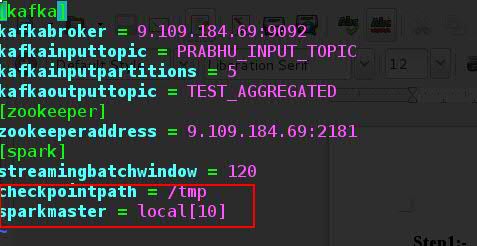
**Step1:-**

Complete all the steps in Kafka Steps.docx .

This basically verifies if the Kafka Producer was able to successfully send to the Kafka Input Topic which will be used by the Spark Streaming as Input.

**Step2:-**

**Edit the following file projectrootdir/config/Configfile.properties**



1)Edit the Sparkmaster to the Sparkmaster node. I

t has the default value of local[10] as this code was run on my personal Laptop.

2) Edit the Spark streaming checkpoint directory if needed. If the Code is run on a Spark Cluster with Master and Worker node this directory needs to be in HDFS.

Example

checkpointpath = hdfs://9.109.184.69:8020/tmp/

3) Edit the streamingbatchwindow if required currently its set to 120 seconds as per the requirement in the mail.

**Step3:-**

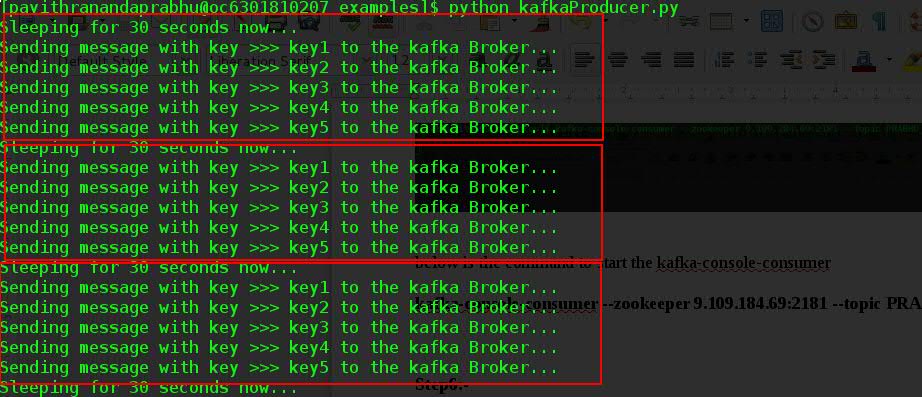
**Start the Kafka Producer program.**

1) cd projectrootdirectory/examples

2) execute the following command

**python kafkaProducer.py**

this is a program which runs forever and generates messages to kafka every 30 seconds.

****

We can see in the above output that kafkaProducer has generated three sets of messages every 30

**Step 4:**

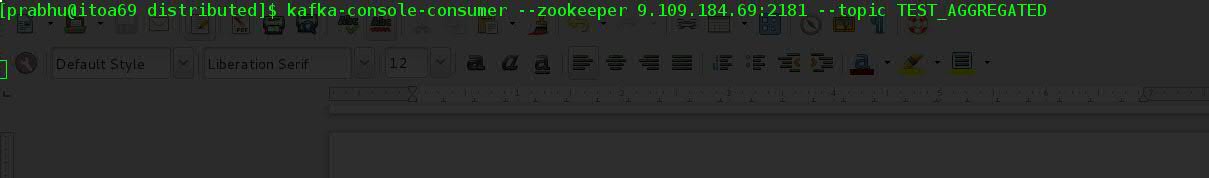
**Start the kafka-console-consumer program on the Output Topic.**

Output\_Topic currently being used is TEST\_AGGREGATED.

Issue the below command to Start the Kafka-console-consumer

**kafka-console-consumer --zookeeper 9.109.184.69:2181 --topic TEST\_AGGREGATED**

change the zookeeper address as per the environment



**Step 5:-**

**Start the Spark Streaming program.**

Spark Streaming uses the spark-streaming-kafka-0-8 Jar which is used to read the Streaming data from Kafka.

There are two ways to Start the Spark Streaming program.

**Method 1:- Using the Spark-submit command**

**1)** cd projectrootdirectory/examples

**2)** Execute the below command..

**spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8\_2.11:2.0.0 sparkStreaming.py**

**--packages => parameter downloads the necessary dependencies from the maven repository**

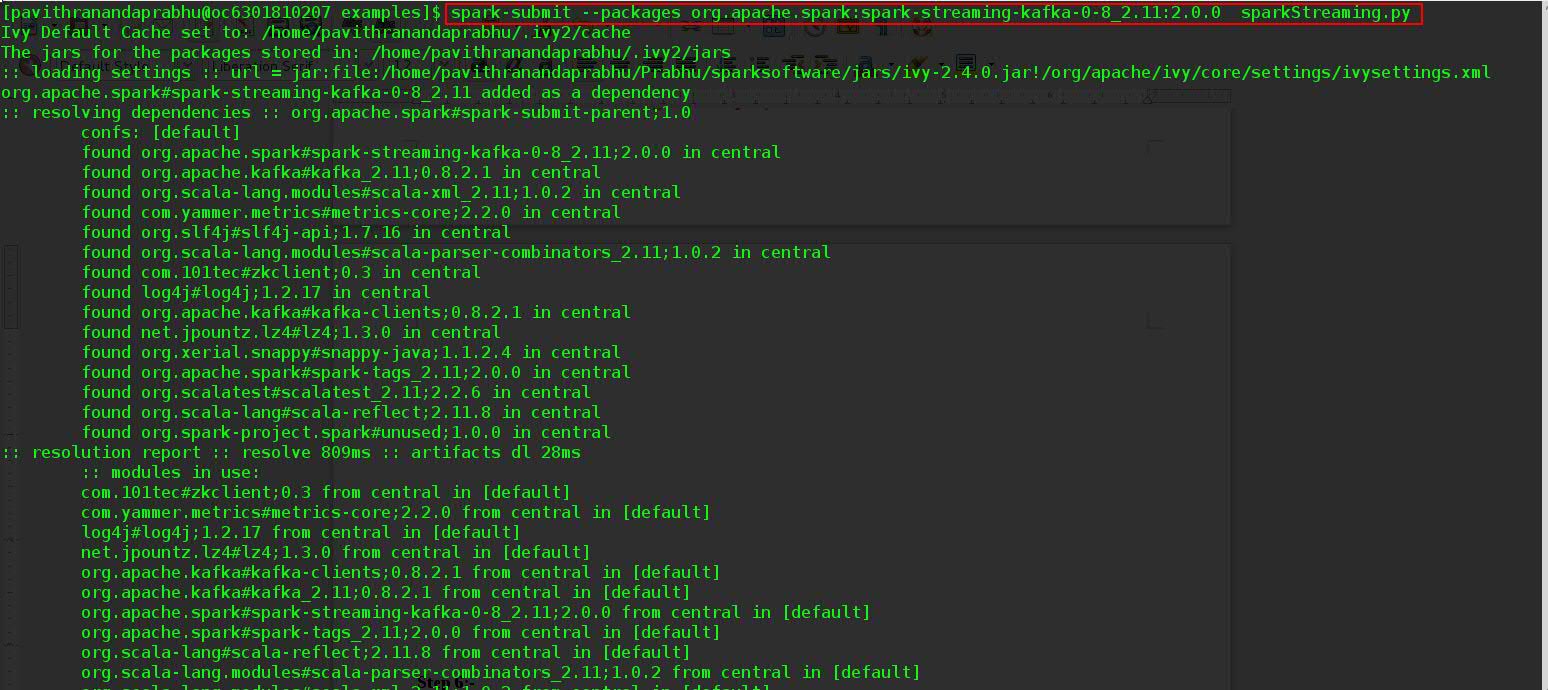
**org.apache.spark:spark-streaming-kafka-0-8\_2.11:2.0.0**

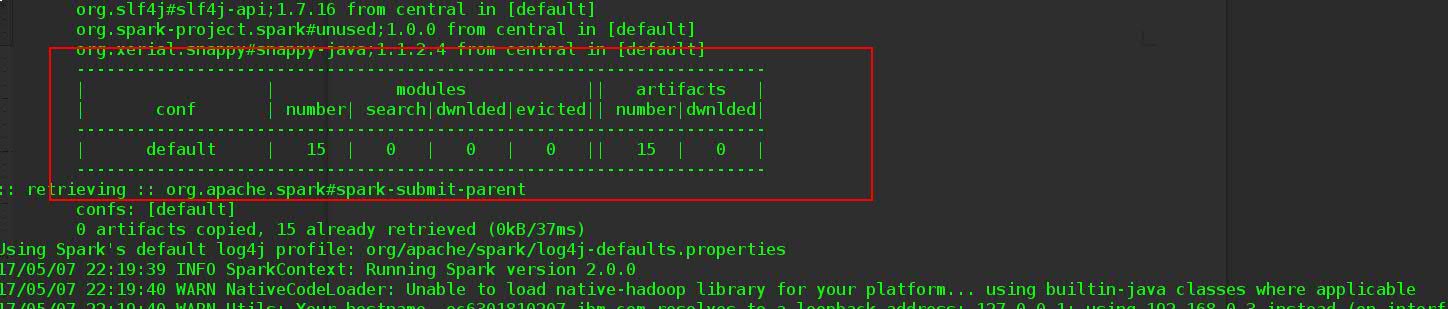
**here**

**2.11 denotes the Scala Version**

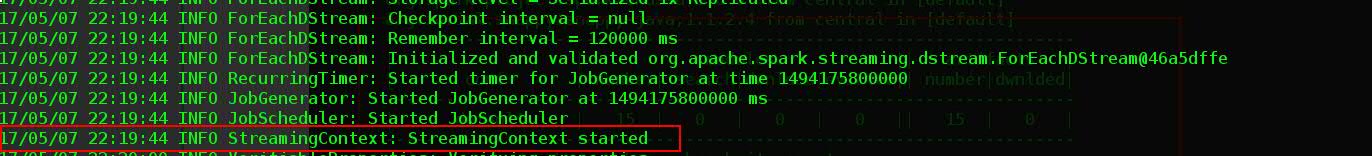
**2.0.0 denotes the Spark version.**

**Please make sure to edit this Argument as per your environment.**

****

****

We can see from the above pic that the spark-submit worked without errors and also the dependent jars are also downloaded from maven repository.



StreamingContext was created successfully.

**Method 2 :- using the Shell command**

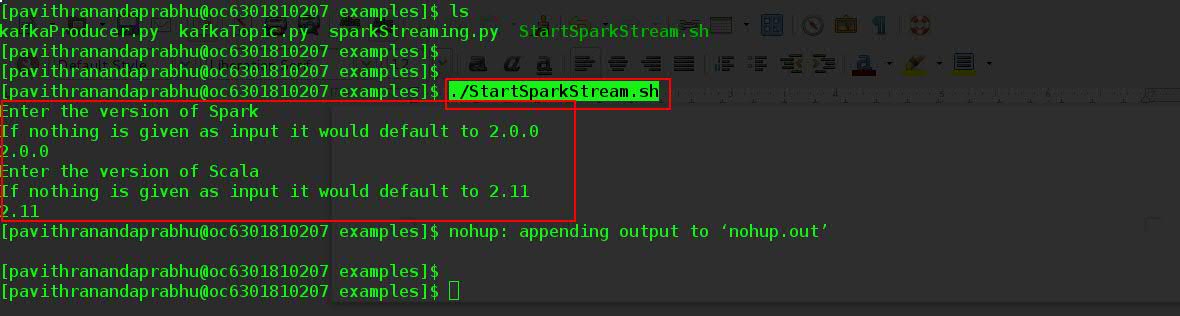
1)

**1)** cd projectrootdirectory/examples

**2)** Execute the below command..

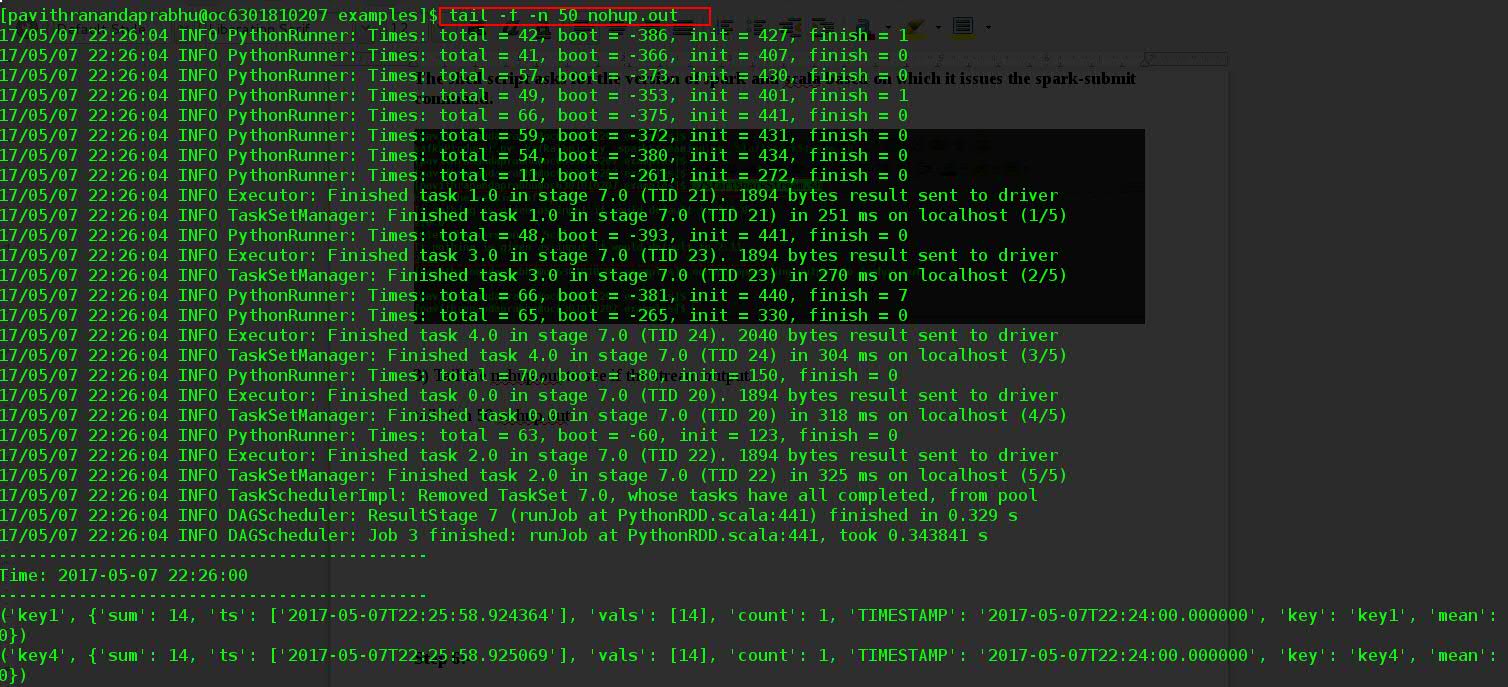
**./StartSparkStream.sh**

**The shell script asks for the version of spark and scala based on which it issues the spark-submit command.**



**3)** Tail the nohup.out to see if the stream output.

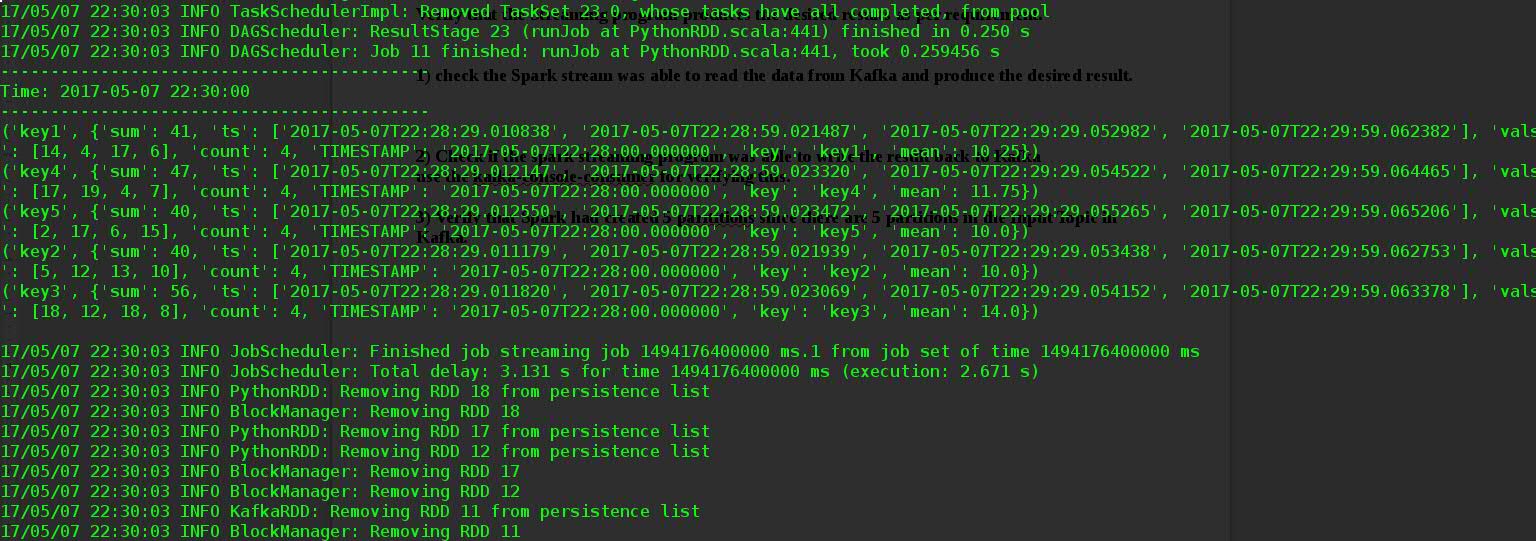
tail -f -n 50 nohup.out

****

**Step 6:-**

**Verify that the Streaming program produces the desired results as per requirement.**

**1) check the Spark stream was able to read the data from Kafka and produce the desired result.**

****

Above is the spark output for the window

Starting time :- 2017-05-07 22:28:00

ending Time: 2017-05-07 22:30:00

('key1', {'sum': 41, 'ts': ['2017-05-07T22:28:29.010838', '2017-05-07T22:28:59.021487', '2017-05-07T22:29:29.052982', '2017-05-07T22:29:59.062382'], 'vals': [14, 4, 17, 6], 'count': 4, 'TIMESTAMP': '2017-05-07T22:28:00.000000', 'key': 'key1', 'mean': 10.25})

('key4', {'sum': 47, 'ts': ['2017-05-07T22:28:29.012147', '2017-05-07T22:28:59.023320', '2017-05-07T22:29:29.054522', '2017-05-07T22:29:59.064465'], 'vals': [17, 19, 4, 7], 'count': 4, 'TIMESTAMP': '2017-05-07T22:28:00.000000', 'key': 'key4', 'mean': 11.75})

('key5', {'sum': 40, 'ts': ['2017-05-07T22:28:29.012550', '2017-05-07T22:28:59.023472', '2017-05-07T22:29:29.055265', '2017-05-07T22:29:59.065206'], 'vals': [2, 17, 6, 15], 'count': 4, 'TIMESTAMP': '2017-05-07T22:28:00.000000', 'key': 'key5', 'mean': 10.0})

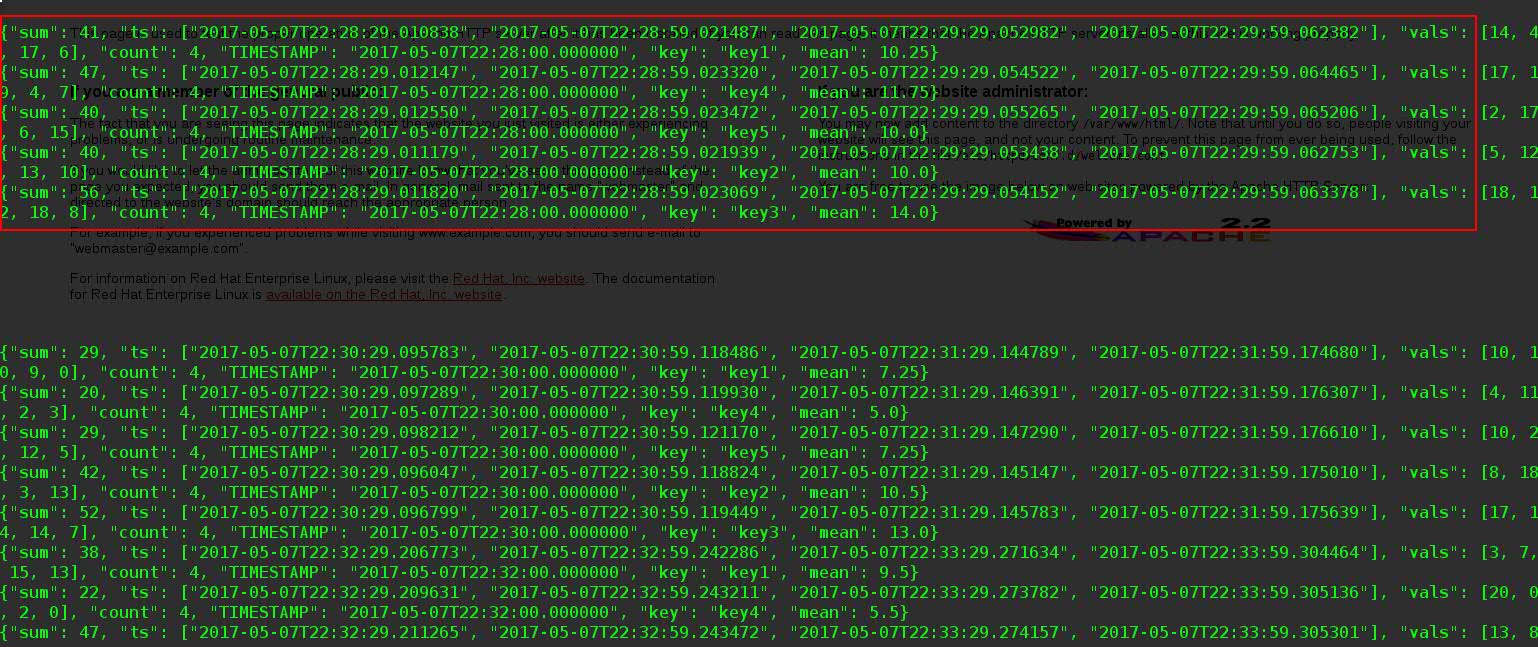
('key2', {'sum': 40, 'ts': ['2017-05-07T22:28:29.011179', '2017-05-07T22:28:59.021939', '2017-05-07T22:29:29.053438', '2017-05-07T22:29:59.062753'], 'vals': [5, 12, 13, 10], 'count': 4, 'TIMESTAMP': '2017-05-07T22:28:00.000000', 'key': 'key2', 'mean': 10.0})

('key3', {'sum': 56, 'ts': ['2017-05-07T22:28:29.011820', '2017-05-07T22:28:59.023069', '2017-05-07T22:29:29.054152', '2017-05-07T22:29:59.063378'], 'vals': [18, 12, 18, 8], 'count': 4, 'TIMESTAMP': '2017-05-07T22:28:00.000000', 'key': 'key3', 'mean': 14.0})

**we can see from the above output that it had produced the output as per the requirement..**

**2) Check if the spark streaming program was able to write the result back to Kafka**

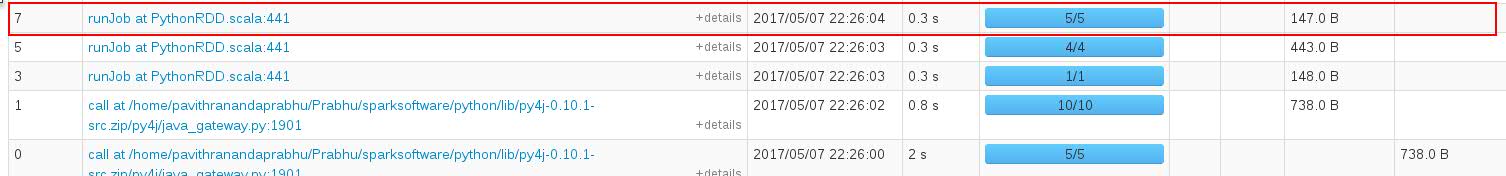
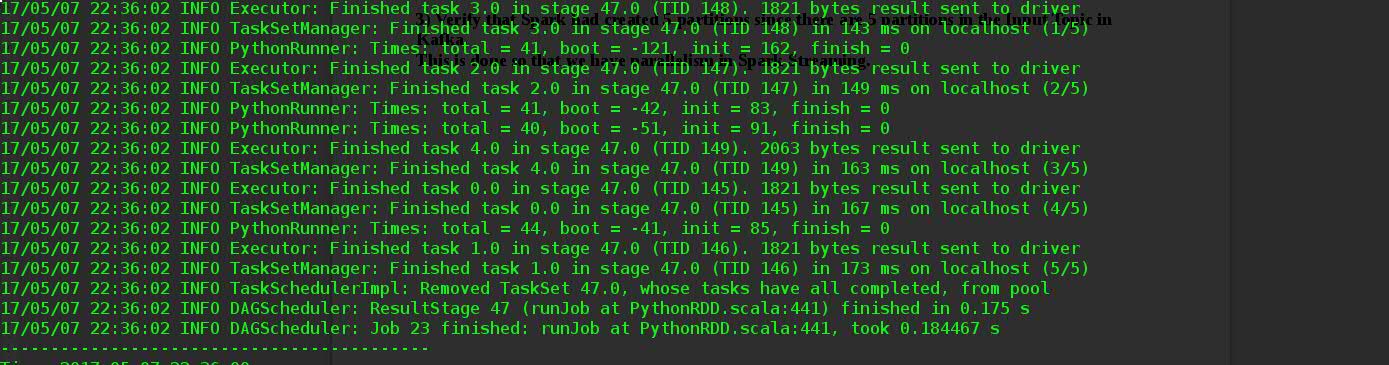
**use the kafka-console-consumer for verifying this.**

****

Above is the output from the Kafka-console-consumer and we can see that the output generated from the spark-streaming is available in the Kafka-output-topic which is **TEST\_AGGREGATED**.

**3) Verify that Spark had created 5 partitions since there are 5 partitions in the Input Topic in Kafka.**

**This is done so that we have parallelism in Spark Streaming.**



From the above output from UI and Spark Streaming output we see that Spark-Streaming had 5 partitions and hence there are 5 tasks.