Girish Rajani
A20503736
CSP-554 Big Data Technologies
Homework 8

Exercise 1) 5 points

1. At Twitter, it was observed that the ETL pipelines were difficult to build and maintain throughout the ETL process, and more generally, latency was encountered throughout this process. Introduced latency meant that data analytics was being conducted using day-old data which impacted the company's ability to make decisions.

2. It was mentioned that at Twitter, a case where the lambda architecture would be appropriate is if they wanted to count tweet impressions. The lambda architecture would allow them to receive not only real-time updates on what users are doing right now but also get historical counts from tweets posted a while ago.

3. Twitter found that two of the limitations of using the lambda architecture were increased complexity (everything must be written twice, once for the batch layer and once for the real-time layer, with the implementation being different in each platform in most cases) and trading off expressiveness (no arbitrary computations) for simplicity (not needing separate batch and realtime implementations).

4. The Kappa architecture introduces stream processing through historical data. This allows for real-time and batch processing using only this technology compared to the lambda architecture, where you would have a different implementation for batch processing and a different one for real-time processing. All the data is handled through this single-stream processing engine in Kappa (a one-size-fits-all solution).

5. One distinguishing feature of Apache Beam is instead of distinguishing between batch and streaming computations, it distinguishes between bounded and unbounded datasets. This means that Apache Beam is an instance of the Kappa architecture, so it uses a single model for both batch and stream processing.

Exercise 2) 5 points

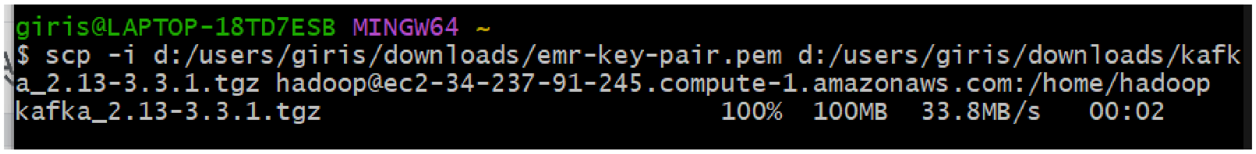
1. The Kappa architecture consists of a stream processing system that performs all the computations compared to the lambda architecture, which needs a separate system, the batch layer. This means that the lambda architecture has a batch layer for processing data periodically (such as once a day) and a speed layer for processing data not processed by the batch layer. In some systems, such as Hadoop-based systems that do not provide a streaming API, it can be difficult to deploy and maintain the lambda architecture and that is where the Kappa architecture comes into play by performing the job of the batch and speed layers within one standalone stream processing system.

2. The advantage of pure streaming versus micro-batch real-time processing systems is that streaming has lower latency than micro-batch, while the drawback is pure streaming has lower throughput than micro-batch.

3. Storm is a distributed stream processing system, thus having low latency, and is compatible with multiple languages. Storm introduces scalability, fault tolerance, and elasticity. In a Storm pipeline, data starts flowing through spout nodes from the streaming layer, and the output (tuple) is passed down to bolt nodes which perform processing, writing data to external storage and passing down data until it reaches the serving layer. Storm has fault tolerance in that by tracking the status of the tuples, if any bolt fails within the pipeline, Storm will replay the tuple.

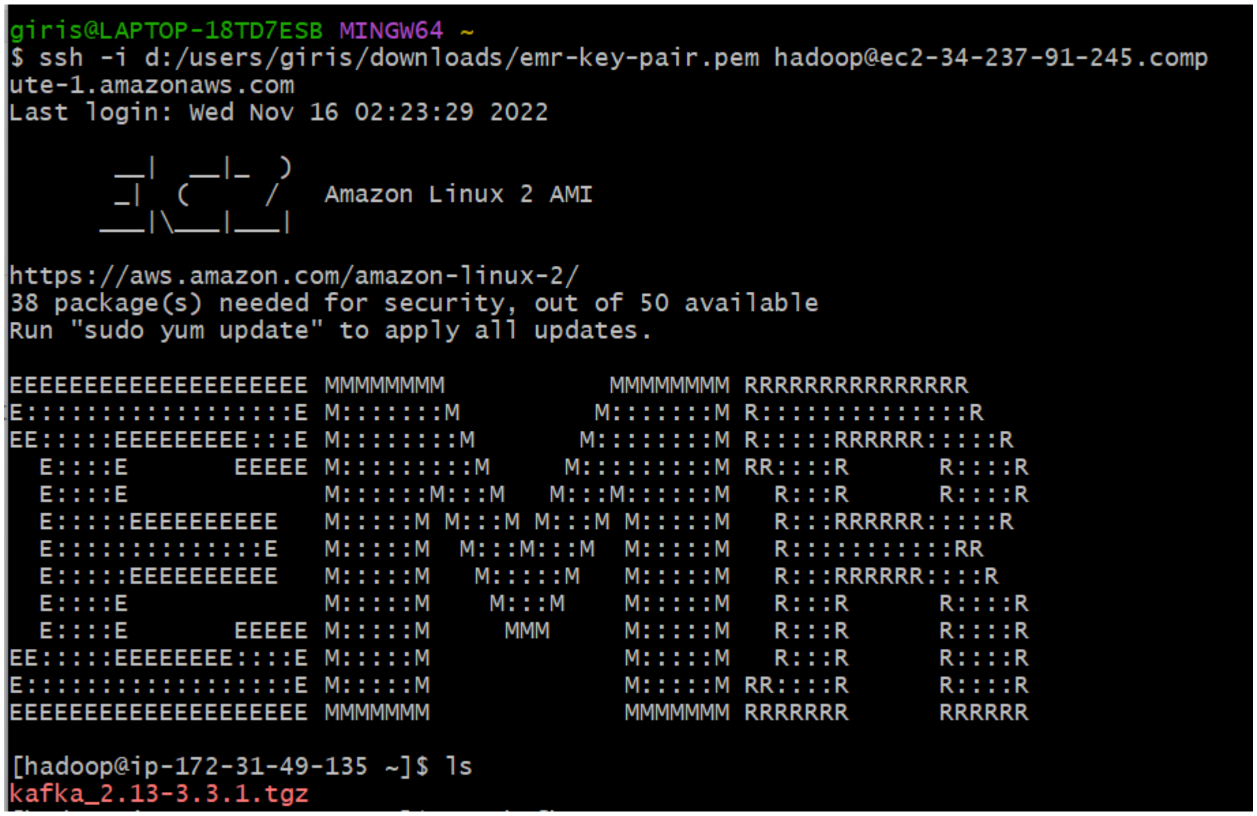
4. Spark streaming shifts the Spark batch processing approach to work on real-time data streams by dividing the data into small batches, transforming them into a sequence of RDDs (DStream), and then processing them through worker nodes to execute the application logic just like it would in batch processing.

Exercise 3) 5 points



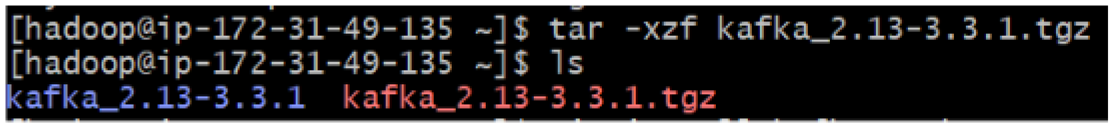
```
giris@LAPTOP-18TD7ESB MINGW64 ~
$ scp -i d:/users/giris/downloads/emr-key-pair.pem d:/users/giris/downloads/kafk
a_2.13-3.3.1.tgz hadoop@ec2-34-237-91-245.compute-1.amazonaws.com:/home/hadoop
kafka_2.13-3.3.1.tgz                          100%  100MB  33.8MB/s   00:02
```

Figure 1 - Using scp command to upload the downloaded Kafka software



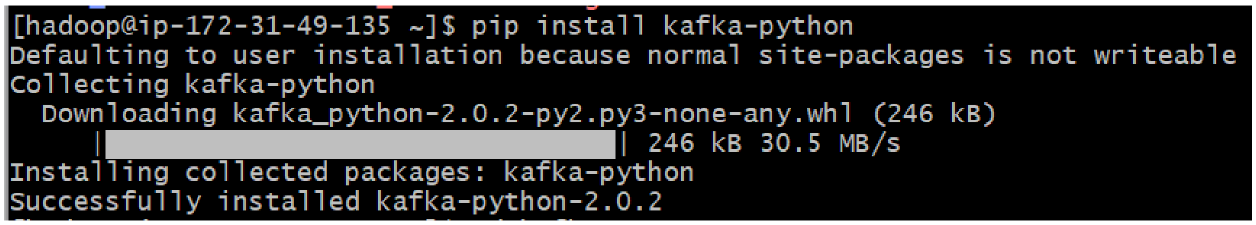
```
giris@LAPTOP-18TD7ESB MINGW64 ~
$ ssh -i d:/users/giris/downloads/emr-key-pair.pem hadoop@ec2-34-237-91-245.comp
ute-1.amazonaws.com
Last login: Wed Nov 16 02:23:29 2022

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
38 package(s) needed for security, out of 50 available
Run "sudo yum update" to apply all updates.

EEEEEEEEEEEEEEEEEEEE MMMMMMMM           MMMMMMMM RRRRRRRRRRRRRRR
E::::::::::::::::::E M:::::::M         M:::::::M R::::::::::::::R
EE:::::EEEEEEEEE:::E M::::::::M       M::::::::M R:::::RRRRRR:::::R
  E::::E       EEEEE M:::::::::M     M:::::::::M RR::::R      R::::R
  E::::E             M::::::M:::M   M:::M::::::M   R:::R      R::::R
  E:::::EEEEEEEEEE   M:::::M M:::M M:::M M:::::M   R:::RRRRRR:::::R
  E::::::::::::::E   M:::::M  M:::M:::M  M:::::M   R:::::::::::RR
  E:::::EEEEEEEEEE   M:::::M   M:::::M   M:::::M   R:::RRRRRR::::R
  E::::E             M:::::M    M:::M    M:::::M   R:::R      R::::R
  E::::E       EEEEE M:::::M     MMM     M:::::M   R:::R      R::::R
EE:::::EEEEEEEE::::E M:::::M             M:::::M   R:::R      R::::R
E::::::::::::::::::E M:::::M             M:::::M RR::::R      R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMM             MMMMMMM RRRRRRR      RRRRRR

[hadoop@ip-172-31-49-135 ~]$ ls
kafka_2.13-3.3.1.tgz
```

Figure 2 - Showing ssh command used to connect to the EMR cluster



```
[hadoop@ip-172-31-49-135 ~]$ tar -xzf kafka_2.13-3.3.1.tgz
[hadoop@ip-172-31-49-135 ~]$ ls
kafka_2.13-3.3.1  kafka_2.13-3.3.1.tgz
```

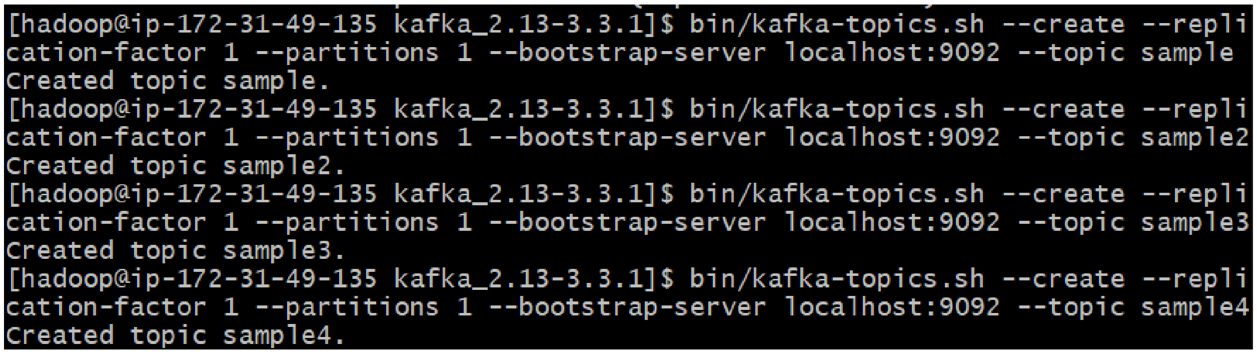
Figure 3 - Executing the tar -xzf kafka_2.13-3.3.1.tgz which creates the new directory (kafka_2.13-3.3.1) holding the kakfa software release.



```
[hadoop@ip-172-31-49-135 ~]$ pip install kafka-python
Defaulting to user installation because normal site-packages is not writeable
Collecting kafka-python
  Downloading kafka_python-2.0.2-py2.py3-none-any.whl (246 kB)
     |████████████████████████████████| 246 kB 30.5 MB/s
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
```

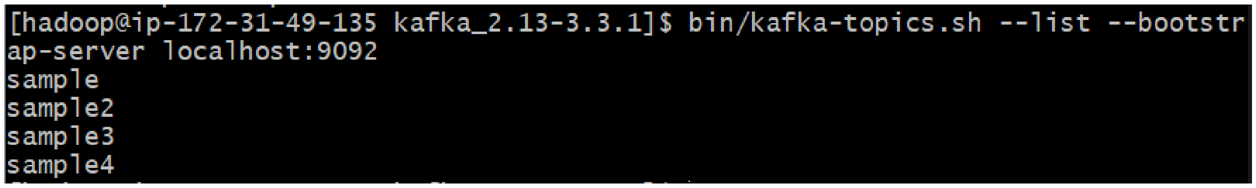
Figure 4 - Showing successful installation of kafka-python

Step E:



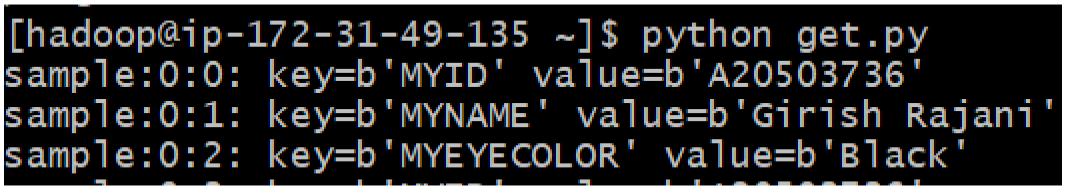
```
[hadoop@ip-172-31-49-135 kafka_2.13-3.3.1]$ bin/kafka-topics.sh --create --repli
cation-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample
Created topic sample.
[hadoop@ip-172-31-49-135 kafka_2.13-3.3.1]$ bin/kafka-topics.sh --create --repli
cation-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample2
Created topic sample2.
[hadoop@ip-172-31-49-135 kafka_2.13-3.3.1]$ bin/kafka-topics.sh --create --repli
cation-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample3
Created topic sample3.
[hadoop@ip-172-31-49-135 kafka_2.13-3.3.1]$ bin/kafka-topics.sh --create --repli
cation-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample4
Created topic sample4.
```

Figure 5 - Creating new kafka topics sample, sample2, sample3, and sample4, respectively.



```
[hadoop@ip-172-31-49-135 kafka_2.13-3.3.1]$ bin/kafka-topics.sh --list --bootstr
ap-server localhost:9092
sample
sample2
sample3
sample4
```

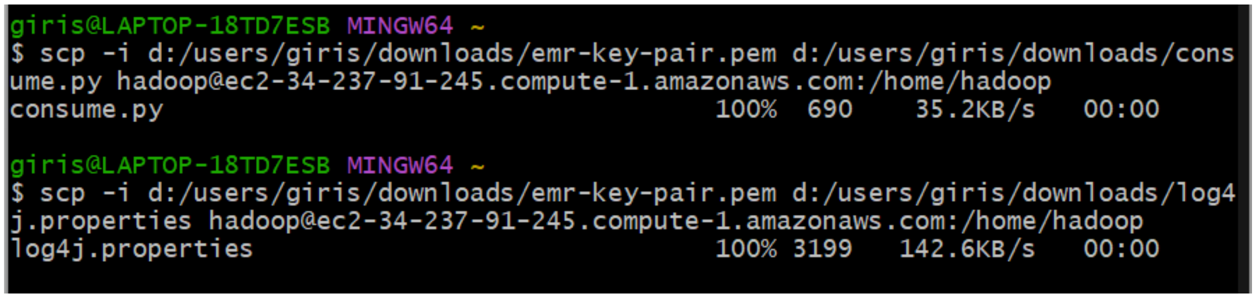
Figure 6 - bin/kafka-topics.sh --list --bootstrap-server localhost:9092 to list the topics



```
[hadoop@ip-172-31-49-135 ~]$ python get.py
sample:0:0: key=b'MYID' value=b'A20503736'
sample:0:1: key=b'MYNAME' value=b'Girish Rajani'
sample:0:2: key=b'MYEYECOLOR' value=b'Black'
```

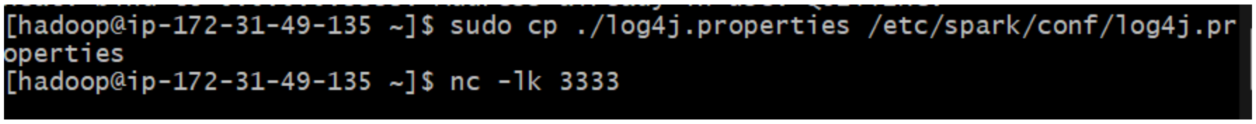
Figure 7 - After executing put.py in the Producer, the above shows the get.py file being executed in Consumer which shows the messages that were put into the kafka topic sample

Exercise 4) 5 points



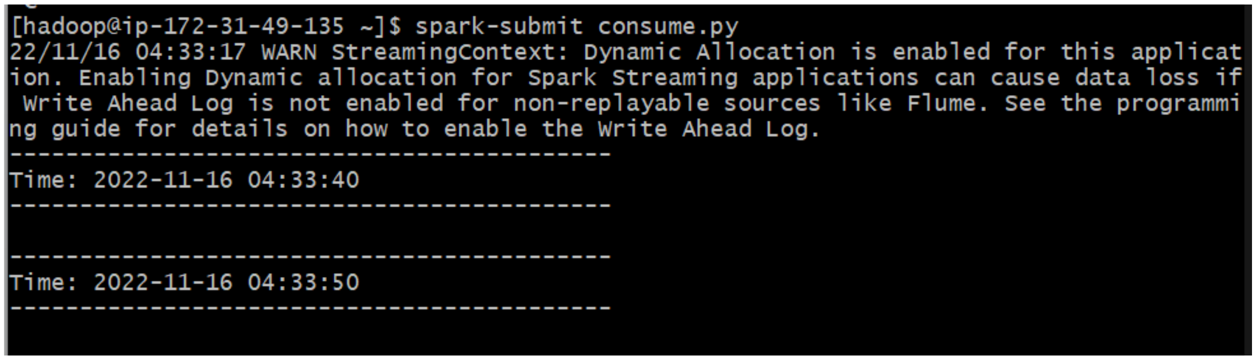
```
giris@LAPTOP-18TD7ESB MINGW64 ~
$ scp -i d:/users/giris/downloads/emr-key-pair.pem d:/users/giris/downloads/cons
ume.py hadoop@ec2-34-237-91-245.compute-1.amazonaws.com:/home/hadoop
consume.py                                    100%  690    35.2KB/s   00:00

giris@LAPTOP-18TD7ESB MINGW64 ~
$ scp -i d:/users/giris/downloads/emr-key-pair.pem d:/users/giris/downloads/log4
j.properties hadoop@ec2-34-237-91-245.compute-1.amazonaws.com:/home/hadoop
log4j.properties                              100% 3199   142.6KB/s   00:00
```

Figure 8 - Using SCP to upload modified consumer.py file and log4j.properties



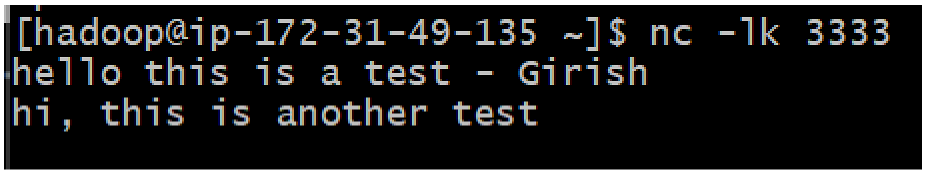
```
[hadoop@ip-172-31-49-135 ~]$ sudo cp ./log4j.properties /etc/spark/conf/log4j.pr
operties
[hadoop@ip-172-31-49-135 ~]$ nc -lk 3333
```

Figure 9 - In a new terminal, after connecting to the master node via ssh, the above command was run to change the logging properties to turn off “INFO” messages to allow easier viewing of the results of the stream processing job



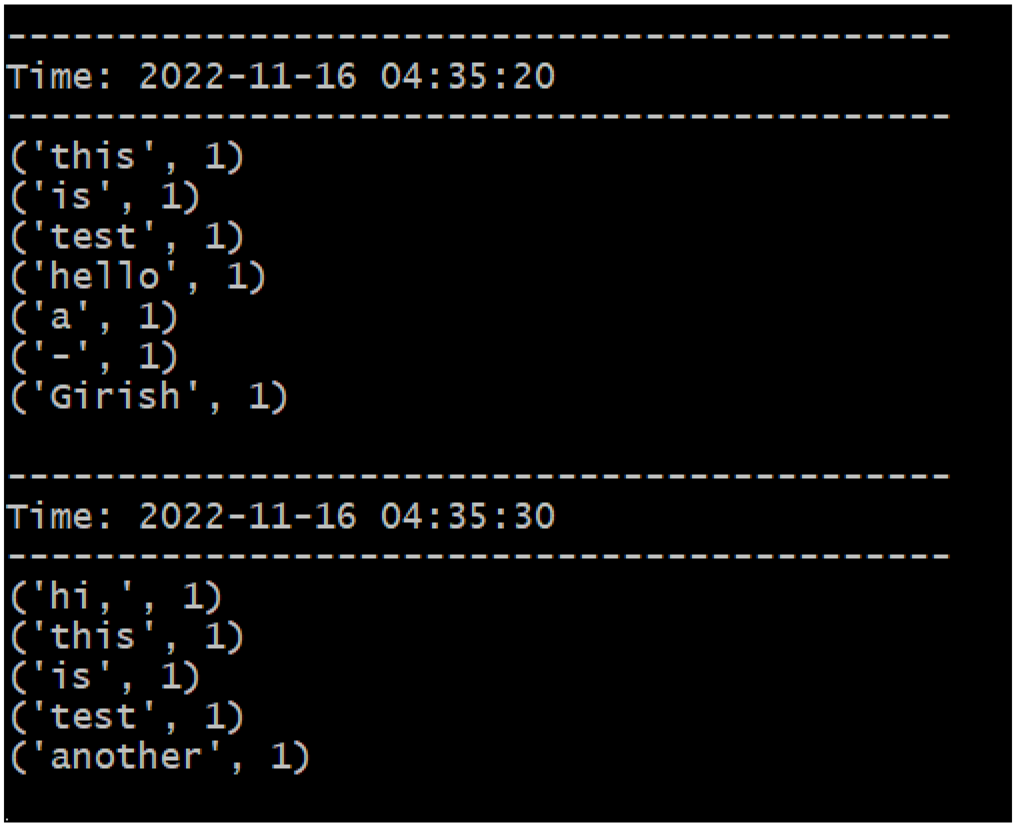
```
[hadoop@ip-172-31-49-135 ~]$ spark-submit consume.py
22/11/16 04:33:17 WARN StreamingContext: Dynamic Allocation is enabled for this applicat
ion. Enabling Dynamic allocation for Spark Streaming applications can cause data loss if
 Write Ahead Log is not enabled for non-replayable sources like Flume. See the programmi
ng guide for details on how to enable the Write Ahead Log.
-------------------------------------------
Time: 2022-11-16 04:33:40
-------------------------------------------

-------------------------------------------
Time: 2022-11-16 04:33:50
-------------------------------------------
```

Figure 10 - In a second terminal, after connecting to the master node via ssh, the above command was run. This will create word count results of sample messages which will be displayed later



```
[hadoop@ip-172-31-49-135 ~]$ nc -lk 3333
hello this is a test - Girish
hi, this is another test
```

Figure 11 - In the original terminal, the above command was used to open a TCP connection on port 3333 and sample lines of text were added.



```
-------------------------------------------
Time: 2022-11-16 04:35:20
-------------------------------------------
('this', 1)
('is', 1)
('test', 1)
('hello', 1)
('a', 1)
('-', 1)
('Girish', 1)

-------------------------------------------
Time: 2022-11-16 04:35:30
-------------------------------------------
('hi,', 1)
('this', 1)
('is', 1)
('test', 1)
('another', 1)
```

Figure 12 - When coming back to the second terminal from figure 10, we can now see the word count results