Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

BDI

Name: Kartik Jolapara **Branch:** Computer Engineering

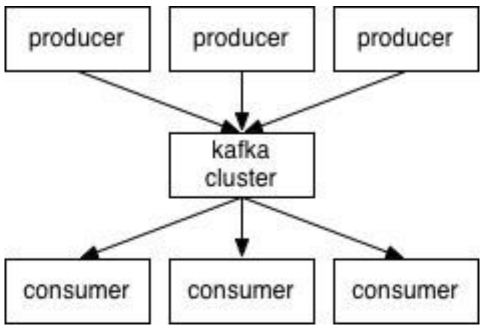
SAP ID: 60004200107 **Batch:** B1

EXPERIMENT NO. 9
Read streaming data using Kafka

AIM: Read streaming data using Kafka.

THEORY:
What is Apache Kafka?

Apache Kafka is a software platform which is based on a distributed streaming process. It is a publish-subscribe messaging system which let exchanging of data between applications, servers, and processors as well. Apache Kafka was originally developed by LinkedIn, and later it was donated to the Apache Software Foundation. Currently, it is maintained by Confluent under Apache Software Foundation. Apache Kafka has resolved the lethargic trouble of data communication between a sender and a receiver.



What is a messaging system?

A messaging system is a simple exchange of messages between two or more persons, devices, etc. A publish-subscribe messaging system allows a sender to send/write the message and a receiver to read that message. In Apache Kafka, a sender is known as a producer who publishes messages, and a receiver is known as a consumer who consumes that message by subscribing it.

What is Streaming process?

A streaming process is the processing of data in parallelly connected systems. This process allows different applications to limit the parallel execution of the data, where one record executes without waiting for the output of the previous record. Therefore, a distributed streaming platform enables the user to simplify the task of the streaming process and parallel execution. Therefore, a streaming platform in Kafka has the following key capabilities:

1. As soon as the streams of records occur, it processes it.
2. It works similar to an enterprise messaging system where it publishes and subscribes streams of records.
3. It stores the streams of records in a fault-tolerant durable way.

To learn and understand Apache Kafka, the aspirants should know the following four core APIs:

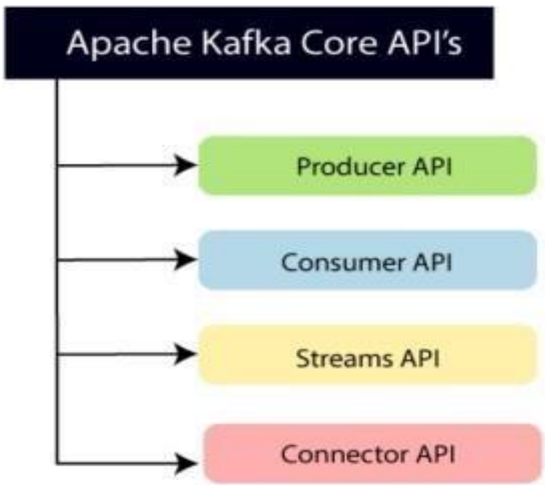
Producer API: This API allows/permits an application to publish streams of records to one or more topics.

Consumer API: This API allows an application to subscribe one or more topics and process the stream of records produced to them.

Streams API: This API allows an application to effectively transform the input streams to the output streams. It permits an application to act as a stream processor which consumes an input stream from one or more topics, and produce an output stream to one or more output topics.

Connector API: This API executes the reusable producer and consumer APIs with the existing data systems or applications.

Messages are published to topics by Kafka Producers. Those who subscribe to them, on the other hand, are called Kafka Consumers. Communication to and from Apache Kafka from Producers and Consumers is through a language agnostic TCP protocol, which is high-performing and simple. It assumes the responsibility for facilitating communications involving servers and clients.



Topics and Logs

A topic is considered a key abstraction in Kafka. A topic can be considered a feed name or category where the messages will be published. Each topic is further subdivided into partitions. Partitions split data across different nodes by Kafka brokers. In each of the messages within the partition, there is an ID number assigned, which is technically known as the offset.

Kafka Topic Partitions

All of the messages published can be retained in the cluster, regardless if they have been consumed or not. For instance, if the configuration states that it will be available only for one week, then it can be retained within such period only. Once the period has lapsed, it will be automatically deleted, which, in turn, will provide the system with additional space. On a per consumer basis, only the meta-data, which is also technically known as the offset, is going to be retained. The consumers will have complete control of this retention. It is often consumed linearly, but the user can often go back and reprocess, basically because he or she is in control. Consumers of Kafka are lightweight, cheap, and they do not have a significant impact on the way the clusters will perform. This differs from more traditional message systems. Every consumer has the ability to maintain their own offset, and hence, their actions will not affect other consumers.

Kafka Partitions

Partitions are leveraged for two purposes. The first is to adjust the size of the topic to make it fit on a single node. The second purpose of partition is for parallelism or performance tuning. It makes it possible for one consumer to simultaneously browse messages in concurrent threads.

Distribution

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Every partition is replicated for fault tolerance. In every partition, there is a single server, which is labeled as the "Leader". The leader is responsible for the handling of the requests for read-write. There will also be "Followers", which can be zero or more nodes in the Kafka cluster. They will be responsible for replicating the actions undertaken by the leader. In the case of the failure of the leader, one of the followers will immediately assume the leadership position.

So, Why Kafka? When?

When compared to the conventional messaging system, one of the benefits of Kafka is that it has ordering guarantees. In the case of a traditional queue, messages are kept on the server based on the order at which they are kept. Then messages are pushed out based on the order of their storage and there is no specific timing requirement in their transmission. This means that their arrival to different consumers could be random. Hold the phone. Did you just write "could be random"? Think about that for 5 seconds. That may be fine in some use cases, but not others. Kafka is often deployed as the foundational element in Streaming Architectures because it can facilitate loose coupling between various components in architectures such as databases, microservices, Hadoop, data warehouses, distributed files systems, search applications, etc.

Guarantees

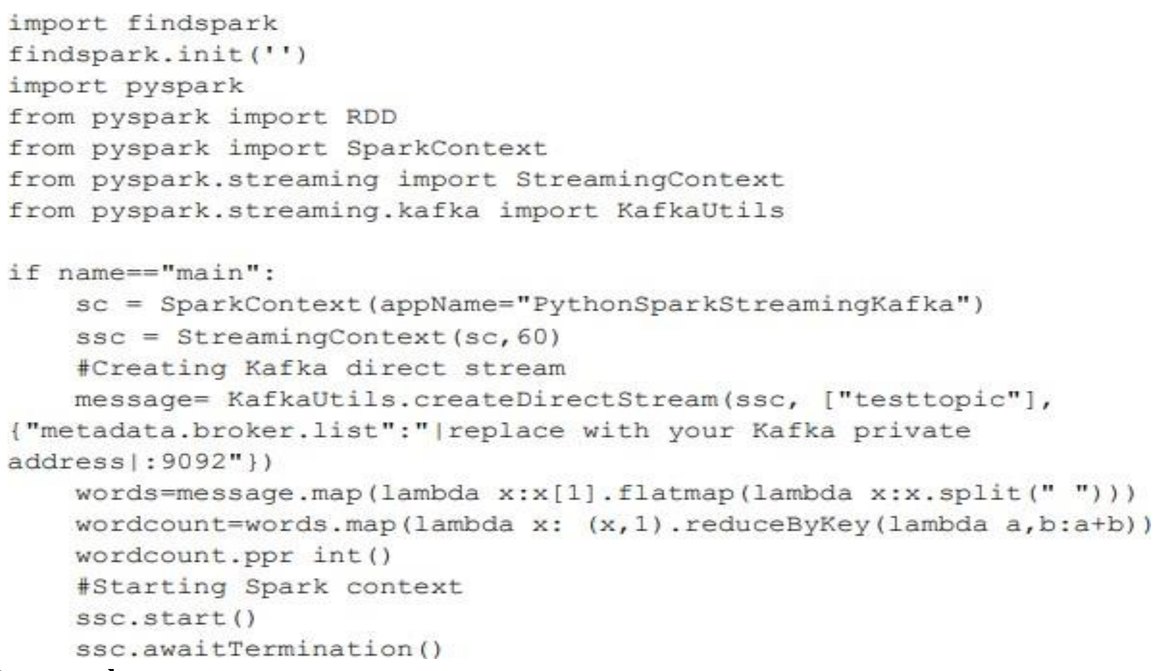
In a Kafka messaging system, there are essentially three guarantees:

1. A message sent by a producer to a topic partition is appended to the Kafka log based on the order sent.

2. Consumers see messages in the order of their log storage.

3. For topics with replication factor N, Kafka will tolerate N-1 number of failures without losing the messages previously committed to the log.

CODE



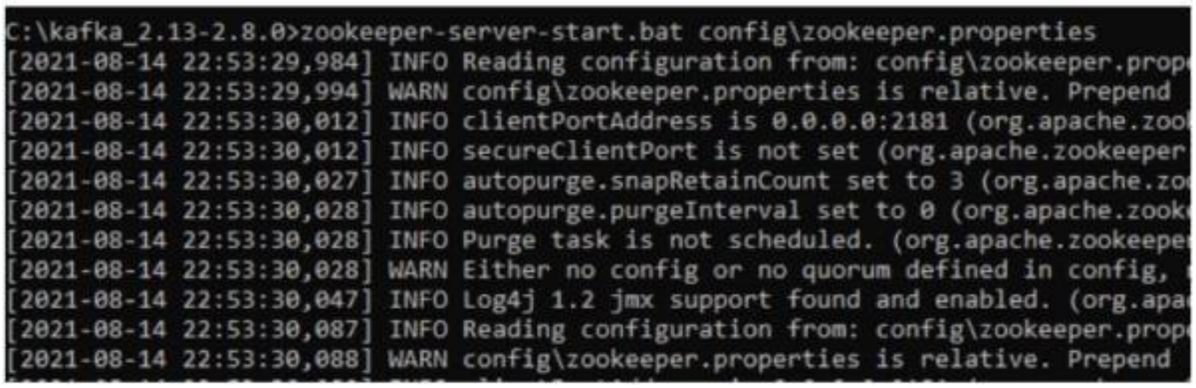
```
import findspark
findspark.init('')
import pyspark
from pyspark import RDD
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils

if name=="main":
    sc = SparkContext(appName="PythonSparkStreamingKafka")
    ssc = StreamingContext(sc,60)
    #Creating Kafka direct stream
    message= KafkaUtils.createDirectStream(ssc, ["testtopic"],
{"metadata.broker.list":"|replace with your Kafka private
address|:9092"})
    words=message.map(lambda x:x[1].flatmap(lambda x:x.split(" ")))
    wordcount=words.map(lambda x: (x,1).reduceByKey(lambda a,b:a+b))
    wordcount.ppr int()
    #Starting Spark context
    ssc.start()
    ssc.awaitTermination()
```

Commands:

Open 4 command line terminals On first terminal enter the

command: **zookeeper-server-start.bat**

.\config\zookeeper.properties Output:



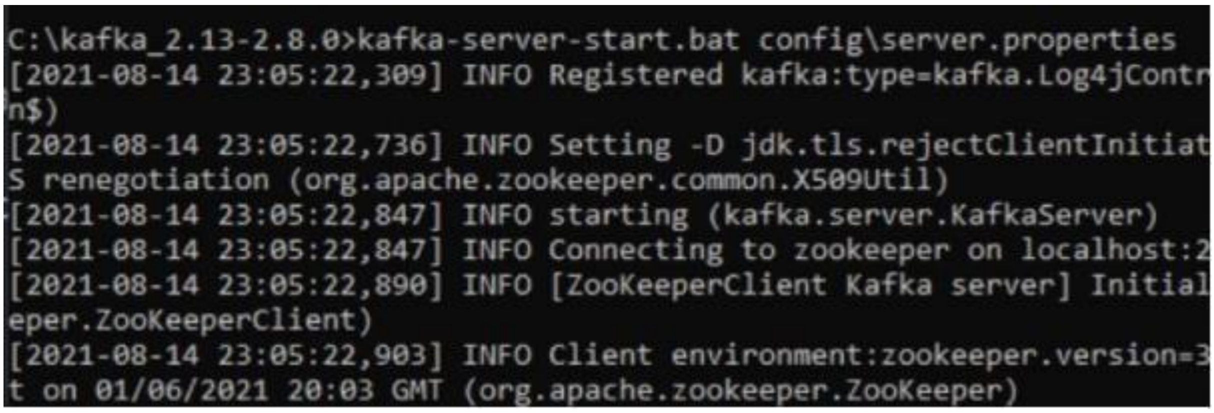
```
C:\kafka_2.13-2.8.0>zookeeper-server-start.bat config\zookeeper.properties
[2021-08-14 22:53:29,984] INFO Reading configuration from: config\zookeeper.prop
[2021-08-14 22:53:29,994] WARN config\zookeeper.properties is relative. Prepend
[2021-08-14 22:53:30,012] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zoo
[2021-08-14 22:53:30,012] INFO secureClientPort is not set (org.apache.zookeeper
[2021-08-14 22:53:30,027] INFO autopurge.snapRetainCount set to 3 (org.apache.zo
[2021-08-14 22:53:30,028] INFO autopurge.purgeInterval set to 0 (org.apache.zook
[2021-08-14 22:53:30,028] INFO Purge task is not scheduled. (org.apache.zookeepe
[2021-08-14 22:53:30,028] WARN Either no config or no quorum defined in config,
[2021-08-14 22:53:30,047] INFO Log4j 1.2 jmx support found and enabled. (org.apa
[2021-08-14 22:53:30,087] INFO Reading configuration from: config\zookeeper.prop
[2021-08-14 22:53:30,088] WARN config\zookeeper.properties is relative. Prepend
```

Zookeeper is up and running at the specified port. On

second terminal enter the command:

kafka-server-start.bat .\config\server.properties

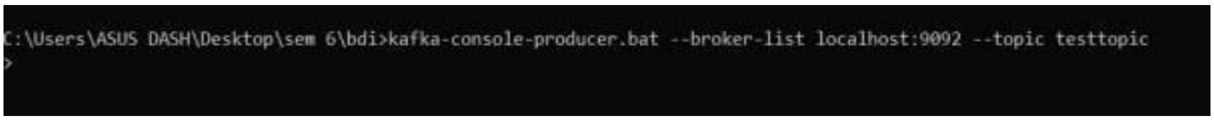
Output:



```
C:\kafka_2.13-2.8.0>kafka-server-start.bat config\server.properties
[2021-08-14 23:05:22,309] INFO Registered kafka:type=kafka.Log4jContr
n$)
[2021-08-14 23:05:22,736] INFO Setting -D jdk.tls.rejectClientInitiat
S renegotiation (org.apache.zookeeper.common.X509Util)
[2021-08-14 23:05:22,847] INFO starting (kafka.server.KafkaServer)
[2021-08-14 23:05:22,847] INFO Connecting to zookeeper on localhost:2
[2021-08-14 23:05:22,890] INFO [ZooKeeperClient Kafka server] Initial
eper.ZooKeeperClient)
[2021-08-14 23:05:22,903] INFO Client environment:zookeeper.version=3
t on 01/06/2021 20:03 GMT (org.apache.zookeeper.ZooKeeper)
```

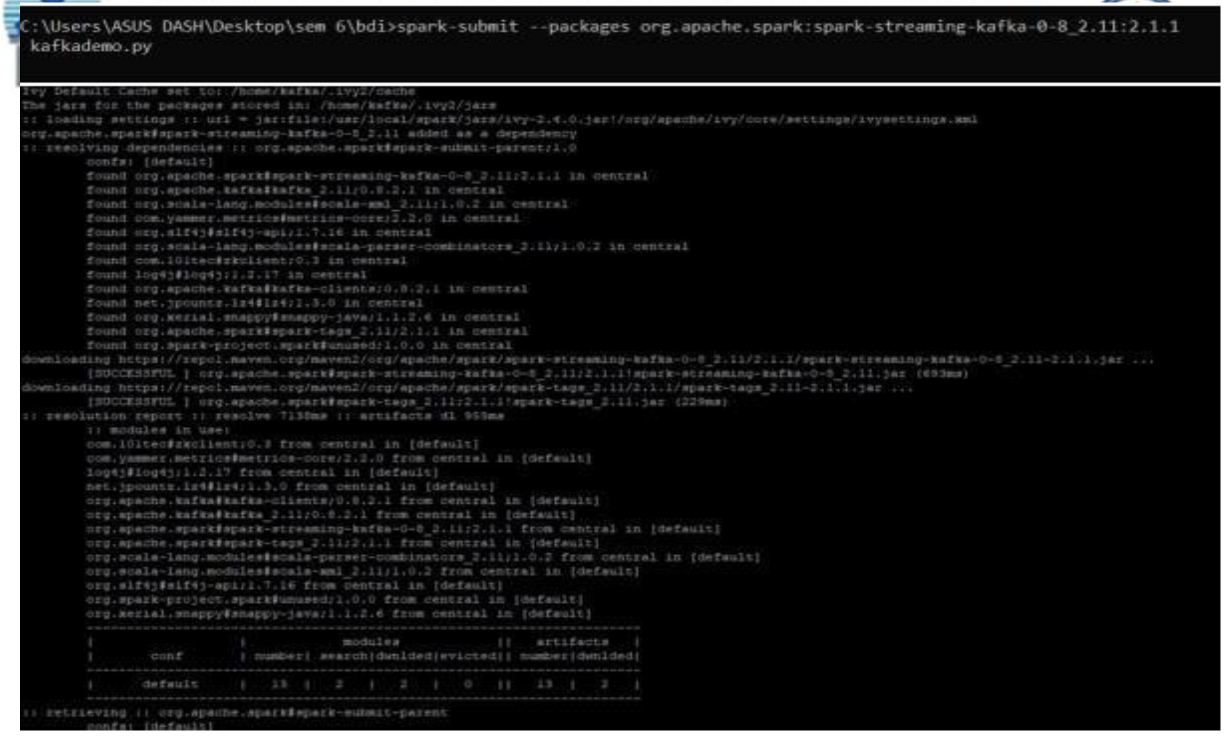
Now, you have the kafka running in one command line window and zookeeper running in another. If you want to start working with them, you will have to open another command line window and start writing commands.

On third terminal we run the command: **kafka-console-producer.bat --broker-list localhost:9092 --topic testtopic**



```
C:\Users\ASUS DASH\Desktop\sem 6\bdi>kafka-console-producer.bat --broker-list localhost:9092 --topic testtopic
>
```

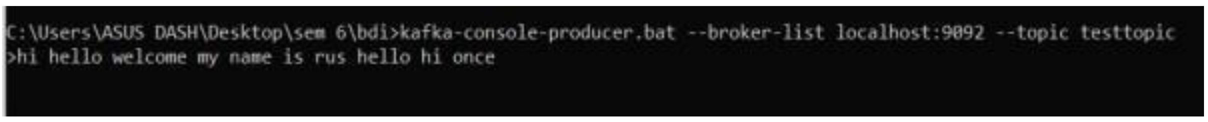
On fourth terminal we run the command:



```
C:\Users\ASUS DASH\Desktop\sem 6\bdi>spark-submit --packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.1
 kafkademo.py
```

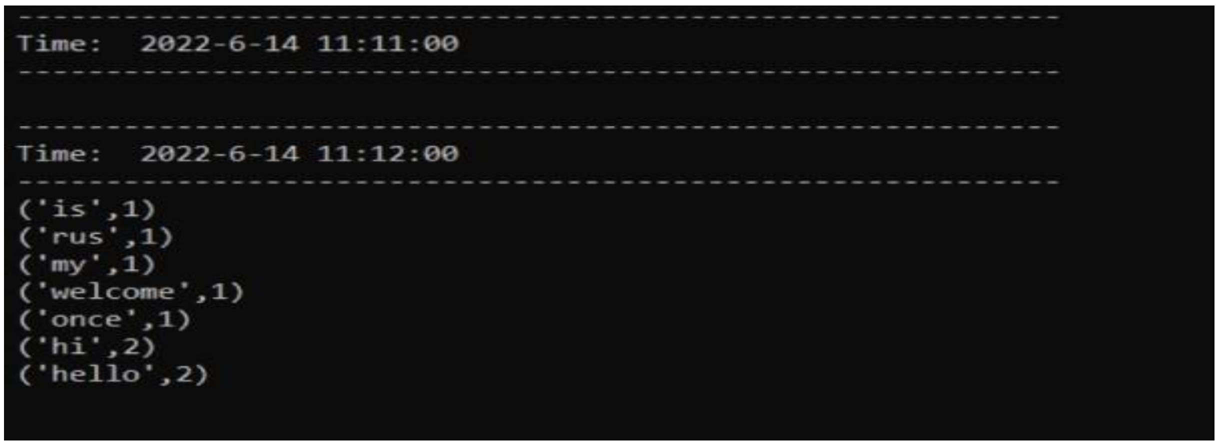
```
Ivy Default Cache set to: /home/kafka/.ivy2/cache
The jars for the packages stored in: /home/kafka/.ivy2/jars
:: loading settings :: url = jar:file:/usr/local/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
org.apache.spark#spark-streaming-kafka-0-8_2.11 added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent;1.0
        confs: [default]
        found org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.1 in central
        found org.apache.kafka#kafka_2.11;0.8.2.1 in central
        found org.scala-lang.modules#scala-xml_2.11;1.0.2 in central
        found com.yammer.metrics#metrics-core;2.2.0 in central
        found org.slf4j#slf4j-api;1.7.16 in central
        found org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 in central
        found com.101tec#zkclient;0.3 in central
        found log4j#log4j;1.2.17 in central
        found org.apache.kafka#kafka-clients;0.8.2.1 in central
        found net.jpountz.lz4#lz4;1.3.0 in central
        found org.xerial.snappy#snappy-java;1.1.2.6 in central
        found org.apache.spark#spark-tags_2.11;2.1.1 in central
        found org.spark-project.spark#unused;1.0.0 in central
downloading https://repo1.maven.org/maven2/org/apache/spark/spark-streaming-kafka-0-8_2.11/2.1.1/spark-streaming-kafka-0-8_2.11-2.1.1.jar ...
        [SUCCESSFUL ] org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.1!spark-streaming-kafka-0-8_2.11.jar (693ms)
downloading https://repo1.maven.org/maven2/org/apache/spark/spark-tags_2.11/2.1.1/spark-tags_2.11-2.1.1.jar ...
        [SUCCESSFUL ] org.apache.spark#spark-tags_2.11;2.1.1!spark-tags_2.11.jar (229ms)
:: resolution report :: resolve 7138ms :: artifacts dl 959ms
        :: modules in use:
        com.101tec#zkclient;0.3 from central in [default]
        com.yammer.metrics#metrics-core;2.2.0 from central in [default]
        log4j#log4j;1.2.17 from central in [default]
        net.jpountz.lz4#lz4;1.3.0 from central in [default]
        org.apache.kafka#kafka-clients;0.8.2.1 from central in [default]
        org.apache.kafka#kafka_2.11;0.8.2.1 from central in [default]
        org.apache.spark#spark-streaming-kafka-0-8_2.11;2.1.1 from central in [default]
        org.apache.spark#spark-tags_2.11;2.1.1 from central in [default]
        org.scala-lang.modules#scala-parser-combinators_2.11;1.0.2 from central in [default]
        org.scala-lang.modules#scala-xml_2.11;1.0.2 from central in [default]
        org.slf4j#slf4j-api;1.7.16 from central in [default]
        org.spark-project.spark#unused;1.0.0 from central in [default]
        org.xerial.snappy#snappy-java;1.1.2.6 from central in [default]
        ---------------------------------------------------------------------
        |                  |            modules            ||   artifacts   |
        |       conf       | number| search|dwnlded|evicted|| number|dwnlded|
        ---------------------------------------------------------------------
        |      default     |   13  |   2   |   2   |   0   ||   13  |   2   |
        ---------------------------------------------------------------------
:: retrieving :: org.apache.spark#spark-submit-parent
        confs: [default]
```

Now our command line is waiting for the data we enter the data in terminal 3:



```
C:\Users\ASUS DASH\Desktop\sem 6\bdi>kafka-console-producer.bat --broker-list localhost:9092 --topic testtopic
>hi hello welcome my name is rus hello hi once
```

And the output is obtained at terminal 4:



```
-------------------------------------------
Time:  2022-6-14 11:11:00
-------------------------------------------

-------------------------------------------
Time:  2022-6-14 11:12:00
-------------------------------------------
('is',1)
('rus',1)
('my',1)
('welcome',1)
('once',1)
('hi',2)
('hello',2)
```

CONCLUSION: Hence, with the help of the above experiment, we have integrated kafka and spark and read streaming data using Kafka.