

# CSP-554 Big Data Technologies

## Bairi Rohith Reddy – Assignment 8

### Exercise 1)

Read the article “The Lambda and the Kappa” found on our blackboard site in the “Articles” section and answer the following questions using between 1-3 sentences each. Note this, article provides a real-world and critical view of the lambda pattern and some related big data processing patterns:

1. (1 point) Extract-transform-load (ETL) is the process of taking transactional business data (think of data collected about the purchases you make at a grocery store) and converting that data into a format more appropriate for reporting or analytic exploration. What problems was encountering with the ETL process at Twitter (and more generally) that impacted data analytics?

**Ans:**

Twitter encountered challenges with the ETL process, which impacted data analytics by limiting access to fresh data for better decision making. One of the main issues was the slow speed of ETL pipelines, which often resulted in data being at least a day old by the time it reached the analytics database. Increasing the frequency of ETL processes was not a viable solution due to the risk of overloading the pipeline.

2. (1 point) What example is mentioned about Twitter of a case where the lambda architecture would be appropriate?

**Ans:**

The example mentioned in the article about Twitter's use case for the lambda architecture is when they wanted to count the number of tweet impressions in real-time while also including historical data. The batch layer provided the complete count of all impressions, while the real-time layer provided an up-to-date count that included only the latest impressions. The results from the real-time layer were transient and could be discarded when the batch computations provided the complete count. The lambda architecture allowed Twitter to process historical data in batches while also processing real-time data in real-time, providing a complete view of the data.

3. (2 points) What did Twitter find were the two of the limitations of using the lambda architecture?

**Ans:**

Twitter found two limitations of using the lambda architecture: complexity and maintenance challenges. The lambda architecture is complex and can be difficult to maintain, which can lead to issues with scalability and reliability. Additionally, the lambda architecture can be challenging to implement, which can result in longer development cycles and higher costs.

**4. (1 point) What is the Kappa architecture?****Ans:**

It is a simplified version of the Lambda architecture that relies solely on stream processing. All the data is processed in real-time as it arrives, eliminating the need for a batch processing layer. It provides a simpler and more streamlined approach to processing large volumes of data, and it can be more cost-effective and easier to maintain than the Lambda architecture.

**5. (1 point) Apache Beam is one framework that implements a kappa architecture. What is one of the distinguishing features of Apache Beam?****Ans:**

Apache Beam is a framework that provides a unified programming model for both batch and stream processing, and it supports multiple programming languages. One of the unique features of Apache Beam is that it explicitly recognizes the difference between event time (when an event occurred) and processing time (when the event is observed in the system), and it offers an abstraction for managing these complexities.

**Exercise 2)**

**Read the article “Real-time stream processing for Big Data” available on the blackboard in the ‘Articles’ section and then answer the following questions:**

**a) (1.25 points) What is the Kappa architecture and how does it differ from the lambda architecture?****Ans:**

The Kappa architecture is a big data processing architecture that is a simplification of the Lambda architecture. In the Kappa architecture, data flows in the form of streams, unlike the Lambda architecture, which uses both batch and stream processing. The central premise behind the Kappa architecture is that it can perform both real-time and batch processing with a single technology stack, making it more streamlined and easier to maintain than the Lambda architecture. One of the biggest advantages of the Kappa architecture is that it allows for only streaming processing, eliminating the need for a batch processing layer

**b) (1.25 points) What are the advantages and drawbacks of pure streaming versus micro-batch real-time processing systems?****Ans:**

Pure streaming systems offer real-time data processing, which allows for immediate results and faster decision-making. They can also handle high volumes of data and are more scalable than micro-batch real-time processing systems. However, pure streaming systems can be more complex and require more resources to maintain. On the other hand, micro-batch real-time processing systems are simpler and more cost-effective to

maintain. They can also handle large volumes of data, but with a slight delay in processing. However, micro-batch real-time processing systems may not be suitable for use cases that require immediate results or real-time decision-making

**c) (1.25 points) In few sentences describe the data processing pipeline in Storm.**

**Ans:**

Apache Storm is a distributed real-time big data processing system designed to process vast amounts of data in a fault-tolerant and horizontal scalable manner. The data processing pipeline in Storm consists of a topology, which is a directed acyclic graph of spouts and bolts. Spouts are responsible for ingesting data into the pipeline, while bolts are responsible for processing the data. The data is processed in real-time as it flows through the pipeline, and the results can be stored in a variety of data stores, including Hadoop, HBase, and Cassandra.

**d) (1.25 points) How does Spark streaming shift the Spark batch processing approach to work on real-time data streams?**

**Ans:**

Spark streaming provides a high level abstraction called **Discretized stream** or **DStream**. Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

DStream is a continuous stream of data or a sequence of RDD's.  
The flow is as follows--

DStream ---> RDD'S ---> Messages (Entities)

Each RDD can have lot of messages.

We can perform operations/transformations at Dstream level.

**Exercise 3)**

Step A – Start an EMR cluster.

Step B – Copy the Kafka software to the EMR master node.

- `scp -i /c/Users/rohit/OneDrive/Desktop/all_files/BIGDATA/emr-key-pair.cer c/Users/rohit/OneDrive/Desktop/all_files/BIGDATA/Homework-Assignments/Homework-8/kafka-3.4.0-src.tgz hadoop@ec2-54-160-83-245.compute-1.amazonaws.com:/home/hadoop`

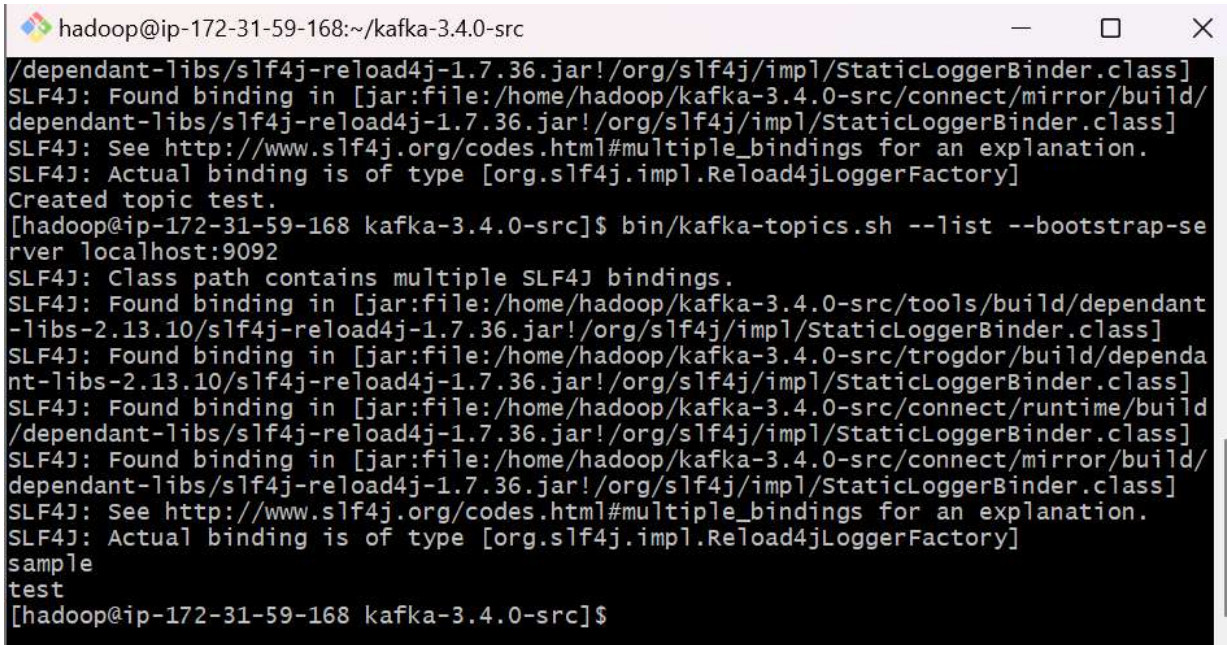
Step C – Install the Kafka software and start it

- `ssh -i /c/Users/rohit/OneDrive/Desktop/all_files/BIGDATA/emr-key-pair.cer hadoop@ec2-54-160-83-245.compute-1.amazonaws.com`
- `tar -xzf kafka-3.4.0-src.tgz`
- `pip install kafka-python`
- `cd kafka-3.4.0-src`

- `bin/zookeeper-server-start.sh config/zookeeper.properties &`
- `bin/kafka-server-start.sh config/server.properties &`

Step D – Prepare to run Kafka producers and consumers

- `cd kafka-3.4.0-src`
- `bin/kafka-topics.sh --create --replication-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic sample`
- `bin/kafka-topics.sh --create --replication-factor 1 --partitions 1 --bootstrap-server localhost:9092 --topic test`
- `bin/kafka-topics.sh --list --bootstrap-server localhost:9092`



```
hadoop@ip-172-31-59-168:~/kafka-3.4.0-src
/dependant-libs/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/connect/mirror/build/dependant-libs/slf4j-reload4j-1.7.36.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.Reload4jLoggerFactory]
Created topic test.
[hadoop@ip-172-31-59-168 kafka-3.4.0-src]$ bin/kafka-topics.sh --list --bootstrap-server localhost:9092
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/tools/build/dependant-libs-2.13.10/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/trogdor/build/dependant-libs-2.13.10/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/connect/runtime/build/dependant-libs/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/connect/mirror/build/dependant-libs/slf4j-reload4j-1.7.36.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.Reload4jLoggerFactory]
sample
test
[hadoop@ip-172-31-59-168 kafka-3.4.0-src]$
```

#### a) put.py

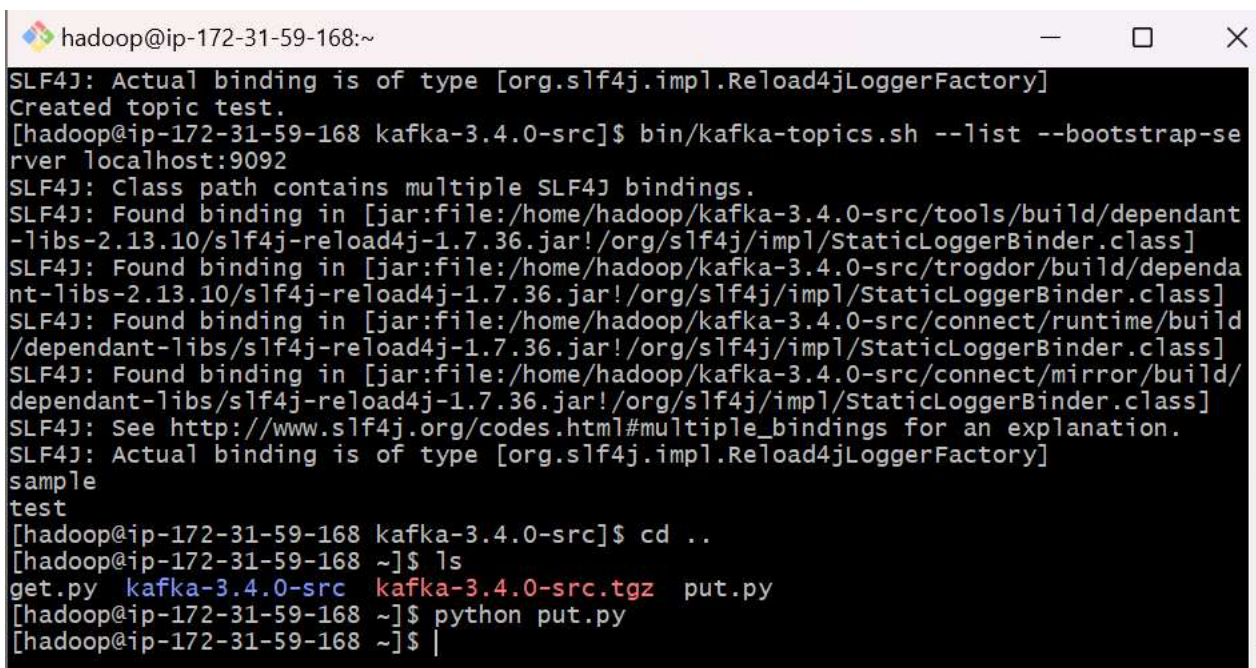
```
from json import dumps
from kafka import KafkaProducer
producer = KafkaProducer(bootstrap_servers=['localhost:9092'])
p = {
    "MYID": "A20526972",
    "MYNAME": "Rohith Reddy Bairi",
    "MYEYECOLOR": "Black"
}
for k in p.keys():
    bkeys = bytes(k, 'utf-8')
    bvalue = bytes(p[k], 'utf-8')
    producer.send('sample', value = bvalue, key=bkeys)

producer.close()
```

```

1  from json import dumps
2  from kafka import KafkaProducer
3  producer = KafkaProducer(bootstrap_servers=['localhost:9092'])
4  p = {
5      "MYID": "A20526972",
6      "MYNAME": "Rohith Reddy Bairi",
7      "MYEYECOLOR": "Black"
8  }
9  for k in p.keys():
10     bkeys = bytes(k, 'utf-8')
11     bvalue=bytes(p[k], 'utf-8')
12     producer.send('sample',value = bvalue,key=bkeys)
13
14  producer.close()
15

```



A terminal window titled 'hadoop@ip-172-31-59-168:~' showing the execution of Kafka-related commands. It displays SLF4J warnings about multiple bindings, the creation of a 'test' topic, and the execution of 'bin/kafka-topics.sh --list --bootstrap-server localhost:9092'. It then shows the execution of 'python put.py' which produces output to the terminal.

```

hadoop@ip-172-31-59-168:~
SLF4J: Actual binding is of type [org.slf4j.impl.Reload4jLoggerFactory]
Created topic test.
[hadoop@ip-172-31-59-168 kafka-3.4.0-src]$ bin/kafka-topics.sh --list --bootstrap-se
rver localhost:9092
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/tools/build/dependant
-libs-2.13.10/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/trogdor/build/dependa
nt-libs-2.13.10/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/connect/runtime/build
/dependant-libs/slf4j-reload4j-1.7.36.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/home/hadoop/kafka-3.4.0-src/connect/mirror/build/
dependant-libs/slf4j-reload4j-1.7.36.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.Reload4jLoggerFactory]
sample
test
[hadoop@ip-172-31-59-168 kafka-3.4.0-src]$ cd ..
[hadoop@ip-172-31-59-168 ~]$ ls
get.py  kafka-3.4.0-src  kafka-3.4.0-src.tgz  put.py
[hadoop@ip-172-31-59-168 ~]$ python put.py
[hadoop@ip-172-31-59-168 ~]$ |

```

## b) get.py

```

from kafka import KafkaConsumer
consumer = KafkaConsumer(
    'sample',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    consumer_timeout_ms=10000,
    group_id='my-group')
for message in consumer:
    print("key=%s value=%s" % (message.key.decode('utf-
8'),message.value.decode('utf8')))
consumer.close()

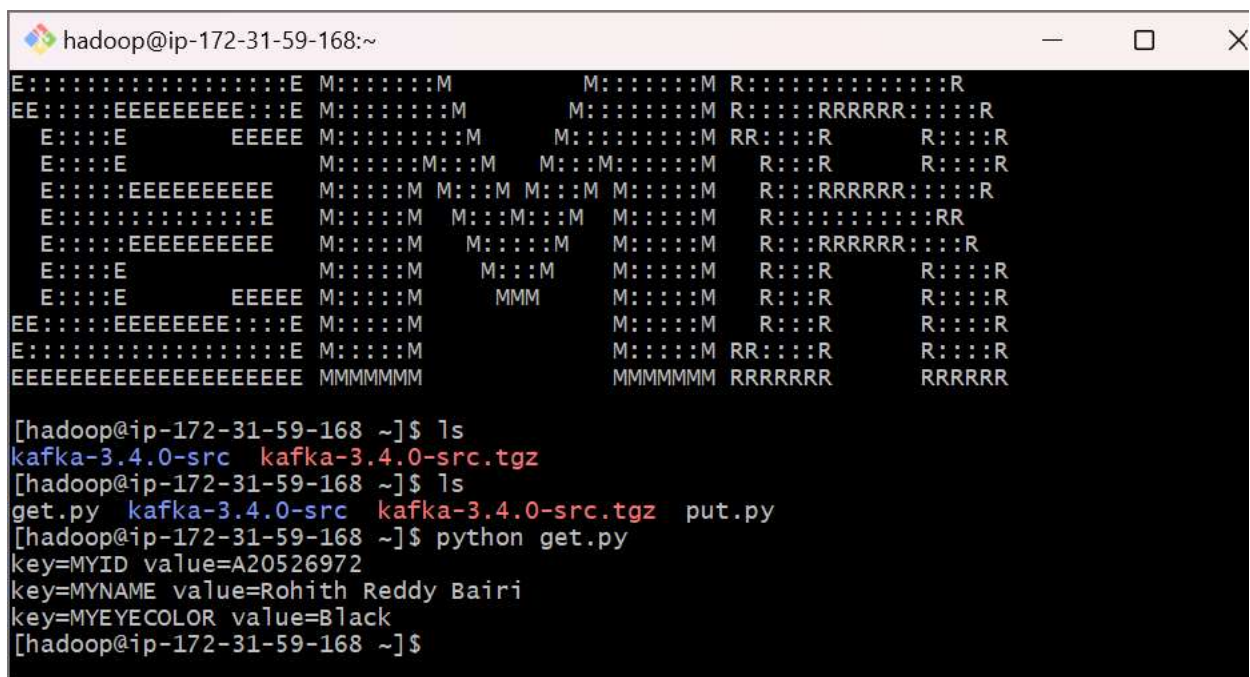
```



```

from kafka import KafkaConsumer
consumer = KafkaConsumer(
    'sample',
    bootstrap_servers=['localhost:9092'],
    auto_offset_reset='earliest',
    consumer_timeout_ms=10000,
    group_id='my-group')
for message in consumer:
    print("key=%s value=%s" % (message.key.decode('utf-8'),message.value.decode('utf8')))
consumer.close()

```



```

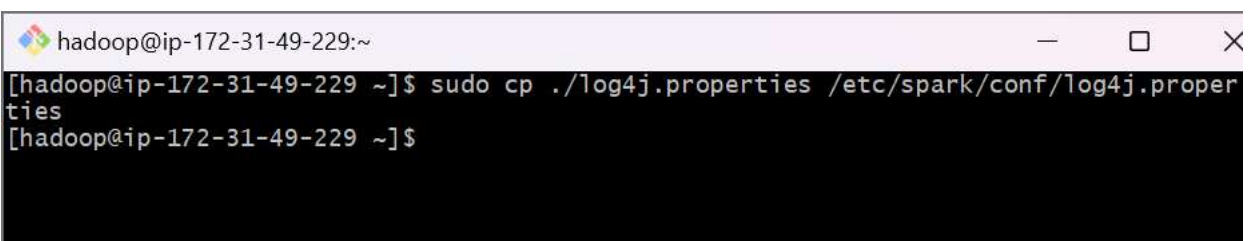
hadoop@ip-172-31-59-168:~
E::::::::::::::::::::E M::::::::M M::::::::M R::::::::R
EE::::::::::::::::::E M::::::::M M::::::::M R::::::::RRRRRR::::R
 E::::E EEEEE M::::::::M M::::::::M RR::::R R::::R
 E::::E EEEEE M::::::::M M::::::::M R::::R R::::R
 E::::EEEEEEEEEE M::::M M::M M::M M::::M R::RRRRRR::::R
 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::::::::::::::::::E M::::M M::::M R::R R::::R
E::::::::::::E M::::M M::::M RR::::R R::::R
EEEEEEEEEEEEEEEEEEEE MMMMMMMM MMMMMMMM RRRRRRR RRRRRR

[hadoop@ip-172-31-59-168 ~]$ ls
kafka-3.4.0-src kafka-3.4.0-src.tgz
[hadoop@ip-172-31-59-168 ~]$ ls
get.py kafka-3.4.0-src kafka-3.4.0-src.tgz put.py
[hadoop@ip-172-31-59-168 ~]$ python get.py
key=MYID value=A20526972
key=MYNAME value=Rohith Reddy Bairi
key=MYEYECOLOR value=Black
[hadoop@ip-172-31-59-168 ~]$

```

#### Exercise 4)

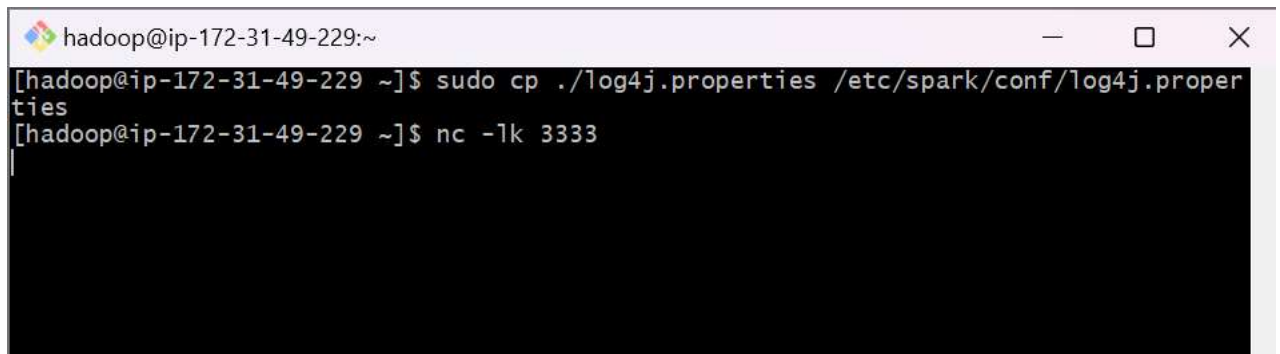
1. scp the file log4j.properties to your EMR cluster master node.
2. In the EC2-1 window enter the following command to open a TCP (socket) connection on port 3333  
**nc -lk 3333**



```

hadoop@ip-172-31-49-229:~
[hadoop@ip-172-31-49-229 ~]$ sudo cp ./log4j.properties /etc/spark/conf/log4j.properties
[hadoop@ip-172-31-49-229 ~]$

```

A terminal window titled 'hadoop@ip-172-31-49-229:~' with standard window controls. It shows two commands being executed: 'sudo cp ./log4j.properties /etc/spark/conf/log4j.properties' and 'nc -lk 3333'.

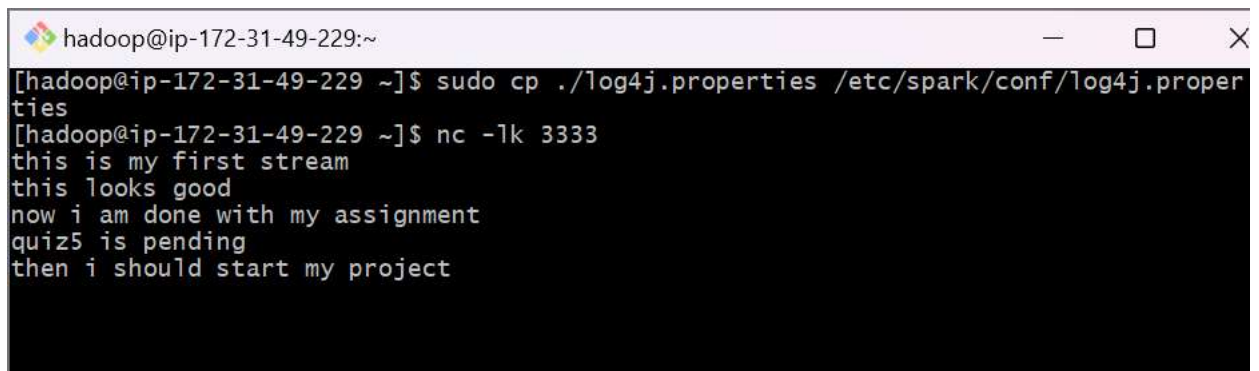
```
hadoop@ip-172-31-49-229:~  
[hadoop@ip-172-31-49-229 ~]$ sudo cp ./log4j.properties /etc/spark/conf/log4j.properties  
[hadoop@ip-172-31-49-229 ~]$ nc -lk 3333
```

3. In the EC2-2 window enter the following command:

**spark-submit consume.py**

4. Now in the EC2-1 window enter one or more lines of text and press Enter/Return after each one including the last. You should see the word count results scroll by in the EC2-2 window

## EC2-1 window

A terminal window titled 'hadoop@ip-172-31-49-229:~' with standard window controls. It shows the same two commands as the EC2-2 window, followed by several lines of text input: 'this is my first stream', 'this looks good', 'now i am done with my assignment', 'quiz5 is pending', and 'then i should start my project'.

```
hadoop@ip-172-31-49-229:~  
[hadoop@ip-172-31-49-229 ~]$ sudo cp ./log4j.properties /etc/spark/conf/log4j.properties  
[hadoop@ip-172-31-49-229 ~]$ nc -lk 3333  
this is my first stream  
this looks good  
now i am done with my assignment  
quiz5 is pending  
then i should start my project
```

## EC2-2 window

 hadoop@ip-172-31-49-229:~

-----  
Time: 2023-04-23 20:45:00  
-----

('this', 1)  
( 'is', 1)  
( 'my', 1)  
( 'first', 1)  
( 'stream', 1)

-----  
Time: 2023-04-23 20:45:10  
-----

('this', 1)  
( 'looks', 1)  
( 'good', 1)

-----  
Time: 2023-04-23 20:45:20  
-----

('now', 1)  
( 'i', 1)  
( 'am', 1)  
( 'done', 1)  
( 'with', 1)  
( 'my', 1)  
( 'assignment', 1)

-----  
Time: 2023-04-23 20:45:30  
-----

('quiz5', 1)  
( 'is', 1)  
( 'pending', 1)

-----  
Time: 2023-04-23 20:45:40  
-----

('i', 1)  
( 'start', 1)  
( 'project', 1)  
( 'then', 1)  
( 'should', 1)  
( 'my', 1)

-----  
Time: 2023-04-23 20:45:50  
-----