ETL process using Kafka, flume and spark streaming

Hadoop

FINAL PROJECT

Soumya Kataria: 100897536

Pre-requisites: zookeeper, kafka and confluent installation

Flume is getting the data and is storing it in hdfs which is then used by spark for sorting and streaming

**FLUME**

Spark is used to stream the data and scala is also used for sorting the data. Kafka is connected to spark.

We are copying data from local file system to HDFS which is acting as our trigger and sparks takes that data for sorting which is done in scala and then streams it

**BEHIND THE SCENES FOR KAKFKA AND SPARK**

* We created two topics in kafka called “IDLE” and “ACTIVE”
* We also created two consumers in kafka called “IDLE” and “ACTIVE”
* Scala is running queries to sort the data as active or idle according to the values given in gt column.
* We also have two value streams one for idle and one for active
* Topic IDLE is used to stream data for idle value stream and topic ACTIVE is used to stream data for active value stream

HDFS

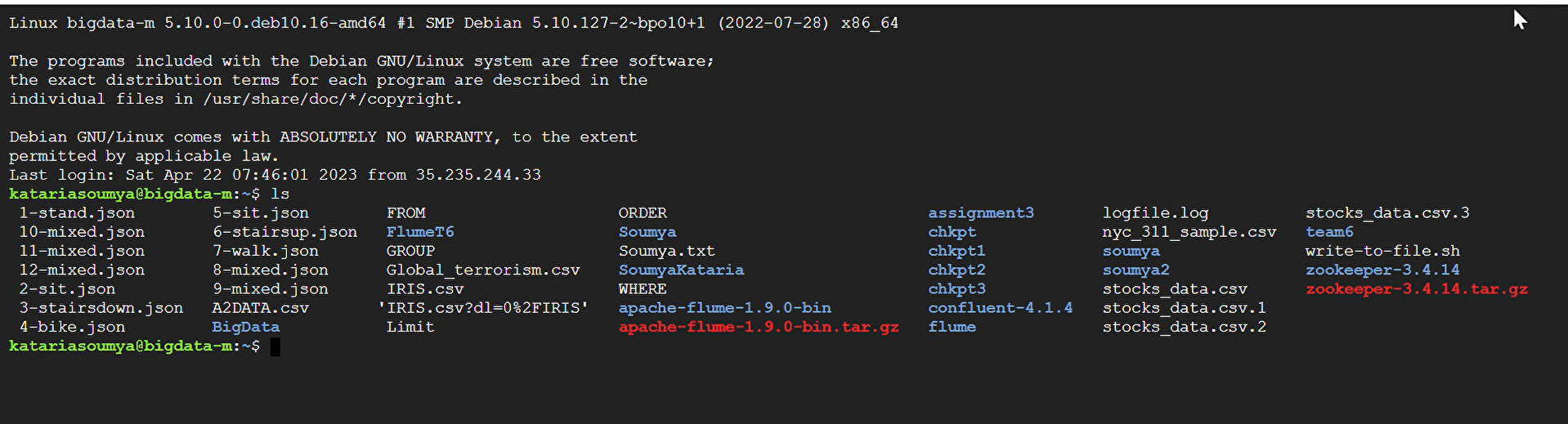
**KAFKA**

**SPARK**

**HDFS**

# **Streaming from FLUME**

In the ETL process we have one flume agent taking data from folder name team6 and sending it one by one in HDFS. This has been done by creating a loop in the shell script called write-to-file.sh. This script has to be run to send the data along with another flume command.

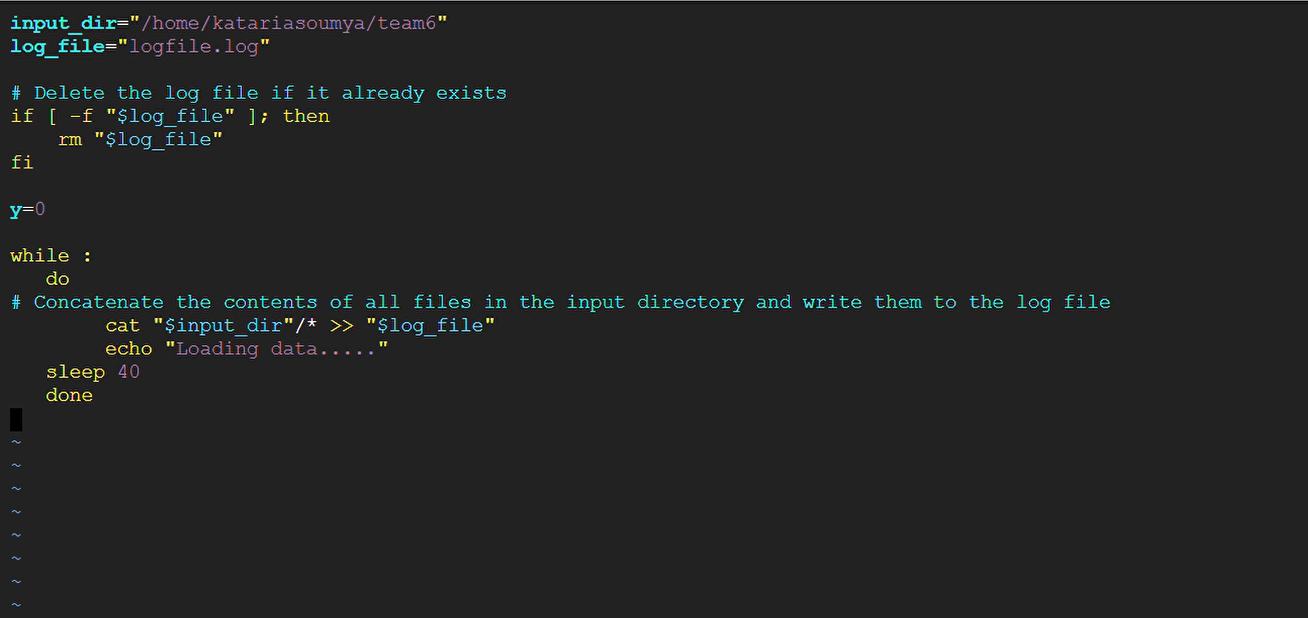


# The folder where all the IOT dataset files are saved

The data has been taken from this folder and then send to spark for sorting as idle or active.

# **Shell script and configuration file**

Shell script that we have created is given below

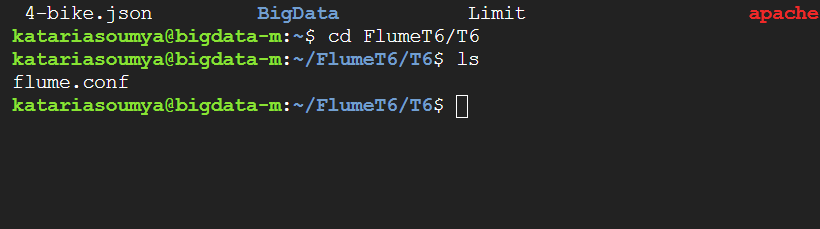


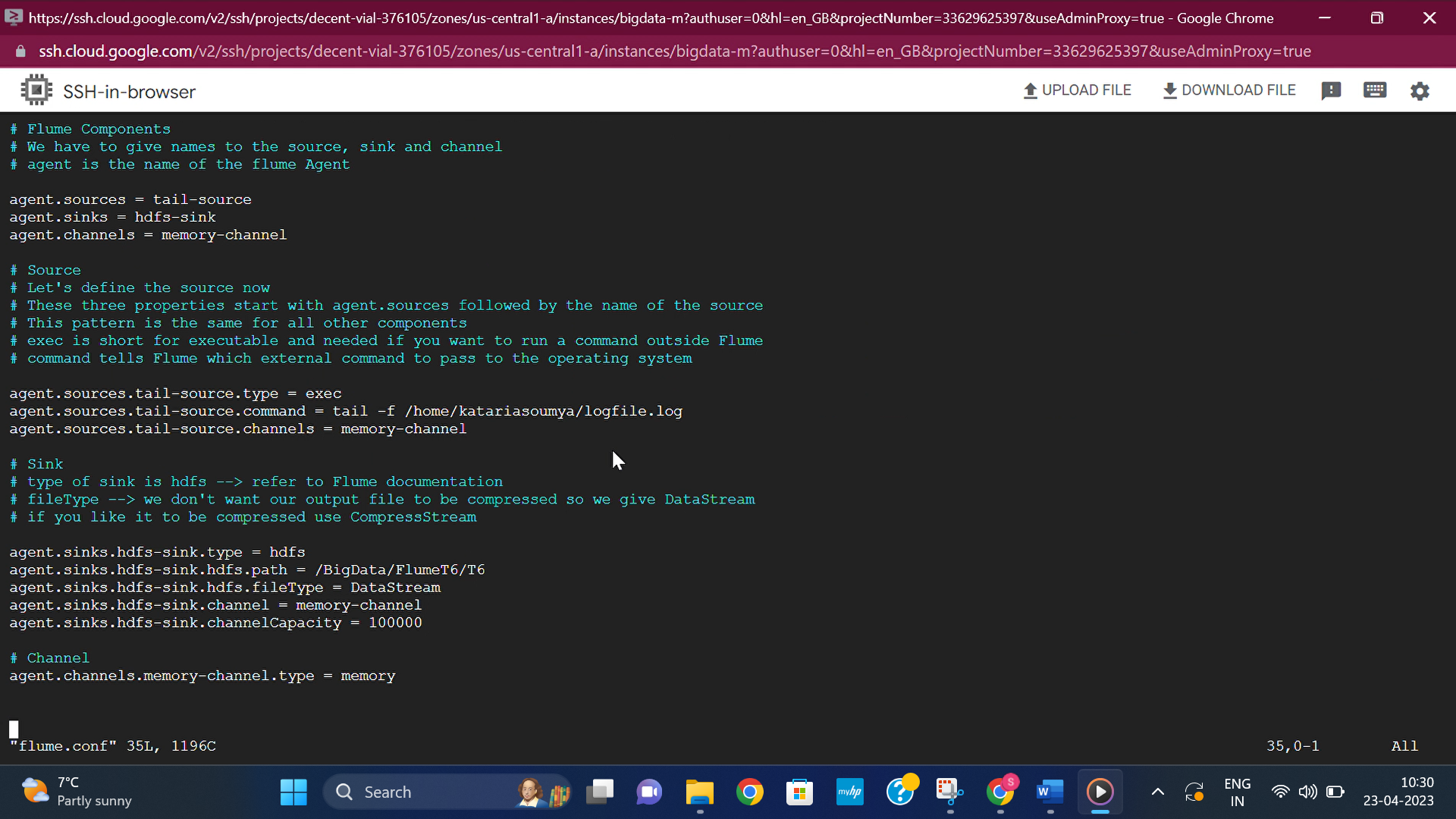
This shell script performs the following tasks:

* It sets the variable input\_dir to the path /home/katariasoumya/team6.
* It sets the variable log\_file to the filename logfile.log.
* It checks if the log file already exists using an if statement and removes it using the rm command if it does.
* It initializes the variable y to zero.
* It enters an infinite loop using the while : statement.
* Within the loop, it concatenates the contents of all files in the input\_dir directory using the cat command and appends them to the log\_file using the >> operator.
* It prints the message "Loading data....." to the console using the echo command.
* It sleeps for 40 seconds using the sleep command.

The script is designed to continuously monitor the input\_dir directory for changes and write the contents of any new or modified files to the log\_file file. The script will continue running indefinitely until it is manually stopped.

The following is the configuration file avaible in FlumeT6/T6 folder.





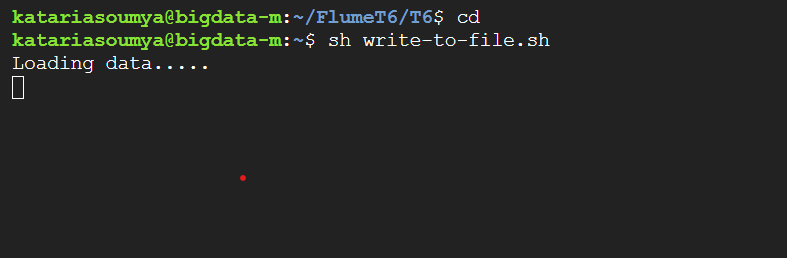
The configuration specifies three components: sources, sinks, and channels.

* Sources: The source specified in this configuration is a tail-source, which is of exec type. The command attribute specifies the command to be executed, which is tail -f /home/katariasoumya/logfile.log. This means that the source will continuously read new data from the logfile.log file in the specified directory. The channels attribute specifies the channel to which the source will write the data. In this case, it is the memory-channel.
* Sinks: The sink specified in this configuration is an hdfs-sink, which is of hdfs type. The hdfs.path attribute specifies the HDFS (Hadoop Distributed File System) directory where the data will be written. In this case, it is /BigData/FlumeT6/T6. The hdfs.fileType attribute specifies the file format used for writing the data to HDFS, which is DataStream. The channel attribute specifies the channel from which the sink will read the data. In this case, it is the memory-channel. The channelCapacity attribute specifies the maximum number of events that the channel can hold in memory before it starts blocking.
* Channels: The channel specified in this configuration is a memory-channel, which is of memory type. This channel stores the data in memory before it is consumed by the sink. The type attribute specifies the channel type, which is memory.

Overall, this Flume configuration reads data from the logfile.log file using the tail command, stores it in a memory channel, and then writes it to HDFS in the DataStream format. This is a common pattern used in real-time data processing pipelines where data is continuously generated and needs to be processed and analysed in real-time.

Then we ran the shell script and using

**sh Write-to-file.sh**

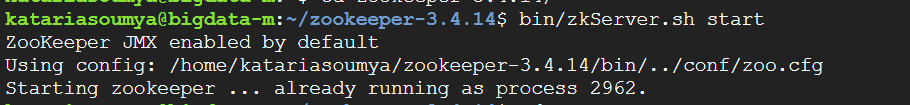
****

And then ran the following command

flume-ng agent --conf /home/katariasoumya/FlumeT6/T6/ -f /home/katariasoumya/FlumeT6/T6/flume.conf -Dflume.root.logger=DEBUG,console -n agent

# **Zookeeper and Confluent**

ran **bin/zkServer.sh start** command in zookeeper folder and **nohup bin/kafka-server-start etc/kafka/server.properties > /dev/null 2>&1 &** command in confluent folder

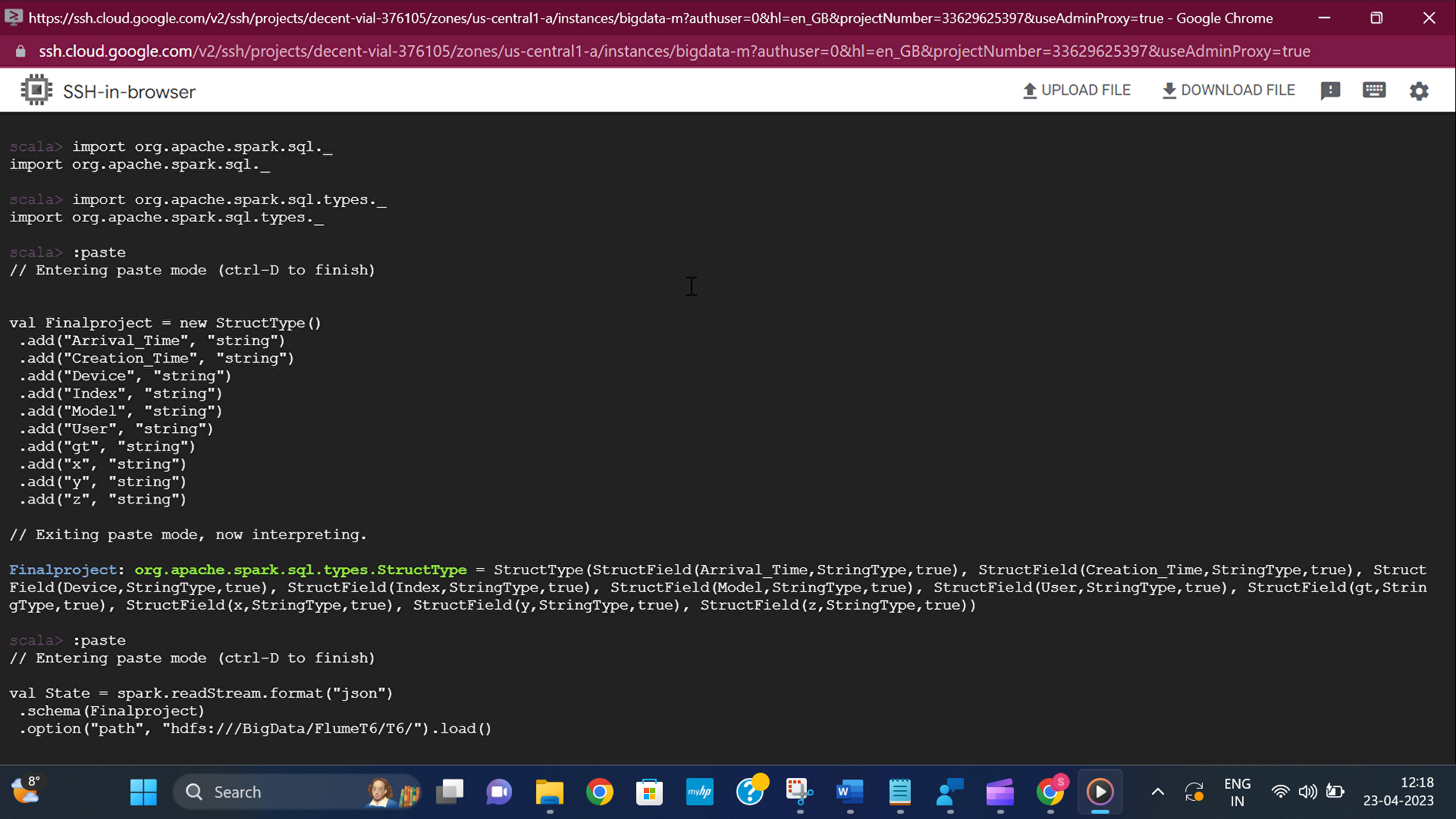




# **SCALA**

This code defines a StructType schema called Finalproject with ten fields of type String. It then reads JSON files from a Spark Structured Streaming source, using the spark.readStream method, and specifies the schema using the schema method. The JSON files are read from the Hadoop Distributed File System (HDFS) directory located at "hdfs:///BigData/" using the option method.

The load method creates a DataFrame that represents a streaming dataset. This DataFrame is continuously updated with new data as new JSON files are written to the specified HDFS directory. Note that this code only sets up the streaming query, but it does not start the query. The streaming query must be started separately using the start method.



**The path is the location where flume will save the files while streaming**

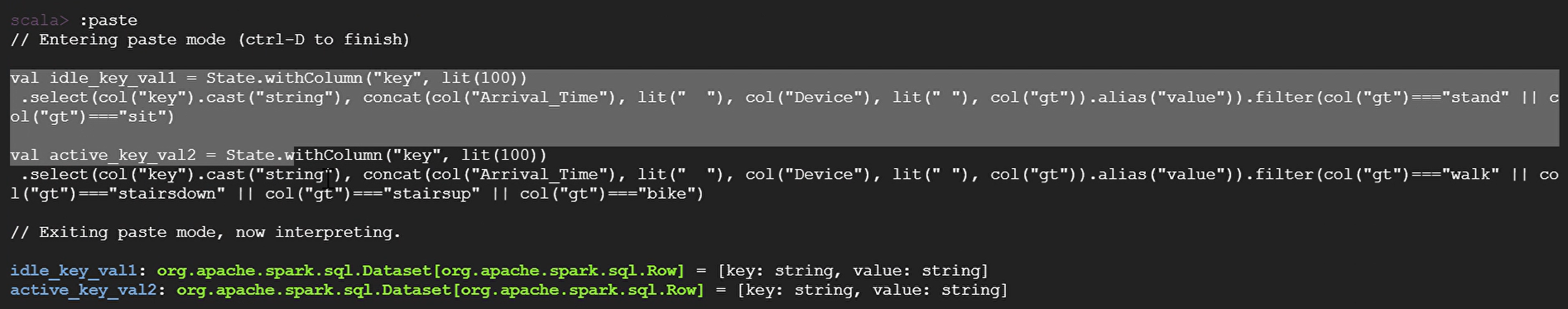
## Queries for sorting:

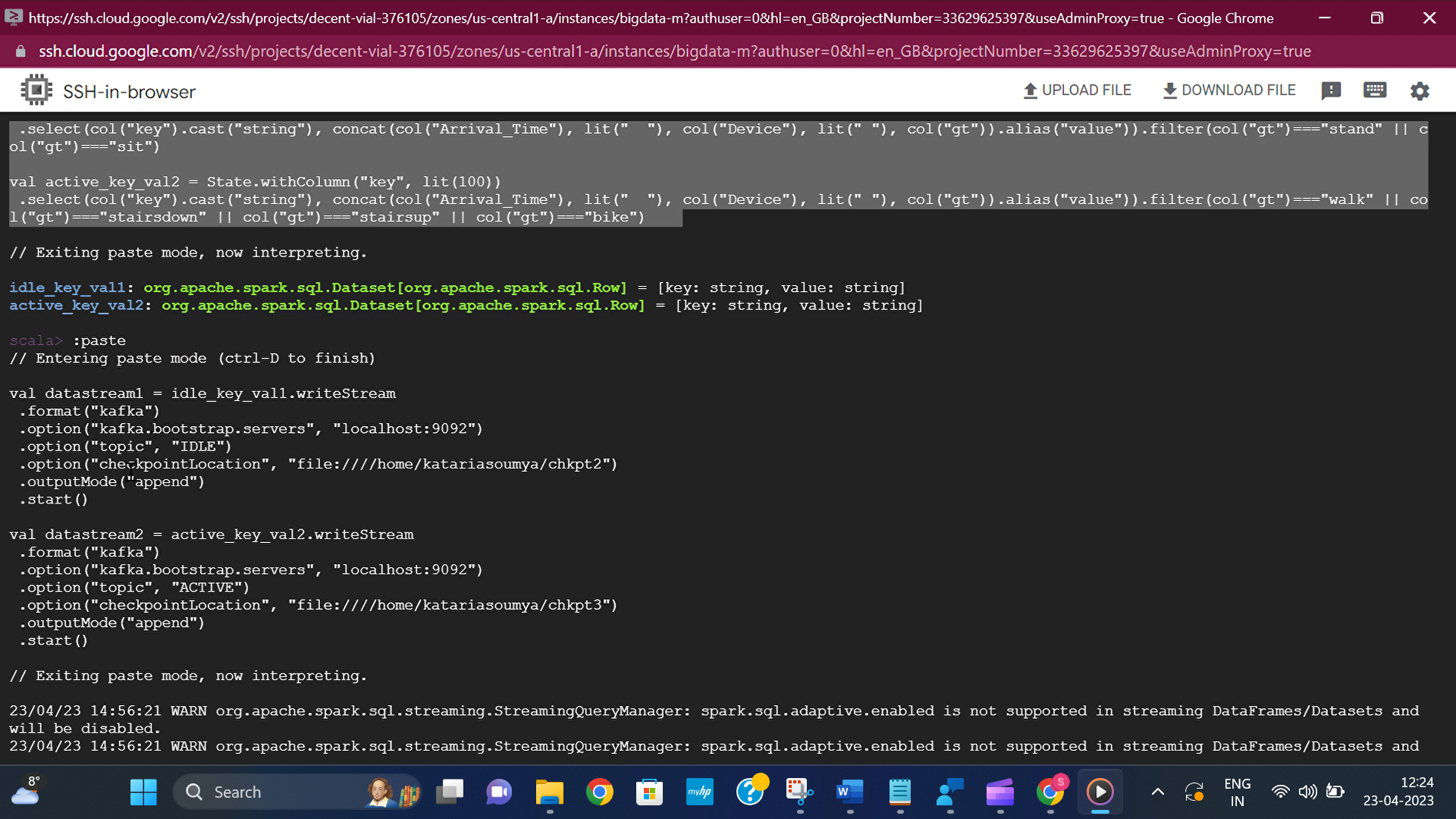
**val idle\_key\_val1 = State.withColumn("key", lit(100))**

**.select(col("key").cast("string"), concat(col("Arrival\_Time"), lit(" "), col("Device"), lit(" "), col("gt")).alias("value")).filter(col("gt")==="stand" || col("gt")==="sit")**

**val active\_key\_val2 = State.withColumn("key", lit(100))**

**.select(col("key").cast("string"), concat(col("Arrival\_Time"), lit(" "), col("Device"), lit(" "), col("gt")).alias("value")).filter(col("gt")==="walk" || col("gt")==="stairsdown" || col("gt")==="stairsup" || col("gt")==="bike")[[1]](#footnote-1)**





These commands are setting up two streaming queries that write data to Kafka topics.

* idle\_key\_val1 is the DataFrame representing the data with "stand" or "sit" activity, which is being written to the "IDLE" Kafka topic.
* active\_key\_val2 is the DataFrame representing the data with "walk", "stairsdown", "stairsup", or "bike" activity, which is being written to the "ACTIVE" Kafka topic.
* Both queries use the writeStream method to create a streaming query, and the format method to specify that the output should be in the Kafka format.
* The option method is used to specify the Kafka server and topic to write to, as well as the checkpoint location where the query metadata will be stored.
* Finally, the outputMode method is set to "append", which means that only new rows added to the output DataFrame will be written to Kafka.
* The queries are started using the start method, which begins the streaming query and writes the output to the specified Kafka topics.

# **Kafka Consumer and Topics (same for HDFS as well)**

We only need to create two consumers and two topics for kafka and each consumer will receive the data from HDFS and flume streaming to HDFS.

Topics that we have are IDLE and ACTIVE

Commands used

**bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic IDLE**

**bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic ACTIVE**

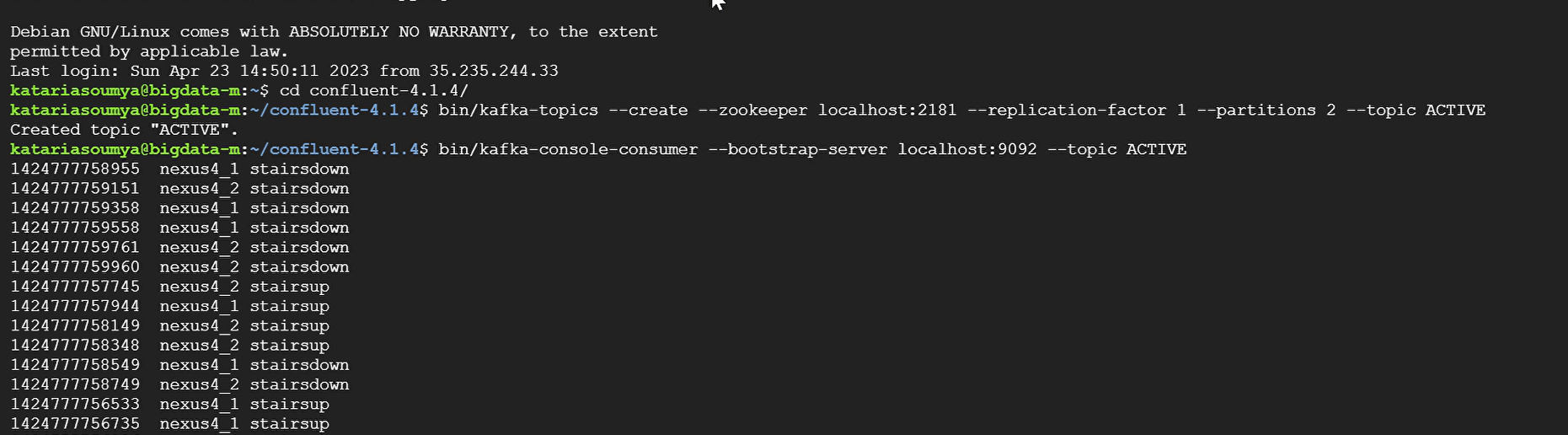
Consumers in kafks- ACTIVE and IDLE

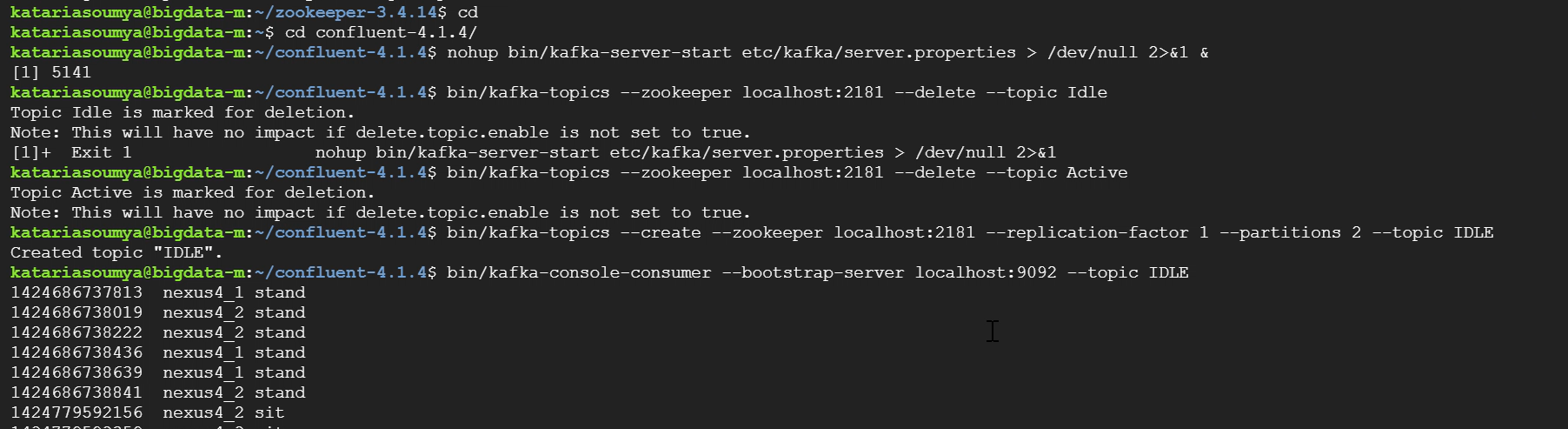
Commands used

**bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic IDLE**

**bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic ACTIVE**

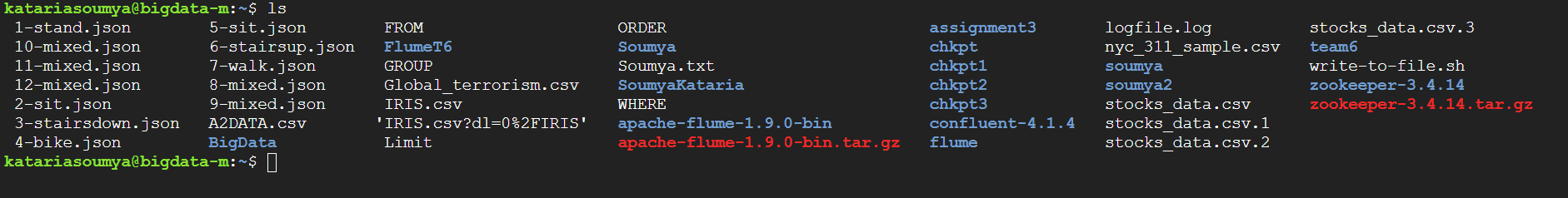
**\*\*note-**both the consumer and the topics are created in confluent directory.





# **Streaming from HDFS**

We have all our files saved in local file system as well.

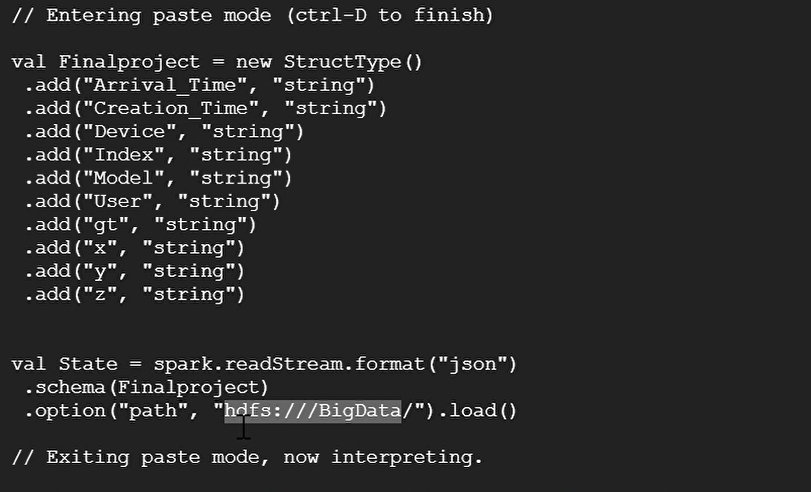


Files stored in linux

We’ll send these files in BigData folder which will be treated as our trigger and when we’ll copy the files from local system to big data folder i.e in HDFS the values will then get distributes as per the kafka topic and then sent to the relevant consumer which will display our values.

# SCALA

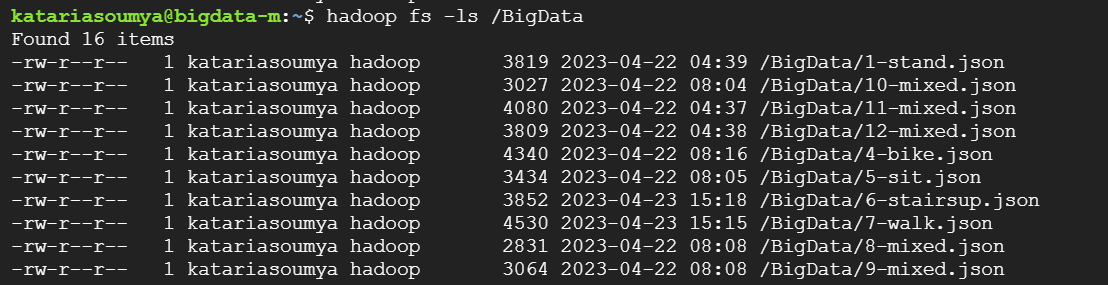
The scala queries and coding is exact same for the HDFS streaming as well except the following



The path is the location where we’ll copy the files from the local system

Command for copying: **hadoop fs -copyFromLocal <filename> /BigData/.**

**Files copies by us in BigData folder**

****

1. We tried using other query given below instead of the second query but it was giving error in scala

   val active\_key\_val2 = State.withColumn("key", lit(100))

   .select(col("key").cast("string"), concat(col("Arrival\_Time"), lit(" "), col("Device"), lit(" "), col("gt")).alias("value"))

   .filter(!(col("gt")==="stand" || col("gt")==="sit")) [↑](#footnote-ref-1)