**SETUP OF THE Alert SYSTEM**

To setup the system, IoT data from a Nexus phone is downloaded (12 json files) in the form of JSON files. These files contain different set of information; however, we are focusing on the Arrival\_Time, gt and device.

Steps:

1. Created log file to send data with the help of flume agent to HDFS folder.
2. Copied the data from local to HDFS folder to create a second source for the system “BigDataProject/localcopied”.
3. Configure the flume agent write data into a HDFS folder named “/BigDataProject/flume/basic”.
4. Created Kafka consumers with separate topics “Action” and “Idle”.
5. Implemented Spark streaming to read the data from the above two sources and send it to different Kafka sinks. If the “gt” data consists stand or sit, the message will send to the Kafka topic “Idle” while remaining goes to “Action”.

**First Source**

FLUMES: The flume is a distributed, reliable, and available service for efficiently collecting, and moving large amounts of log data.

Here, we have collected all the JSON files and created the logfile using the script below.

createlog.sh

#

log\_file="logfile.log"

json\_files=$(find . -type f -name '.json' -o -name '.JSON')

# Check if the log file exists and remove it if it does

if [ -f $log\_file ] ; then

rm $log\_file

fi

# Loop through the JSON files and append the data to the log file and implementing this to unlimited times

while :

do

for file in $json\_files; do

echo "Reading data from file: $file"

while read -r line; do

echo "$line" >> $log\_file

done < "$file"

done

sleep 3

done

The above script helps to generate a logfile which is basically copying all the json files into the **logfile.log** infinite times. To run the script command used: **sh createlog.sh**

Afterwards, in the configuration file for the Flume agent all the configurations are set to send data from the logfile to HDFS folder.

Configuration file in which the location of logfile is given to the agent source command and the location of the HDFS folder “BigDataProject/flume/basic”

Basic-flume.conf

# Flume Components

# We have to give names to the source, sink and channel

# agent is the name of the flume Agent

agent.sources = tail-source

agent.sinks = hdfs-sink

agent.channels = memory-channel

# Source

# Let's define the source now

# These three properties start with agent.sources followed by the name of the source

# This pattern is the same for all other components

# exec is short for executable and needed if you want to run a command outside Flume

# command tells Flume which external command to pass to the operating system

agent.sources.tail-source.type = exec

agent.sources.tail-source.command = tail -f /home/iharvinder/logfile.log

agent.sources.tail-source.channels = memory-channel

# Sink

# type of sink is hdfs --> refer to Flume documentation

# fileType --> we don't want our output file to be compressed so we give DataStream

# if you like it to be compressed use CompressStream

agent.sinks.hdfs-sink.type = hdfs

agent.sinks.hdfs-sink.hdfs.path = /BigDataProject/flume/basic

agent.sinks.hdfs-sink.hdfs.fileType = DataStream

agent.sinks.hdfs-sink.channel = memory-channel

# Channel

agent.channels.memory-channel.type = memory

After configuring the agent, the flume ng command is run to call the agent and perform the task to transfer the logfile to HDFS folder.

Flume agent command

flume-ng agent --conf /home/iharvinder15/flume/ -f /home/iharvinder15/flume/basic-flume.conf -Dflume.root.logger=DEBUG,console -n agent

**Second Source**

To create the second source all of the 12 Json files is complied into one json file named **iotdata.json.** Then this file is directly copied from the local to HDFS folder using the below command.

hadoop fs -copyFromLocal iotdata.json /BigDataProject/localcopied

Start zookeeper and kafka broker server for further implementations of the system.

To start zookeeper, go to zookeeper directory and run the command.

bin/zkServer.sh start

To start kafka broker (which helps to setup communication between producer and consumers), go to confluent directory and run the command.

nohup bin/kafka-server-start etc/kafka/server.properties > /dev/null 2>&1 &

the above command will start the broker in silently on the port and will not show any logs for it. After this create consumers.

**Kafka consumers**

To create kafka consumers with different topics “action” and “idle”.

Inside the confluent directory.

To create kafka topic “idle”

bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic idle

Start listening on the idle.

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

To create kafka topic “action”

bin/kafka-topics --create --zookeeper localhost:2181 --replication-factor 1 --partitions 3 --topic action

Start listening on the action.

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

**Spark Streaming**

Then, create a folder to create a checkpoint for Spark Streaming, which is used to store data which is sent to kafka sinks using the mkdir command. (mkdir chkpt)

Start spark shell to implement the spark code to complete the setup.

spark-shell --master local --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.0

The above command will open the spark shell where the scala queries will be implemented.

Import necessary libraries

import org.apache.spark.sql.\_

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

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

Create schema to read the json format

// data is in json format so we need to provide the schema

val userSchema = new StructType()

.add("Arrival\_Time", "string")

.add("Device", "string")

.add("gt", "string")

Read data from the first source.

// Reading data from the first source

val iot = spark.readStream.format("json")

.schema(userSchema)

.option("path", "hdfs:///BigDataProject/flume/basic/\*").load()

Read data from the second source.

// Reading data from the second source

val localcopied = spark.readStream.format("json")

.schema(userSchema)

.option("path", "hdfs:///BigDataProject/localcopied/").load()

Here, the concatenation takes place to put one more column so that the data can be identified from which source is it coming.

// Concatenate the source name to the data

val iot\_with\_source = iot.withColumn("source", lit("iot"))

val localcopied\_with\_source = localcopied.withColumn("source", lit("localcopied"))

Then, creating a dataframe by joining both of the data frames.

// Union the two sources

val union\_df = iot\_with\_source.union(localcopied\_with\_source)

Filtering data to send the data according to two different topics “action” and “idle”

// Filter the data and send to corresponding Kafka sink

val idle\_df = union\_df.filter(col("gt") === "stand" || col("gt") === "sit")

val action\_df = union\_df.filter(!(col("gt") === "stand" || col("gt") === "sit"))

val idle\_key\_val = idle\_df.withColumn("key", lit(100))

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

val action\_key\_val = action\_df.withColumn("key", lit(100))

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

#Writing data as stream to the specified kafka topic named “idle”

val idle\_stream = idle\_key\_val.writeStream

.format("kafka")

.option("kafka.bootstrap.servers", "localhost:9092")

.option("topic", "idle")

.option("checkpointLocation", "file:///home/iharvinder15/chkpt/idle")

.outputMode("append")

.start()

#Writing data as stream to the specified kafka topic named “action”

val action\_stream = action\_key\_val.writeStream

.format("kafka")

.option("kafka.bootstrap.servers", "localhost:9092")

.option("topic", "action")

.option("checkpointLocation", "file:///home/iharvinder15/chkpt/action")

.outputMode("append")

.start()

// Start both the streams

idle\_stream.awaitTermination()

action\_stream.awaitTermination()

The output of the above script is shown in the Appendix.

**APPENDIX**

Text

Description automatically generated**Creation of log file:**

**Running Flume Agent:**

Text

Description automatically generatedText

Description automatically generated**Flume Source – Action Topic:**

Text

Description automatically generated**Local Source:**

**Flume Source – Idle Topic:**Text

Description automatically generated

Text

Description automatically generated**Spark:**

**Segregating the data based on the sink:**

Text

Description automatically generated