Kafka + Spark Streaming + Cassandra HW2/Task 2

Wyatt Melin: wmelin2@illinois.ede

I. Abstract

Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including (but not limited to) Kafka, flume and Amazon Kinesis. [1] Since homework 2 requires the use of Kafka and Spark Streaming, this paper will mention the steps used to achieve such of a task and finally stream results into Cassandra DB.

II. Architecture

The full end to end architecture of the assignment is outlined in figure 1 below. Much like homework 1, attaching the volume to an EMR cluster, loop through the data and finally, write only csv files to HDFS. Once CSV files are loaded in HDFS, then bulk querying of the data the get only the results needed to answer each question which is consolidated into a CSV file to be sent off to S3. The CSV file is then loaded from S3 to an EC2 instance running Python3,and using the Kafka python framework. The process of emitting code to the Kafka cluster can be found here. [5]

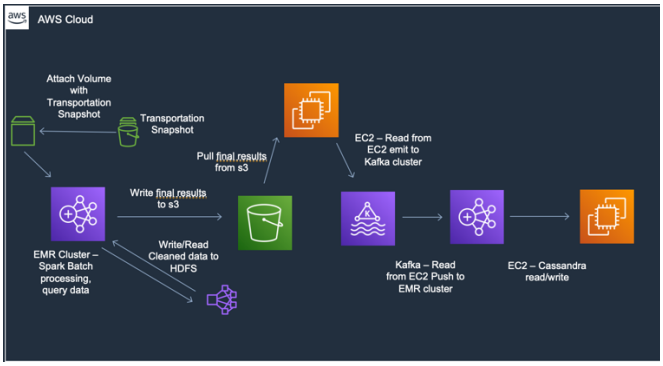

Figure 1 – Describes the full end to end architecture of HW assignment 2.

This paper will focus more on the architecture described in figure 2, and assumes there is already prior background knowledge in Spark batch processing and HDFS.

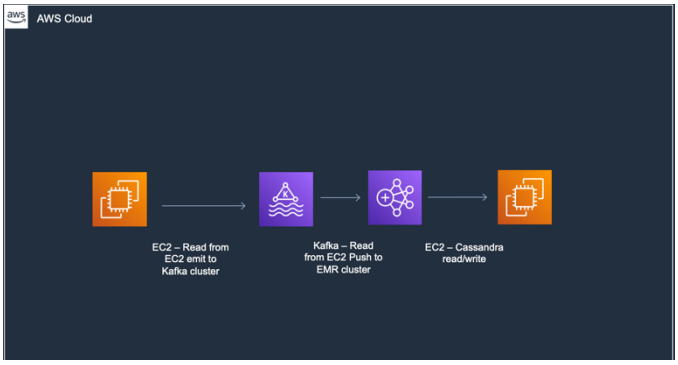

Figure 2 – Describes the architecture of emitting data to the Kafka topic to be listened to by a EMR cluster for Spark streaming.

III. Kafka

Once the files are downloaded from S3 to EC2, we can then begin the process of streaming to Kafka. As mentioned earlier, to emit data to the Kafka topic, I wrote a python script that used pandas to read each CSV file. Then for each row in the CSV file, I then formatted the string to be delimited by commas on the Spark streaming. (More on that later) I did run into hurdle to get the Spark cluster to work properly. It was to my surprise that I needed to run a separate script with the "**KafkaAdminClient**" to properly setup the Kafka topic. Code can be found here [5] demonstrating the setup of the Kafka topic via Python.

IV. Spark Streaming + Cassandra

In order to connect Cassandra with Spark, first is configuring any firewall rules would apply. Second, is the –conf parameter to specify the IP address of the EC2 instance Cassandra was running on. The full command ended up like this:

```
spark-submit -packages org.apache.spark:spark-streaming-kafka-08_211:21.0,come.datastax.spark:spark-cassandra-connector_2.11:2.3.0–conf spark.cassandra.connection.host=XXX.XX.X.XX X Assignment2Version2.py localhost:9092
```

V. Results

Question 1.1

```
cqlsh:homework> select * from oneone;

 uid                                  | destination | numberofflights
--------------------------------------+-------------+----------------
 75614f70-df88-11eb-9589-9db2c4145ca6 |         ATL |        11540422
 d35cc230-df88-11eb-9589-9db2c4145ca6 |         DEN |         6273787
 f7a71280-df88-11eb-9589-9db2c4145ca6 |         SFO |         5171023
 68c6fdf0-df88-11eb-9589-9db2c4145ca6 |         ORD |        12449354
 80f52d70-df88-11eb-9589-9db2c4145ca6 |         DFW |        10799303
 dd01a170-df88-11eb-9589-9db2c4145ca6 |         DTW |         5636622
 e61452d0-df88-11eb-9589-9db2c4145ca6 |         IAH |         5480734
 8a85c160-df88-11eb-9589-9db2c4145ca6 |         LAX |         7723596
 eeb50880-df88-11eb-9589-9db2c4145ca6 |         MSP |         5199213
 93f20470-df88-11eb-9589-9db2c4145ca6 |         PHX |         6585534

(10 rows)
cqlsh:homework>
```

Question 1.2

```
cqlsh:homework> select * from onetwo;

 uid                                  | airline | averagedelayinminutes
--------------------------------------+---------+----------------------
 21e002c0-df87-11eb-9589-9db2c4145ca6 |      HA |                -1.01
 322b0030-df87-11eb-9589-9db2c4145ca6 |      AQ |                 1.16
 8ada3f70-df87-11eb-9589-9db2c4145ca6 |      OO |                 5.74
 94596e40-df87-11eb-9589-9db2c4145ca6 |      9E |                 5.87
 7c63b3e0-df87-11eb-9589-9db2c4145ca6 |      NW |                 5.56
 6294f3c0-df87-11eb-9589-9db2c4145ca6 |  ML (1) |                 4.75
 3a76df20-df87-11eb-9589-9db2c4145ca6 |      PS |                 1.45
 73ff9200-df87-11eb-9589-9db2c4145ca6 |      F9 |                 5.47
 6ab86320-df87-11eb-9589-9db2c4145ca6 |  PA (1) |                 5.32
 835050f0-df87-11eb-9589-9db2c4145ca6 |      WN |                 5.56

(10 rows)
```

Questions 2.1, 2.2, 2.3, 2.4

airline	averagedelay	origin
EA	5.93739	CMH
NW	4.04155	CMH
UA	3.95212	CRQ
EA	8.89143	BOS
DH	8.74298	JFK
XE	1.48977	CRQ
ML (1)	4.36646	CMH
TZ	-0.381997	CRQ
DL	4.86918	CRQ
AA	10.08074	JFK
US	3.9684	CRQ
CO	8.20121	JFK
DL	7.44534	BOS
OO	2.70582	SEA
YV	5.12226	SEA
XE	8.11374	JFK
US	5.9933	CMH
NW	4.85636	CRQ
PS	4.72064	SEA
TW	12.63907	JFK
PI	5.20129	CMH
NW	6.49876	SEA
PA (1)	11.52348	JFK
AA	3.51393	CMH
YV	7.96119	CMH
AA	8.73351	BOS
TZ	6.345	SEA
NW	11.63782	JFK
US	6.41238	SEA
B6	11.1271	JFK
DL	11.98667	JFK
DL	4.71344	CMH
TW	4.30468	CRQ
MQ	5.35059	CRQ

I have merged the above question results together because they had much similar data characteristics and therefore, can be streamed together when it came to Spark Streaming and writing to the Datastax data frame via the Cassandra connector.

Question 3.2

arrivaldelay	dest	origin	scheduleddeparture
-25	JFK	PHX	2008-09-08 00:00:00+0000
7	ATL	BOS	2008-03-04 00:00:00+0000
-14	STL	ORD	2008-01-26 00:00:00+0000
-19	LAX	MIA	2008-05-16 00:00:00+0000
10	MIA	LAX	2008-05-16 00:00:00+0000
-25	MSP	JFK	2008-09-09 00:00:00+0000
-2	BOS	ATL	2008-05-08 00:00:00+0000
-14	STL	DFW	2008-01-24 00:00:00+0000

VI. Optimizations

- Submitting rows only – Submitting only rows and delimiting by string in comma is a performance optimization I have chosen in my architecture. Starting with submitting data to the Kafka topic and ending in Cassandra, this will help with data transit both on the up & downstream of all components in the data pipeline.
- Stream filtered data only – Streaming filtered data only reduces the processing noise in the Spark streaming cluster. Allocating cluster resources only to results that are final will help throughput end to end.

VII. Final Summary

Connecting the Datastax data frame was learning curve for me. Understanding the Datastax Cassandra connector took also a thorough understanding of the Spark development lifecycle.[2] More specifically, I needed to understand what were the underlying components to the spark-submit command, particularly when it came to the –packages command and the –conf command. [3]

The formal documentation on the Spark website was a bit bland for me to understand. A quick google led me to a stack overflow post, that explained that packages are maven coordinates of jars to be included on the driver and executor class paths. [4]

When it comes to the -conf command, the Spark documentation was easier for me to digest. Simply put, the conf parameter is key value format Spark configuration property.

Putting it altogether, my Spark streaming application uses the Kafka and Cassandra connector maven packages. Both these packages together, embark the core of responding to a Kafka topic and conveniently saving the data to Cassandra natively in Spark via RDD. Both Task 1 and Task 2 have given me hands on experience into Spark batch processing and Spark streaming with Kafka! Yes, formatting the data correctly for Cassandra to insert from Spark can be a hassle. However, when placing Cassandra under the CAP theorem, it's one of the better open-source storage options for throughput and availability!

REFERENCES:

[1] https://databricks.com/glossary/what-is-spark-streaming
[2] https://spark.apache.org/docs/latest/submitting-applications.html
[3] https://medium.datadriveninvestor.com/spark-architecture-and-application-lifecycle-77e347badcbe
[4] https://stackoverflow.com/questions/51434808/spark-submit-packages-vs-jars
[5] https://github.com/wyguy321/SparkStreaming