**Spark Hackathon Project -** By Shunmuga Dinesh [WE 33]

**Project Contents:**

1. **Spark Core and SQL script**
2. **Spark streaming script**
3. **Allmethods (Resubale methods)**
4. **Spark UI Visualization snapshtos**

**1.Spark Core and SQL script**

package org.inceptez.sparkhack

import org.inceptez.hack.\_

import org.apache.spark.SparkContext

import org.apache.spark.sql.SparkSession

import org.apache.spark.rdd.\_

import org.apache.hadoop.fs.\_

import org.apache.spark.streaming.\_

import java.net.URI;

import org.apache.hadoop.conf.Configuration

object sparkhack {

val spark = SparkSession.builder().appName("sparkHackthon").master("local[\*]")

.config("spark.history.fs.logDirectory", "file:///tmp/spark-events")

.config("spark.eventLog.dir", "file:////tmp/spark-events")

.config("spark.eventLog.enabled", "true")

.config("hive.metastore.uris","thrift://localhost:9083")

.config("spark.sql.warehouse.dir","hdfs://localhost:54310/user/hive/warehouse")

.enableHiveSupport().getOrCreate();

val sc= spark.sparkContext;

val sqlc=spark.sqlContext;

val reusableObj = new allmethods();

val configuration = new Configuration();

val fs = FileSystem.get(new URI("hdfs://localhost:54310"), configuration);

def main(args:Array[String]){

val insurancefile1 = "hdfs://localhost:54310/user/hduser/sparkhack2/insuranceinfo1.csv";

val insurancefile2 = "hdfs://localhost:54310/user/hduser/sparkhack2/insuranceinfo2.csv";

val hdfsFileLocation="hdfs://localhost:54310/user/hduser/hackathon/";

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Part A\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//1. Load the file1 (insuranceinfo1.csv) from HDFS using textFile API into an RDD insuredata

val insuredata = sc.textFile(insurancefile1);

/\*2. Remove the header line from the RDD contains column names.

Hint: Use rdd1.first -> rdd1.filter to remove the first line identified to remove the header. \*/

val insuranceHeader=insuredata.first.toString;

var rdd1 = insuredata.filter(x=>x!=insuranceHeader);

//3. Display the count and show few rows and check whether header is removed.

println("Total count in insuranceinfo1 without header: "+rdd1.count);

println("sample data: " + rdd1.take(3).foreach(println));

//4. Remove the blank lines in the rdd.

//Hint: Before splitting, trim it, calculate the length and filter only row size is not = 0.

var trimrrdd = rdd1.map(x=>x.trim).filter(x=>x.size!=0);

//5. Map and split using ‘,’ delimiter.

var splitrdd = trimrrdd.map(x=>x.split(",",-1));

//6. Filter number of fields are equal to 10 columns only - analyze why we are doing this

var filteredrdd= splitrdd.filter(\_.length==10);

//7. Add case class namely insureclass with the field names used as per the header record in the file and apply to the above data to create schemaed RDD.

// declared outside out object

var schemaInsuredata1=filteredrdd.map(x=> insureclass(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8),x(9)))

//8. Take the count of the RDD created in step 7 and step 1 and print how many rows are removed in the cleanup process of removing fields does not equals 10.

println("No of rows are removed in the cleanup process of removing fields does not equals 10");

rowsRemoved(insuredata, schemaInsuredata1);

//9. Create another RDD namely rejectdata and store the row that does not equals 10 columns

val rejectdata1 = splitrdd.filter(\_.length!=10);

//10. Load the file2 (insuranceinfo2.csv) from HDFS using textFile API into an RDD insuredata2

var insuredata2 = sc.textFile(insurancefile2);

println("insuranceinfo2 process started");

/\*11. Repeat from step 2 to 9 for this file also and

eg: remove the records with pattern given below.

,,,,,,,13,Individual,Yes \*/

val rdd2 = reusableObj.removeHeader(insuredata2); // Step 2 and 3

val trimrrdd2 = rdd2.map(x=>x.trim).filter(x=>x.size!=0); // Step 4

val splitrdd2 = trimrrdd2.map(x=>x.split(",",-1)); // Step 5

val filteredrdd2= splitrdd2.filter(\_.length==10); //Step 6

var schemaInsuredata2=filteredrdd2.map(x=> insureclass(x(0),x(1),x(2),x(3),x(4),x(5),x(6),x(7),x(8),x(9))) //Step 7

val rejectdata2 = splitrdd2.filter(\_.length!=10); //Step 9

//create the final rdd including the filtering of the records that contains blank IssuerId,IssuerId2

schemaInsuredata2 = schemaInsuredata2.filter(x=>x.IssuerId!="" && x.IssuerId2!="");

rowsRemoved(insuredata2, schemaInsuredata2);

println("Final rows in insuranceinfo2 after removing empty issuerId");

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part A 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// 12. Merge the both header removed RDDs derived in steps 7 and 11 into an RDD namely insuredatamerged

val insuredatamerged = schemaInsuredata1.union(schemaInsuredata2);

//13. Cache it either to memory or any other persistence levels you want, display only first few rows

insuredatamerged.persist();

println("sample data: " + insuredatamerged.take(3).foreach(println));

//14. Calculate the count of rdds created in step 7+11 and rdd in step 12, check whether they are matching.

println("No of rows before merge: " + (schemaInsuredata1.count + schemaInsuredata2.count));

println("No of rows after merge: " + insuredatamerged.count);

//15. Remove duplicates from this merged RDD created in step 12 and print how many duplicate rows are there.

val distinctData = insuredatamerged.distinct();

println("No of duplicate rows: " + (insuredatamerged.count - distinctData.count));

//16. Increase the number of partitions to 8 and name it as insuredatarepart.

val insuredatarepart = distinctData.repartition(8);

//17. Split the above RDD using the businessdate field into rdd\_20191001 and rdd\_ 20191002 based on the BusinessDate of 2019-10-01 and 2019-10-02 respectively

val rdd\_20191001 = insuredatarepart.filter { x => x.BusinessDate== "2019-10-01" };

val rdd\_20191002 = insuredatarepart.filter { x => x.BusinessDate== "2019-10-02" };

//18. Store the RDDs created in step 9, 12, 17 into HDFS locations.

val filePaths=Array(hdfsFileLocation+"rejectdata1", hdfsFileLocation+"insuredatamerged", hdfsFileLocation+"rdd\_20191001", hdfsFileLocation+"rdd\_20191002");

reusableObj.deleteFileIfExist(fs, filePaths, true); // delete file if already exists

rejectdata1.saveAsTextFile("hdfs://localhost:54310/user/hduser/hackathon/rejectdata1");

insuredatamerged.saveAsTextFile("hdfs://localhost:54310/user/hduser/hackathon/insuredatamerged");

rdd\_20191001.saveAsTextFile("hdfs://localhost:54310/user/hduser/hackathon/rdd\_20191001");

rdd\_20191002.saveAsTextFile("hdfs://localhost:54310/user/hduser/hackathon/rdd\_20191002");

println("Rdds area saved in hdfs successfully");

/\*20. Create structuretype for all the columns as per the insuranceinfo1.csv.

Hint: Do it carefully without making typo mistakes. Fields issuerid, issuerid2 should be of

IntegerType, businessDate should be DateType and all other fields are StringType,

ensure to import sql.types library.

\*/

import org.apache.spark.sql.types.{StructType, StructField, StringType, IntegerType, DateType}

val insureschema = StructType(Array(StructField("IssuerId", IntegerType,true),

StructField("IssuerId2", IntegerType, true),StructField("BusinessDate", DateType,true),

StructField("StateCode", StringType, true),StructField("SourceName", StringType,true),

StructField("NetworkName", StringType, true),StructField("NetworkURL", StringType,true),

StructField("custnum", StringType, true),StructField("MarketCoverage", StringType,true),

StructField("DentalOnlyPlan", StringType, true)));

/\*19. Convert the RDD created in step 16 above into Dataframe namely insuredaterepartdf

applying the structtype created in the step 20 given in the next usecase.

Hint: Think of converting df to rdd then use createdataframe to apply schema some

thing like this..

Eg: createDataframe(df19.rdd,structuretype) \*/

import spark.implicits.\_;

import org.apache.spark.sql.functions.\_

val tempdf = insuredatarepart.toDF("IssuerId","IssuerId2","BusinessDate","StateCode","SourceName","NetworkName","NetworkURL","custnum","MarketCoverage","DentalOnlyPlan");

val updateddf = tempdf.withColumn("IssuerId",col("IssuerId").cast(IntegerType)).withColumn("IssuerId2", col("IssuerId2").cast(IntegerType)).

withColumn("BusinessDate", col("BusinessDate").cast(DateType));

updateddf.printSchema()

var insuredaterepartdf = spark.createDataFrame(updateddf.rdd, insureschema);

println("Data Frame created using strcuture");

insuredaterepartdf.printSchema()

println("sample Date: "+insuredaterepartdf.count());

println("sample Date after applying schema: "+insuredaterepartdf.sort('IssuerId.desc).show(5));

insuredaterepartdf.createOrReplaceTempView("partA");

println("not null: "+spark.sql("Select \* from partA where BusinessDate is not null").count());

println("null: "+spark.sql("Select \* from partA where BusinessDate is null").count());

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part A 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Part B\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

21. Create dataset using the csv module with option to escape ‘,’ accessing the

insuranceinfo1.csv and insuranceinfo2.csv files, apply the schema of the structure type

created in the step 20.

\*/

val csvdf= spark.read.option("header", true).option("escape", ",").option("dateFormat", "dd-MM-yyyy").schema(insureschema).csv(insurancefile1,insurancefile2);

/\*

22. Apply the below DSL functions in the DFs created in step 21.

a. Rename the fields StateCode and SourceName as stcd and srcnm respectively.

b. Concat IssuerId,IssuerId2 as issueridcomposite and make it as a new field

Hint : Cast to string and concat.

c. Remove DentalOnlyPlan column

d. Add columns that should show the current system date and timestamp with the

fields name of sysdt and systs respectively.

\*/

val date=java.time.LocalDate.now.toString;

val time=java.time.LocalTime.now.toString;

val modifiedinsuredf = csvdf.withColumnRenamed("StateCode", "stcd").withColumnRenamed("SourceName", "srcnm").

withColumn("issueridcomposite", concat('IssuerId.cast(StringType),lit(" "),'IssuerId2.cast(StringType))).

withColumn("sysdt", current\_date()).withColumn("systs", current\_timestamp()).drop("DentalOnlyPlan");

/\*

23. Remove the rows contains null in any one of the field and count the number of rows

which contains all columns with some value.

Hint: Use drop.na options

\*/

val insuredfWithNoNull=modifiedinsuredf.na.drop();

println("no of rows contains all call: " + insuredfWithNoNull.count());

// 24. -> reusableObj.remspecialchar(value)

// 25. Import the package, instantiate the class and register the method generated in step 24 as a udf for invoking in the DSL function.

val udfRemoveSpecialChar = udf(reusableObj.remspecialchar \_);

val sampleStr="01[WelCome]\_To-{Hakathon}";

println("Remove special char: "+sampleStr);

println("After passing to function: "+ reusableObj.remspecialchar(sampleStr));

// 26. Call the above udf in the DSL by passing NetworkName column as an argument to get the special characters removed DF.

val specialCharRemoveddf = insuredfWithNoNull.withColumn("NetworkName",udfRemoveSpecialChar('NetworkName));

// 27. Save the DF generated in step 26 in JSON into HDFS with overwrite option.

specialCharRemoveddf.write.mode("overwrite").json(hdfsFileLocation+"insureSpecialCharRemovedjson");

spark.read.json(hdfsFileLocation+"insureSpecialCharRemovedjson").show(5,false);

println("JSON read write")

// 28. Save the DF generated in step 26 into CSV format with header name as per the DF and delimited by ~ into HDFS with overwrite option.

specialCharRemoveddf.coalesce(1).write.mode("overwrite").option("header", true).option("delimiter", "~").csv(hdfsFileLocation+"insureSpecialCharRemovedcsv");

spark.read.option("header","true").csv(hdfsFileLocation+"insureSpecialCharRemovedcsv").show(5,false);

println("CSV read write")

// 29. Save the DF generated in step 26 into hive external table and append the data without overwriting it.

//spark.sql("create external table if not exists customerhive1())")

specialCharRemoveddf.write.mode("append").saveAsTable("default.insureSpecialCharRemovedhive");

spark.sql("select count(1) from default.insureSpecialCharRemovedhive").show();

spark.sql("select \* from default.insureSpecialCharRemovedhive limit 5").show();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part B 3\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

4. Tale of handling RDDs, DFs and TempViews (20% Completion) –

Total 75%

Loading RDDs, split RDDs, Load DFs, Split DFs, Load Views, Split Views, write UDF, register to

use in Spark SQL, Transform, Aggregate, store in disk/DB

Use RDD functions:

30. Load the file3 (custs\_states.csv) from the HDFS location, using textfile API in an RDD

custstates, this file contains 2 type of data one with 5 columns contains customer

master info and other data with statecode and description of 2 columns.

\*/

val custstatespath = "hdfs://localhost:54310/user/hduser/sparkhack2/custs\_states.csv";

val custstates = sc.textFile(custstatespath);

/\*31. Split the above data into 2 RDDs, first RDD namely custfilter should be loaded only with

5 columns data and second RDD namely statesfilter should be only loaded with 2

columns data. \*/

val custfilter = custstates.map(\_.split(",") ).filter(x=> x.length==5);

println("CSV cust data: " + custfilter.take(2));

val statesfilter = custstates.map(\_.split(",") ).filter(x=> x.length==2);

println("CSV state data: " +statesfilter.take(2));

/\*

Use DSL functions:

32. Load the file3 (custs\_states.csv) from the HDFS location, using CSV Module in a DF

custstatesdf, this file contains 2 type of data one with 5 columns contains customer

master info and other data with statecode and description of 2 columns.

\*/

val custstatesdf = spark.read.csv(custstatespath);

/\*

33. Split the above data into 2 DFs, first DF namely custfilterdf should be loaded only with 5

columns data and second DF namely statesfilterdf should be only loaded with 2 columns

data.

Hint: Use filter DSL function to check isnull or isnotnull to achieve the above

functionality then rename and drop columns in the above 2 DFs accordingly.

\*/

val custfilterdf = custstatesdf.filter("\_c4 is not null").toDF("custid","name","address","age","profession")

println("DSL cust data: " +custfilterdf.show(5));

val statesfilterdf = custstatesdf.filter("\_c2 is null").withColumnRenamed("\_c0", "statecode").withColumnRenamed("\_c1", "description").drop("\_c2","\_c3","\_c4");

println("DSL state data: " +statesfilterdf.show(5));

/\*

Use SQL Queries: 34. Register the above DFs as temporary views as custview and statesview.

\*/

custfilterdf.createOrReplaceTempView("custview");

statesfilterdf.createOrReplaceTempView("statesview");

// 35. Register the DF generated in step 22 as a tempview namely insureview

modifiedinsuredf.createOrReplaceTempView("insureview");

//36. Import the package, instantiate the class and Register the method created in step 24 in the name of remspecialcharudf using spark udf registration.

spark.udf.register("remspecialcharudf",reusableObj.remspecialchar \_);

//37. Write an SQL query with the below processing

// a. Pass NetworkName to remspecialcharudf and get the new column called cleannetworkname

// b. Add current date, current timestamp fields as curdt and curts.

// c. Extract the year and month from the businessdate field and get it as 2 new fields called yr,mth respectively.

/\* d. Extract from the protocol either http/https from the NetworkURL column, if no

protocol found then display noprotocol. For Eg: if http://www2.dentemax.com/

then show http if www.bridgespanhealth.com then show as noprotocol store in

a column called protocol.\*/

/\* e. Display all the columns from insureview including the columns derived from

above a, b, c, d steps with statedesc column from statesview with age,profession

column from custview . Do an Inner Join of insureview with statesview using

stcd=stated and join insureview with custview using custnum=custid.

\*/

spark.sql("select \* from insureview limit 1").show(1);

val cleannetworkname = spark.sql("""select \*,remspecialcharudf(NetworkName) as cleannetworkname,

current\_date() as curdt, current\_timestamp as curts,

year(BusinessDate) as yr, month(BusinessDate) as mth,

case when NetworkURL like 'http:%' then 'http' else case when NetworkURL like 'https:%' then 'https' else 'noprotocol' end end as protocol

from insureview""");

cleannetworkname.createOrReplaceTempView("insureview2");

println("cleannetworkname: "+ cleannetworkname.show(5));

spark.sql("select \* from custview limit 5").show(5);

println("custview");

spark.sql("select \* from statesview limit 5").show(5);

println("statesview");

val insurejoined = spark.sql("""Select i.\*,s.description as statedesc, c.age,c.profession from insureview2 i inner join

statesview s on i.stcd=s.statecode join custview c on i.custnum=c.custid""");

insurejoined.show(5);

println("Insure joined data from dataframe");

insurejoined.createOrReplaceTempView("insurejoineddata");

// 38. Store the above selected Dataframe in Parquet formats in a HDFS location as a single file.

insurejoined.coalesce(1).write.mode("overwrite").parquet("hdfs://localhost:54310/user/hduser/hackathon/insurejoinedparquet");

spark.read.parquet("hdfs://localhost:54310/user/hduser/hackathon/insurejoinedparquet").show(5);

println("Insure joined data read from hdfs");

/\*

39. Very very important interview question : Write an SQL query to identify average age,

count group by statedesc, protocol, profession including a seqno column added which

should have running sequence number partitioned based on protocol and ordered

based on count descending.

For eg.

Seqno,Avgage,count,statedesc,protocol,profession

1,48.4,10000, Alabama,http,Pilot

2,72.3,300, Colorado,http,Economist

1,48.4,3000, Atlanta,https,Health worker

2,72.3,2000, New Jersey,https,Economist

\*/

val partitionquery = spark.sql("""select row\_number() over(partition by protocol order by count(1) desc) as Seqno,avg(age) as avgage,

count(1) as count,statedesc,protocol,profession from insurejoineddata group by age,statedesc,protocol,profession""");

println("interview question result , using rank partition group");

partitionquery.coalesce(1).write.mode("overwrite").csv("hdfs://localhost:54310/user/hduser/hackathon/finalinsureviewcsv");

//spark.sql("select \* from finalinsureview").show(false);

println("total rows in seqno partition and group by: "+partitionquery.count());

println("total http rows: "+partitionquery.filter("protocol='http'").count());

println("total https rows: "+partitionquery.filter("protocol='https'").count());

println("total noprotocol rows: "+partitionquery.filter("protocol='noprotocol'").count());

insuredatamerged.unpersist();

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part B 4\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//5. Visualization (5% Completion) – Total 80%

// captured snapshot and check at end of the document

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part B 5\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Part C - Spark Streaming with DF & SQL

// 6. Realtime Streaming (20% Completion) – Total 100%

// check  **2.Spark streaming script** for this part

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part C 6\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

}

**def rowsRemoved**(initialRdd:RDD[String],finalRdd:RDD[insureclass]):Unit=

{

val initialRows=initialRdd.count;

val finalRows=finalRdd.count;

println("initial rows: " + initialRows + ", final rows: "+finalRows);

println("Rows removed: "+ (initialRows-finalRows));

}

}

**case class insureclass**

(IssuerId:String,IssuerId2:String,BusinessDate:String ,StateCode:String,SourceName:String,NetworkName:String,NetworkURL:String,custnum:String,MarketCoverage:String,DentalOnlyPlan:String)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*End of part B\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**Spark Streaming Hackathon Project**

**2.Spark streaming script**

package org.inceptez.hackstreaming

import org.apache.spark.\_

import org.apache.spark.streaming.\_

import org.apache.spark.sql.SparkSession

import org.apache.spark.SparkContext

import org.apache.spark.rdd.\_

import org.apache.spark.sql.hive.\_

object filestream {

def main(args:Array[String])

{

// Create the context

val spark = SparkSession.builder().appName("sparkStreamingHackthon").master("local[\*]")

.config("spark.history.fs.logDirectory", "file:///tmp/spark-events")

.config("spark.eventLog.dir", "file:////tmp/spark-events")

.config("spark.eventLog.enabled", "true")

.config("hive.metastore.uris","thrift://localhost:9083")

.config("spark.sql.warehouse.dir","hdfs://localhost:54310/user/hive/warehouse")

.enableHiveSupport().getOrCreate();

val sc = spark .sparkContext;

sc.setLogLevel("ERROR")

// 1. Load the chatdata into a Dataframe using ~ delimiter by pasting in kafka or streaming file source or socket with the seconds of 50.

val ssc = new StreamingContext(sc, Seconds(50))

val chatdstream = ssc.textFileStream("file:///home/hduser/sparkdata/streaming/")

import spark.implicits.\_;

chatdstream.foreachRDD { rdd =>

if (!rdd.isEmpty())

{

var chatdf = rdd.map(\_.split("~")).map ( x => (x(0).toInt,x(1).trim(),x(2).trim())).toDF();

// 2. Define the column names as id,chat,'type'

chatdf= chatdf.withColumnRenamed("\_1", "id").withColumnRenamed("\_2", "chat").withColumnRenamed("\_3", "type");

// 3. Filter only the records contains 'type' as 'c

var filterChat = chatdf.filter("type='c'");

filterChat.show(1);

// 4. Remove the column 'type' from the above dataframe, hence the resultant dataframe contains only id and chat and convert to tempview.

filterChat.drop("type").createOrReplaceTempView("tempview");

/\* 5. Use SQL split function on the 'chat' column with the delimiter as ' ' (space) to tokenize all words for

eg. if a chat looks like this (i have internet issue from yesterday) and convert the chat column as array

type and name it as chat\_tokens which has to looks like [i,have,internet,issue,from,yesterday]

id,chat\_tokens

1,[i,have,internet,issue,from,yesterday] \*/

val temp1 = spark.sql("select id, split(chat,' ') as chat\_tokens from tempview");

temp1.createOrReplaceTempView("chattokenview");

println("chattokenview: " + temp1.show(3));

/\* 6. Use SQL explode function to pivot the above data created in step 5, then the exploded/pivoted data

looks like below, register this as a tempview.

id,chat\_splits

1,i

1,have \*/

val temp2 = spark.sql("select id, explode(chat\_tokens) as chat\_splits from chattokenview")

temp2.createOrReplaceTempView("chattempview")

println("chattempview: " + temp2.show(3));

/\* 8. Load the stopwords from linux file location /home/hduser/stopwordsdir/stopwords (A single column file found in the web that contains values such as (a,above,accross,after,an,as,at) etc. which is used in any English sentences for punctuations or as fillers) into dataframe with the column name stopword and convert to tempview. \*/

val temp3 = spark.read.text("file:///home/hduser/install/IZ\_WORKOUTS/SPARK\_HACKATHON\_2020/realtimedataset/stopwords").toDF("stopword");

temp3.createOrReplaceTempView("stopwordtempview");

println("stopwordtempview: " + temp3.show(3));

/\* 9. Write a left outer join between chat tempview created in step 6 and stopwords tempview created in step 8 and filter all nulls (or) use subquery with not in option to filter all stop words from the actual chat

tempview using the stopwords tempview created above. \*/

val joneddata = spark.sql("select c.id,c.chat\_splits from chattempview c left outer join stopwordtempview s on c.chat\_splits=s.stopword where s.stopword is null");

println("joneddata: " + joneddata.show(3));

// 10. Load the final result into a hive table should have only result as given below using append option.

spark.sql("create database if not exists hackathon");

joneddata.write.mode("append").saveAsTable("hackathon.chatstopword");

/\* 11. Identify the most recurring keywords used by the customer in all the chats by grouping based on the

keywords used with count of keywords. use group by and count functions in the sql

for eg.

issues,2

notworking,2

tv,2 \*/

val chatKeywords = spark.sql("select chat\_splits, count(1) as count from hackathon.chatstopword group by chat\_splits");

println("chatKeywords: " + chatKeywords.show(3));

// 12. Store the above result in a json format with column names as chatkeywords, occurance

chatKeywords.toDF("chatkeywords","occurance").coalesce(1).write.mode("overwrite").json("hdfs://localhost:54310/user/hduser/hackathon/chatKeywordsoccurancejson");

}

} // end of for each

ssc.start()

ssc.awaitTermination()

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of Step 6 - Streaming\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

**3.Allmethods (Resubale methods)**

package org.inceptez.hack

import org.apache.spark.SparkContext

import org.apache.spark.\_

import org.apache.spark.rdd.\_

import org.apache.hadoop.fs.\_

class allmethods {

// Remove all special characters and numbers 0 to 9 - ? , / \_ ( ) [ ]

**def remspecialchar(**value:String):String=

{

var modifiedValue="";

modifiedValue = value.replaceAll("[[0-9~`\\[\\]\\\\!@#$%^&\*()\_+={};':\",./<>?|-]]","").replaceAll(" ", "");

return modifiedValue

}

/\*2. Remove the header line from the RDD contains column names.

Hint: Use rdd1.first -> rdd1.filter to remove the first line identified to remove the header. \*/

**def removeHeader**(rdd:RDD[String]):RDD[String]=

{

val header=rdd.first.toString;

val rdd1 = rdd.filter(x=>x!=header);

//3. Display the count and show few rows and check whether header is removed.

println("Total count without header: "+rdd1.count);

println("sample data: " + rdd1.take(3).foreach(println));

return rdd1;

}

// Delete Existing files

**def deleteFileIfExist**(fs:FileSystem, filePaths:Array[String],isDeleteRecursively:Boolean)

{

filePaths.foreach { x =>

var path = new Path(x);

if(fs.delete(path,isDeleteRecursively))

{

println("file exists, hence deleting. "+ x);

}

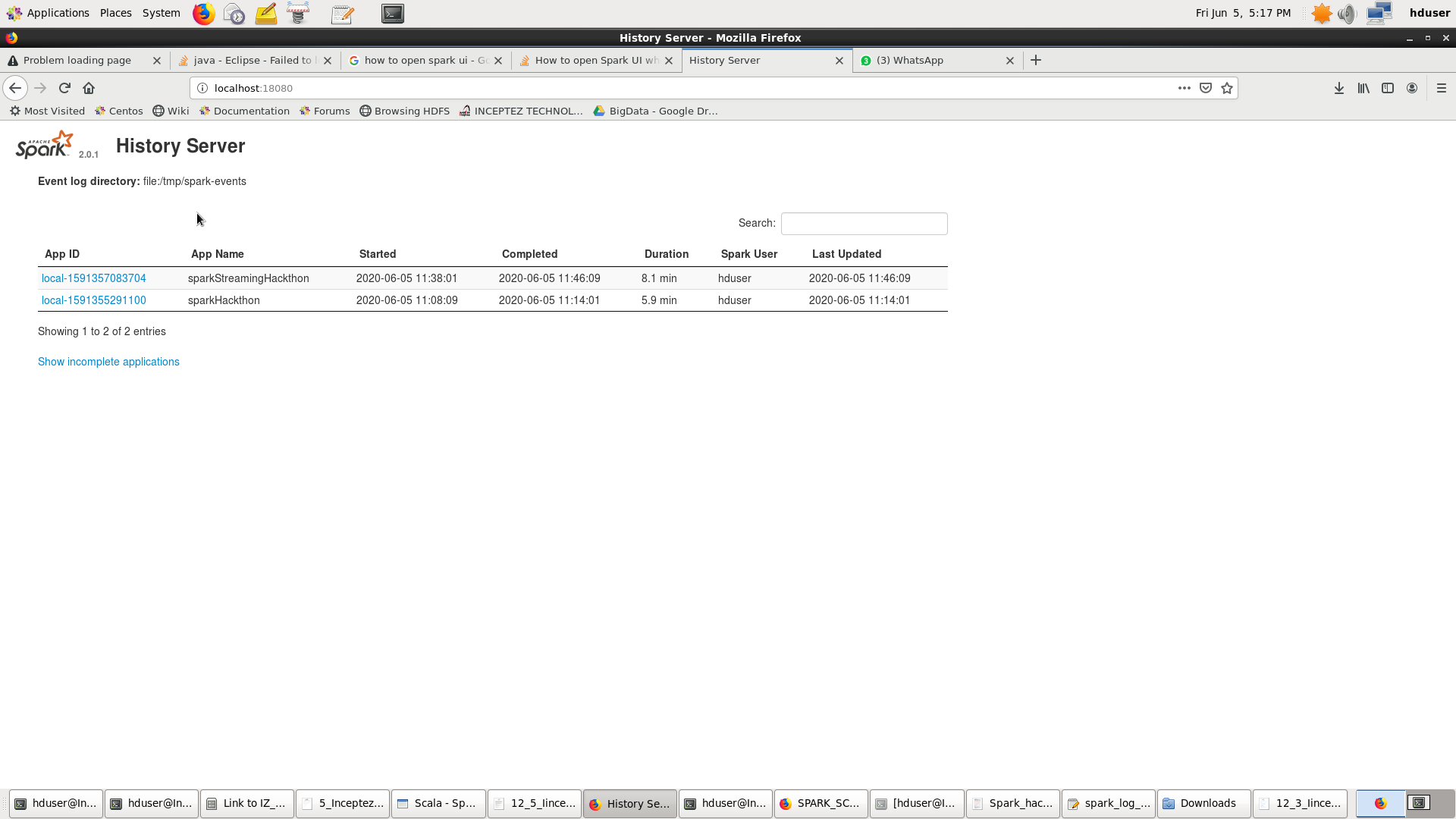
}

}

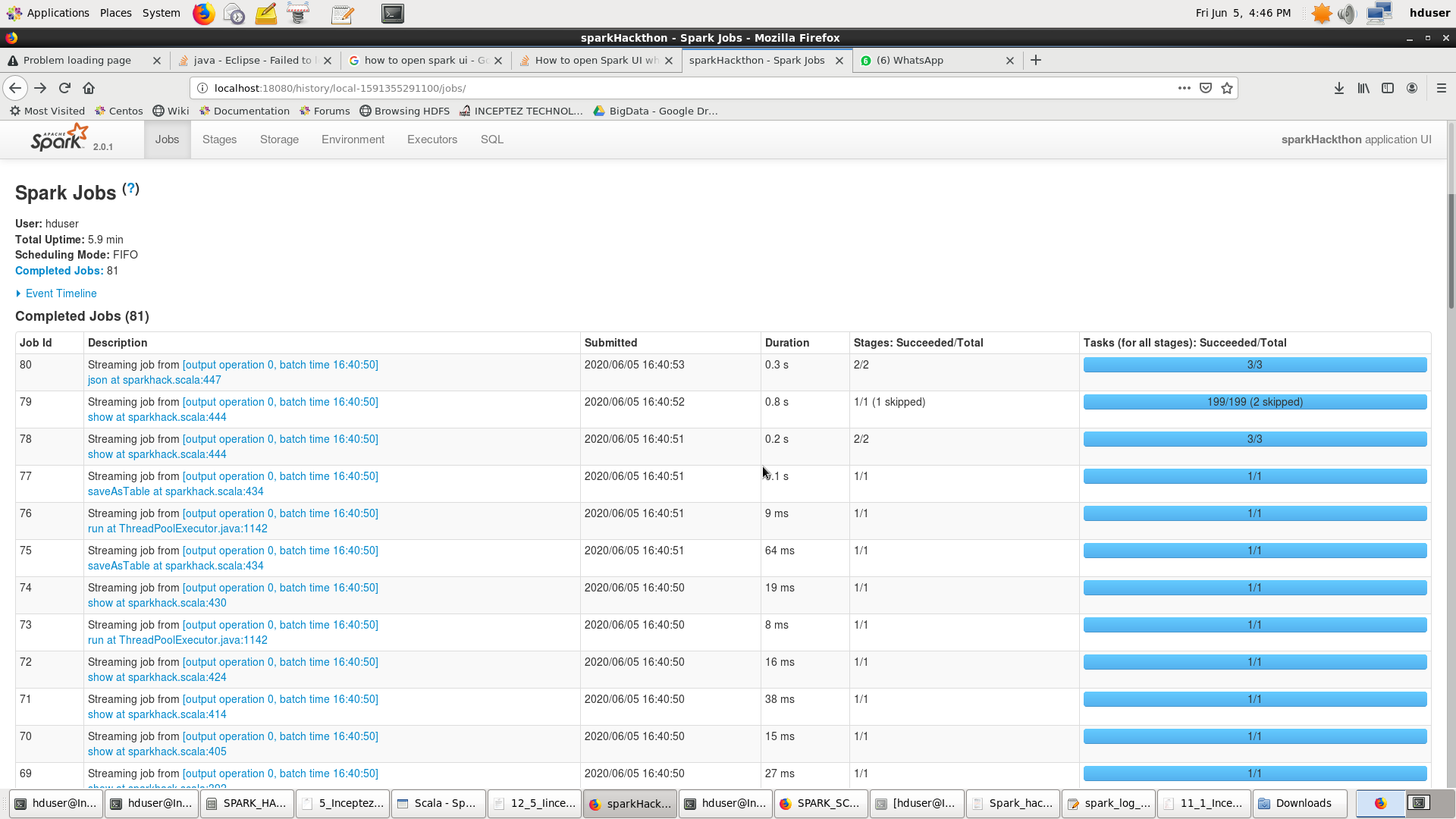
}

**4.Spark UI Visualization snapshtos [Part B - Step 5]**

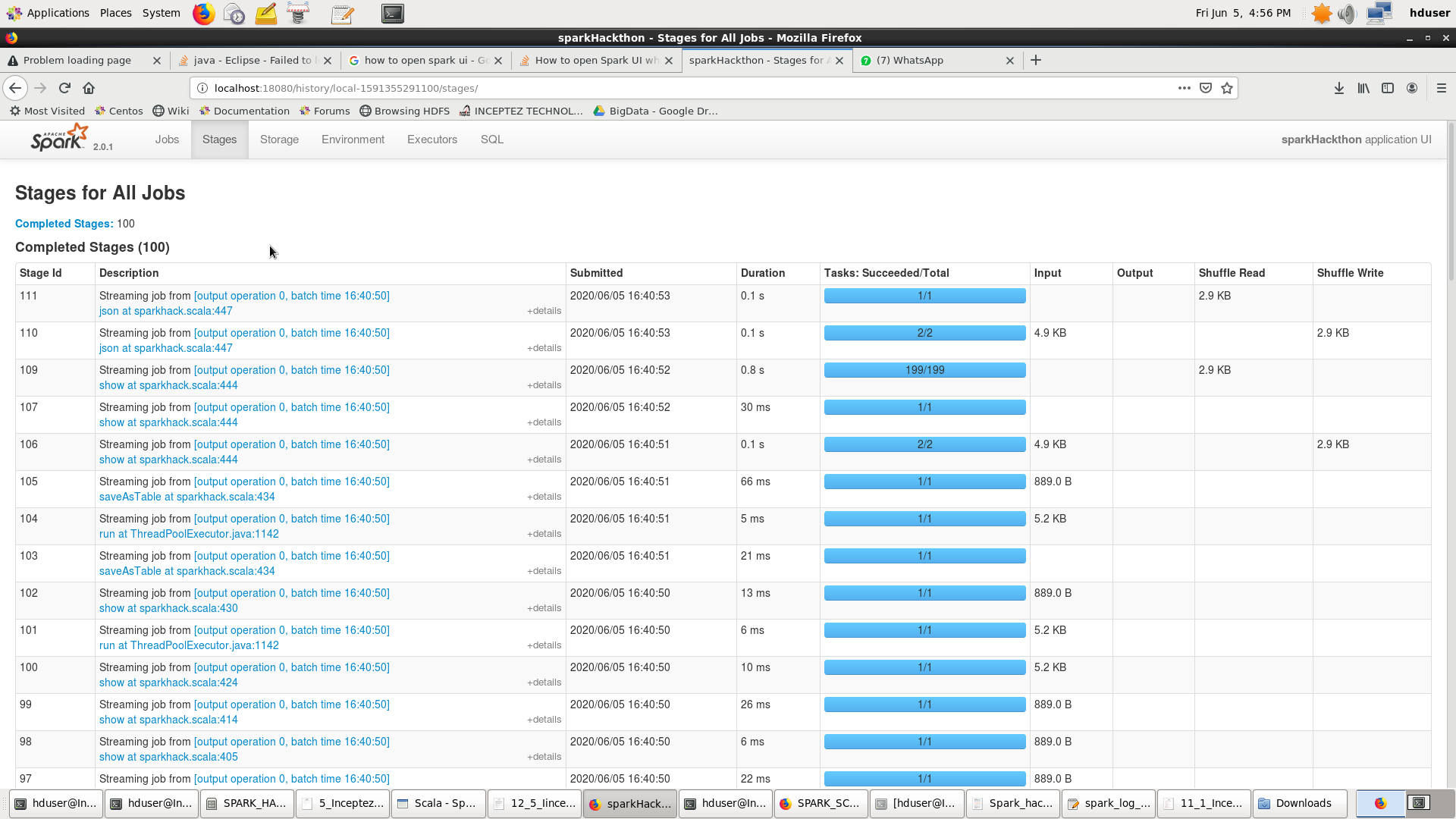
**Applications list:**

****

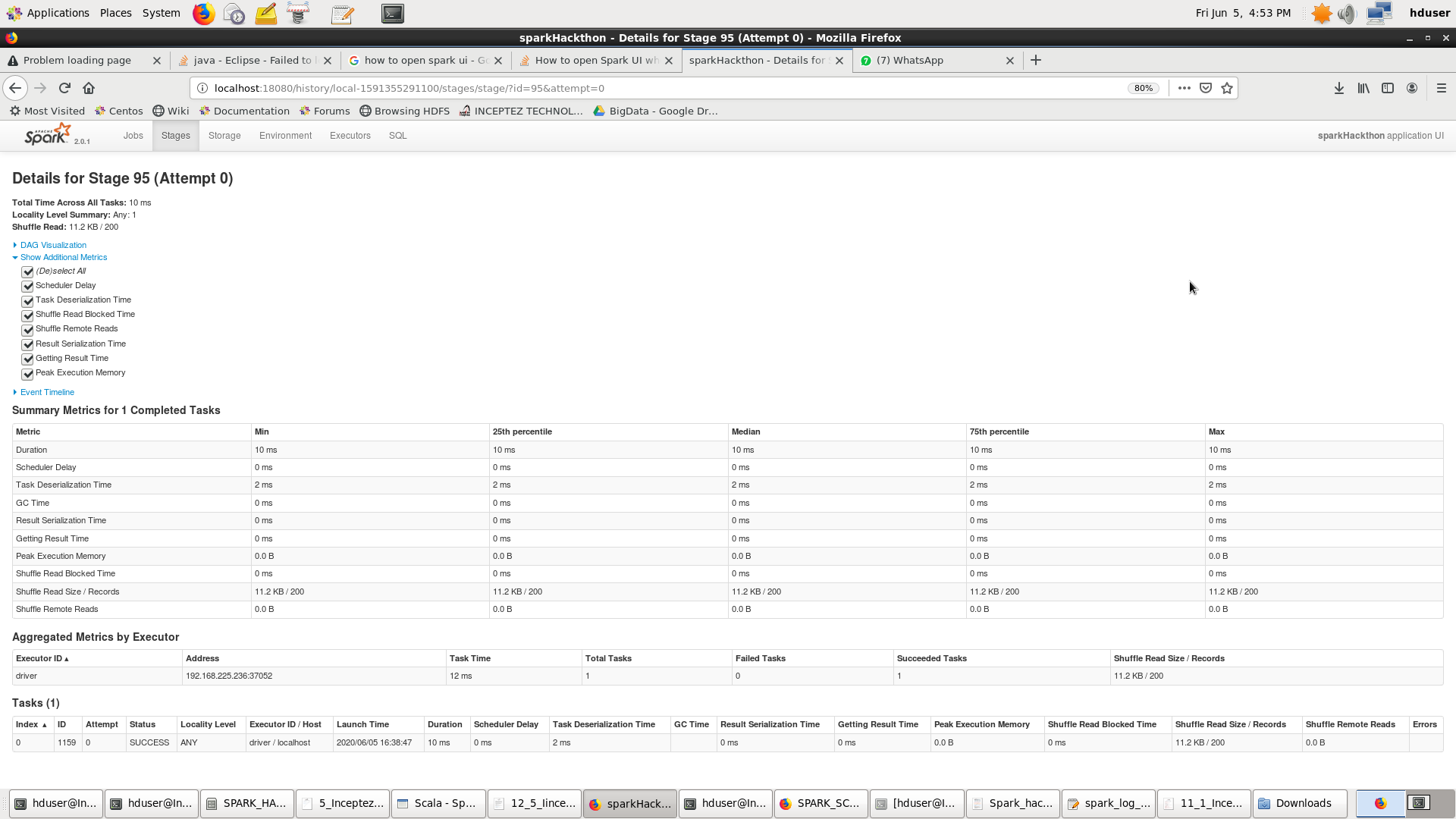
**Jobs:**

****

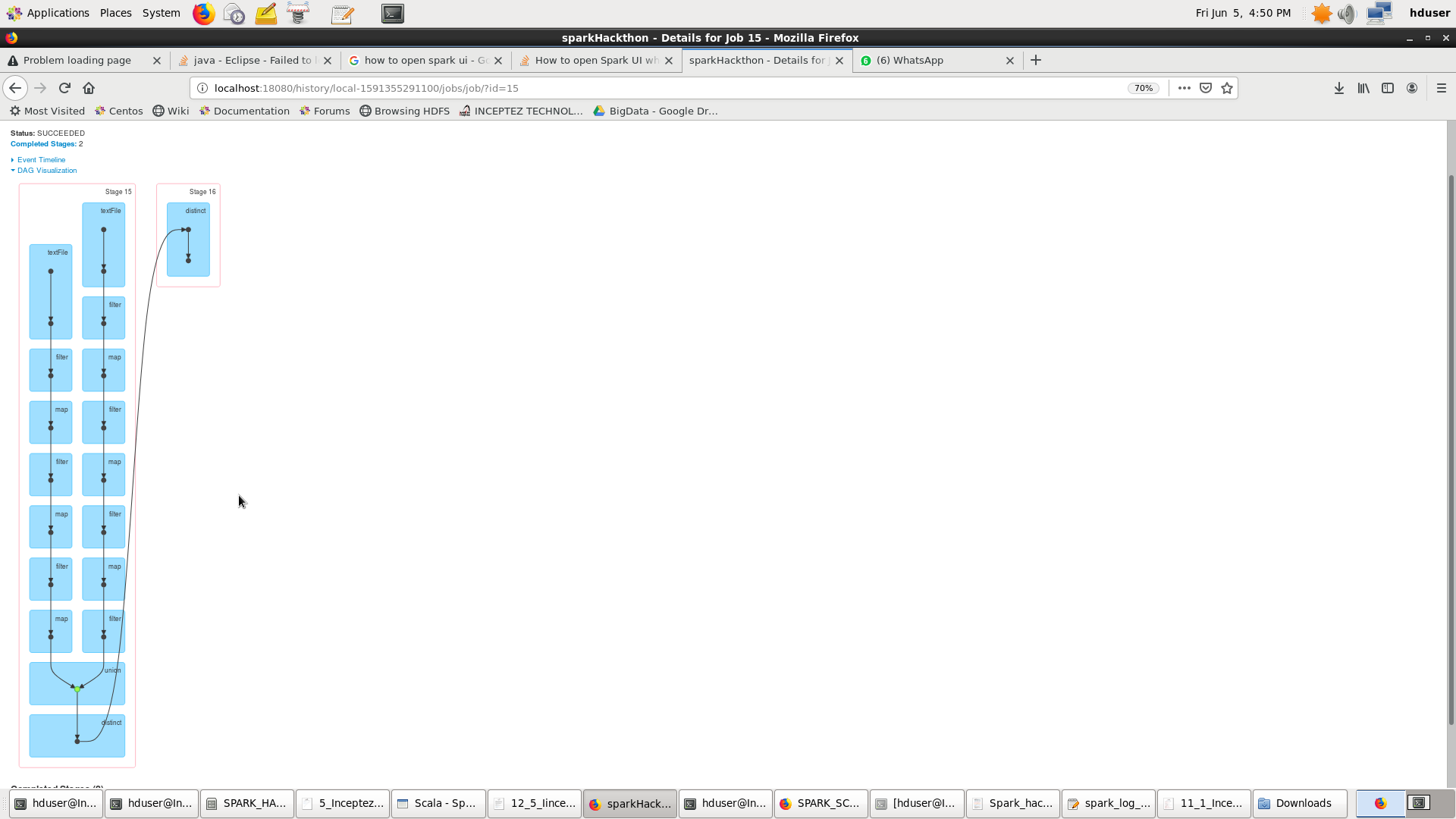
**Stages:**

****

**Stage Details:**

****

**DAG:**

****