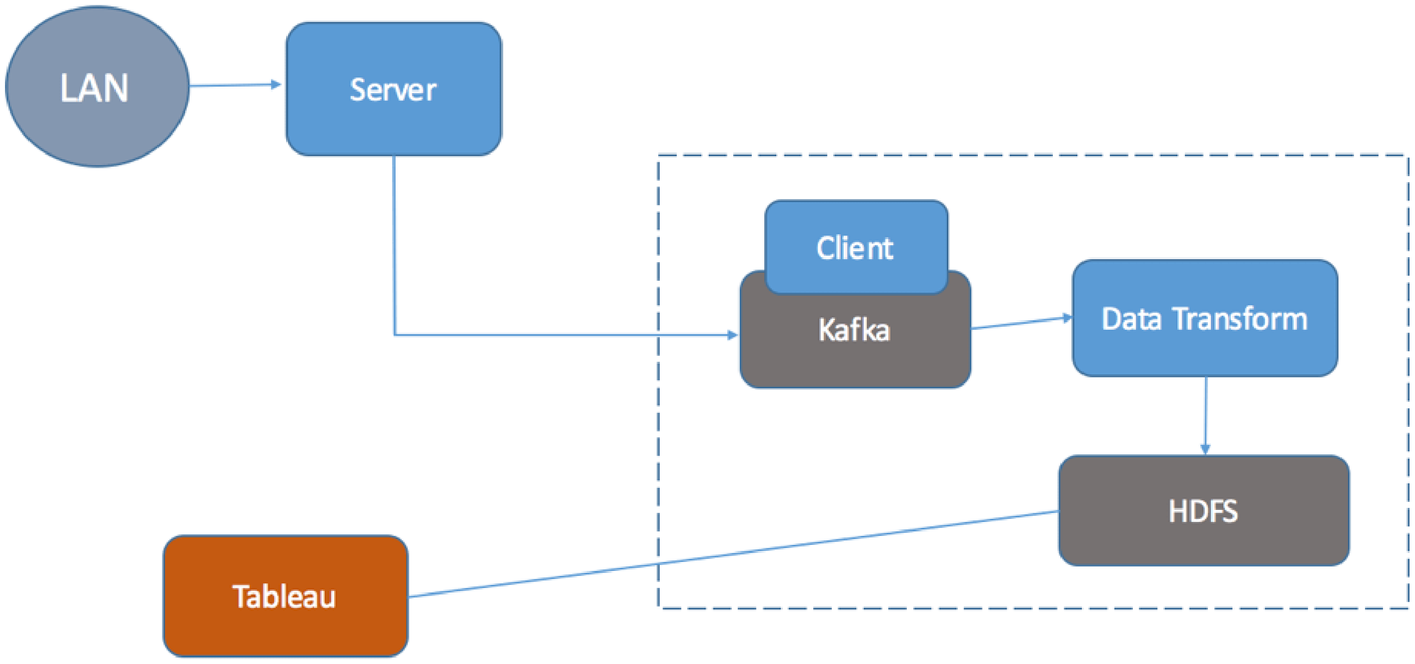
Network Data Ingestion and Transformation

A data collection server, shown in the diagram below, is collecting data in real time from the local network. The data collected by the Server is working with a Client residing in DLP to transfer the network data collected through Kafka. Using Kafka socket code, we are making a connection to the client and capture the network data and send it to a Kafka topic. From this topic, data will be moved to HDFS by a Consumer program. With the data present in HDFS, we can then further transform it using our program and visualize the data before and after the transformation in different chart format such as pie chart, bar chart etc. using Tableau tool. The diagram shows how exactly network data flows from a local network through Kafka in HDFS and gets transformed.



In this lab, the network stream will be pre-created, so it will be available live in the platform for access. The steps for creating a network stream is described below. User doesn't need to operate these to create a stream.

<font color='red'>Request access to the Data Learning Platform by sending a message to:</font> datalearningplatform@cisco.com

Lab Objectives

- How to ingest a network stream in DLP Platform

- How to get network data from the HDFS.
- Learn the process of transform network data by creating a transform function using IDE
- Learn how to save a file to HDFS and visualize data

Prerequisites

- Knowledge on Hadoop to store the network data.
- Studied platform user guide.
- Basic knowledge of how spark works.
- Use of Chrome OS

Lab Settings

“Data Repository” section is allowing you to create network real-time data stream. Kafka’s producer will push the Network traffic generated data to Kafka cluster and Consumer will consume that data and save that into HDFS in real-time. From HDFS, we can visualize the data using visualization tool. This application will allow you to use the existing Kafka’s Producer & Consumer function to ingest your real time network data into HDFS environment.

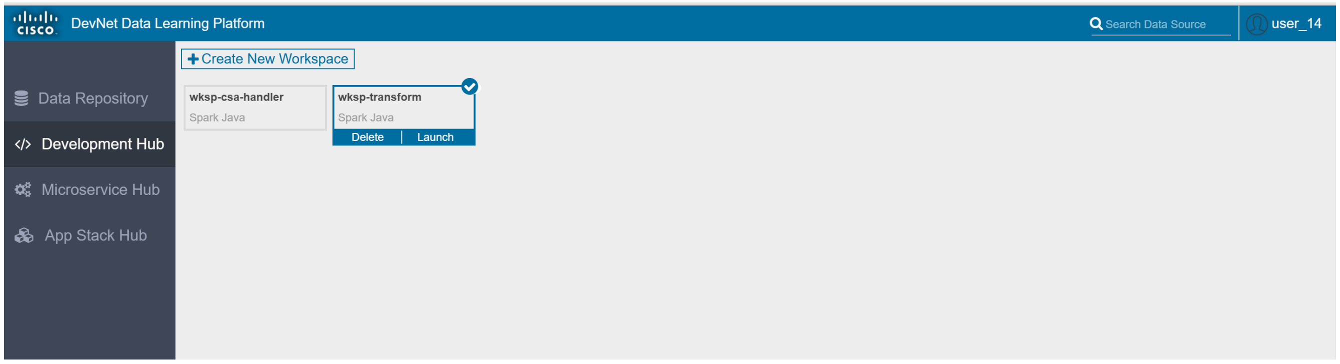
<b>N.B.</b> DLP platform provide a default Kafka’s Producer, Consumer and Network traffic generator. User can select there own network data and use DLP provided Kafka’s component to process the network data. User can define and deploy their own service also.

Step 1: Explore Data Learning Platform (DLP)

<font color=‘red’>Request access to the Data Learning Platform by sending a message to:</font> datalearningplatform@cisco.com

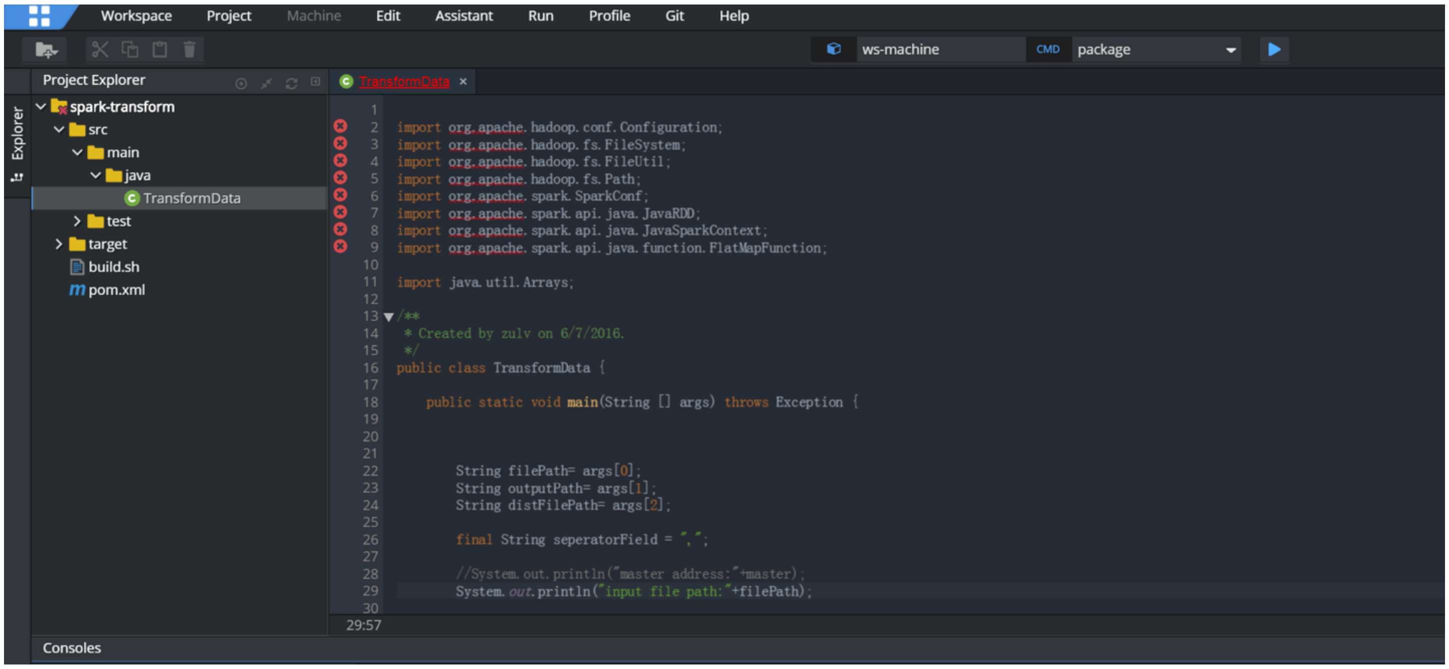
1. Select Workspace from “Development Hub”. The pre-defined workspace is <b>“wksp-transform”</b>

2. Select the workspace and launch the IDE by clicking the Launch link.



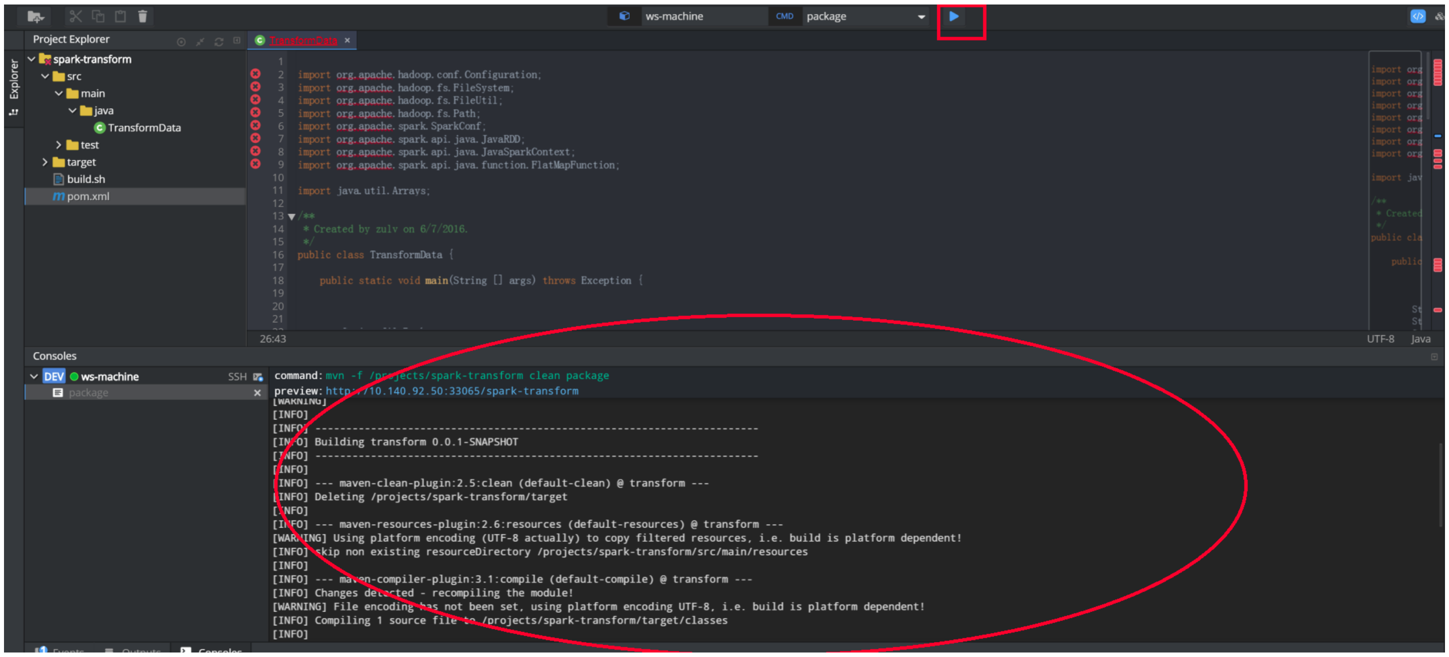
Step 2: Developing code in IDE

a) Go to Workspace to put your code (sample code in code-section) here.



```java

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileUtil;
import org.apache.hadoop.fs.Path;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;

import java.util.Arrays;

/**
 * Created by zulv on 6/7/2016.
 */
public class TransformData {

    public static void main(String [] args) throws Exception {



        String filePath= args[0];
        String outputPath= args[1];
        String distFilePath= args[2];

        final String seperatorField = ",";

        //System.out.println("master address:"+master);
        System.out.println("input file path:"+filePath);

```

b) Select the <b>package</b> and click on <b>Blue</b> button to build. After successful build process, change it to <b>run</b> and then click on <b>Blue</b> button again.

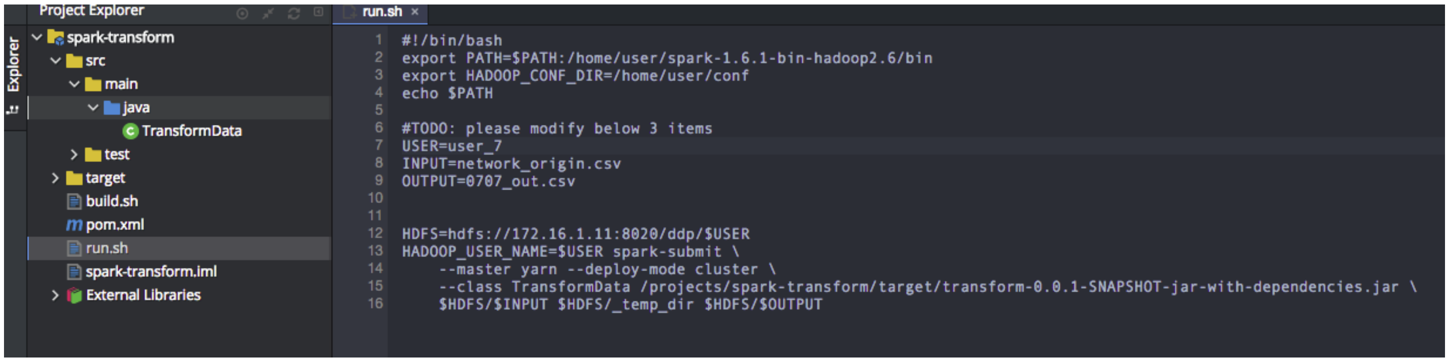


```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FileUtil;
import org.apache.hadoop.fs.Path;
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;
import org.apache.spark.api.java.function.FlatMapFunction;

import java.util.Arrays;

/**
 * Created by zulv on 6/7/2016.
 */
public class TransformData {

    public static void main(String [] args) throws Exception {
```

```
command: mvn -f /projects/spark-transform clean package
preview: http://10.140.92.50:33065/spark-transform
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building transform 0.0.1-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ transform ---
[INFO] Deleting /projects/spark-transform/target
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ transform ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /projects/spark-transform/src/main/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ transform ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 1 source file to /projects/spark-transform/target/classes
[INFO]
```

Step 3: Run the Program

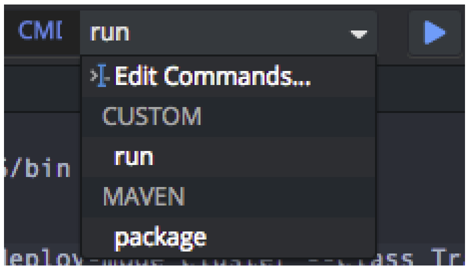
a) Double click run.sh to open it, change parameters as described below



```
#!/bin/bash
export PATH=$PATH:/home/user/spark-1.6.1-bin-hadoop2.6/bin
export HADOOP_CONF_DIR=/home/user/conf
echo $PATH

#TODO: please modify below 3 items
USER=user_7
INPUT=network_origin.csv
OUTPUT=0707_out.csv


HDFS=hdfs://172.16.1.11:8020/ddp/$USER
HADOOP_USER_NAME=$USER spark-submit \
    --master yarn --deploy-mode cluster \
    --class TransformData /projects/spark-transform/target/transform-0.0.1-SNAPSHOT-jar-with-dependencies.jar \
    $HDFS/$INPUT $HDFS/_temp_dir $HDFS/$OUTPUT
```

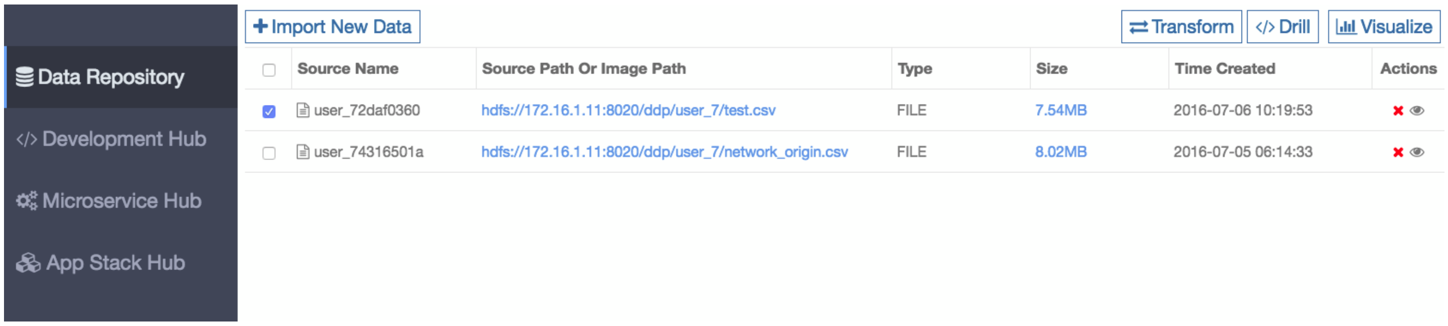
- USER: your current login user name
- INPUT: The file name in “Data Repository” section which you want to transform.
- OUTPUT: The output file name that your code will generate.

b) Choose a Command run under Edit Commands, then click the blue button ▶ to run, as in the Terminal.



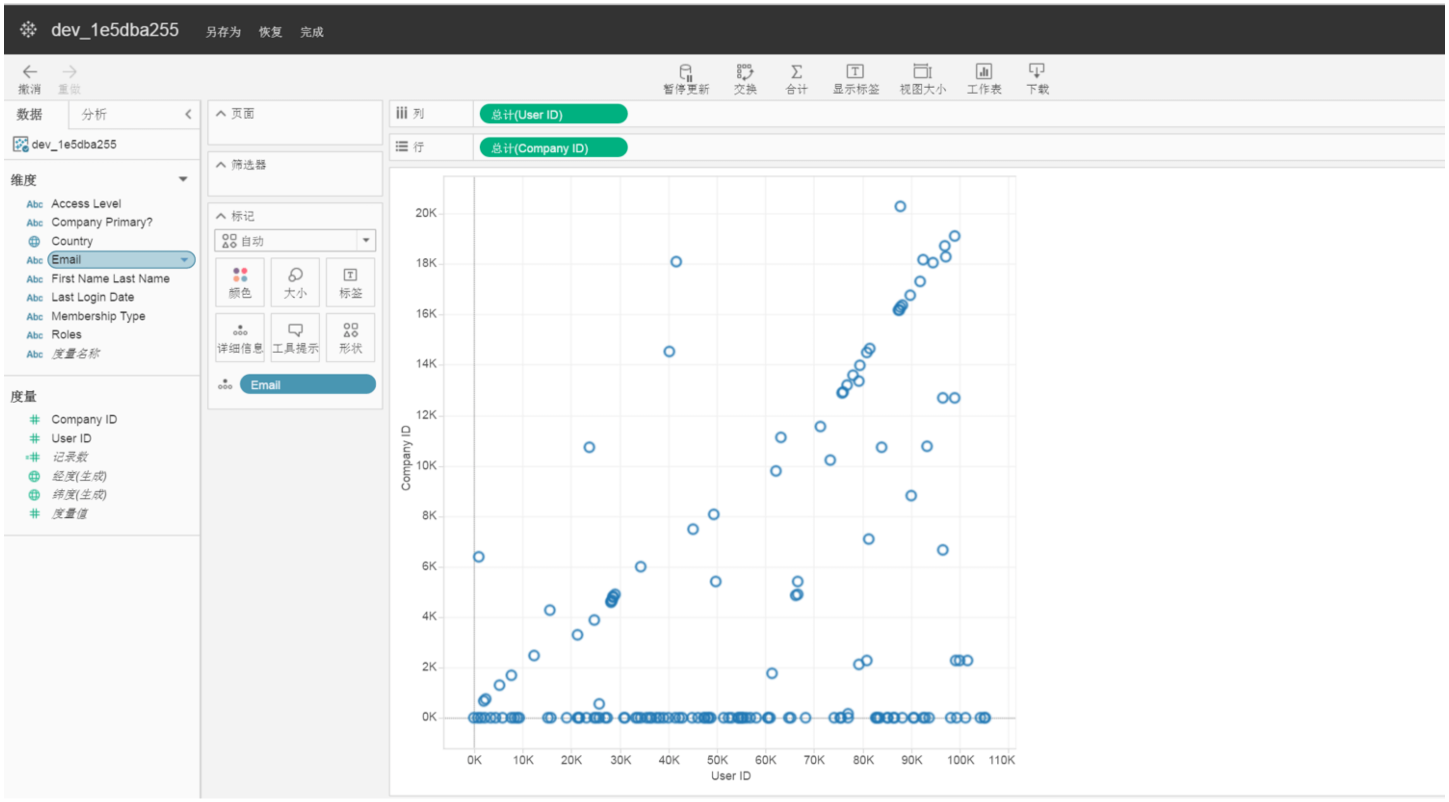
Step 3: View Data in Visualization Tool

Select file (your output file) you want to visualize, then click Visualize button.



	Source Name	Source Path Or Image Path	Type	Size	Time Created	Actions
✓	user_72daf0360	hdfs://172.16.1.11:8020/ddp/user_7/test.csv	FILE	7.54MB	2016-07-06 10:19:53	✖ 👁
	user_74316501a	hdfs://172.16.1.11:8020/ddp/user_7/network_origin.csv	FILE	8.02MB	2016-07-05 06:14:33	✖ 👁

Visualize View:



Sample Code: Transform a network data file and store transformed file to HDFS.

Below is the sample code which is used for getting the csv file of network data from HDFS environment, perform the transformation and store the transformed data into HDFS.

```
//Import the below libraries which are used to run the code.
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.FileUtil;

import org.apache.hadoop.fs.Path;

import org.apache.spark.SparkConf;

import org.apache.spark.api.java.JavaRDD;

import org.apache.spark.api.java.JavaSparkContext;

import org.apache.spark.api.java.function.FlatMapFunction;


import java.util.Arrays;

//Class: TransformData. A class that transform the network data

public class TransformData {


    public static void main(String [] args) throws Exception {




  //three parameters here

        String filePath= args[0];

        String outputPath= args[1];

        String distFilePath= args[2];


        final String seperatorField = ",";


        //System.out.println("master address:"+master);

        System.out.println("input file path:"+filePath);

// here we are giving application name as “File transform”

        SparkConf conf = new SparkConf().setAppName("File transform");

        JavaSparkContext sc = new JavaSparkContext(conf);


        JavaRDD<String> textFile = sc.textFile(filePath);


        JavaRDD<String> cols = textFile.flatMap(new FlatMapFunction<String, String>() {

            public Iterable<String> call(String s) {

                String [] splits = s.split(seperatorField) ;
                String extractedCols = "";

                for(int i= 1 ;i<= 6 ;i ++ ){
                    extractedCols += splits[i ] + seperatorField ;
                }

                if(extractedCols.endsWith(seperatorField)){
                    extractedCols = extractedCols.substring(0,extractedCols.length() - 1)
 ;
                }

                return Arrays.asList(extractedCols);
            }
        });



       cols.coalesce(1).saveAsTextFile(outputPath);

       boolean bool = getMergeInHdfs(outputPath,distFilePath);
        System.out.println("merge success:"+bool);




    }

    public static boolean getMergeInHdfs(String src, String dest) throws IllegalArgumentEx
ception, Exception {

        Configuration config = new Configuration();
        FileSystem fs = FileSystem.get(config);
        Path srcPath = new Path(src);
        Path dstPath = new Path(dest);

        // Check if the path already exists
        if (!(fs.exists(srcPath))) {
            System.out.println("Path " + src + " does not exists!");
            return false;
        }

        if ((fs.exists(dstPath))) {
            System.out.println("File Path " + dest + " exists!");
            return false;
        }

        return FileUtil.copyMerge(fs, srcPath, fs, dstPath, true, config, null);
    }
}
```