

Lambda And Kappa Architecture for Computing Stocks Price Statistics

Problem Statement:

Compute NVIDIA closing stock price statistics using Lambda and Kappa Big Data Architectures.
Speed Layer will compute immediate data statistics.
Batch layer will compute stored batch data statistics.

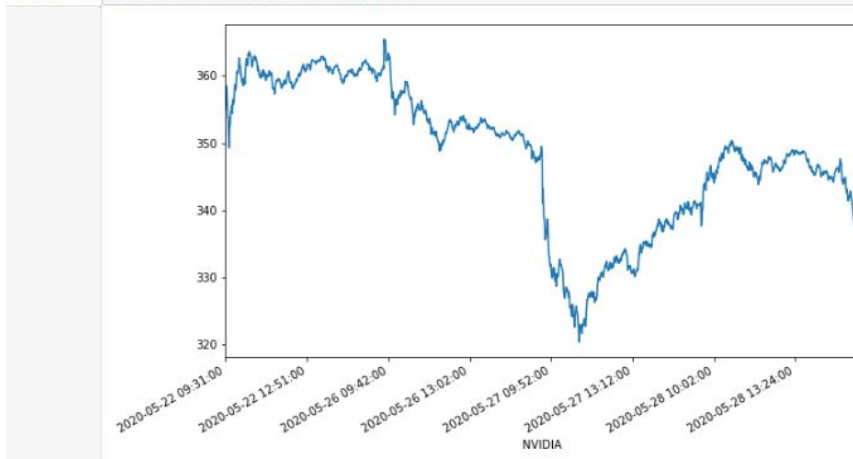
Technology Stack:

Alpha Vantage API for NVIDIA stock price data (7 days data with 1 min interval)

(https://github.com/RomeITorres/alpha_vantage)

Notebook - AlphaVantageTS - this notebook is available in spark folder

```
In [10]: plot_time_series(nvda, "NVIDIA")
```



Falcon Web Service for posting stored stocks data in JSON as kafka streams (works as a kafka producer). **Kafka** is used to process streaming of stock prices

Zookeeper to manage kafka resources

Spark is used to implement the business logic in speed and batch layers in architectures

Steps to reproduce the lambda and kappa architectures:

1. docker-compose.yml:
 - a. Change volumes as per your system drive in docker-compose.yml
 - b. cd to folder where docker-compose.yml resides
 - c. execute docker-compose up --build
 - d. Above command will fetch and create containers for falcon, kafka, cassandra, zookeeper and spark

```
Terminal
murtaza@Machine ~ $ xrandr --output LVDS1 --brightness 0.70
murtaza@Machine ~ $ cd BigDataArchitectures/
murtaza@Machine ~/BigDataArchitectures $ docker-compose up --build
Building falcon
Step 1/5 : FROM python:2.7.13-slim
--> 426d65ab9a72
Step 2/5 : ADD . /project
--> Using cache
--> a66f91484b99
Step 3/5 : RUN chmod a+x /project/src/run.sh
--> Using cache
--> c161eb828b56
Step 4/5 : EXPOSE 8000
--> Using cache
--> 9411213709e7
Step 5/5 : CMD /project/src/run.sh
--> Using cache
--> 658b05526109
Successfully built 658b05526109
Successfully tagged bigdataarchitectures_falcon:latest
Starting bigdataarchitectures_cassandra_1 ... done
Starting bigdataarchitectures_zookeeper_1 ... done
Starting bigdataarchitectures_kafka_1 ... done
Starting bigdataarchitectures_spark_1 ... done
Starting bigdataarchitectures_falcon_1 ... done
Attaching to bigdataarchitectures_zookeeper_1, bigdataarchitectures_cassandra_1, bigdataarchitectures_kafka_1, bigdataarchitectures_falcon_1, bigdataarchitectures_spark_1
ZooKeeper JMX enabled by default
Using config: /opt/zookeeper-3.4.13/bin/../conf/zoo.cfg
waiting for kafka to be ready
waiting for kafka to be ready
cassandra container
a. execute init.sql to create tables for data storage
b. gsqlsh -f hdrinit.sql
3. Simulate streaming data from falcon service and input to kafka producer for streaming
a. cd data folder
b. ./user-simulator.py
```

```
Terminal
CONTAINER ID        IMAGE                               STATUS              PORTS              NAMES
d4f96af470e8       bigdataarchitectures_falcon       Up About a minute   0.0.0.0:8000->8000/tcp  bigdataarchitectures_falcon_1
fb50054b9f39       jupyter/pyspark-notebook:b17be6205231  Up About a minute   0.0.0.0:4040-4041->4040-4041/tcp, 0.0.0.0:8888->8888/tcp  bigdataarchitectures_spark_1
c876f1991c54       wurstmeister/kafka:0.10.2.0        Up About a minute   0.0.0.0:9092->9092/tcp  bigdataarchitectures_kafka_1
79fa872c61c5       cassandra:3                        Up About a minute   0.0.0.0:7000-7001->7000-7001/tcp, 0.0.0.0:7199->7199/tcp, 0.0.0.0:9042->9042/tcp, 0.0.0.0:9160->9160/tcp  bigdataarchitectures_cassandra_1
7b4bb961107d       wurstmeister/zookeeper             Up About a minute   22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp  bigdataarchitectures_zookeeper_1
murtaza@Machine ~/BigDataArchitectures $
```

2. sh into cassandra container
 - a. execute init.sql to create tables for data storage
 - b. csqslsh -f bdr/init.sql

```
murtaza@Machine ~ $ docker exec -it 79fa872c61c5 bash
root@79fa872c61c5:/# cqlsh -f bdr/init.py
Can't open 'bdr/init.py': [Errno 2] No such file or directory: 'bdr/init.py'
root@79fa872c61c5:/# ls
bdr  boot  dev          etc      lib      media  opt    root  sbin  sys  usr
bin  data  docker-entrypoint.sh  home  lib64  mnt    proc  run  srv  tmp  var
root@79fa872c61c5:/# cqlsh -f bdr/init.sql
root@79fa872c61c5:/#
```

3. Simulate streaming data from falcon service and input to kafka producer for streaming
 - a. cd data folder
 - b. ./user-simulator.py

[illegible]

4. LAMBDA ARCHITECTURE:

The batch and speed layer of the lambda architectures are implemented using PySpark. The streaming data is fetched from Kafka producer, processed to compute stock price statistics and finally the data and results are saved to Cassandra, and later the results are served using Falcon service.

5. Process Streaming Data Using PySpark

- Open Notebook - Lambda - Stream - Stocks Statistics on localhost:8888 home:
- Execute the cells to implement speed layer of the lambda architecture

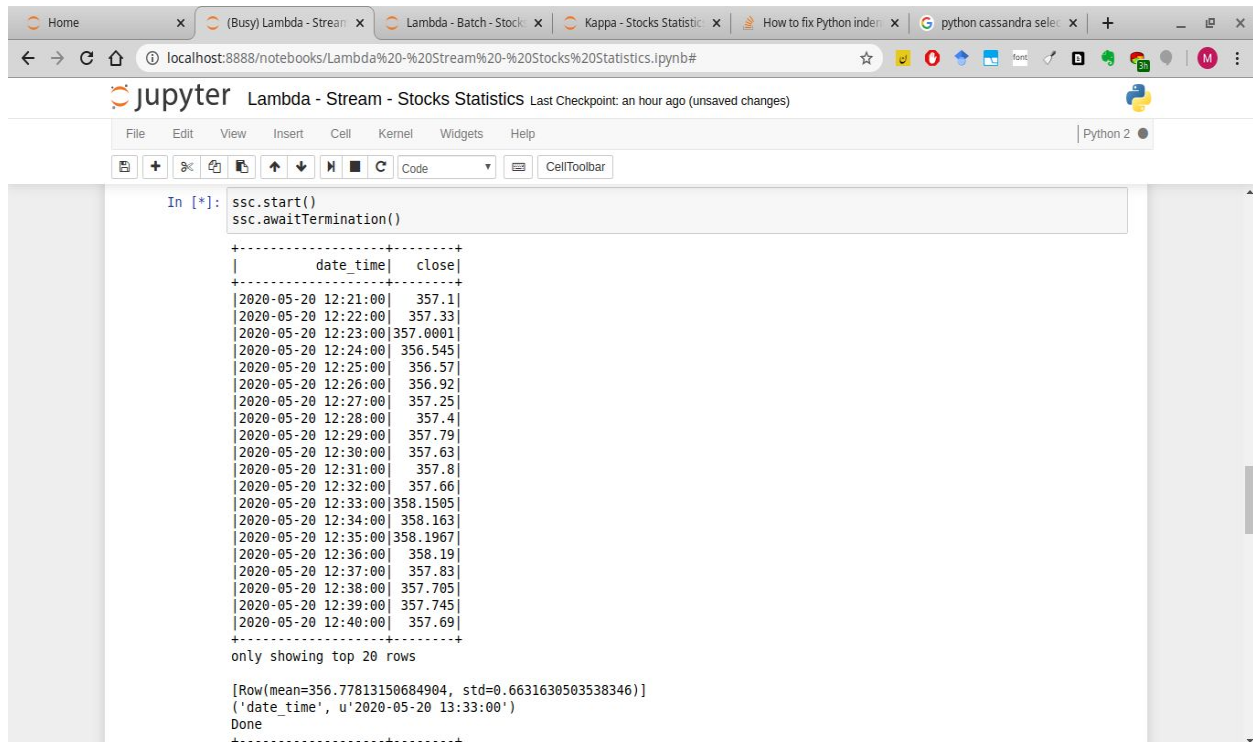
```
In [ ]: def processStream(rdd):
        df = sql.read.json(rdd)
        if(len(df.columns)):
            priceDf = df.select('date_time', 'close').cache()
            close_price_statistics(priceDf).\
                write.format("org.apache.spark.sql.cassandra").\
                mode('append').options(table="nvda_stock_close_stats_lambda", keyspace="bdr").save()

            #store stocks data for the batch process
            df.write.format("org.apache.spark.sql.cassandra").\
                mode('append').options(table="nvda_stock_data_lambda", keyspace="bdr").save()

        print "Done"
    else:
        print "Empty"
```

```
In [ ]: parsed.foreachRDD(lambda rdd: processStream(rdd))
```

- Speed Layer process the immediate Spark RDD stream from Kafka producer



The screenshot shows a Jupyter Notebook interface with the following components:

- Browser Tabs:** Home, (Busy) Lambda - Stream, Lambda - Batch - Stock, Kappa - Stocks Statistic, How to fix Python index, python cassandra sele, and a plus sign for more tabs.
- Address Bar:** localhost:8888/notebooks/Lambda%20-%20Stream%20-%20Stocks%20Statistics.ipynb#
- Page Header:** jupyter Lambda - Stream - Stocks Statistics Last Checkpoint: an hour ago (unsaved changes)
- Menu Bar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help
- Toolbar:** Includes icons for file operations, cell execution, and a CellToolBar.
- Code Cell:**

```
In [*]: ssc.start()
        ssc.awaitTermination()
```
- Output:**

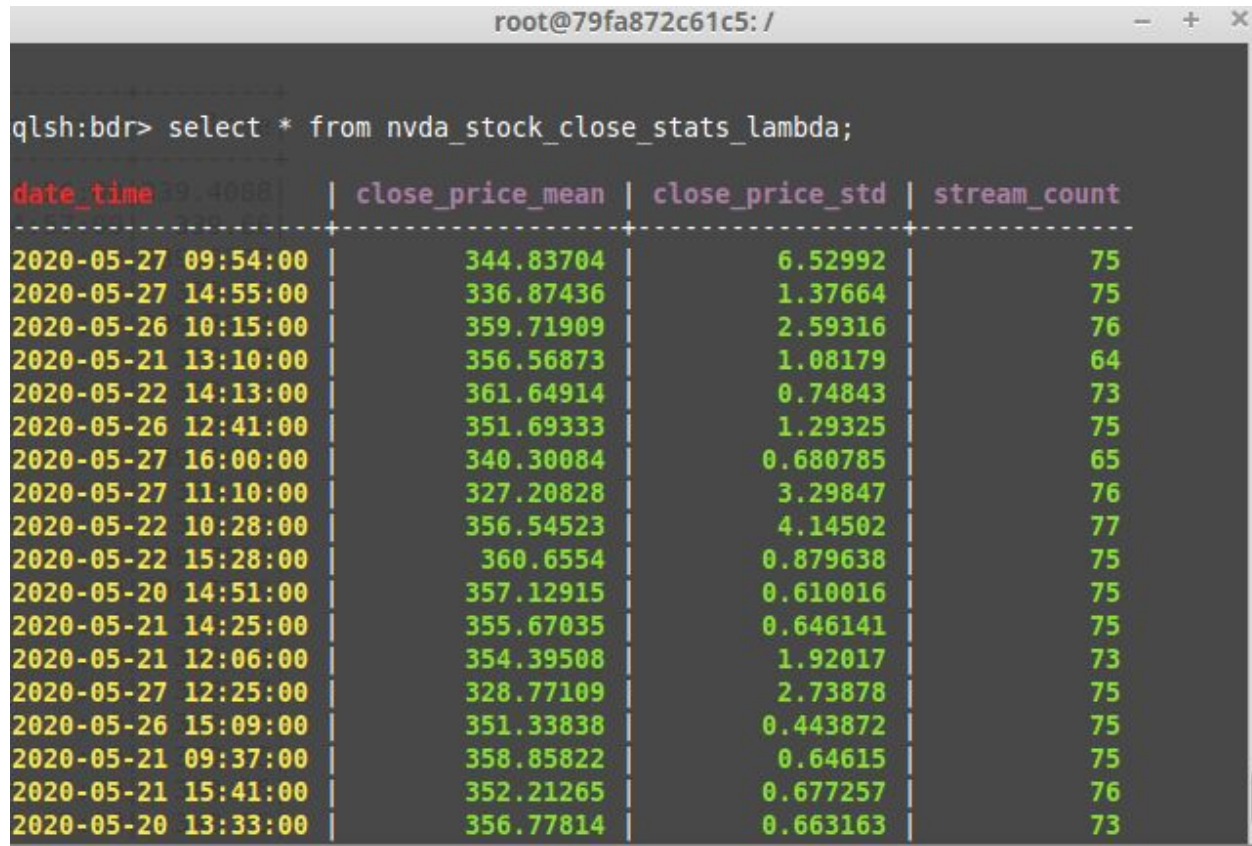
date_time	close
2020-05-20 12:21:00	357.1
2020-05-20 12:22:00	357.33
2020-05-20 12:23:00	357.0001
2020-05-20 12:24:00	356.545
2020-05-20 12:25:00	356.57
2020-05-20 12:26:00	356.92
2020-05-20 12:27:00	357.25
2020-05-20 12:28:00	357.4
2020-05-20 12:29:00	357.79
2020-05-20 12:30:00	357.63
2020-05-20 12:31:00	357.8
2020-05-20 12:32:00	357.66
2020-05-20 12:33:00	358.1505
2020-05-20 12:34:00	358.163
2020-05-20 12:35:00	358.1967
2020-05-20 12:36:00	358.19
2020-05-20 12:37:00	357.83
2020-05-20 12:38:00	357.705
2020-05-20 12:39:00	357.745
2020-05-20 12:40:00	357.69

only showing top 20 rows

```
[Row(mean=356.77813150684904, std=0.6631630503538346)]
('date_time', u'2020-05-20 13:33:00')
Done
```

- d. Stream Statistics are saved to lambda stream statistics table

The following screenshot shows the stream statistics. The table shows date, mean value and standard deviation with number of records in the stream. The date is not in sorted order, so the data is not ordered. Nevertheless we can see the stream statistics on different time slots.



```
root@79fa872c61c5: /
qlsh:bdr> select * from nvda_stock_close_stats_lambda;
```

date_time	close_price_mean	close_price_std	stream_count
2020-05-27 09:54:00	344.83704	6.52992	75
2020-05-27 14:55:00	336.87436	1.37664	75
2020-05-26 10:15:00	359.71909	2.59316	76
2020-05-21 13:10:00	356.56873	1.08179	64
2020-05-22 14:13:00	361.64914	0.74843	73
2020-05-26 12:41:00	351.69333	1.29325	75
2020-05-27 16:00:00	340.30084	0.680785	65
2020-05-27 11:10:00	327.20828	3.29847	76
2020-05-22 10:28:00	356.54523	4.14502	77
2020-05-22 15:28:00	360.6554	0.879638	75
2020-05-20 14:51:00	357.12915	0.610016	75
2020-05-21 14:25:00	355.67035	0.646141	75
2020-05-21 12:06:00	354.39508	1.92017	73
2020-05-27 12:25:00	328.77109	2.73878	75
2020-05-26 15:09:00	351.33838	0.443872	75
2020-05-21 09:37:00	358.85822	0.64615	75
2020-05-21 15:41:00	352.21265	0.677257	76
2020-05-20 13:33:00	356.77814	0.663163	73

- e. And all the streamed data is saved to cassandra for batch computation

```
root@79fa872c61c5: /
root@79fa872c61c5:/# cqlsh
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.11.6 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh> use bdr;
cqlsh:bdr> describe tables

nvda_stock_stats kappa          nvda_stock_data_kappa
nvda_stock_data_stats_lambda    nvda_stock_data_lambda
nvda_stock_close_stats_lambda

cqlsh:bdr> select * from nvda_stock_data_lambda
... ;

```

date_time	close	high	low	open	volume
2020-05-20 14:21:00	356.29251	356.55981	356.17499	356.32001	1698
2020-05-26 12:54:00	352.495	353.095	352.29001	353.07999	2884
2020-05-27 11:25:00	327.64999	327.74991	327.20001	327.41	7330
2020-05-21 15:10:00	353.06	353.22	353.03	353.14999	1375

6. Process Batch Data Using Spark:

- a. Open Notebook - Lambda - Batch - Stocks Statistics on localhost:8888 home

```
In [16]: def processBatch():
df = sql.read.format("org.apache.spark.sql.cassandra").\
options(table="nvda_stock_data_lambda", keyspace="bdr").load().cache()

if(len(df.columns)):
    priceDf = df.select('date_time', 'close').cache()
    close_price_statistics(priceDf).\
        write.format("org.apache.spark.sql.cassandra").\
        mode('append').options(table="nvda_stock_data_stats_lambda", keyspace="bdr").save()

    print "Done"
else:
    print "Empty"
```

- b. Execute cells to execute the batch layer of the lambda architecture

only showing top 20 rows

```
[Row(mean=351.258466814068, std=10.477217980568085)]
('date_time', u'2020-05-27 14:07:00')
Done
```

- c. The statistics of stocks data batch are saved to cassandra
- d. Access stored results using Falcon requests

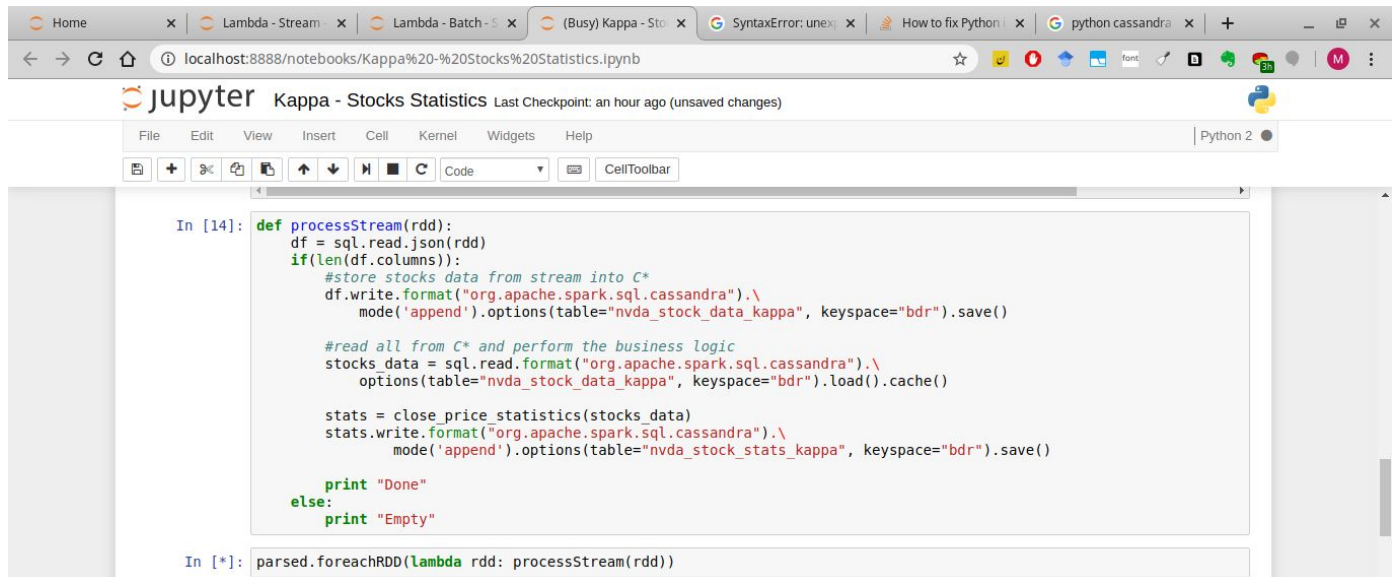
```
127.0.0.1:8000/bdr?stocks-lam...
127.0.0.1:8000/bdr?stocks-lambda=stocks-lambda

{"NVIDIA Stocks Stream Statistcs - Lambda": [{"2020-05-20 15:15:00", 357.444091796875, 0.799302339553833, 78}, {"2020-05-21 12:22:00", 355.7643737792969, 1.8710006475448608, 64}, {"2020-05-20 14:03:00", 356.8752746582031, 0.7247077226638794, 70}, {"2020-05-22 12:13:00", 359.3701477050781, 0.8271702527999878, 70}, {"2020-05-26 14:09:00", 352.4309387207031, 0.5740280151367188, 77}, {"2020-05-22 09:51:00", 352.61505126953125, 1.8961962461471558, 73}, {"2020-05-21 12:42:00", 356.6593017578125, 1.0361791849136353, 73}, {"2020-05-27 11:50:00", 325.52044677734375, 2.183438301086426, 77}, {"2020-05-22 13:26:00", 360.9294738769531, 1.314462423324585, 73}, {"2020-05-20 12:58:00", 357.7593078613281, 1.0654906034469604, 79}, {"2020-05-22 12:02:00", 359.57843017578125, 0.9666242599487305, 75}, {"2020-05-26 15:04:00", 351.44647216796875, 0.5392846465110779, 78}, {"2020-05-21 10:40:00", 358.3649597167969, 1.711965560913086, 77}, {"2020-05-26 12:52:00", 351.97967529296875, 1.4529130458831787, 75}, {"2020-05-20 15:18:00", 357.4889221191406, 0.8325269818305969, 75}, {"2020-05-21 11:00:00", 357.0888977050781, 2.060117483139038, 76}, {"2020-05-26 11:56:00", 353.0216369628906, 2.1239945888519287, 77}, {"2020-05-27 15:41:00", 339.1783142089844, 1.2250617742538452, 77}, {"2020-05-21 12:57:00", 356.54705810546875, 0.9483515620231628, 35}, {"2020-05-26 14:02:00", 352.60565185546875, 0.552599310874939, 76}, {"2020-05-20 10:23:00", 358.5992431640625, 0.8916797041893005, 53}, {"2020-05-26 15:26:00", 350.99664306640625, 0.5995392203330994, 77}, {"2020-05-26 12:46:00", 352.4554443359375, 1.0623294115066528, 51}, {"2020-05-22 13:19:00", 360.6234130859375, 1.2789578437805176, 77}, {"2020-05-20 11:18:00", 357.74444580078125, 0.7509284615516663, 75}, {"2020-05-20 12:50:00", 358.0611572265625, 1.081049919128418, 76}, {"2020-05-26 13:46:00", 352.8387145996094, 0.6202677488327026, 76}, {"2020-05-20 13:57:00", 356.803466796875, 0.7266151309013367, 77}, {"2020-05-27 14:24:00", 334.49310302734375, 2.4668476581573486, 77}, {"2020-05-21 12:38:00", 356.2783508300781, 1.5805453062057495, 77}, {"2020-05-26 15:19:00", 351.1554870605469, 0.4402821362018585, 77}, {"2020-05-22 12:58:00", 359.7853088378906, 1.0949773788452148, 77}, {"2020-05-20 13:03:00", 357.0825500488281, 0.7133272886276245, 53}, {"2020-05-27 10:14:00", 340.00909423828125, 7.982994556427002, 78}, {"2020-05-27 12:26:00", 328.7369079589844, 2.8152031898498535, 77}, {"2020-05-20 12:37:00", 358.47381591796875, 0.8925294876098633, 79}, {"2020-05-27 15:22:00", 338.4986267089844, 1.1841654777526855, 76}, {"2020-05-21 13:30:00", 356.1719055175781, 0.9879884123802185, 76}, {"2020-05-26 14:28:00", 352.1687316894531, 0.7996277809143066, 75}, {"2020-05-27 13:05:00", 332.0389709472656, 1.1355196237564087, 76}, {"2020-05-27 16:00:00", 340.8548278808594, 0.29457807540893555, 22}, {"2020-05-27 12:36:00", 329.99151611328125, 2.1755056381225586, 76}, {"2020-05-27 13:52:00", 332.9056091308594, 1.6038281917572021, 76}, {"2020-05-21 11:54:00", 354.11895751953125, 1.657448649406433, 74}, {"2020-05-27 14:21:00", 334.30767822265625, 2.457120895385742, 76}, {"2020-05-26 10:43:00", 358.65130615234375, 2.80637788772583, 77}, {"2020-05-22 10:26:00", 356.2325744628906, 4.089753150939941, 77}, {"2020-05-22 14:16:00", 361.5541687011719, 0.8287563920021057, 78}, {"2020-05-21 09:44:00", 359.1014709472656, 0.7613022327423096, 63}, {"2020-05-22 14:36:00", 360.98382568359375, 1.0746381282806396, 77}, {"2020-05-21 15:08:00", 353.4382629394531, 1.5548748970031738, 69}, {"2020-05-20 10:16:00", 358.7252502441406, 0.8819521069526672, 46}, {"2020-05-22 12:36:00", 359.1671447753906, 0.7606996893882751, 77}, {"2020-05-21 11:18:00", 355.4595642089844, 2.4212734699249268, 77}, {"2020-05-26 11:13:00", 356.1764221191406, 1.4853794574737549, 77}, {"2020-05-20 11:34:00", 357.96588134765625, 0.9099789261817932, 78}, {"2020-05-21 15:14:00", 353.3870544433594, 1.5441246032714844, 76}, {"2020-05-20 15:52:00", 358.56298828125, 0.7185864448547363, 72}, {"2020-05-26 11:37:00", 354.8883056640625, 2.1280665397644043, 76}, {"2020-05-20 15:35:00", 357.9229431152344, 0.7991465926170349, 75}, {"2020-05-22 15:55:00", 360.79827880859375, 0.7367756366729736, 75}, {"2020-05-20 14:40:00", 356.96441650390625, 0.6885085701942444, 73}, {"2020-05-26 13:13:00", 352.6002197265625, 0.7824697494506836, 74}, {"2020-05-21 14:24:00", 355.6967468261719, 0.6052044034004211, 75}, {"2020-05-27 11:32:00", 325.91278076171875, 2.6991777420043945, 78}, {"2020-05-20 13:24:00", 356.7198791503906, 0.4002355933189392, 25}, {"2020-05-21 13:58:00", 355.7777404785156, 0.5665695667266846, 76}, {"2020-05-22 10:47:00", 359.42755126953125, 3.445847272872925, 74}, {"2020-05-27 10:33:00", 335.1180725097656, 7.0751519203186035, 77}, {"2020-05-26 10:39:00", 358.9137878417969, 2.7077231407165527, 76}, {"2020-05-20 15:32:00", 357.8423767089844, 0.8294095993041992, 75}, {"2020-05-22 11:41:00", 360.625244140625, 1.56346595287323, 75}, {"2020-05-27 14:59:00", 337.0906982421875, 1.4475420713424683, 76}, {"2020-05-27 13:07:00", 332.0533447265625, 1.1082197427749634, 77}, {"2020-05-22 13:52:00", 361.70574951171875, 0.653119683265686, 76}, {"2020-05-26 11:55:00", 353.0619201660156, 2.108288288116455, 76}, {"2020-05-21 10:13:00", 359.2970886230469, 0.7126681208610535, 67}, {"2020-05-20 14:20:00", 356.893798828125, 0.6922262907028198, 74}, {"2020-05-21 11:21:00", 355.18084716796875, 2.456285238265991, 77}, {"2020-05-21 10:04:00", 359.2069396972656, 0.6871939897537231, 78}, {"2020-05-21 12:14:00", 354.7332458496094, 2.176048755645752, 74}, {"2020-05-27 15:07:00", 337.5833740234375, 1.3552634716033936, 75}, {"2020-05-20 11:39:00", 358.11505126953125, 1.039027452468872, 76}, {"2020-05-22 11:03:00", 360.94293212890625, 1.6338447332382202, 72}, {"2020-05-27 12:49:00", 330.8858337402344, 2.014518976211548, 77}, {"2020-05-21 14:45:00", 354.84759521484375, 1.6474218368530273, 75}, {"2020-05-21 15:39:00", 352.2629699707031, 0.6777615547180176, 75}, {"2020-05-22 09:33:00", 352.121337890625, 1.216566562652588, 78}, {"2020-05-22 14:59:00", 360.4228820800781, 0.668528676603302, 74}, {"2020-05-22 15:09:00", 360.4736328125, 0.7009450793264296, 77}, {"2020-05-26 09:56:00", 361.0291442871094, 1.7715320587158203, 76}, {"2020-05-27 13:43:00", 332.6474304199219, 1.4111813306808472, 77}, {"2020-05-21 13:09:00", 356.57757568359375, 1.037994384765625, 75}, {"2020-05-22 09:59:00", 353.22528076171875, 2.5933749675750732, 76}, {"2020-05-26 12:30:00", 351.7801208496094, 1.3750232458114624, 77}, {"2020-05-26 10:21:00", 359.4808654785156, 2.6209042072296143, 77}, {"2020-05-21 10:01:00", 359.26104736328125,
```

KAPPA ARCHITECTURE:

7. In Kappa architecture, every data is considered as a stream and processed accordingly. We only need to implement one layer. This layer would be compute intensive. As the stream size can be varied according to requirements.

a. Open Notebook - Kappa - Stocks Statistics on localhost:8888



```
In [14]: def processStream(rdd):
df = sql.read.json(rdd)
if(len(df.columns)):
    #store stocks data from stream into C*
    df.write.format("org.apache.spark.sql.cassandra").\
        mode('append').options(table="nvda_stock_data_kappa", keyspace="bdr").save()

    #read all from C* and perform the business logic
    stocks_data = sql.read.format("org.apache.spark.sql.cassandra").\
        options(table="nvda_stock_data_kappa", keyspace="bdr").load().cache()

    stats = close_price_statistics(stocks_data)
    stats.write.format("org.apache.spark.sql.cassandra").\
        mode('append').options(table="nvda_stock_stats_kappa", keyspace="bdr").save()

    print "Done"
else:
    print "Empty"
```

```
In [*]: parsed.foreachRDD(lambda rdd: processStream(rdd))
```

b. Execute cells for Kappa Stream layer implementation

```
In [*]: ssc.start()
ssc.awaitTermination()
```

date_time	close	high	low	open	volume
2020-05-20 12:24:00	356.545	357.08	356.5	357.0	33881.0
2020-05-20 11:32:00	359.77	359.8	359.33	359.39	21405.0
2020-05-20 10:57:00	357.425	357.74	357.38	357.6287	19459.0
2020-05-20 15:44:00	359.93	360.03	359.852	360.025	28452.0
2020-05-20 09:40:00	358.43	359.01	357.4	357.71	68430.0
2020-05-20 12:06:00	358.5998	358.6298	358.21	358.21	14295.0
2020-05-21 12:39:00	356.4545	357.21	356.13	357.1	42701.0
2020-05-20 15:50:00	358.99	359.15	358.92	359.14	24544.0
2020-05-20 12:57:00	356.37	356.5081	356.155	356.155	15535.0
2020-05-20 09:37:00	359.42	359.8065	359.145	359.325	72543.0
2020-05-21 12:01:00	355.8812	356.0125	355.42	355.49	24361.0
2020-05-20 15:34:00	358.41	358.7341	358.38	358.7341	15711.0
2020-05-21 10:49:00	355.145	355.55	354.98	354.992	47021.0
2020-05-21 10:37:00	355.0	355.2	354.22	354.2235	58152.0
2020-05-21 11:46:00	356.6983	356.6983	355.8401	355.898	42077.0
2020-05-20 13:52:00	358.07	358.37	358.0	358.2399	19432.0
2020-05-20 12:18:00	357.0	357.56	356.88	357.53	36660.0
2020-05-21 10:21:00	357.86	358.44	357.65	358.335	29574.0
2020-05-20 12:05:00	358.2	358.3	358.12	358.19	4714.0

```
[Row(mean=357.3354017583024, std=1.8005524099300811)]
('date_time', u'2020-05-20 14:17:00')
Done
```

date_time	close	high	low	open	volume
2020-05-20 12:24:00	356.545	357.08	356.5	357.0	33881.0
2020-05-20 11:32:00	359.77	359.8	359.33	359.39	21405.0
2020-05-20 10:57:00	357.425	357.74	357.38	357.6287	19459.0
2020-05-20 15:44:00	359.93	360.03	359.852	360.025	28452.0
2020-05-20 09:40:00	358.43	359.01	357.4	357.71	68430.0
2020-05-20 12:06:00	358.5998	358.6298	358.21	358.21	14295.0
2020-05-21 12:39:00	356.4545	357.21	356.13	357.1	42701.0
2020-05-20 15:50:00	358.99	359.15	358.92	359.14	24544.0
2020-05-20 12:57:00	356.37	356.5081	356.155	356.155	15535.0
2020-05-20 09:37:00	359.42	359.8065	359.145	359.325	72543.0

- c. Saved the data and results to Cassandra (two streams of data size 1821 and 1945)

```
root@79fa872c61c5: /
```

```
cqlsh:bdr> describe tables;
```

```
nvda_stock_stats_kappa      nvda_stock_data_kappa
nvda_stock_data_stats_lambda nvda_stock_data_lambda
nvda_stock_close_stats_lambda
```

```
reach800@lambda rdd: processstream(rdd))
```

```
cqlsh:bdr> select * from nvda_stock_stats_kappa;
```

date_time	close_price_mean	close_price_std	stream_count
2020-05-20 14:17:00	352.76117	9.94823	1821
2020-05-27 14:07:00	351.87891	10.21136	1945

```
(2 rows)
```

```
cqlsh:bdr>
```

```
ing top 20 rows
```

```
-351.878906495023, std=10.2113638108614951)
```

- d. Access results using Falcon requests

