.setAppName("My First Spark Streaming Application");
	    //Define the URL of the Spark Master.
	    //Useful only if you are executing Scala App directly //from the
console.
	    //We will comment it for now but will use later
	    //conf.setMaster("spark://ip-10-237-224-94:7077")

	    System.out.println("Retreiving Streaming Context from Spark Conf");
	    //Retrieving Streaming Context from SparkConf Object.
	    //Second parameter is the time interval at which //streaming data will
be divided into batches

	    JavaStreamingContext streamCtx = new JavaStreamingContext(conf,
Durations.seconds(2));

	    //Define the type of Stream. Here we are using TCP //Socket as text
stream,
	    //It will keep watching for the incoming data from a //specific machine
(localhost) and port (9087)
	    //Once the data is retrieved it will be saved in the //memory and in
case memory
	    //is not sufficient, then it will store it on the Disk.
	    //It will further read the Data and convert it into DStream

	    JavaReceiverInputDStream<String> lines =
streamCtx.socketTextStream("localhost",
9087,StorageLevel.MEMORY_AND_DISK_SER_2());

	    //Apply the x.split() function to all elements of
//JavaReceiverInputDStream
	    //which will further generate multiple new records from //each record
in Source Stream
	    //And then use flatmap to consolidate all records and //create a new
JavaDStream.

	    JavaDStream<String> words = lines.flatMap( new FlatMapFunction<String,
String>() {
	        @Override public Iterable<String> call(String x) {
	            return Arrays.asList(x.split(" "));
```

```
            }
        });


        //Now, we will count these words by applying a using //mapToPair()
        //mapToPair() helps in applying a given function to //each element in
an RDD
        //And further will return the Scala Tuple with "word" //as key and
value as "count".

        JavaPairDStream<String, Integer> pairs = words.mapToPair(
            new PairFunction<String, String, Integer>() {
                @Override
                public Tuple2<String, Integer> call(String s) throws Exception
{
                    return new Tuple2<String, Integer>(s, 1);
                }
        });


        //Further we will aggregate the value of each key by //using/applying
the given function.

        JavaPairDStream<String, Integer> wordCounts = pairs.reduceByKey(
            new Function2<Integer, Integer, Integer>() {
            @Override public Integer call(Integer i1, Integer i2) throws
Exception {
                return i1 + i2;
            }
        });


        //Lastly we will print First 10 Words.
        //We can also implement custom print method for //printing all values,
        //as we did in Scala example.

        wordCounts.print(10);
        //Most important statement which will initiate the //Streaming Context

        streamCtx.start();
        //Wait till the execution is completed.

        streamCtx.awaitTermination();
    }
}
```

==========➔