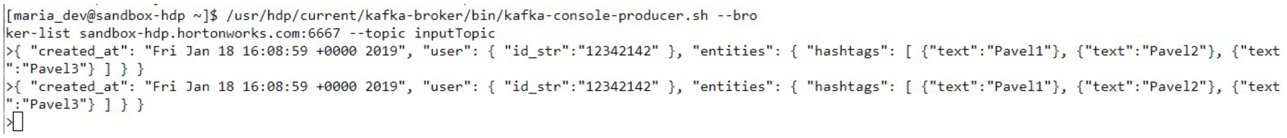
Advanced streaming report.

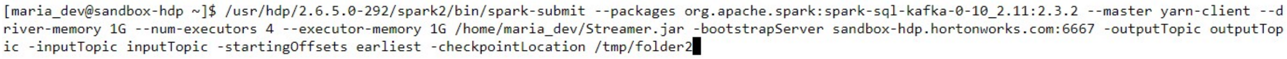
Let’s create an input topic that we will be reading from, and an output topic that we will be writing into.



```
[maria_dev@sandbox-hdp ~]$ /usr/hdp/current/kafka-broker/bin/kafka-console-producer.sh --bro
ker-list sandbox-hdp.hortonworks.com:6667 --topic inputTopic
>{ "created_at": "Fri Jan 18 16:08:59 +0000 2019", "user": { "id_str":"12342142" }, "entities": { "hashtags": [ {"text":"Pavel1"}, {"text":"Pavel2"}, {"text
":"Pavel3"} ] } }
>{ "created_at": "Fri Jan 18 16:08:59 +0000 2019", "user": { "id_str":"12342142" }, "entities": { "hashtags": [ {"text":"Pavel1"}, {"text":"Pavel2"}, {"text
":"Pavel3"} ] } }
>
```

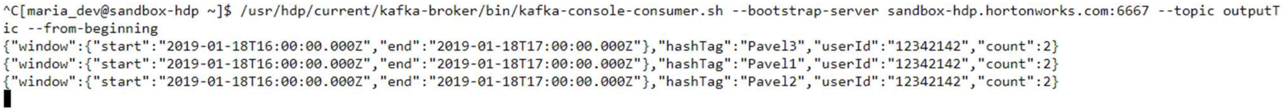
As you can see, I have 2 messages, each has 3 hashtags, the hashtags are the same in both messages and the hour is the same also. So, we should get one window per each hashtag output to hour new topic.

Now, we will ingest this data and group it by <1 hour time window on column “dateTime”, hashtag, userId>.



```
[maria_dev@sandbox-hdp ~]$ /usr/hdp/2.6.5.0-292/spark2/bin/spark-submit --packages org.apache.spark:spark-sql-kafka-0-10_2.11:2.3.2 --master yarn-client --d
river-memory 1G --num-executors 4 --executor-memory 1G /home/maria_dev/Streamer.jar -bootstrapServer sandbox-hdp.hortonworks.com:6667 -outputTopic outputTop
ic -inputTopic inputTopic -startingOffsets earliest -checkpointLocation /tmp/folder2
```

This is the result that I get if I consume from the output topic:

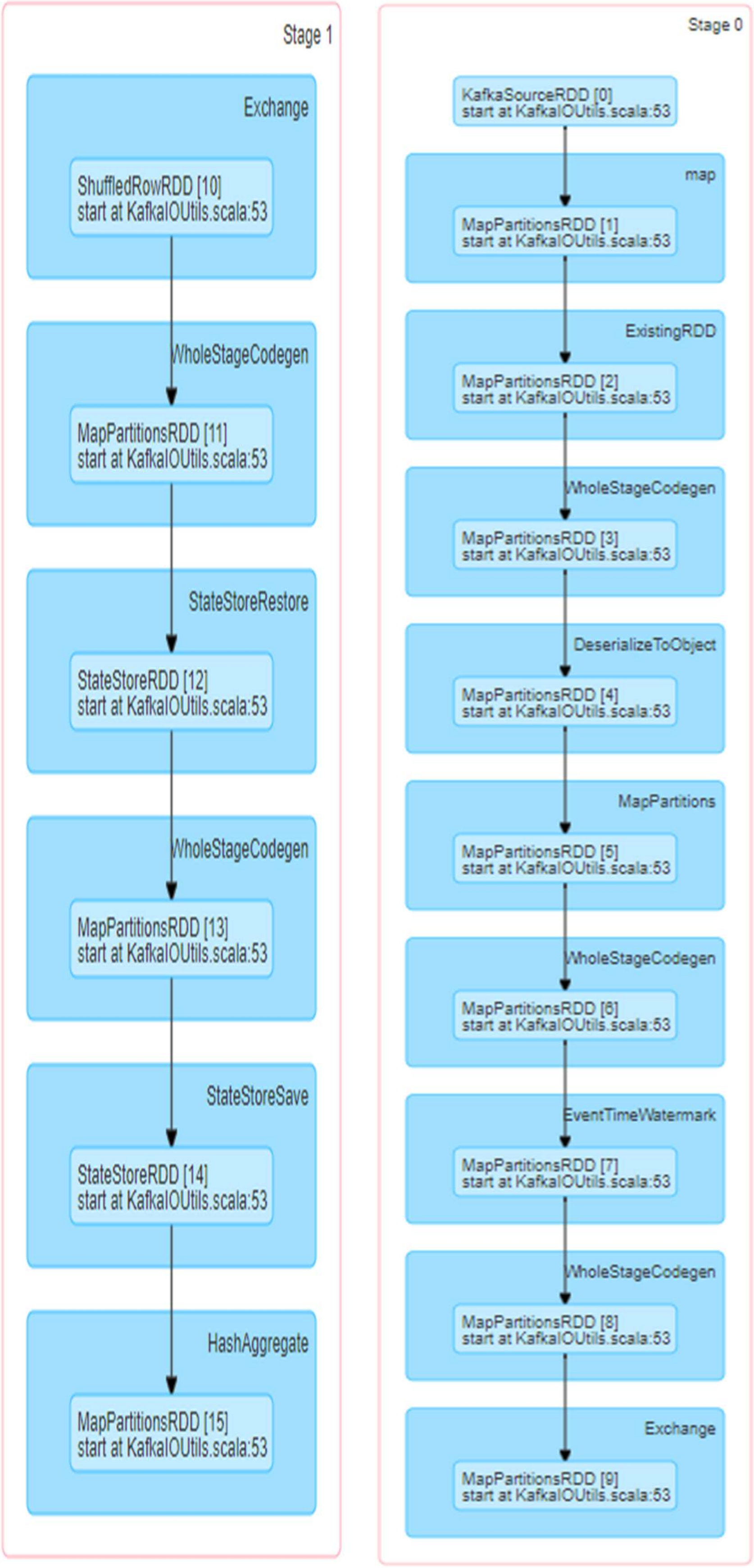


```
^C[maria_dev@sandbox-hdp ~]$ /usr/hdp/current/kafka-broker/bin/kafka-console-consumer.sh --bootstrap-server sandbox-hdp.hortonworks.com:6667 --topic outputT
ic --from-beginning
{"window":{"start":"2019-01-18T16:00:00.000Z","end":"2019-01-18T17:00:00.000Z"},"hashTag":"Pavel3","userId":"12342142","count":2}
{"window":{"start":"2019-01-18T16:00:00.000Z","end":"2019-01-18T17:00:00.000Z"},"hashTag":"Pavel1","userId":"12342142","count":2}
{"window":{"start":"2019-01-18T16:00:00.000Z","end":"2019-01-18T17:00:00.000Z"},"hashTag":"Pavel2","userId":"12342142","count":2}
```

Since I used an hourly watermark, in case I was to get a kafka message that was lagging 2 hours behind the last message that I received here, it would not have been included into the aggregated result and would have been dropped. In any case, the result is just as expected.

QED.

On the next page there are DAG pictures of this job’s stages.


Stage 1
Exchange
ShuffledRowRDD [10]
start at KafkaIOUtils.scala:53
WholeStageCodegen
MapPartitionsRDD [11]
start at KafkaIOUtils.scala:53
StateStoreRestore
StateStoreRDD [12]
start at KafkaIOUtils.scala:53
WholeStageCodegen
MapPartitionsRDD [13]
start at KafkaIOUtils.scala:53
StateStoreSave
StateStoreRDD [14]
start at KafkaIOUtils.scala:53
HashAggregate
MapPartitionsRDD [15]
start at KafkaIOUtils.scala:53
Stage 0
KafkaSourceRDD [0]
start at KafkaIOUtils.scala:53
map
MapPartitionsRDD [1]
start at KafkaIOUtils.scala:53
ExistingRDD
MapPartitionsRDD [2]
start at KafkaIOUtils.scala:53
WholeStageCodegen
MapPartitionsRDD [3]
start at KafkaIOUtils.scala:53
DeserializeToObject
MapPartitionsRDD [4]
start at KafkaIOUtils.scala:53
MapPartitions
MapPartitionsRDD [5]
start at KafkaIOUtils.scala:53
WholeStageCodegen
MapPartitionsRDD [6]
start at KafkaIOUtils.scala:53
EventTimeWatermark
MapPartitionsRDD [7]
start at KafkaIOUtils.scala:53
WholeStageCodegen
MapPartitionsRDD [8]
start at KafkaIOUtils.scala:53
Exchange
MapPartitionsRDD [9]
start at KafkaIOUtils.scala:53