## HU Extension Assignment 09 E63 Big Data Analytics

### Handed out: 10/28/2017 Due by 4:00 PM EST on Saturday, 11/04/2017

**Problem 1.** Install Kafka on you Linux VM**.** If on your own VM with CentOS7.4 you should be able to install Kafka using yum:

$ sudo yum install kafka

Kafka code is most probably installed in the directory /usr/hdp/current/kafka-broker. Create an environmental variable KAFKA\_HOME pointing to that directory. Place the directory /usr/hdp/current/kafka-broker/bin in the PATH variable in the .bash\_profile file in your home directory. Source .bash\_profile (e.i. issue command $ source .bash\_profile ), so that you can invoke Kafka scripts from any directory. Make sure that Zookeeper server is started. Kafka configuration files reside in the directories: $KAFKA\_HOME/config. Create a topic**.** Demonstrate that provided scripting clientkafka-console-producer.sh receives and displays messages produced by kafka-console-consumer.sh client.

**START ZOOKEEPER**

|  |
| --- |
| [cloudera@quickstart zookeeper]$ sudo service zookeeper-server start  JMX enabled by default  Using config: /etc/zookeeper/conf/zoo.cfg  Starting zookeeper ... already running as process 21220. |

**START KAFKA**

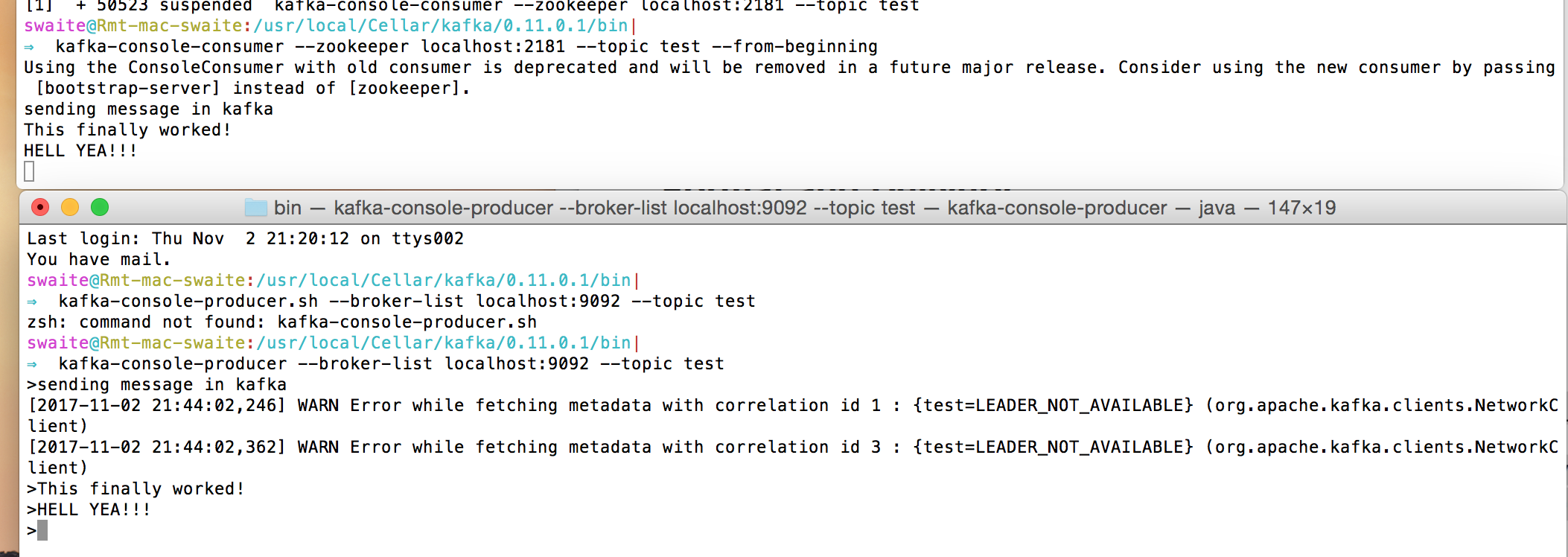
|  |
| --- |
| [cloudera@quickstart tmp]$ sudo service kafka-server start  Starting Kafka Server (kafka-server):                      [  OK  ]  Kafka Server is running                                    [  OK  ] |

**CREATE TOPIC**

|  |
| --- |
| **[cloudera@quickstart config]$ $KAFKA\_HOME/bin/kafka-topics.sh --list --zookeeper localhost:2181**  SLF4J: Class path contains multiple SLF4J bindings.  SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.21.jar!/org/slf4j/impl/StaticLoggerBinder.class]  SLF4J: Found binding in [jar:file:/usr/lib/kafka/libs/slf4j-log4j12-1.7.5.jar!/org/slf4j/impl/StaticLoggerBinder.class]  SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.  SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]  test  test2 |

***I had issues with KAFKA in cloudera VM so I installed it on my Macbook to continue***

|  |
| --- |
| brew install kafka  brew services start zookeeper  brew services start kafka    swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒  brew services start zookeeper  ==> **Successfully started `zookeeper` (label: homebrew.mxcl.zookeeper)**  swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒  brew services start kafka  ==> **Successfully started `kafka` (label: homebrew.mxcl.kafka)** |



**Problem 2**. Make supplied python script kafka\_consumer.py receive messages produced by supplied python script kafka\_producer.py. Modify kafka\_producer.py so that you can pass server name and the port of the Kafka broker and the name of Kafka topic on the command line. Also, modify that script so that it continuously reads your terminal inputs and sends every line to Kafka consumer. Demonstrate that kafka\_consumer.py can read and display messages of modified kafka\_producer.py. Provide working code of modified kafka\_producer.py. Describe to us the process of installing Python packages, if any, you needed for this problem.

**KAFKA\_PRODUCER.py Code**

|  |
| --- |
| **from** kafka **import** KafkaProducer **import** time **import** sys  **if** len(sys.argv) != 4:  **print** "Please pass 3 arguments to script ex: python2.7 kafka\_producer.py localhost 9092 topic1"  exit(1)  producer = KafkaProducer(bootstrap\_servers=str(sys.argv[1]) + ':' + str(sys.argv[2])) kafka\_topic = str(sys.argv[3])  inp = raw\_input("\nPlease type a message or EXIT to exit script:\n") **while** inp != "EXIT":  inp = raw\_input()  producer.send(sys.argv[3], b"" + str(inp)) |

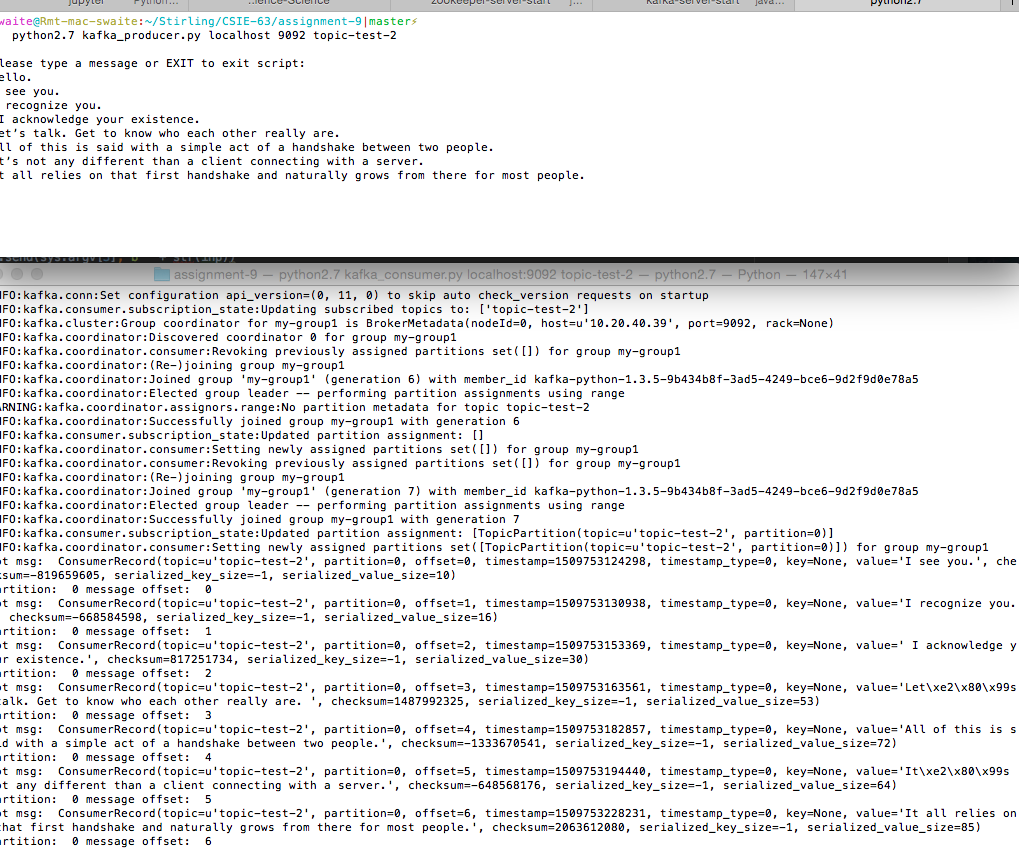
**Command Line Input Kafka Producer**

|  |
| --- |
| swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ python2.7 kafka\_producer.py localhost 9092 topic-test-2  Please type a message or EXIT to exit script:  Hello.  I see you.  I recognize you.  I acknowledge your existence.  Let’s talk. Get to know who each other really are.  All of this is said with a simple act of a handshake between two people.  It’s not any different than a client connecting with a server.  It all relies on that first handshake and naturally grows from there for most people. |

**Command Line Input Kafka Consumer**

|  |
| --- |
| swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ python2.7 kafka\_consumer.py localhost:9092 topic-test-2  Topic is: topic-test-2  Group is: my-group1  INFO:kafka.client:Bootstrapping cluster metadata from [('localhost', 9092, 0)]  INFO:kafka.conn:<BrokerConnection node\_id=bootstrap host=localhost/::1 port=9092>: connecting to ::1:9092  INFO:kafka.client:Bootstrap succeeded: found 1 brokers and 5 topics.  INFO:kafka.conn:<BrokerConnection node\_id=bootstrap host=localhost/::1 port=9092>: Closing connection.  INFO:kafka.conn:<BrokerConnection node\_id=0 host=10.20.40.39/10.20.40.39 port=9092>: connecting to 10.20.40.39:9092  INFO:kafka.conn:Broker version identifed as 0.11.0  INFO:kafka.conn:Set configuration api\_version=(0, 11, 0) to skip auto check\_version requests on startup  INFO:kafka.consumer.subscription\_state:Updating subscribed topics to: ['topic-test-2']  INFO:kafka.cluster:Group coordinator for my-group1 is BrokerMetadata(nodeId=0, host=u'10.20.40.39', port=9092, rack=None)  INFO:kafka.coordinator:Discovered coordinator 0 for group my-group1  INFO:kafka.coordinator.consumer:Revoking previously assigned partitions set([]) for group my-group1  INFO:kafka.coordinator:(Re-)joining group my-group1  INFO:kafka.coordinator:Joined group 'my-group1' (generation 6) with member\_id kafka-python-1.3.5-9b434b8f-3ad5-4249-bce6-9d2f9d0e78a5  INFO:kafka.coordinator:Elected group leader -- performing partition assignments using range  WARNING:kafka.coordinator.assignors.range:No partition metadata for topic topic-test-2  INFO:kafka.coordinator:Successfully joined group my-group1 with generation 6  INFO:kafka.consumer.subscription\_state:Updated partition assignment: []  INFO:kafka.coordinator.consumer:Setting newly assigned partitions set([]) for group my-group1  INFO:kafka.coordinator.consumer:Revoking previously assigned partitions set([]) for group my-group1  INFO:kafka.coordinator:(Re-)joining group my-group1  INFO:kafka.coordinator:Joined group 'my-group1' (generation 7) with member\_id kafka-python-1.3.5-9b434b8f-3ad5-4249-bce6-9d2f9d0e78a5  INFO:kafka.coordinator:Elected group leader -- performing partition assignments using range  INFO:kafka.coordinator:Successfully joined group my-group1 with generation 7  INFO:kafka.consumer.subscription\_state:Updated partition assignment: [TopicPartition(topic=u'topic-test-2', partition=0)]  INFO:kafka.coordinator.consumer:Setting newly assigned partitions set([TopicPartition(topic=u'topic-test-2', partition=0)]) for group my-group1  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=0, timestamp=1509753124298, timestamp\_type=0, key=None, value='I see you.', checksum=-819659605, serialized\_key\_size=-1, serialized\_value\_size=10)  partition: 0 message offset: 0  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=1, timestamp=1509753130938, timestamp\_type=0, key=None, value='I recognize you.', checksum=-668584598, serialized\_key\_size=-1, serialized\_value\_size=16)  partition: 0 message offset: 1  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=2, timestamp=1509753153369, timestamp\_type=0, key=None, value=' I acknowledge your existence.', checksum=817251734, serialized\_key\_size=-1, serialized\_value\_size=30)  partition: 0 message offset: 2  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=3, timestamp=1509753163561, timestamp\_type=0, key=None, value='Let\xe2\x80\x99s talk. Get to know who each other really are. ', checksum=1487992325, serialized\_key\_size=-1, serialized\_value\_size=53)  partition: 0 message offset: 3  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=4, timestamp=1509753182857, timestamp\_type=0, key=None, value='All of this is said with a simple act of a handshake between two people.', checksum=-1333670541, serialized\_key\_size=-1, serialized\_value\_size=72)  partition: 0 message offset: 4  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=5, timestamp=1509753194440, timestamp\_type=0, key=None, value='It\xe2\x80\x99s not any different than a client connecting with a server.', checksum=-648568176, serialized\_key\_size=-1, serialized\_value\_size=64)  partition: 0 message offset: 5  got msg: ConsumerRecord(topic=u'topic-test-2', partition=0, offset=6, timestamp=1509753228231, timestamp\_type=0, key=None, value='It all relies on that first handshake and naturally grows from there for most people.', checksum=2063612080, serialized\_key\_size=-1, serialized\_value\_size=85)  partition: 0 message offset: 6 |

**Screenshot of Message Passing Exchange**



**Problem 3.**  Rather than using splitAndSend.sh bash script to generate traffic towards Spark Streaming engine, write a Kafka Producer which will read orders.txt file and send 1,000 orders to a Kafka topic every second. Write a Kafka consumer that will deliver those batches of orders to Spark Streaming engine. Base your Kafka consumer on provided direct\_word\_count.py script. Let Spark streaming engine count the number of orders different stocks where bought in each batch. Display for us a section of results in your solution. Describe to us the process of installing and invoking Python packages, if any, you needed for this problem.

**python2.7 kafka-orders-producer.py localhost:9092 p3-topic-1**

**CODE**

|  |
| --- |
| **from** kafka **import** KafkaProducer **from** itertools **import** islice **import** time **import** sys  infile\_path = "/Users/swaite/Stirling/CSIE-63/assignment-9/data/orders.txt" chunk\_size = 1000 producer = KafkaProducer(bootstrap\_servers=str(sys.argv[1])) topic = str(sys.argv[2]) # topic = "p3-topic-1" # producer = KafkaProducer(bootstrap\_servers="localhost:9092")  order\_count = 0 **with** open(infile\_path) **as** f:  **while** True:  chunks = list(islice(f, chunk\_size))  order\_count += 1  **print**("message\_count: " + str(order\_count) + "\n" + "chunk\_size: " + str(len(chunks)))  chunked\_message = "".join(chunks)  **print**(chunked\_message)  producer.send(topic, chunked\_message)  **if not** chunks:  **break** time.sleep(1) |

**OUTPUT**

|  |
| --- |
| 2016-03-22 20:25:29,82975,46,KMI,591,101.00,B  2016-03-22 20:25:29,82976,92,YHOO,74,90.00,S  2016-03-22 20:25:29,82977,6,MRO,997,15.00,S  2016-03-22 20:25:29,82978,63,AFFX,243,2.00,S  2016-03-22 20:25:29,82979,11,Z,714,25.00,B  2016-03-22 20:25:29,82980,76,WYNN,899,46.00,S  2016-03-22 20:25:29,82981,27,SBGL,290,88.00,B  2016-03-22 20:25:29,82982,70,KGC,771,16.00,B  2016-03-22 20:25:29,82983,79,OPK,36,35.00,S  2016-03-22 20:25:29,82984,11,INO,140,94.00,S  2016-03-22 20:25:29,82985,63,FB,883,64.00,S  2016-03-22 20:25:29,82986,72,AMD,441,98.00,S  2016-03-22 20:25:29,82987,18,NFLX,659,74.00,B  2016-03-22 20:25:29,82988,96,AG,101,90.00,S  2016-03-22 20:25:29,82989,58,UAL,500,37.00,B  2016-03-22 20:25:29,82990,38,PETX,808,93.00,B  2016-03-22 20:25:29,82991,56,WNC,328,94.00,B  2016-03-22 20:25:29,82992,64,AU,889,26.00,B  2016-03-22 20:25:29,82993,3,AMD,283,48.00,S  2016-03-22 20:25:29,82994,43,BP,834,16.00,B  2016-03-22 20:25:29,82995,93,NEM,606,61.00,S  2016-03-22 20:25:29,82996,83,GG,929,81.00,B  2016-03-22 20:25:29,82997,7,FB,134,11.00,B  2016-03-22 20:25:29,82998,81,BHP,823,68.00,B  2016-03-22 20:25:29,82999,13,AAPL,375,90.00,B  2016-03-22 20:25:29,83000,14,AG,478,62.00,S |

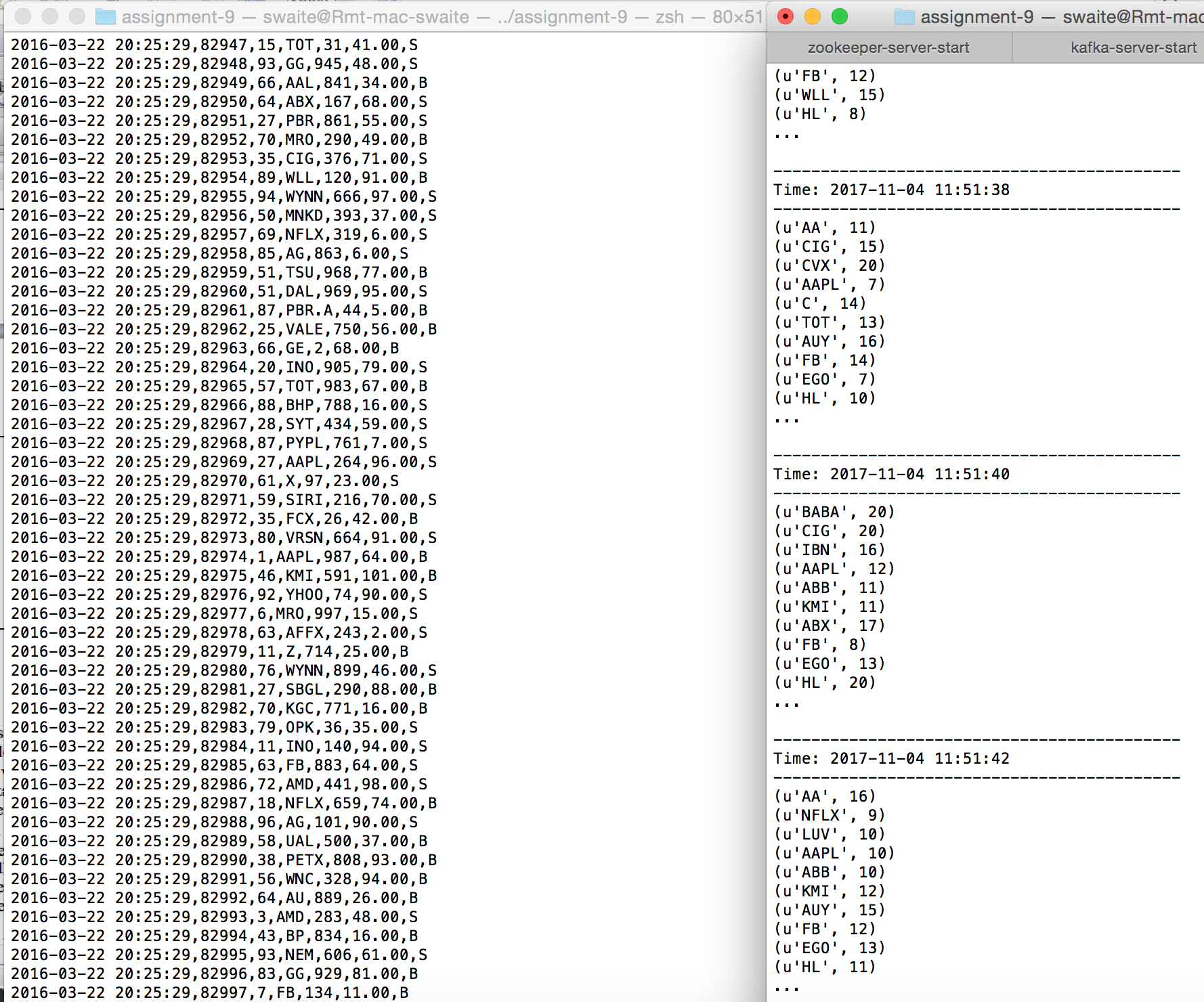
**python2.7 kafka-orders-consumer.py localhost:9092 p3-topic-1**

**CODE**

|  |
| --- |
| **from** pyspark **import** SparkContext **from** pyspark.sql **import** SQLContext, SparkSession, Row **from** pyspark.streaming **import** StreamingContext **from** pyspark.streaming.kafka **import** KafkaUtils **from** itertools **import** islice **from** datetime **import** datetime **from** operator **import** add  # Base your Kafka consumer on provided direct\_word\_count.py script. # Let Spark streaming engine count the number of orders different stocks where bought in each batch. # Display for us a section of results in your solution. # Describe to us the process of installing and invoking Python packages, if any, you needed for this problem.  **def parse\_order**(line):  # Need to split line into an array  l = line.split(",")  **try**:  # Getting some none orders wierdness  # Need to break out  **if** l[6] != u"B" **and** l[6] != u"S":  **raise** Exception("Bad line: ({0})".format(line))  # return parsed line  **return** [{  "order\_date": datetime.strptime(l[0], "%Y-%m-%d %H:%M:%S"),  "order\_id": long(l[1]),  "client\_id": long(l[2]),  "stock\_symbol": l[3],  "amount": int(l[4]),  "stock\_price": float(l[5]),  "order\_type": l[6]  }]  **except** Exception **as** err:  **print**("Bad line: ({0})".format(line))  **return** []  **if** \_\_name\_\_ == "\_\_main\_\_":  **if** len(sys.argv) != 3:  **print**("Usage: direct\_kafka\_wordcount.py <broker\_list> <topic>", file=sys.stderr)  exit(-1)   # ### Initialize streaming context  # Base your Kafka consumer on provided direct\_word\_count.py script.  # 1. Let Spark streaming engine count the number of orders different stocks where bought in each batch.  # 2. Display for us a section of results in your solution.  # 3. Describe to us the process of installing and invoking Python packages, if any, you needed for this problem.   conf = SparkConf() \  .setMaster("local[2]") \  .setAppName("KafkaSparkStreaming") \  .set("spark.executor.memory", "2g") \  .set("spark.driver.extraClassPath", "./libs/spark-streaming-kafka-0-8-assembly\_2.11-2.2.0.jar") \  .set("spark.executor.extraClassPath", "./libs/spark-streaming-kafka-0-8-assembly\_2.11-2.2.0.jar")  sc = SparkContext(conf=conf)  sc.setLogLevel("ERROR")  sqlContext = SQLContext(sc)  spark = SparkSession.builder.appName("spark play").getOrCreate()  ssc = StreamingContext(sc, 2)   brokers, topic = sys.argv[1:]  kvs = KafkaUtils.createDirectStream(ssc, [topic], {"metadata.broker.list": brokers})  lines = kvs.map(**lambda** x: x[1]) \  .flatMap(**lambda** x: [line **for** line **in** x.splitlines()])\  .flatMap(parse\_order)   # https://stackoverflow.com/questions/35582516/spark-counting-distinct-values-by-key  # Filtering buys by order batch chunks of 1000  # Let Spark streaming engine count the number of orders different stocks where bought in each batch.  count\_buys = lines.filter(**lambda** b: b["order\_type"] == u"B")\  .map(**lambda** x: (x["stock\_symbol"], 1))\  .reduceByKey(add)  # printing stocks sold per order  count\_buys.pprint()   ssc.start()  ssc.awaitTermination() |

**OUTPUT**

|  |
| --- |
| -------------------------------------------  Time: 2017-11-04 11:51:38  -------------------------------------------  (u'AA', 11)  (u'CIG', 15)  (u'CVX', 20)  (u'AAPL', 7)  (u'C', 14)  (u'TOT', 13)  (u'AUY', 16)  (u'FB', 14)  (u'EGO', 7)  (u'HL', 10)  ...  -------------------------------------------  Time: 2017-11-04 11:51:40  -------------------------------------------  (u'BABA', 20)  (u'CIG', 20)  (u'IBN', 16)  (u'AAPL', 12)  (u'ABB', 11)  (u'KMI', 11)  (u'ABX', 17)  (u'FB', 8)  (u'EGO', 13)  (u'HL', 20)  ...  -------------------------------------------  Time: 2017-11-04 11:51:42  -------------------------------------------  (u'AA', 16)  (u'NFLX', 9)  (u'LUV', 10)  (u'AAPL', 10)  (u'ABB', 10)  (u'KMI', 12)  (u'AUY', 15)  (u'FB', 12)  (u'EGO', 13)  (u'HL', 11)  ... |



**Problem 4.** Install Cassandra server on your VM. Use Cassandra SQL Client, cqlsh, to create and populate table person. Let every person by described by his or her first and last name, and city where he or she lives. Let every person possess up to three cell phones. Populate your table with three individuals using cqlsh client. Demonstrate that you can select the content of your table person including individuals’ cell phones. Write a simple client in a language of your choice that will populate 3 rows in Casandra’s table person, subsequently update one of those rows, for example change the city where a person lives, and finally retrieve that modify row from Cassandra and write its content to the console. Describe to us the process of installing and invoking Java, Scala or Python packages, if any, you needed for this problem.

**Installing CQL and Cassandra**

|  |
| --- |
| swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ sudo pip install cql  swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ brew install cassandra |

**Starting/Stopping Cassandra**

|  |
| --- |
| swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ brew services start cassandra  swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ brew services stop cassandra |

**Use Cassandra SQL Client, cqlsh, to create and populate table person**.

|  |
| --- |
| swaite@Rmt-mac-swaite:/usr/local/Cellar/cassandra/3.11.1/bin|  ⇒ cd /usr/local/Cellar/cassandra/3.11.1/bin  swaite@Rmt-mac-swaite:/usr/local/Cellar/cassandra/3.11.1/bin|  ⇒ ./cqlsh |

Create keyspace

|  |
| --- |
| cqlsh> create keyspace mykeyspace with replication = { 'class' : 'SimpleStrategy', 'replication\_factor' : 1 };  cqlsh> use mykeyspace; |

**Let every person by described by his or her first and last name, and city where he or she lives.**

|  |
| --- |
| cqlsh:mykeyspace> CREATE TABLE person (  ... person\_id int,  ... first\_name text,  ... last\_name text,  ... city text,  ... cell\_1 text,  ... cell\_2 text,  ... cell\_3 text,  ... PRIMARY KEY (person\_id)); |
| cqlsh:mykeyspace> insert into person (person\_id, first\_name, last\_name, cell\_1, cell\_2, cell\_3, city) VALUES (1, 'Stirling', 'Waite', '234234', '324234', '34235', 'SLC');  insert into person (person\_id, first\_name, last\_name, cell\_1, cell\_2, cell\_3, city) VALUES (2, 'Jack', 'Smith', '48383', '58484', '2933', 'LAX');  insert into person (person\_id, first\_name, last\_name, cell\_1, cell\_2, cell\_3, city) VALUES (3, 'Elliott', 'Robot', '2343', '3244542', '3434234', 'NYC');  cqlsh:mykeyspace> select \* from person  ... ;  **person\_id** | **cell\_1** | **cell\_2** | **cell\_3** | **city** | **first\_name** | **last\_name**  -----------+--------+---------+---------+------+------------+-----------  **1** | **234234** | **324234** | **34235** | **SLC** | **Stirling** | **Waite**  **2** | **48383** | **58484** | **2933** | **LAX** | **Jack** | **Smith**  **3** | **2343** | **3244542** | **3434234** | **NYC** | **Elliott** | **Robot** |

**Write a simple client in a language of your choice that will populate 3 rows in Casandra’s table person, subsequently update one of those rows, for example change the city where a person lives, and finally retrieve that modify row from Cassandra and write its content to the console.**

|  |
| --- |
| swaite@Rmt-mac-swaite:/usr/local/Cellar/cassandra/3.11.1/bin|  ⇒ sudo pip install cassandra-drive |

**CODE**

See p4.py

|  |
| --- |
| **from** cassandra.cluster **import** Cluster cluster = Cluster() session = cluster.connect('mykeyspace') session.execute("INSERT INTO person (person\_id, first\_name, last\_name, city, cell\_1, cell\_2, cell\_3) VALUES (4, 'Tony', 'Tiger', 'PHX', '111', '2222', '3333')") session.execute("INSERT INTO person (person\_id, first\_name, last\_name, city, cell\_1, cell\_2, cell\_3) VALUES (5, 'John', 'Claxton', 'SFO', '444', '5555', '6666')") session.execute("UPDATE person set city = 'NYC' where person\_id=5") result = session.execute("SELECT \* FROM person WHERE person\_id=5") row = result[0] **print** row |

**OUTPUT**

|  |
| --- |
| swaite@Rmt-mac-swaite:~/Stirling/CSIE-63/assignment-9|master⚡  ⇒ python2.7 p4.py  Row(person\_id=5, cell\_1=u'444', cell\_2=u'5555', cell\_3=u'6666', city=u'NYC', first\_name=u'John', last\_name=u'Claxton') |

**TABLE CHECK**

|  |
| --- |
| cqlsh:mykeyspace> select \* from person  ... ;  **person\_id** | **cell\_1** | **cell\_2** | **cell\_3** | **city** | **first\_name** | **last\_name**  -----------+--------+---------+---------+------+------------+-----------  **5** | **444** | **5555** | **6666** | **NYC** | **John** | **Claxton**  **1** | **234234** | **324234** | **34235** | **SLC** | **Stirling** | **Waite**  **2** | **48383** | **58484** | **2933** | **LAX** | **Jack** | **Smith**  **4** | **111** | **2222** | **3333** | **PHX** | **Tony** | **Tiger**  **3** | **2343** | **3244542** | **3434234** | **NYC** | **Elliott** | **Robot** |

Please, describe every step of your work and present all intermediate and final results in a Word document. **Please, copy and past text version of all essential command and snippets of results into the Word document with explanations of the purpose of those commands. We cannot retype text that is in JPG images**. Please, always submit a separate copy of the original, working scripts and/or class files you used. Sometimes we need to run your code and retyping is too costly. Please include in your MS Word document only relevant portions of the console output or output files. Sometime either console output or the result file is too long and including it into the MS Word document makes that document too hard to read. PLEASE DO NOT EMBED files into your MS Word document. For issues and comments visit the class Discussion Board. If you use some other language other than Python in your daily work with NLP, please be free to use that language and a framework of your choice to do this assignment.