## Owen Galvin

## HU Extension Assignment 08 E63 Big Data Analytics

### Handed out: 03/26/2016 Due by 11:30PM EST, 04/01/2016

Please, describe every step of your work and present all intermediate and final results in a Word document. Please, copy past text version of all essential command and snippets of results into the Word document. 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 as separate files. 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. Please, submit to the class drop box. For issues and comments visit the class Discussion Board. You are not obliged to use Java or Eclipse. You are welcome to use any language and any IDE of your choice.

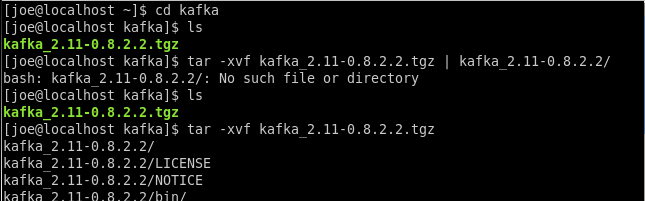
Use most excellent and very detailed notes created by Marina Popova for Section 07 and Kafka and Streaming as you main guide for this assignment.

**Problem 1)** On your Cloudera VM or any other VM you might be using install Kafka. Just in case, install one of the recent Kafka 0.8 versions. Demonstrate that you can create a topic, publish messages to that topic and consume messages sent to that topic. Use Kafka command line interface.

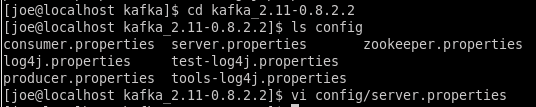
**Solution:**

**I am going back to the CentOS VM I had created from scratch and last used in Assignment 04, since it has Java 8 and the Lab notes should be easier to follow using that as the base. I had done most of my previous work as user = Joe, so that is how I’ve logged in this time.**

**First task is to un-tar the 0.8 kafka .tgz I downloaded, same version from the lab. I chose to make folder named kafka in root of joe, had been going to try in /user dir originally but realized there might be permission issues down the road and wanted to avoid anything like that for now.**



**Move to new directory and confirm contents of /config look good and then open server.properties in the vi editor.**



**I like the idea of keeping things in one place also, so I alter log.dirs to be /home/joe/kafka/kafka-logs.**



**And I create that directory, don’t know if kafka would auto-create it on its own.**



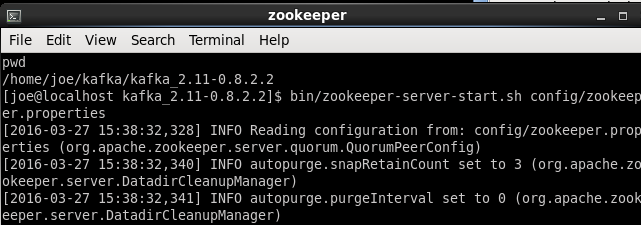
**Below shows similar for the zookeeper.properties file, along with creating zookeeper directory in /home/joe/kafka.**



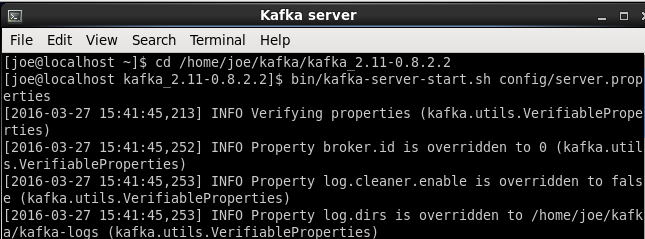
**Altered contents of zookeeper.properties.**



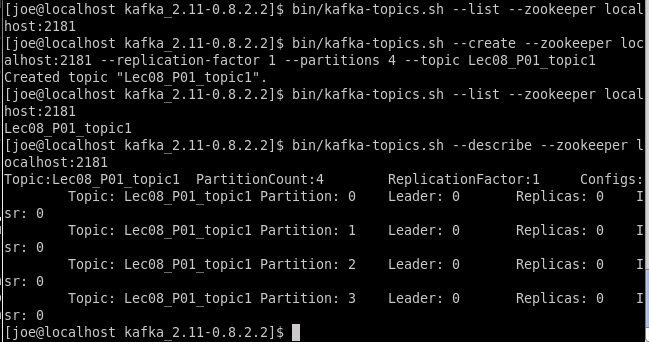
**Going to follow practice of separate “named” terminal windows. In the new zookeeper one, run the startup script, passing in the zookeeper.properties file I had earlier modified.**



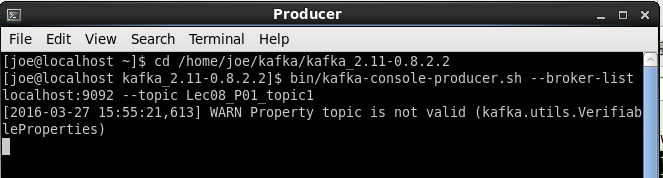
**Similarly for the Kafka server terminal, call the kafka startup script and pass in the server.properties file.**



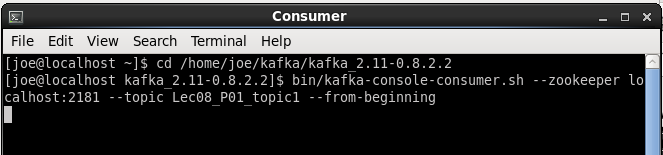
**I run the kafka-topics script with “list” param to confirm that, unsurprisingly, no topics in this brand new kafka service. Then I create a topic named “Lec08\_P01\_topic1” using the syntax from the lab pdf. Feedback indicates it was created and calling kafka-topics.sh again, once with –list and then again with –describe argument, confirms existence of new topic, with 4 partitions as expected.**



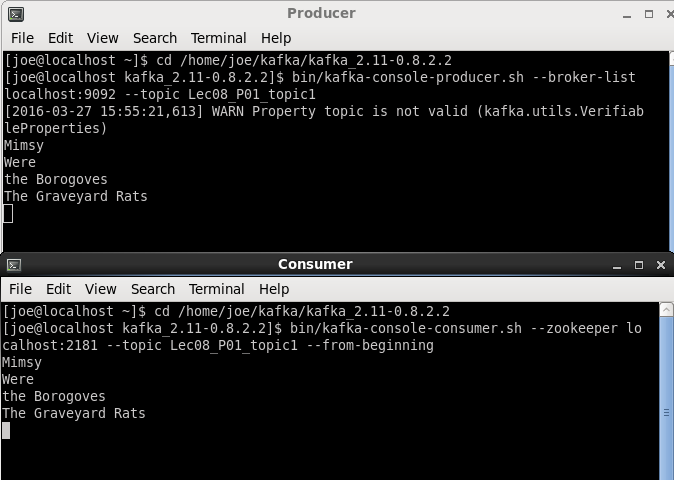
**Another new Terminal session, named Producer, run the kafka-console.producer.sh and pass in my previously created topic to the topic arg. Get the expected warning message and then it sits in a pending state.**



**One more new Terminal window = Consumer and run the consumer.sh script, passing in my topic name to topic argument.**



**Back to the Producer terminal and enter some words and lines of text. Each shows up in Consumer window within a moment or two, indicating successful publish of data from producer -> consumption by consumer.**



|  |
| --- |
| #Commands text etc.  ls  tar -xvf kafka\_2.11-0.8.2.2.tgz | kafka\_2.11-0.8.2.2/  cd kafka\_2.11-0.8.2.2  ls config  vi config/server.properties  mkdir ../kafka-logs  vi config/zookeeper.properties  mkdir ../zookeeper  #zookeeper  cd /home/joe/kafka/kafka\_2.11-0.8.2.2  pwd  bin/zookeeper-server-start.sh config/zookeeper.properties  #Kafka server  cd /home/joe/kafka/kafka\_2.11-0.8.2.2  bin/kafka-server-start.sh config/server.properties  bin/kafka-topics.sh --list --zookeeper localhost:2181  bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 4 --topic Lec08\_P01\_topic1  bin/kafka-topics.sh --describe --zookeeper localhost:2181  #Producer  cd /home/joe/kafka/kafka\_2.11-0.8.2.2  bin/kafka-console-producer.sh --broker-list localhost:9092 --topic Lec08\_P01\_topic1    #Consumer  cd /home/joe/kafka/kafka\_2.11-0.8.2.2  bin/kafka-console-consumer.sh --zookeeper localhost:2181 --topic Lec08\_P01\_topic1 --from-beginning    #Producer  Mimsy  Were  the Borogoves  The Graveyard Rats |

**Problem 2)** Using Java or Python or any other (even scripting) language of your choice construct a producer and a consumer object. Let producer generate one random number between 0 or 1 and 10 every second. Let both producer and consumer run indefinitely or until you kill them. Demonstrate that your consumer is receiving messages by printing both the stream if numbers generated on the producer and the stream of numbers fetched by the consumer. You might find it easier to print to files and examine files afterwards. Once you terminate the exchange, examine Kafka’s log.

Instructions on how to write Java producer and consumer you can find on this URLs:

<https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+Producer+Example>

<https://cwiki.apache.org/confluence/display/KAFKA/0.8.0+SimpleConsumer+Example>

Instructions on how to write Python clients for Kafka you could find on this URL:

<https://cwiki.apache.org/confluence/display/KAFKA/Clients#Clients-Python>

Instructions for Scala could be found here:

<https://cwiki.apache.org/confluence/display/KAFKA/Clients#Clients-ScalaDSL>

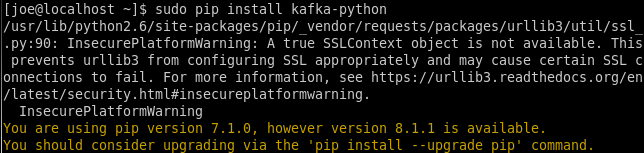
You are welcome to follow any other instructions and use any other programming or scripting language.

**Solution:**

**First task was to install kafka-python since the CentOS has Python 2.6. Unfortunately, this VM also did not have pip, so ran through a quick series of command to get that installed. (Not really a component of the Assignment, so I’ll include the screenshots but no real description – and the commands were mostly a copy & paste from web searches.)**

|  |
| --- |
| **…** |

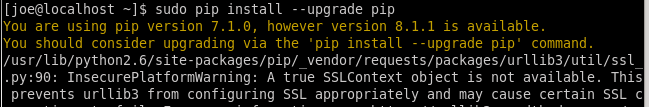
**With pip now available, the install command for kafka-python goes right through.**



**…**



**And then I ran into the section in the lab video describing the pip install, so I saw that it would be best to also run upgrade on it.**



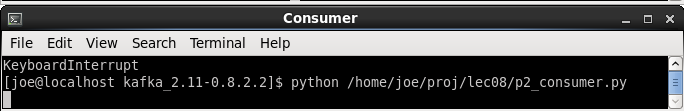
**Next is a code walkthrough of Python my producer file (p2\_producer.py) where I highlight below the changes I made in the sample kafka\_python\_producer.py in order to match Assignment’s goal. The first two highlights are to import items from the random and datetime modules, the former to generate the random numbers and the later to provide timestamps for the messages being sent. There is a simple print statement to indicate transmission has begun and then the remaining code is in an indefinite ‘while True’ loop that will never terminate on its own. Within the loop is a variable named msg that holds the text being sent over the wire, i.e. the random number generated by the randint function. After that is a print statement that displays the current time along with the msg that is being sent out. Send() is called on the KafkaProducer object using a topic name = 'Lec08\_P02\_topic1', with the second parameter set to the actual text (msg variable) being published. Extraneous print statements removed and sleep time has been cut down to 1 second.**

|  |
| --- |
| from kafka import KafkaProducer  from random import randint  from datetime import datetime  import time  producer = KafkaProducer(bootstrap\_servers=*'localhost:9092'*)  print *'Initiating random numbers '*  while True:  msg = str(randint(0,10))  print *'{0} - Random number created by PRODUCER: {1}'*.format(datetime.now().strftime(*'%H:%M:%S:%f'*), msg)  producer.send(*'Lec08\_P02\_topic1'*, msg )    time.sleep(1) |

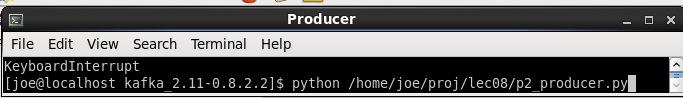
**There was no sample for the consumer side so I’ll describe the contents of p2\_consumer.py entirely. The first two items are imports, beginning with the KafkaConsumer module that corresponds to the KafkaProducer module from the last producer .py file, followed by datetime again. Then an KafkaConsumer object is created, first passing in the name of the topic, which must be a match for the topic name on the producer side. The other param is for bootstrap\_servers definition and matches the ‘localhost:9092’ value used on the producer side. (I also played around with the auto\_offset\_reset param but didn’t use it for the screenshot examples). Then each time a message is received by this consumer object the contents are printed out, surrounded by some identifying text, including the time at which the message was received. As designed this script will simply sit there, waiting indefinitely for any topics that match the described name.**

|  |
| --- |
| from kafka import KafkaConsumer  from datetime import datetime  consumer = KafkaConsumer(*'Lec08\_P02\_topic1'*,  #auto\_offset\_reset='earliest',  bootstrap\_servers=[*'localhost:9092'*])  for message in consumer:  print *'{0} - Random number received by CONSUMER: {1}'*.format(datetime.now().strftime(*'%H:%M:%S:%f'*), message.value) |

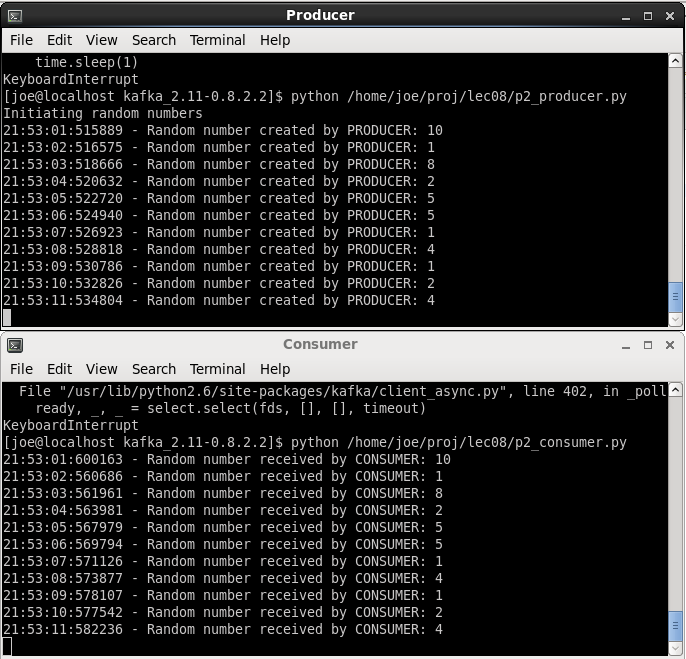
**For the demonstration, I start by running the consumer script, so that it is ready and waiting for the producer to begin producing.**

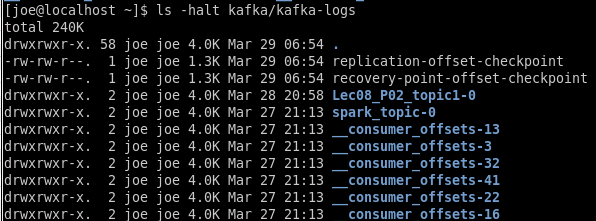


**Then I get ready to kick off the producer script…**



**And below is what they look like once the producer side has started.**

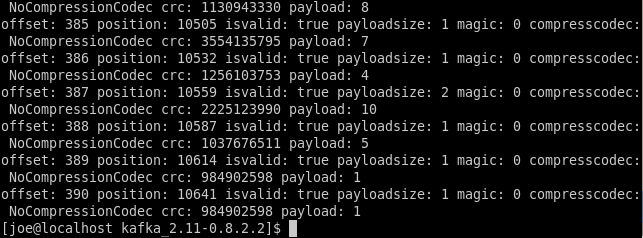


**For the final task I begin to examine the logs by doing an ls on the contents of my kafka-logs directory.**

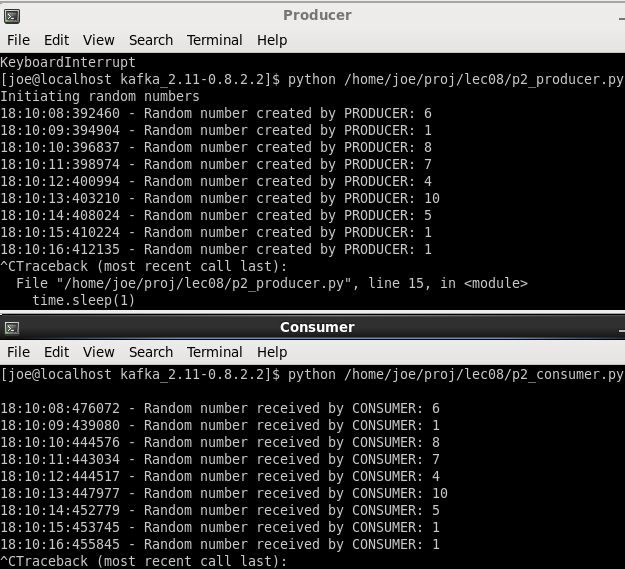
**And list contents of the directory matching the topic I used for Problem 2.**

**Now I can run the DumpLogSegments tool on the .log file above, including the --print-data-log argument to include the value of the payload received.**  


**Which produces quite a bit of output, ending with below. We see offset/message number for each along with its byte position in the log file. All of them appear to have been valid messages. The displayed were all size = 1, except for the fourth from bottom. I had run the scripts again after taking the initial screenshots and If I go back to that matching Consumer window, I can see that size=2 corresponds to the number 10, two characters.**



**Producer/Consumer screenshot matching above log contents.**



|  |
| --- |
| #Command text etc.  #installing pip  cd tmp  wget http://dl.fedoraproject.org/pub/epel/6/x86\_64/epel-release-6-8.noarch.rpm  yum install epel-release-6-8.noarch.rpm  yum -y install python-pip  sudo pip install --upgrade pip  sudo pip install kafka-python  python /home/joe/proj/lec08/p2\_consumer.py  python /home/joe/proj/lec08/p2\_producer.py  cd /home/joe/kafka/kafka\_2.11-0.8.2.2  bin/kafka-run-class.sh kafka.tools.DumpLogSegments --files /home/joe/kafka/kafka-logs/Lec08\_P02\_topic1-0/00000000000000000000.log --print-data-log |

**Problem 3)** Starting from one of the attached Spark Streaming clients DirectKafkaWordCount in Java, Scala or Python write a consumer client that will replace the consumer from the previous problem. However, rather than simply printing every message it receives from the producer, let it print for us every 5 seconds the rolling count of numbers between 1 and 10 it received in the last 30 seconds. You might find it simpler to print to files and then examine those files afterwards. For Java build simple Maven Project with a single Java class and pom.xml file similar to the one provided. Build your projects following the process we used in Assignment 4.

**Solution:**

**Makes sense to begin with a code review. The Producer is almost identical to that from Problem 2, two minor differences where it is now generating a number between 1 and 10 (instead of 0 to 10) and the topic name has been updated for this problem.**

|  |
| --- |
| while True:  msg = str(randint(1,10))  print *b'{0} - Random number created by P3 PRODUCER: {1}'*.format(datetime.now().strftime(*'%H:%M:%S:%f'*), msg)  producer.send(*'Lec08\_P03\_topic1'*, msg ) |

**Moving onto the consumer, will compare my final version vs. the supplied direct\_kafka\_wordcount.py and point out differences, the first of which is a simple import that will allow print to work as a function. My python is 2.6 so the existing print syntax won’t work and in order to get the stderr to print out correctly it seemed this might be the easiest approach.**

|  |
| --- |
| from \_\_future\_\_ import print\_function |

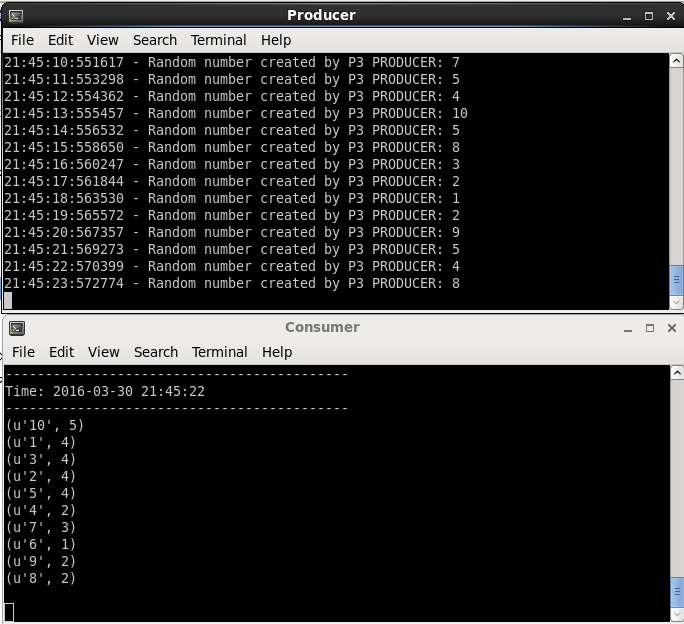
**In the section below, the batch interval is decreased to 1 second – the window length (30 for this problem) and sliding interval (5) must be divisible by this number, so original “2” value won’t work. The preferred ReduceByKeyAndWindow() function used later on requires checkpoint functionality to be in place, so use checkpoint() function on StreamingContext object to do so, passing a string. That string in turn apparently becomes a HDFS directory holding checkpoint data.**

|  |
| --- |
| sc = SparkContext(appName=*"PythonStreamingDirectKafkaWordCount"*)  ssc = StreamingContext(sc, 1) #batch interval to 1 second  ssc.checkpoint(*'lec08checkpoint'*) |

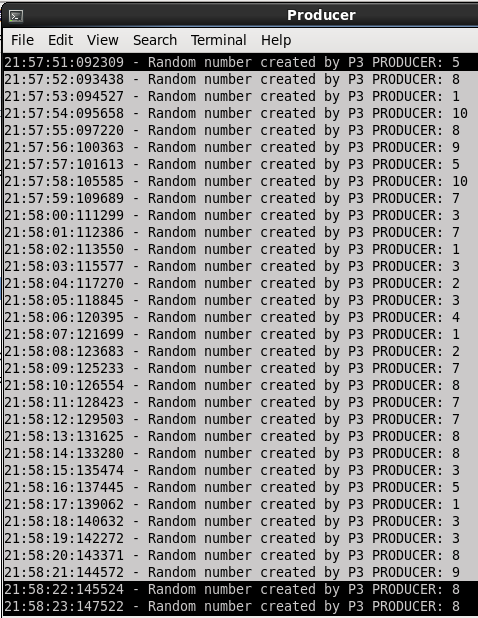
**Remaining changes are below, beginning with a general renaming of some variables to be more consistent with a number count scenario and removal of the flatMap transformation, which doesn’t make sense now since values can be processed directly as they arrive. Then .reduceByKeyAndWindow is used in place of .reduceByKey, in order to implement primary sliding window goal of this assignment. The first lambda arg performs the number count while the second one acts as an inverse reduce function, cleaning out the old data that should not be included in the window. Window length of 30 seconds and slide interval of 5 seconds represent the remaining arguments. While testing with some hard-coded data I noticed the consumer would output lines for numbers that were not present in a given window, displpaying with a count = 0. That would be a rarer event for a full 30 second window with 1 to 10 digits but I figured it made sense to add a filter() transformation at the end that removed any digits with a count = 0.**

|  |
| --- |
| brokers, topic = sys.argv[1:]  kvs = KafkaUtils.createDirectStream(ssc, [topic], {*"metadata.broker.list"*: brokers})  # Please use windowing operations. The window is 30 seconds long and it moves every 5 seconds. Batch interval could remain 1 second.  # Frequency (the count) for each digit in 30 second interval. That number should fluctuate. In one 30 second interval you will have 5 1-s, 7 2-s, 4 3-s, etc.  vals = kvs.map(lambda x: x[1])  counts = (vals.map(lambda num: (num, 1))  .reduceByKeyAndWindow(lambda a, b: a + b, lambda a, b: a - b, 30, 5)  .filter(lambda tup: tup[1] > 0))  counts.pprint()  ssc.start()  ssc.awaitTermination() |

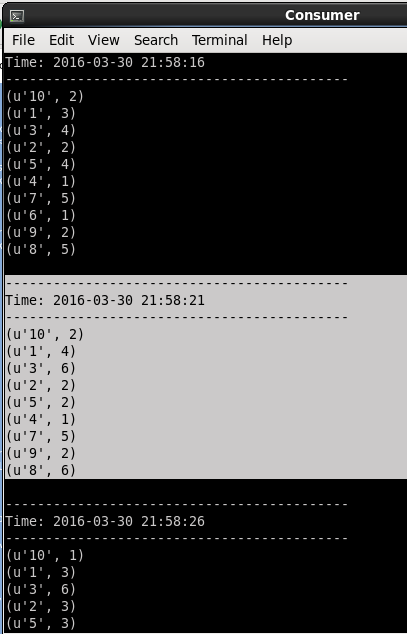
**Below is a screenshot of the above-described producer and p3\_direct\_kafka\_wordcount.py in action.**



**And a more specific one, where I was able to find the matching set of Producer-output numbers & their corresponding accumulated consumer-digit-count rows. The 30 rows on Producer side are highlighted:**



**As is the 30 second window report matching above:**



**My upload includes output from both windows that covers those matching windows.**

|  |
| --- |
| #Problem 3 Terminal window text  python /home/joe/proj/lec08/p3\_producer.py  spark-submit p3\_direct\_kafka\_wordcount.py 'localhost:9092' 'Lec08\_P03\_topic1' |