

Kafka + Spark Streaming + PySpark

Spark Streaming

Connecting the Dots

Student Name: Norina Akhtar – 19643

Introduction

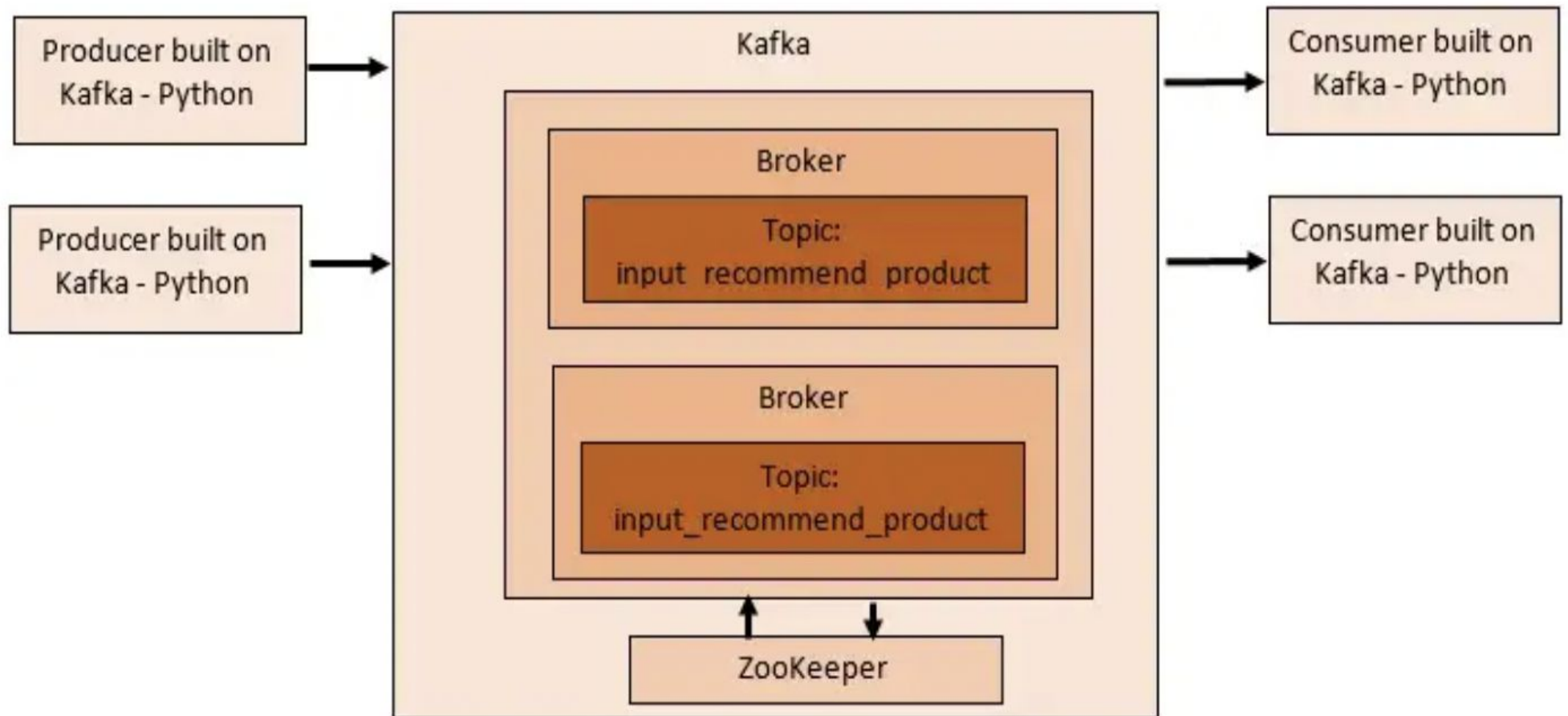
Real-time data ingesting is a common problem in real-time analytics, because in a platform such as e-commerce, active users in a given time and the number of events created by each active user are many. Hence, recommendations (i.e., predictions) for each event or groups of events are expected to be near real-time.

It is a distributed streaming platform, which helps to build real-time streaming data pipelines

The primary concerns are, *How we will [consume, produce, and process] these events efficiently?*



Design



Implementing step 1 (Apache Kafka + Kafka-Python)

1.The latest version of Kafka binary distribution is available at

<https://kafka.apache.org/downloads>.

2. Unzip the folder

Start the zookeeper on first terminal using the following command:

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

```
[root login: Fri Dec 2 21:05:40 on ttys001]
[norinaakhtar@Norinas-Air kafka-2.12-3.3.1 % bin/zookeeper-server-start.sh config/zookeeper.properties]
[2022-12-02 21:06:19,243] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,244] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,246] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,246] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,246] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,246] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,248] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-12-02 21:06:19,248] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-12-02 21:06:19,248] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
[2022-12-02 21:06:19,248] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerMain)
[2022-12-02 21:06:19,248] INFO Log4j 1.2 jmx support not found; jmx disabled. (org.apache.zookeeper.jmx.ManagedUtil)
[2022-12-02 21:06:19,249] INFO Reading configuration from: config/zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,249] WARN config/zookeeper.properties is relative. Prepend ./ to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,249] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,249] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,249] INFO observerMasterPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,249] INFO metricsProvider.className is org.apache.zookeeper.metrics.impl.DefaultMetricsProvider (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2022-12-02 21:06:19,249] INFO Starting server (org.apache.zookeeper.server.ZooKeeperServerMain)
[2022-12-02 21:06:19,254] INFO ServerMetrics initialized with provider org.apache.zookeeper.metrics.impl.DefaultMetricsProvider@15bb6bea (org.apache.zookeeper.server.ServerMetrics)
[2022-12-02 21:06:19,256] INFO zookeeper.snapshot.trust.empty : false (org.apache.zookeeper.server.persistence.FileTxnSnapLog)
[2022-12-02 21:06:19,260] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-12-02 21:06:19,260] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-12-02 21:06:19,260] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-12-02 21:06:19,261] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-12-02 21:06:19,261] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-12-02 21:06:19,261] INFO (org.apache.zookeeper.server.ZooKeeperServer)
[2022-12-02 21:06:19,261] INFO (org.apache.zookeeper.server.ZooKeeperServer)
```

Implementation

3. Starting Kafka Brokers

Use another terminal:

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

```
...he.zookeeper.server.quorum.QuorumPeerMain config/zookeeper.properties
```

```
...nt-3.3.1.jar:/Users/norinaakhtar/Desktop/kafka.Kafka config/server.properties
```

```
Last login: Fri Dec  2 21:06:03 on ttys000
```

```
[norinaakhtar@Norinas-Air kafka_2.12-3.3.1 % bin/kafka-server-start.sh config/server.properties
```

4. Creating Kafka Topics

Use another command

```
bin/kafka-topics.sh --create --topic input_recommend_product --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1
```

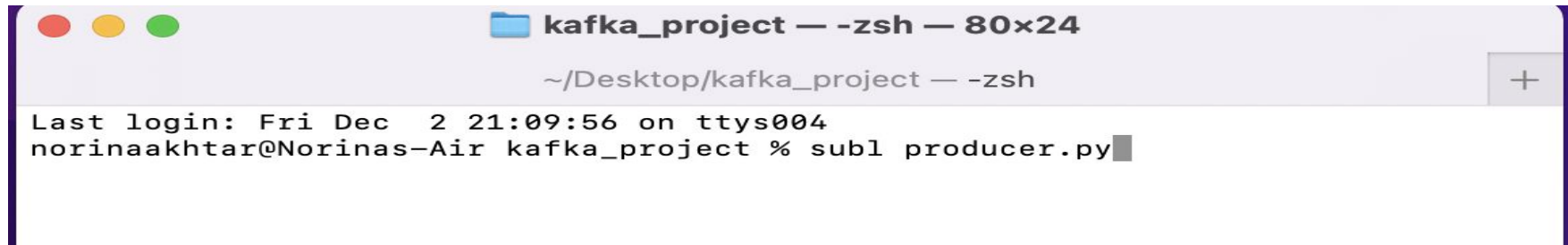
```

...he.zookeeper.server.quorum.QuorumPeerMain config/zookeeper.properties      ...nt-3.3.1.jar:/Users/norinaakhtar/Desktop kafka.Kafka config/server.properties      ~/Desktop/kafka_2.12-3.3.1 --
Last login: Fri Dec  2 21:06:20 on ttys001
[norinaakhtar@Norinas-Air kafka_2.12-3.3.1 % bin/kafka-topics.sh --create --topic input_recommend_product --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1
WARNING: Due to limitations in metric names, topics with a period('.') or underscore('_') could collide. To avoid issues it is best to use either, but not both.
Created topic input_recommend_product.
[norinaakhtar@Norinas-Air kafka_2.12-3.3.1 % ls
LICENSE      NOTICE      bin          config       libs         licenses     logs         site-docs
[norinaakhtar@Norinas-Air kafka_2.12-3.3.1 % bin/kafka-topics.sh --bootstrap-server localhost:9092 --list

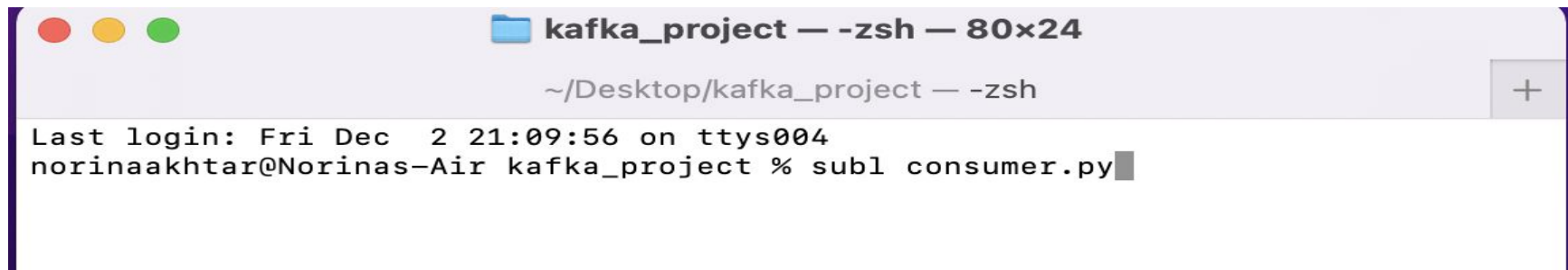
```

Implementation

5. Creating Producer and Consumer using Kafka-python



```
kafka_project — -zsh — 80x24
~/Desktop/kafka_project — -zsh
Last login: Fri Dec  2 21:09:56 on ttys004
norinaakhtar@Norinas-Air kafka_project % subl producer.py
```



```
kafka_project — -zsh — 80x24
~/Desktop/kafka_project — -zsh
Last login: Fri Dec  2 21:09:56 on ttys004
norinaakhtar@Norinas-Air kafka_project % subl consumer.py
```

consumer.py

```
1 from kafka import KafkaConsumer
2
3 consumer = KafkaConsumer('input_recommend_product', bootstrap_servers=['localhost:9092'])
4 for msg in consumer:
5     print(msg.value)
```

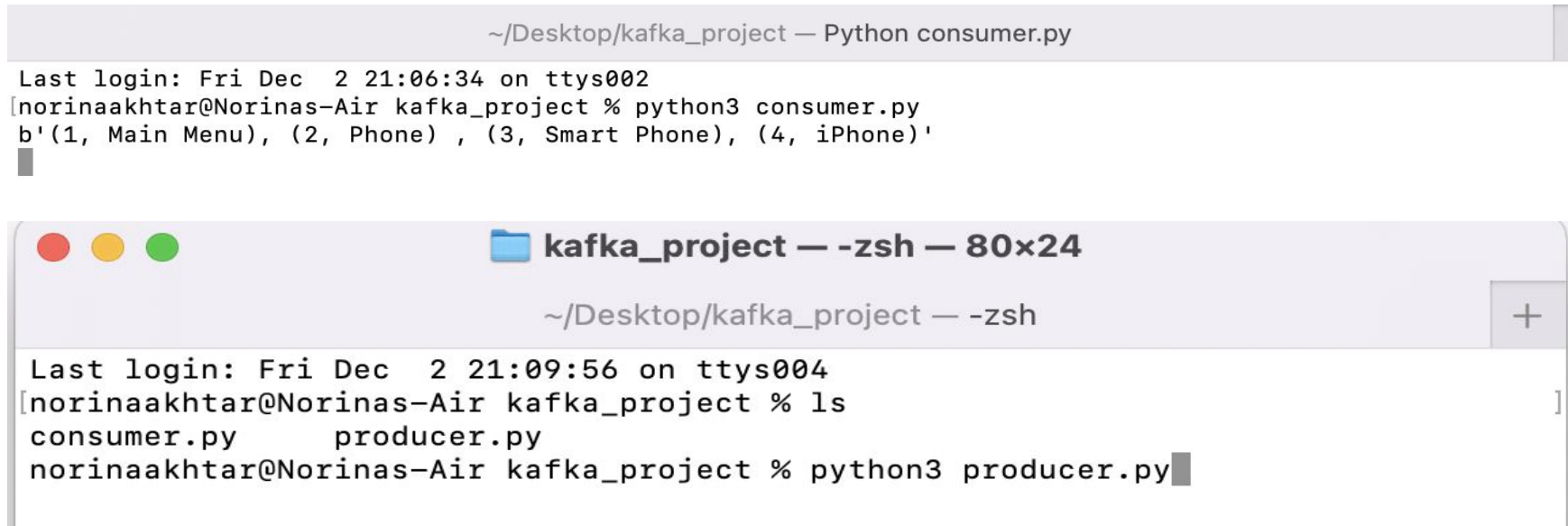
producer.py

```
1 from kafka import KafkaProducer
2
3 producer = KafkaProducer(bootstrap_servers='localhost:9092')
4
5 producer.send('input_recommend_product', b'(1, Main Menu), (2, Phone) , (3, Smart Phone), (4,
6 producer.close()
7
```


Run Files

Steps:

- i. First run consumer.py
- ii. Run producer.py on another terminal
- iii. You can see **output** on first terminal



The image shows two terminal windows. The top window, titled '~/Desktop/kafka_project — Python consumer.py', displays the output of running 'python3 consumer.py', which is a byte string: b'(1, Main Menu), (2, Phone) , (3, Smart Phone), (4, iPhone)'. The bottom window, titled 'kafka_project — -zsh — 80x24', shows the directory listing after running 'ls', which lists 'consumer.py' and 'producer.py', followed by the command 'python3 producer.py' being entered.

```
~/Desktop/kafka_project — Python consumer.py
Last login: Fri Dec  2 21:06:34 on ttys002
[norinaakhtar@Norinas-Air kafka_project % python3 consumer.py
b'(1, Main Menu), (2, Phone) , (3, Smart Phone), (4, iPhone)'
```

```
kafka_project — -zsh — 80x24
~/Desktop/kafka_project — -zsh
Last login: Fri Dec  2 21:09:56 on ttys004
[norinaakhtar@Norinas-Air kafka_project % ls
consumer.py      producer.py
norinaakhtar@Norinas-Air kafka_project % python3 producer.py
```


Spark streaming

Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including (but not limited to) Kafka, Flume, and Amazon Kinesis. This processed data can be pushed out to file systems, databases, and live dashboards.



Spark streaming



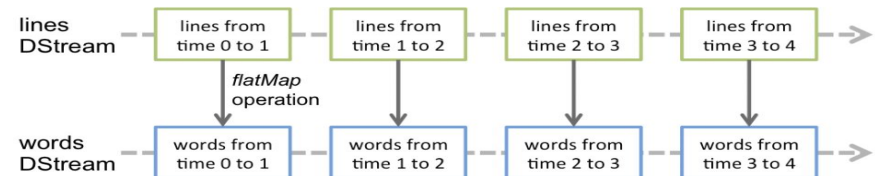
Spark Streaming provides a high-level abstraction called *discretized stream (DStream)*, which represents a continuous stream of data.

- **DStreams** can be created
 - from input data streams from sources such as [Kafka](#), [Flume](#), and Kinesis, or
 - by applying high-level operations on other **DStreams**.
- Internally, a **DStream** is represented as a sequence of **RDDs**.



A **DStreams** is represented by a continuous series of **RDDs**, which is Spark's abstraction of an **immutable, distributed dataset**.

Each **RDD** in a **DStreams** contains data from a certain interval,



The **flatMap** operation is applied on each **RDD** in the **lines DStream** to generate the **RDDs** of the **words DStream**.

Implementing spark streaming

Setup

Running scala with sbt:

1. Make sure you have the Java 8 JDK (also known as 1.8)

2. Download SBT:

<https://www.scala-sbt.org/download.html>

MAC:

```
brew install sbt
```

Create the project:

- `cd` to an empty folder.
- Run the following command `sbt new scala/scala3.g8`. This pulls the 'scala3' template from GitHub. It will also create a `target` folder, which you can ignore.
- When prompted, name the application `wordcount`. This will create a project called “`wordcount`”.

Step 1: Write a standalone application in Spark to count the number of words in a received stream.

Paste the code given code in:

```
/Users/norinaakhtar/Desktop/new/hello-world/src/main
```

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel
```

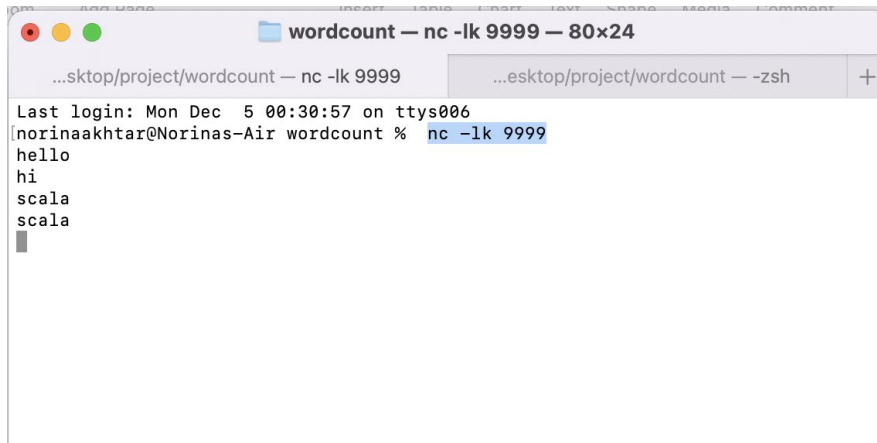
```
object NetworkWordCount {
  def main(args: Array[String]) {
    val ssc = new StreamingContext("local[2]",
      "NetworkWordCount", Seconds(1))
    val lines = ssc.socketTextStream("127.0.0.1", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(x => (x, 1))
    val wordCounts = pairs.reduceByKey(_ + _)
    wordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

Build.sbt should look like this

```
name := "wordcount"
version := "1.0"
scalaVersion := "2.10.3"
libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "0.9.0-incubating",
  "org.apache.spark" %% "spark-streaming" % "0.9.0-incubating"
)
```

Step1: Result

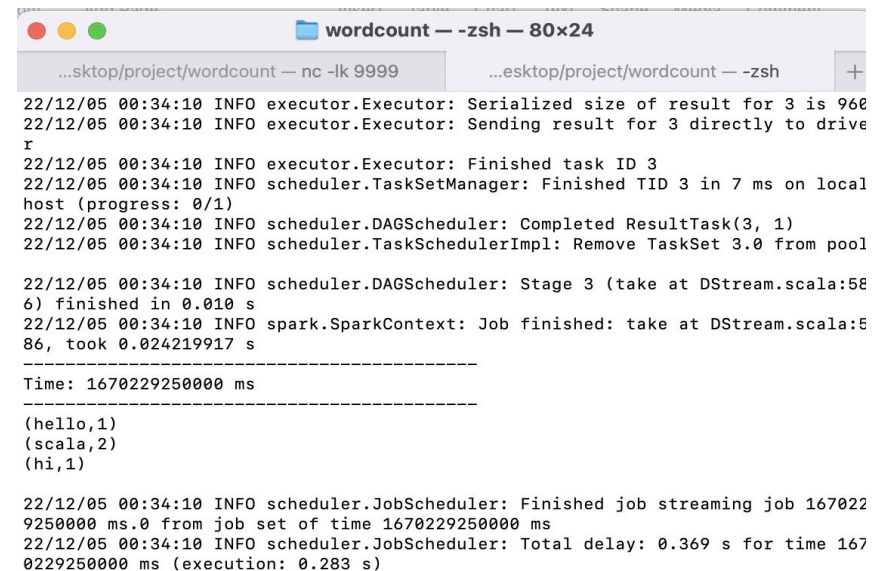
Run `nc -lk 9999` on Terminal 1:

A screenshot of a terminal window titled "wordcount — nc -lk 9999 — 80x24". The terminal shows the command "nc -lk 9999" being executed, which listens for incoming connections. It receives three connections with the messages "hello", "hi", and "scala". The terminal output is as follows:

```
Last login: Mon Dec 5 00:30:57 on ttys006
norinaakhtar@Norinas-Air wordcount % nc -lk 9999
hello
hi
scala
scala
```

Terminal 2

1. Cd into wordcount
2. \$ sbt package
3. \$ sbt run

A screenshot of a terminal window titled "wordcount — -zsh — 80x24". The terminal shows the output of the "sbt run" command, which includes Spark logs and the final result of the wordcount job. The terminal output is as follows:

```
22/12/05 00:34:10 INFO executor.Executor: Serialized size of result for 3 is 960
22/12/05 00:34:10 INFO executor.Executor: Sending result for 3 directly to driver
22/12/05 00:34:10 INFO executor.Executor: Finished task ID 3
22/12/05 00:34:10 INFO scheduler.TaskSetManager: Finished TID 3 in 7 ms on local host (progress: 0/1)
22/12/05 00:34:10 INFO scheduler.DAGScheduler: Completed ResultTask(3, 1)
22/12/05 00:34:10 INFO scheduler.TaskSchedulerImpl: Remove TaskSet 3.0 from pool

22/12/05 00:34:10 INFO scheduler.DAGScheduler: Stage 3 (take at DStream.scala:586) finished in 0.010 s
22/12/05 00:34:10 INFO spark.SparkContext: Job finished: take at DStream.scala:586, took 0.024219917 s

-----
Time: 1670229250000 ms
-----

(hello,1)
(scala,2)
(hi,1)

22/12/05 00:34:10 INFO scheduler.JobScheduler: Finished job streaming job 1670229250000 ms.0 from job set of time 1670229250000 ms
22/12/05 00:34:10 INFO scheduler.JobScheduler: Total delay: 0.369 s for time 1670229250000 ms (execution: 0.283 s)
```

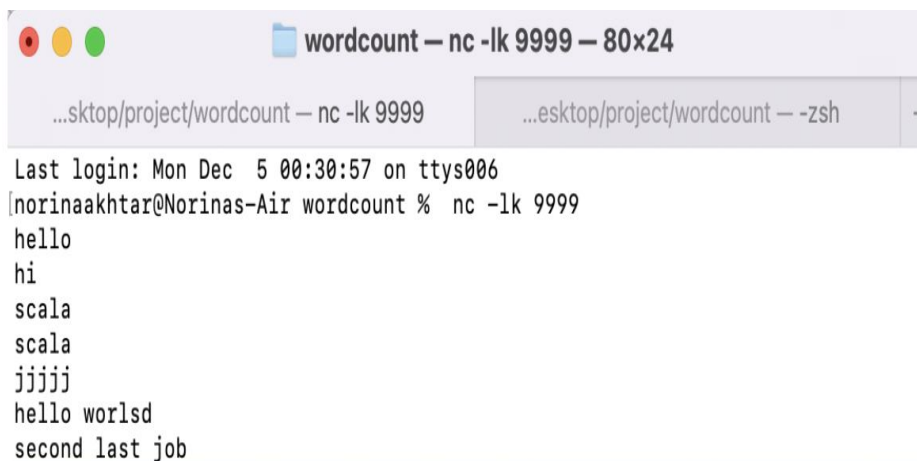
Step 2: Extend the code to generate word count over last 30 seconds of data, and repeat the computation every 10 seconds.

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel
object NetworkWordCount {
  def main(args: Array[String]) {
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }
    val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
    ssc.checkpoint(".")
    val lines = ssc.socketTextStream("127.0.0.1", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val stateWordCounts = pairs.updateStateByKey[Int](updateFunc)
    stateWordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```


Step2 : Result

Terminal 1

Run `nc -lk 9999` on Terminal 1:

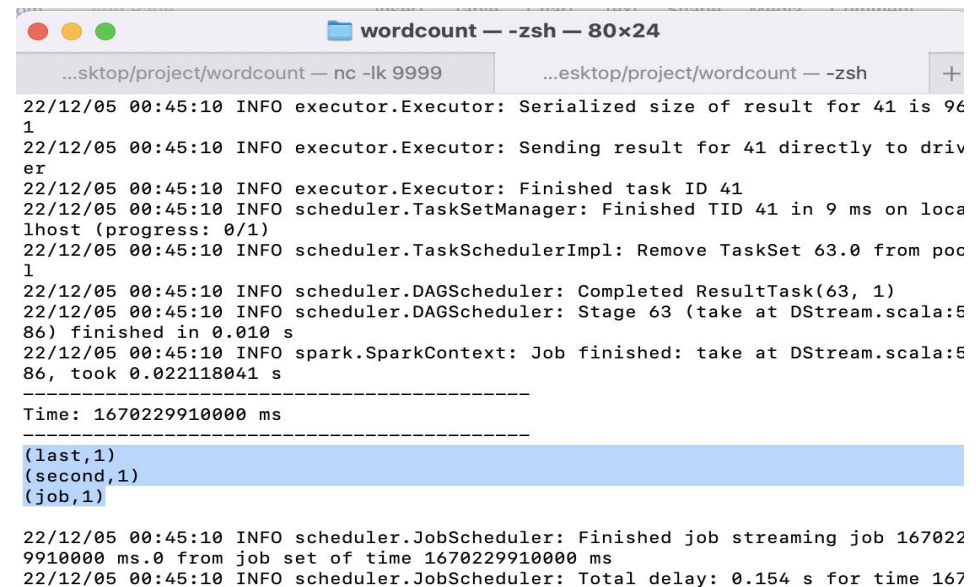
A screenshot of a terminal window titled "wordcount - nc -lk 9999 - 80x24". The terminal shows a netcat listener on port 9999. It receives a connection from "norinaakhtar@Norinas-Air" and the user enters the text "hello\nhi\nscala\nscala\njjjjj\nhello world\nsecond last job".

```
wordcount - nc -lk 9999 - 80x24
...sktop/project/wordcount - nc -lk 9999
...esktop/project/wordcount - zsh

Last login: Mon Dec  5 00:30:57 on ttys006
[norinaakhtar@Norinas-Air wordcount % nc -lk 9999
hello
hi
scala
scala
jjjjj
hello world
second last job
```

Terminal 2

1. Cd into wordcount
2. `$ sbt package`
3. `$ sbt run`

A screenshot of a terminal window titled "wordcount - -zsh - 80x24". It shows the output of running "sbt run" in a project directory. The output includes Spark logs for task execution and job completion, followed by a summary of the job's performance.

```
wordcount - -zsh - 80x24
...sktop/project/wordcount - nc -lk 9999
...esktop/project/wordcount - zsh

22/12/05 00:45:10 INFO executor.Executor: Serialized size of result for 41 is 96
1
22/12/05 00:45:10 INFO executor.Executor: Sending result for 41 directly to driv
er
22/12/05 00:45:10 INFO executor.Executor: Finished task ID 41
22/12/05 00:45:10 INFO scheduler.TaskSetManager: Finished TID 41 in 9 ms on loca
lhost (progress: 0/1)
22/12/05 00:45:10 INFO scheduler.TaskSchedulerImpl: Remove TaskSet 63.0 from poc
l
22/12/05 00:45:10 INFO scheduler.DAGScheduler: Completed ResultTask(63, 1)
22/12/05 00:45:10 INFO scheduler.DAGScheduler: Stage 63 (take at DStream.scala:5
86) finished in 0.010 s
22/12/05 00:45:10 INFO spark.SparkContext: Job finished: take at DStream.scala:5
86, took 0.022118041 s
-----
Time: 1670229910000 ms
-----
(last,1)
(second,1)
(job,1)

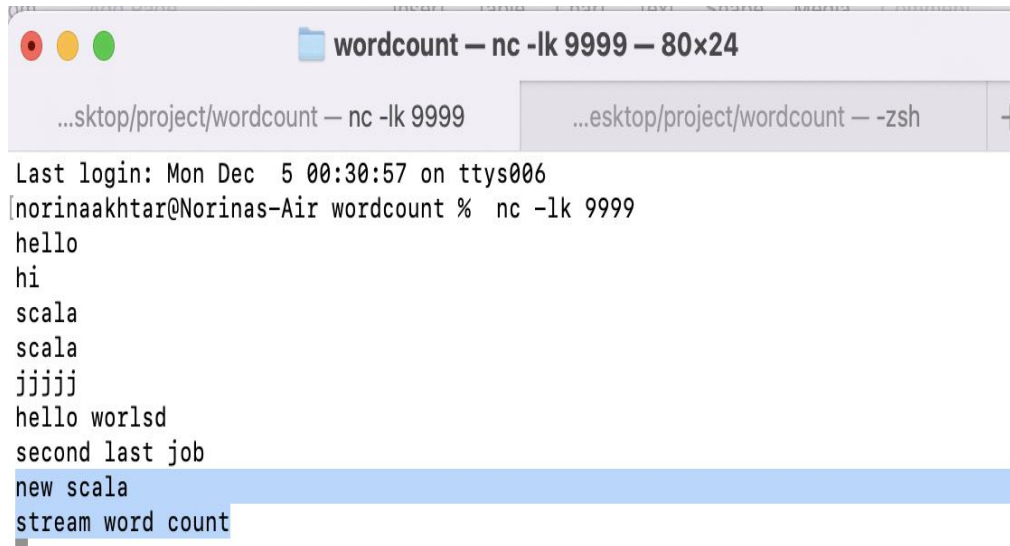
22/12/05 00:45:10 INFO scheduler.JobScheduler: Finished job streaming job 167022
9910000 ms.0 from job set of time 1670229910000 ms
22/12/05 00:45:10 INFO scheduler.JobScheduler: Total delay: 0.154 s for time 167
```

Step 3: Maintain a continuously updated word count for all the words in the stream.

```
import org.apache.spark.streaming.{Seconds, StreamingContext}
import org.apache.spark.streaming.StreamingContext._
import org.apache.spark.storage.StorageLevel
object NetworkWordCount {
  def main(args: Array[String]) {
    val updateFunc = (values: Seq[Int], state: Option[Int]) => {
      val currentCount = values.foldLeft(0)(_ + _)
      val previousCount = state.getOrElse(0)
      Some(currentCount + previousCount)
    }
    val ssc = new StreamingContext("local[2]", "NetworkWordCount", Seconds(1))
    ssc.checkpoint(".")
    val lines = ssc.socketTextStream("127.0.0.1", 9999)
    val words = lines.flatMap(_.split(" "))
    val pairs = words.map(word => (word, 1))
    val stateWordCounts = pairs.updateStateByKey[Int](updateFunc)
    stateWordCounts.print()
    ssc.start()
    ssc.awaitTermination()
  }
}
```

Step 3: Result

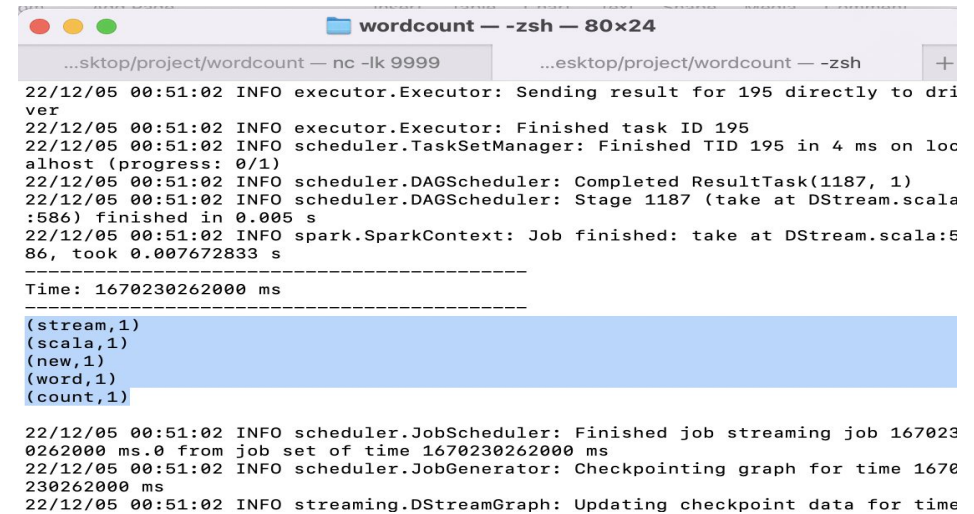
Terminal 1



Terminal 1 window titled "wordcount — nc -lk 9999 — 80x24". The window has two tabs: "...sktop/project/wordcount — nc -lk 9999" and "...esktop/project/wordcount — -zsh". The terminal output shows a login message and a series of commands and responses:

```
Last login: Mon Dec 5 00:30:57 on ttys006
[norinaakhtar@Norinas-Air wordcount % nc -lk 9999
hello
hi
scala
scala
jjjjj
hello world
second last job
new scala
stream word count
```

Terminal 2



Terminal 2 window titled "wordcount — -zsh — 80x24". The window has two tabs: "...sktop/project/wordcount — nc -lk 9999" and "...esktop/project/wordcount — -zsh". The terminal output shows Spark logs for the wordcount application:

```
22/12/05 00:51:02 INFO executor.Executor: Sending result for 195 directly to driver
22/12/05 00:51:02 INFO executor.Executor: Finished task ID 195
22/12/05 00:51:02 INFO scheduler.TaskSetManager: Finished TID 195 in 4 ms on localhost (progress: 0/1)
22/12/05 00:51:02 INFO scheduler.DAGScheduler: Completed ResultTask(1187, 1)
22/12/05 00:51:02 INFO scheduler.DAGScheduler: Stage 1187 (take at DStream.scala:586) finished in 0.005 s
22/12/05 00:51:02 INFO spark.SparkContext: Job finished: take at DStream.scala:586, took 0.007672833 s
-----
Time: 1670230262000 ms
-----
(stream,1)
(scala,1)
(new,1)
(word,1)
(count,1)
22/12/05 00:51:02 INFO scheduler.JobScheduler: Finished job streaming job 1670230262000 ms.0 from job set of time 1670230262000 ms
22/12/05 00:51:02 INFO scheduler.JobGenerator: Checkpointing graph for time 1670230262000 ms
22/12/05 00:51:02 INFO streaming.DStreamGraph: Updating checkpoint data for time
```

Connecting the Dots (Python, Spark, and Kafka)

Python, Spark, and Kafka are vital frameworks in data scientists' day to day activities. It is essential to enable them to integrate these framework

Frequently, Data scientists prefer to use Python (in some cases, R) to develop machine learning models. Here, they have a valid justification since data-driven solutions arrive with many experiments. Numerous interactions with the language we use to develop the models are required to perform experiments, and the libraries and platforms available in python to develop machine-learning models are tremendous. This is a valid argument; however, we confront issues when these models are applied to production

To overcome all the problems, we can identify a set of dots that could be appropriately connected. In this process, I attempt to connect these dots, which are Python, Apache Spark, and Apache Kafka

Implementation of Connecting the Dots (Python, Spark, and Kafka

Running out of credit

Enhancement ideas

- We can integrate kafka stream with **Redpanda** , which natively supports the kafka api so it works out of the box with existing tools and integrations. However its, 10x faster and upto 7x more cost effective.
- Different frameworks and formats i-e avro, parquet and persist can be used to improve the performance and tuning spark job.

Conclusion

- Apache — Kafka is a vital platform in building real-time processing solutions. This article gives you an excellent start to set up Apache — Kafka on a distributed environment and provides easy guidance to produce and consume events.
- Python, Spark, and Kafka are important frameworks in a data scientist's daily activities.
- It can help data scientists to perform their experiments in Python while deploying the final model in a scalable production environment.

References

<https://spark.apache.org/docs/latest/api/python/index.html>

<https://spark.apache.org/docs/latest/index.html#running-the-examples-and-shell>

<https://docs.confluent.io/kafka-clients/python/current/overview.html#ak-python>

<https://spark.apache.org/docs/latest/submitting-applications.html>