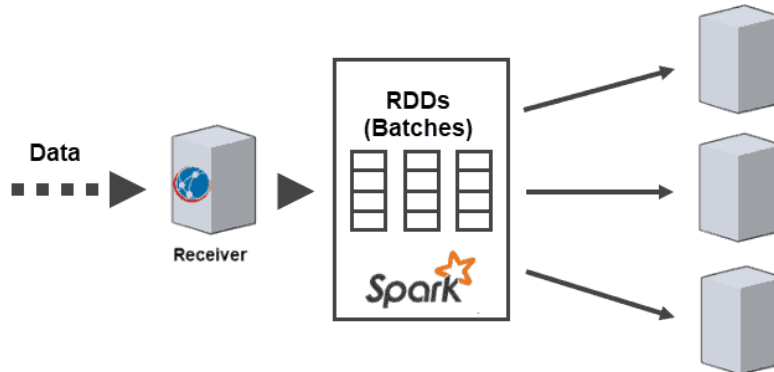


## Bài tập 3

# Spark Streaming

### 1. Tóm tắt Spark Streaming

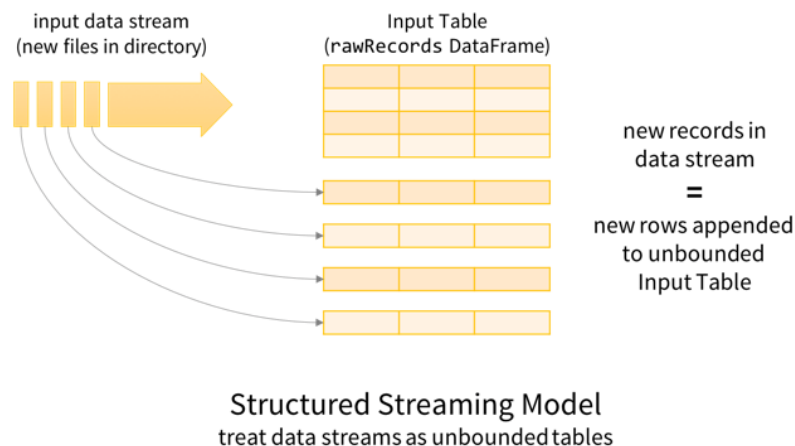
**Streaming** là một phương pháp xử lý dữ liệu trong thời gian thực, trong đó dữ liệu được xử lý gần như ngay lập tức khi nó được tạo ra hoặc nhận được. Streaming thường được áp dụng để xử lý dữ liệu liên tục từ các nguồn như cảm biến, hệ thống máy chủ, ứng dụng di động, và nhiều nguồn dữ liệu khác.



**Spark Streaming** là một thành phần của Apache Spark cho phép xử lý dữ liệu liên tục. Nó cho phép bạn lấy dữ liệu đầu vào từ các nguồn như Kafka, Flume, hoặc nguồn dữ liệu TCP/IP và sau đó xử lý dữ liệu này theo cách phân tán và song song.

- Dữ liệu được chia thành các batch nhỏ và xử lý song song trên các máy khác nhau.
- Spark Streaming hỗ trợ các thao tác xử lý dữ liệu giống như Apache Spark truyền thống, như lọc (filter), ánh xạ (map), thu giảm (reduce) và các phép biến đổi dữ liệu khác.

Spark Streaming bao gồm 2 loại Discretized Stream (DStream) và Structured Streaming. Trong đó Structured Streaming có độ trễ thấp và được sử dụng trong nhiều ứng dụng đòi hỏi thời gian thực cao. Thay vì xử lý dữ liệu dưới dạng batch như trong Spark Streaming, Structured Streaming đối xử với dữ liệu dưới dạng các micro batch với thời gian mỗi batch từ mili giây đến vài giây. Structured Streaming xem dữ liệu stream giống như một bảng dữ liệu nối dài vô tận và quá trình xử lý chính là xử lý các dòng dữ liệu mới được thêm vào.



Structured Streaming hỗ trợ DataFrame và Dataset API, cho phép chúng ta sử dụng các truy vấn cấu trúc (structured queries) giống như khi làm việc với dữ liệu tĩnh. Điều này giúp đơn giản hóa việc phát triển ứng dụng xử lý dữ liệu streaming và tạo ra mã dễ đọc hơn và dễ bảo trì hơn.

## 2. Chuẩn bị môi trường

Các hướng dẫn trong bài lab này được thực hiện trên môi trường Ubuntu 22.04.2.

### 2.1. Java

Cài đặt Java là bắt buộc cho Spark. Gõ lệnh sau vào cửa sổ terminal để kiểm tra xem Java có sẵn và phiên bản của nó là gì:

```
$ java -version
```

Nếu hệ thống chưa có Java, bạn có thể chạy lệnh sau để cài đặt:

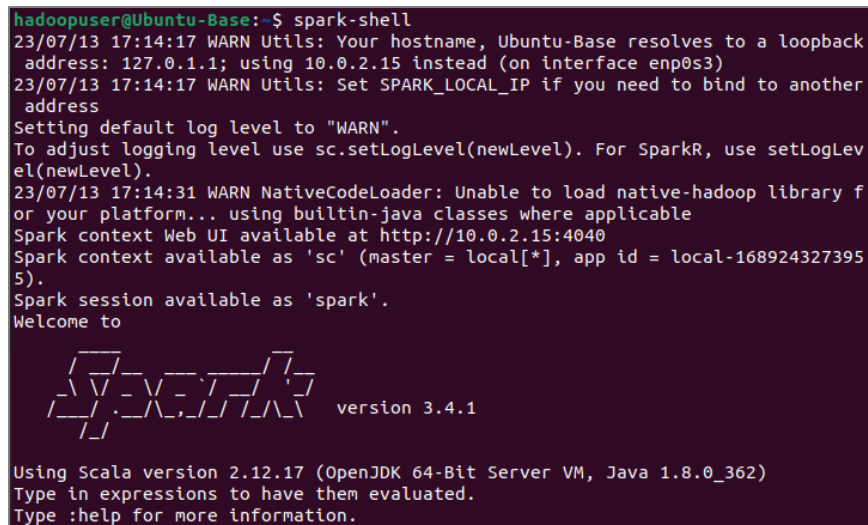
```
$ sudo apt-get update
$ sudo apt-get install openjdk-8-jdk -y
```

Kiểm tra lại sau khi cài đặt.

### 2.2. Spark

Đảm bảo Spark đã được cài đặt trên hệ thống. Bài lab này được tiếp nối với bài cài đặt Spark trong lab trước. Kiểm tra Spark được cài đặt thành công:

```
$ spark-shell
```



```
hadoopuser@Ubuntu-Base:~$ spark-shell
23/07/13 17:14:17 WARN Utils: Your hostname, Ubuntu-Base resolves to a loopback
address: 127.0.0.1; using 10.0.2.15 instead (on interface enp0s3)
23/07/13 17:14:17 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another
address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLev
el(newLevel).
23/07/13 17:14:31 WARN NativeCodeLoader: Unable to load native-hadoop library f
or your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.0.2.15:4040
Spark context available as 'sc' (master = local[*], app id = local-168924327395
5).
Spark session available as 'spark'.
Welcome to

  ____ _
 / ___ \| | | |
/ /___ \| |_| |
 \___ \|____|_|_|

version 3.4.1

Using Scala version 2.12.17 (OpenJDK 64-Bit Server VM, Java 1.8.0_362)
Type in expressions to have them evaluated.
Type :help for more information.
```

### 2.3. Scala

Khi chạy lệnh spark-shell ở trên, màn hình kết quả cũng thể hiện phiên bản của Scala. Bạn kiểm tra phiên bản Scala của hệ điều hành đã có chưa và có phù hợp với bản trong Spark không:

```
$ scala -version
```

Nếu không, bạn cần cập nhật lại để tránh bị xung đột phiên bản cho các lệnh sau này.

Gỡ bỏ phiên bản cũ hơn:

```
sudo apt-get remove scala-library scala
```

Tải phiên bản phù hợp (ở đây tôi chọn 2.12.17 theo bản Spark ở trên):

```
sudo wget https://downloads.lightbend.com/scala/2.12.17/scala-2.12.17.deb
```

Cài Scala:

```
$ sudo dpkg -i scala-2.12.17.deb
```

Khởi động lại hệ điều hành và kiểm tra phiên bản scala.

## 2.4. Hadoop HDFS

Một số phần trong bài lab này có sử dụng HDFS để lưu trữ và quản lý tài nguyên. Kiểm tra hệ thống đã cài đặt thành công HDFS.

## 2.5. Khởi chạy dịch vụ

Sau khi thiết lập xong, chúng ta bắt đầu chạy các dịch vụ để tạo các node (ảo) tương ứng với các cấu hình trong Hadoop.

Chạy tất cả các dịch vụ:

```
$ start-all.sh
```

## 3. Làm việc với các nguồn dữ liệu stream

### 3.1. Nguồn TCP Socket

Socket là một cơ chế giao tiếp cho phép hai tiến trình (process) trên cùng một máy tính hoặc trên các máy tính khác nhau giao tiếp với nhau qua mạng thông qua TCP/IP. Một TCP Socket được định nghĩa bởi một cặp địa chỉ IP (địa chỉ của máy tính) và số cổng (port number) để xác định ứng dụng hoặc dịch vụ cụ thể trên máy tính đó. Khi hai máy tính muốn thiết lập kết nối TCP với nhau, chúng cần xác định IP và cổng của nhau. Sau đó, một trong hai máy tính sẽ đóng vai trò như listener và chờ các yêu cầu kết nối từ máy tính còn lại. Khi yêu cầu kết nối được thiết lập, hai máy tính sẽ có thể gửi và nhận dữ liệu với nhau thông qua TCP Socket đã được thiết lập.

#### 3.1.1. WordCount với Spark Streaming Socket

Trong phần này, tôi trình bày các đoạn code chính để xử lý dữ liệu từ nguồn TCP Socket cùng với các giải thích ý nghĩa trước khi tổng hợp thành một mã nguồn hoàn chỉnh ở cuối phần. Đầu tiên, chúng ta sử dụng `readStream.format("socket")` từ đối tượng Spark session để đọc dữ liệu từ socket và cấu hình cho host và port để có thể lấy dữ liệu từ luồng (stream) tại địa chỉ host và cổng mong muốn.

```
val df = spark.readStream
    .format("socket")
    .option("host", "localhost")
    .option("port", "9090")
    .load()
```

Đoạn code trên lắng nghe TCP Socket trên cùng một máy (localhost) tại cổng 9090. Khi Spark đọc dữ liệu từ socket, nó biểu diễn dữ liệu dưới dạng một DataFrame với một cột được gọi là "value". Bạn có thể chạy lệnh `df.printSchema()` để hiển thị cấu trúc của DataFrame.

```
root
|-- value: string (nullable = true)
```

Tôi vẫn sử dụng bài toán WordCount để minh họa quá trình thực thi. Đầu tiên, chúng ta chia dữ liệu thành các từ dựa trên khoảng trắng, sau đó sử dụng hàm explode để làm phẳng (flatten) nó và tiếp tục áp dụng hàm groupBy.

```
val wordsDF = df.select(explode(split(df("value"), " ")).alias("word"))
val count = wordsDF.groupBy("word").count()
```

Khi đã hoàn thành việc xử lý, chúng ta sử dụng `writeStream.format("console")` để xuất kết quả lên màn hình console. Trong ngữ cảnh ví dụ này, chúng ta sử dụng `outputMode("complete")` khi viết các DataFrame kết quả. Điều này có nghĩa là kết quả của việc tổng hợp dữ liệu sẽ được cập nhật với toàn bộ dữ liệu mỗi lần có dữ liệu mới.

```
val query = count.writeStream
    .format("console")
    .outputMode("complete")
    .start()
    .awaitTermination()
```

Câu truy vấn được khởi chạy với lệnh `start()` và tiến trình sẽ đợi cho đến khi có tín hiệu dừng.

Các đoạn code trên được thực hiện hoàn chỉnh như sau:

```
import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions._

object SparkStreamingFromSocket {
  def main(args: Array[String]): Unit = {

    val spark: SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("SocketSparkExample")
      .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    val df = spark.readStream
      .format("socket")
      .option("host", "localhost")
      .option("port", "9090")
      .load()

    df.printSchema()

    val wordsDF = df.select(explode(split(df("value"), " ")).alias("word"))

    val count = wordsDF.groupBy("word").count()

    val query = count.writeStream
      .format("console")
      .outputMode("complete")
      .start()
      .awaitTermination()
  }
}
```

Lưu code trong một tập tin với tên `SparkStreamingFromSocket.scala` và thực hiện biên dịch mã nguồn này:

```
scalac -classpath "/usr/local/spark/jars/*" SparkStreamingFromSocket.scala
-d SparkStreamingFromSocket.jar
```

### 3.1.2. NetCat

NetCat (thường được viết tắt là nc) là một tiện ích được sử dụng để đọc và ghi dữ liệu trên các kết nối mạng sử dụng giao thức TCP hoặc UDP. Để ghi dữ liệu vào socket TCP bằng NetCat, làm theo các bước sau:

- Trước tiên, bạn cần cài đặt NetCat trên hệ thống của bạn.

```
sudo apt-get update
sudo apt-get install netcat
```

- Sau khi đã cài đặt NetCat, mở terminal. Bắt đầu sử dụng NetCat để ghi dữ liệu vào socket localhost tại port 9090:

```
nc -l -p 9090
```

- Bây giờ, bạn có thể bắt đầu nhập dữ liệu muốn gửi. Nhấn Enter sau khi nhập mỗi mẫu dữ liệu. Ứng dụng Spark sử dụng TCP Socket sẽ chịu trách nhiệm lắng nghe và xử lý dữ liệu nhận được trên cổng đó.

### 3.1.3. Chạy WordCount Spark Streaming

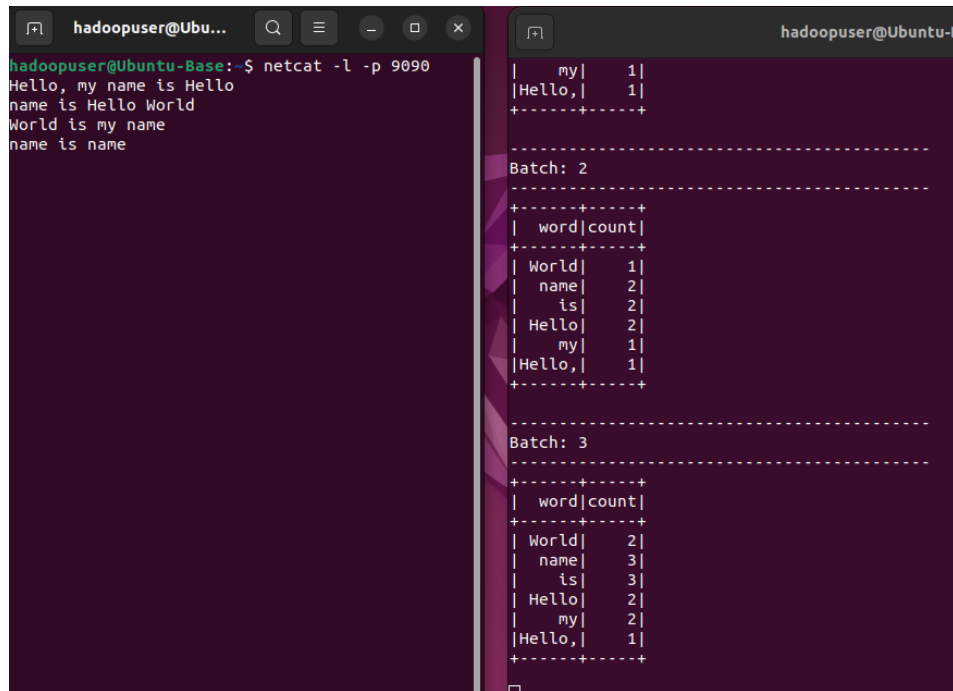
Mở một Terminal khác và chạy ứng dụng SparkStreamingFromSocket.jar:

```
spark-submit --class SparkStreamingFromSocket SparkStreamingFromSocket.jar
```

```
hadoopuser@Ubuntu-Base:~$ netcat -l -p 9090
hadoopuser@Ubuntu-Base:~$

23/07/28 21:16:35 INFO TransportClientFactory: Successfully created connection to /192.168.1.11:33973 after 89 ms (0 ms spent in bootstraps)
23/07/28 21:16:35 INFO Utils: Fetching spark://192.168.1.11:33973/jars/SparkStreamingFromSocket.jar to /tmp/spark-13f9b472-c362-44aa-a639-446c9da0d10b/userFiles-ef00f150-6ead-4bd4-abd0-f93585ab06b0/fetchFileTemp7994057791257752035.tmp
23/07/28 21:16:35 INFO Executor: Adding file:/tmp/spark-13f9b472-c362-44aa-a639-446c9da0d10b/userFiles-ef00f150-6ead-4bd4-abd0-f93585ab06b0/SparkStreamingFromSocket.jar to class loader
23/07/28 21:16:35 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 39565.
23/07/28 21:16:35 INFO NettyBlockTransferService: Server created on 192.168.1.11:39565
23/07/28 21:16:35 INFO BlockManager: Using org.apache.spark.storage.RandomBlockReplicationPolicy for block replication policy
23/07/28 21:16:35 INFO BlockManagerMaster: Registering BlockManager BlockManagerId(driver, 192.168.1.11, 39565, None)
23/07/28 21:16:35 INFO BlockManagerMasterEndpoint: Registering block manager 192.168.1.11:39565 with 366.3 MiB RAM, BlockManagerId(driver, 192.168.1.11, 39565, None)
23/07/28 21:16:35 INFO BlockManagerMaster: Registered BlockManager BlockManagerId(driver, 192.168.1.11, 39565, None)
23/07/28 21:16:35 INFO BlockManager: Initialized BlockManager: BlockManagerId(driver, 192.168.1.11, 39565, None)
-----
Batch: 0
-----
+----+-----+
|word|count|
+----+-----+
+----+-----+
```

Sau khi chương trình không có ERROR nào, bạn chuyển sang bảng điều khiển NetCat và gõ một vài câu và nhấn Enter cho mỗi dòng. Trong quá trình gõ, chúng ta sẽ thấy Spark sẽ xử lý dần dần theo các batch khác nhau.



```
hadoopuser@Ubuntu-Base:~$ netcat -l -p 9090
Hello, my name is Hello
name is Hello World
World is my name
name is name

Batch: 2
+-----+
| word|count|
+-----+
| World| 1|
| name| 2|
| is| 2|
| Hello| 2|
| my| 1|
| Hello,| 1|
+-----+

Batch: 3
+-----+
| word|count|
+-----+
| World| 2|
| name| 3|
| is| 3|
| Hello| 2|
| my| 2|
| Hello,| 1|
+-----+
```

Dừng chương trình lại sau khi hoàn thành xong với Ctrl+C.

### 3.2. Nguồn tập tin trong thư mục

Spark Streaming có thể track các tập tin trong một thư mục. Nếu có tập tin mới, Spark tiến hành xử lý chúng. Trong phần này, chúng ta xử lý các tập tin JSON lưu thông tin về các vùng địa lý. Nhiệm vụ là đếm số lượng vùng có cùng ZipCode. Bạn có thể chép các tập tin này tại link sau:



Streaming sử dụng `readStream` trên đối tượng `SparkSession` để xử lý dữ liệu từ một hệ thống lưu trữ bên ngoài.

```
val df = spark.readStream
  .schema(schema)
  .json("file:///usr/local/spark/stream_data/zipcodes")
```

Bạn cần tạo thư mục `/usr/local/spark/stream_data/zipcodes` để chép dần dần các tập tin vào. Ban đầu, bạn chỉ tạo thư mục, chưa tải tập tin vào đó.

Sử dụng `writeStream.format("console")` để ghi `DataFrame` lên màn hình console.

```
df.writeStream
  .format("console")
  .outputMode("append")
  .start() // Start the computation
  .awaitTermination() // Wait for the computation to terminate
```

Để thực hiện phép tổng hợp `groupBy` và `count` trên dữ liệu luồng, bạn có thể làm như sau:

```
val groupDF = df.select("Zipcode")
  .groupBy("Zipcode").count()
```

`outputMode("complete")` nên được sử dụng khi viết các phép tổng hợp dữ liệu luồng. Ở đây, chúng ta viết kết quả đã được tổng hợp vào màn hình console.

```

groupDF.writeStream
    .format("console")
    .outputMode("complete")
    .start()
    .awaitTermination()

```

Tạo tập tin với tên `SparkStreamingFromDirectory.scala` và nhập mã nguồn hoàn chỉnh như sau:

```

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.types.{IntegerType, StringType, StructField, StructType}

object SparkStreamingFromDirectory {

  def main(args: Array[String]): Unit = {

    val spark: SparkSession = SparkSession.builder()
      .master("local[*]")
      .appName("DirectorySparkExample")
      .getOrCreate()

    spark.sparkContext.setLogLevel("ERROR")

    val schema = StructType(
      List(
        StructField("RecordNumber", IntegerType, true),
        StructField("Zipcode", StringType, true),
        StructField("ZipCodeType", StringType, true),
        StructField("City", StringType, true),
        StructField("State", StringType, true),
        StructField("LocationType", StringType, true),
        StructField("Lat", StringType, true),
        StructField("Long", StringType, true),
        StructField("Xaxis", StringType, true),
        StructField("Yaxis", StringType, true),
        StructField("Zaxis", StringType, true),
        StructField("WorldRegion", StringType, true),
        StructField("Country", StringType, true),
        StructField("LocationText", StringType, true),
        StructField("Location", StringType, true),
        StructField("Decommisioned", StringType, true)
      )
    )

    val df = spark.readStream
      .schema(schema)
      .json("file:///usr/local/spark/stream_data/zipcodes")

    df.printSchema()

    val groupDF = df.select("Zipcode")
      .groupBy("Zipcode").count()
    groupDF.printSchema()

    groupDF.writeStream
      .format("console")
      .outputMode("complete")
      .start()
      .awaitTermination()
  }
}

```

}

Thực hiện biên dịch mã nguồn này:

```
scalac -classpath "/usr/local/spark/jars/*"
SparkStreamingFromDirectory.scala -d SparkStreamingFromDirectory.jar
```

### 3.2.1. Chạy Spark Streaming cho tập tin

Mở một Terminal và chạy ứng dụng SparkStreamingFromDirectory.jar:

```
spark-submit --class SparkStreamingFromDirectory
SparkStreamingFromDirectory.jar
```

Mở một Terminal khác và sao chép từ từ các tập tin JSON vào thư mục đã xác định trước (/usr/local/spark/stream\_data/zipcodes/).

```
hadoopuser@Ubuntu-Base: ~
hadoopuser@Ubuntu-Base:~$ ls zipcodes/
zipcode10.json zipcode2.json zipcode6.json
zipcode11.json zipcode3.json zipcode7.json
zipcode12.json zipcode4.json zipcode8.json
zipcode1.json zipcode5.json zipcode9.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode1.json /usr/
local/spark/stream_data/zipcodes/zipcode1.json
hadoopuser@Ubuntu-Base:~$

hadoopuser@Ubuntu-Base: ~/streaming
RandomBlockReplicationPolicy for block replication policy
23/07/28 22:02:44 INFO BlockManagerMaster: Registering BlockManager
BlockManagerId(driver, 192.168.1.11, 41917, None)
23/07/28 22:02:44 INFO BlockManagerMasterEndpoint: Registering block
manager 192.168.1.11:41917 with 366.3 MB RAM, BlockManagerId(dri
ve r, 192.168.1.11, 41917, None)
23/07/28 22:02:44 INFO BlockManagerMaster: Registered BlockManager B
lockManagerId(driver, 192.168.1.11, 41917, None)
23/07/28 22:02:44 INFO BlockManager: Initialized BlockManager: Block
ManagerId(driver, 192.168.1.11, 41917, None)
root
|-- RecordNumber: integer (nullable = true)
|-- Zipcode: string (nullable = true)
|-- ZipCodeType: string (nullable = true)
|-- City: string (nullable = true)
|-- State: string (nullable = true)
|-- LocationType: string (nullable = true)
|-- Lat: string (nullable = true)
|-- Long: string (nullable = true)
|-- Xaxis: string (nullable = true)
|-- Yaxis: string (nullable = true)
|-- Zaxis: string (nullable = true)
|-- WorldRegion: string (nullable = true)
|-- Country: string (nullable = true)
|-- LocationText: string (nullable = true)
|-- Location: string (nullable = true)
|-- Decommissioned: string (nullable = true)

root
|-- Zipcode: string (nullable = true)
|-- count: long (nullable = false)
```

Kết quả sau khi chép một số tập tin:

```
hadoopuser@Ubuntu-Base: ~
hadoopuser@Ubuntu-Base:~$ ls zipcodes/
zipcode10.json zipcode2.json zipcode6.json
zipcode11.json zipcode3.json zipcode7.json
zipcode12.json zipcode4.json zipcode8.json
zipcode1.json zipcode5.json zipcode9.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode1.json /usr/
local/spark/stream_data/zipcodes/zipcode1.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode2.json /usr/
local/spark/stream_data/zipcodes/zipcode2.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode3.json /usr/
local/spark/stream_data/zipcodes/zipcode3.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode4.json /usr/
local/spark/stream_data/zipcodes/zipcode4.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode5.json /usr/
local/spark/stream_data/zipcodes/zipcode5.json
hadoopuser@Ubuntu-Base:~$ cp zipcodes/zipcode6.json /usr/
local/spark/stream_data/zipcodes/zipcode6.json
hadoopuser@Ubuntu-Base:~$

hadoopuser@Ubuntu-Base: ~/streaming
-----
Batch: 3
-----
+-----+
|Zipcode|count|
+-----+
| 76166|    1|
|  709|    1|
|  704|    3|
| 76177|    2|
+-----+

-----
Batch: 4
-----
+-----+
|Zipcode|count|
+-----+
| 76166|    1|
| 32564|    1|
| 85210|    1|
|  709|    1|
|  708|    1|
| 32046|    1|
| 34445|    1|
|  704|    3|
| 34487|    1|
| 85209|    1|
| 76177|    2|
+-----+
```

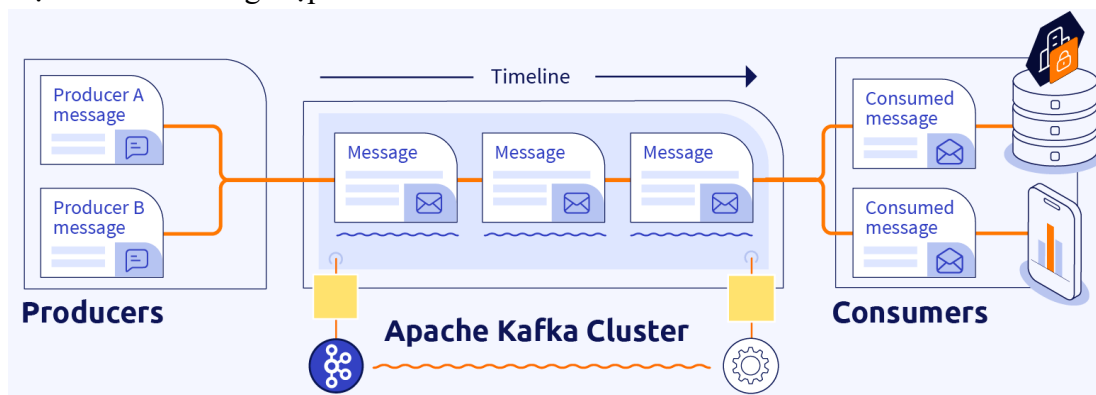
Dừng chương trình sau khi thực nghiệm xong.



### 3.3. Nguồn từ Kafka

#### 3.3.1. Giới thiệu

Apache Kafka là một hệ thống thông điệp phân tán phổ biến được thiết kế để xử lý lượng lớn dữ liệu thời gian thực. Một cụm Kafka có tính mở rộng cao và khả năng chịu lỗi. Nó cũng có khả năng xử lý lưu lượng thông tin cao hơn so với các hệ thống trung gian thông điệp khác như ActiveMQ và RabbitMQ. Mặc dù thông thường được sử dụng như một hệ thống publish/subscribe tin nhắn, nhiều tổ chức cũng sử dụng nó để tổng hợp nhật ký (log aggregation) vì nó cung cấp lưu trữ cố định cho các thông điệp.



Hệ thống publisher/subscriber cho phép một hoặc nhiều nhà xuất bản (producers) tạo các thông điệp mà không cần quan tâm đến số lượng người tiêu thụ (consumers) hoặc cách họ sẽ xử lý các thông điệp. Các bên đã đăng ký sẽ được tự động thông báo về các cập nhật hay tạo mới các thông điệp. Hệ thống này hiệu quả và có tính mở rộng hơn so với các hệ thống nơi khách hàng kiểm tra định kỳ để xác định xem có thông điệp mới nào có sẵn hay không.

#### 3.3.2. Cài đặt Kafka

Kafka cần được cài đặt phù hợp với phiên bản Scala. Trong phần thực hành này, chúng ta sử dụng Scala phiên bản 2.12.x, nên cần tải phiên bản tương thích. Truy cập trang <https://kafka.apache.org/downloads> để tải Kafka. Phiên bản mới nhất phù hợp với Scala 2.12 tại thời điểm bài viết này là 3.5.1. Tải phiên bản và thực hiện giải nén cũng như di chuyển đến nơi phù hợp:

```
$ wget https://downloads.apache.org/kafka/3.5.1/kafka_2.12-3.5.1.tgz
$ tar xzf kafka_2.12-3.5.1.tgz
$ sudo mv kafka_2.12-3.5.1 /usr/local/kafka
```

Đăng ký Kafka và Zookeeper vào trong dịch vụ khởi chạy của hệ thống để mỗi lần khởi động sẽ bật Kafka một cách tự động. Do Kafka cần Zookeeper để điều phối tài nguyên nên chúng ta cũng cần khởi động nó. Zookeeper được tích hợp sẵn trong bản tải của Kafka.

Tạo tập tin Zookeeper.service:

```
$ sudo vi /etc/systemd/system/zookeeper.service
```

Với nội dung sau:

```
[Unit]
Description=Apache Zookeeper server
Documentation=http://zookeeper.apache.org
Requires=network.target remote-fs.target
After=network.target remote-fs.target
```

```
[Service]
Type=simple
ExecStart=/usr/local/kafka/bin/zookeeper-server-start.sh
/usr/local/kafka/config/zookeeper.properties
ExecStop=/usr/local/kafka/bin/zookeeper-server-stop.sh
Restart=on-abnormal

[Install]
WantedBy=multi-user.target
```

Lưu ý, đường dẫn đến thư mục bin của Kafka cần khớp với hệ thống của bạn. Phần tô vàng là cùng một dòng.

Tạo tập tin kafka.service:

```
$ sudo vi /etc/systemd/system/kafka.service
```

Với nội dung sau:

```
[Unit]
Description=Apache Kafka Server
Documentation=http://kafka.apache.org/documentation.html
Requires=zookeeper.service

[Service]
Type=simple
Environment="JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64"
ExecStart=/usr/local/kafka/bin/kafka-server-start.sh
/usr/local/kafka/config/server.properties
ExecStop=/usr/local/kafka/bin/kafka-server-stop.sh

[Install]
WantedBy=multi-user.target
```

Reload lại tiến trình hệ thống:

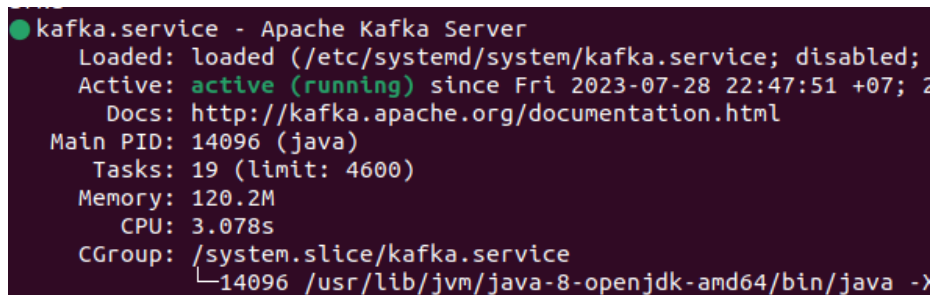
```
$ systemctl daemon-reload
```

Khởi chạy Zookeeper và Kafka:

```
$ sudo systemctl start zookeeper
$ sudo systemctl start kafka
```

Kiểm tra lại dịch vụ đã active:

```
$ sudo systemctl status kafka
```



```
kafka.service - Apache Kafka Server
Loaded: loaded (/etc/systemd/system/kafka.service; disabled;
Active: active (running) since Fri 2023-07-28 22:47:51 +07; 2
Docs: http://kafka.apache.org/documentation.html
Main PID: 14096 (java)
Tasks: 19 (limit: 4600)
Memory: 120.2M
CPU: 3.078s
CGroup: /system.slice/kafka.service
└─14096 /usr/lib/jvm/java-8-openjdk-amd64/bin/java -X
```

Đăng ký đường dẫn Kafka lên hệ điều hành bằng tập tin .bashrc:

```
$ sudo vi .bashrc
```

Thêm hai dòng sau vào cuối tập tin .bashrc:

```
export KAFKA_HOME=/usr/local/kafka
export PATH=$PATH:$KAFKA_HOME/bin
```

Cập nhật lại hệ thống:

```
$ source .bashrc
```

Tạo một chủ đề (topic) trong Kafka để đưa các message vào đó:

```
$ kafka-topics.sh --create --bootstrap-server localhost:9092 --
replication-factor 1 --partitions 1 --topic testTopic
```

Trong lệnh trên, do chúng ta chỉ có 1 máy nên các hệ số lặp lại và partition đều là 1. Chủ đề được đặt tên là testTopic.

Thực hiện kiểm tra chủ đề đã được tạo thành công:

```
$ kafka-topics.sh --list --bootstrap-server localhost:9092
```

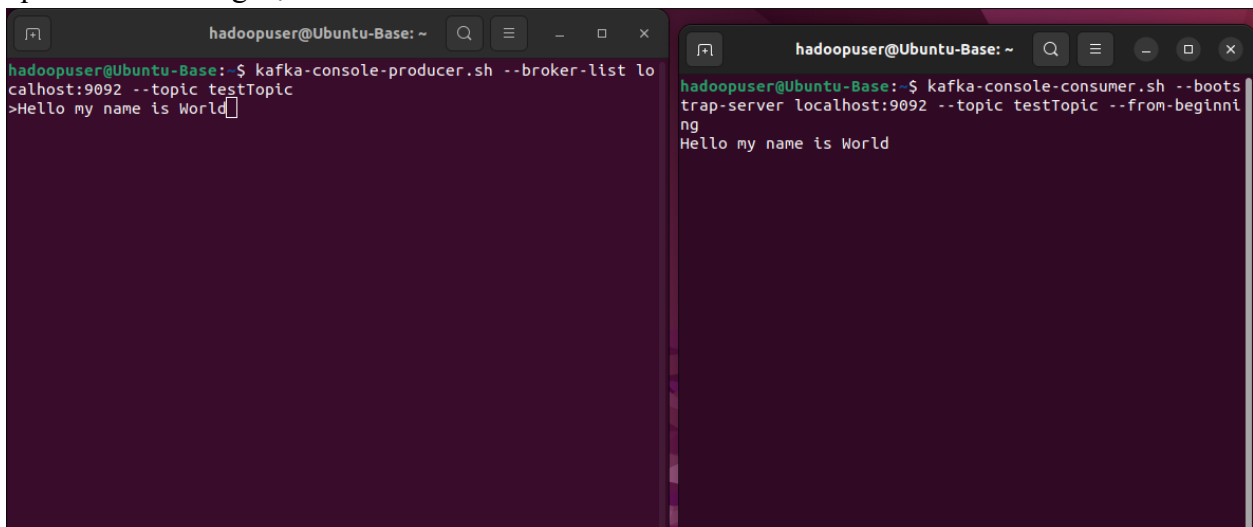
Thực hiện tạo các message vào chủ đề testTopic một cách thủ công qua màn hình Console:

```
$ kafka-console-producer.sh --broker-list localhost:9092 --topic
testTopic
```

Có thể test các message đã có trong hàng đợi của chủ đề bằng cách mở một Terminal khác và chạy dịch vụ consumer trên màn hình Console:

```
$ kafka-console-consumer.sh --bootstrap-server localhost:9092 --
topic testTopic --from-beginning
```

Tham số from-beginning để xác định là lấy message từ đầu đến cuối của topic trong Kafka. Kết quả sau khi thử nghiệm:



Đóng consumer lại, vẫn để producer chạy phục vụ cho Spark Streaming.

### 3.3.3. Lập trình Spark Streaming với Kafka

Spark Streaming sử dụng `readStream()` trên đối tượng `SparkSession` để tải một Streaming Dataset từ Kafka. Tùy chọn `startingOffsets` với giá trị `earliest` được sử dụng để đọc tất cả dữ liệu có sẵn trong Kafka vào thời điểm bắt đầu truy vấn. Chúng ta có thể không sử dụng tùy chọn này thường xuyên và giá trị mặc định của `startingOffsets` là `latest`, điều này chỉ đọc dữ liệu mới chưa được xử lý.

```
val df = spark.readStream
```

```

        .format("kafka")
        .option("kafka.bootstrap.servers", "localhost:9092")
        .option("subscribe", "testTopic")
        .option("startingOffsets", "earliest") // From starting
        .load()

df.printSchema()

```

Đối với dữ liệu luồng từ Kafka, `df.printSchema()` trả về schema của dữ liệu. `DataFrame` được trả về bao gồm tất cả các trường quen thuộc của một bản ghi Kafka và các siêu dữ liệu (metadata) liên quan đến nó.

```

root
 |-- key: binary (nullable = true)
 |-- value: binary (nullable = true)
 |-- topic: string (nullable = true)
 |-- partition: integer (nullable = true)
 |-- offset: long (nullable = true)
 |-- timestamp: timestamp (nullable = true)
 |-- timestampType: integer (nullable = true)

```

Giả sử chúng ta cũng thực hiện WordCount trên dữ liệu từ Kafka:

```

val groupCount = df.select(explode(split(df("value"), "
")).alias("word")).groupBy("word").count()

groupCount.writeStream
    .format("console")
    .outputMode("complete")
    .start()
    .awaitTermination()

```

Tạo tập tin với tên `SparkStreamingConsumeKafka.scala` và nhập mã nguồn hoàn chỉnh như sau:

```

import org.apache.spark.sql.SparkSession
import org.apache.spark.sql.functions.{explode, split}

object SparkStreamingConsumeKafka {

    def main(args: Array[String]): Unit = {

        val spark: SparkSession = SparkSession.builder()
            .master("local[*]")
            .appName("SparkKafkaExample")
            .getOrCreate()

        spark.sparkContext.setLogLevel("ERROR")

        val df = spark.readStream
            .format("kafka")
            .option("kafka.bootstrap.servers", "localhost:9092")
            .option("subscribe", "testTopic")
            // .option("subscribePattern", "topic.*")
            .option("startingOffsets", "earliest") // Other possible values
            assign and latest
            .load()

        df.printSchema()
    }
}

```

```

    val groupCount = df.select(explode(split(df("value"), "
")).alias("word"))
      .groupBy("word").count()

    groupCount.writeStream
      .format("console")
      .outputMode("complete")
      .start()
      .awaitTermination()

  }
}

```

Thực hiện biên dịch mã nguồn này:

```

scalac -classpath "/usr/local/spark/jars/*"
SparkStreamingConsumeKafka.scala -d SparkStreamingConsumeKafka.jar

```

### 3.3.4. Chạy Spark Streaming cho Kafka

Mở một Terminal khác và chạy ứng dụng SparkStreamingFromDirectory.jar:

```

spark-submit --packages org.apache.spark:spark-sql-kafka-0-
10_2.12:3.4.1 --class SparkStreamingConsumeKafka
SparkStreamingConsumeKafka.jar

```

Lưu ý, phải thiết lập dependency kafka cho spark cho tham số --packages, trong đó 2.12 là phiên bản scala và 3.4.1 là phiên bản Spark.

Thực hiện gõ các thông điệp trong Terminal Kafka producer.

```

hadoopuser@Ubuntu-Base: ~$ kafka-console-producer.sh --broker-list lo
calhost:9092 --topic testTopic
>This is the second Hello World
>

```

```

World| 1|
name| 1|
is| 1|
Hello| 1|
my| 1|
+-----+

Batch: 1
+-----+
| word|count|
+-----+
| World| 2|
| name| 1|
| is| 2|
| Hello| 2|
| second| 1|
| the| 1|
| my| 1|
| This| 1|
+-----+

```

## 4. Làm việc với các nơi lưu trữ kết quả stream (sink)

Thực hiện lại các bài trên với nơi lưu trữ (sink) ở dạng khác gồm lưu trữ lên tập tin, lên Kafka.

**Gợi ý cho lưu trữ trên tập tin:**

```

df.writeStream
  .outputMode("append") // Filesink only support Append mode.
  .format("json") // supports these formats : csv, json, orc, parquet
  .option("path", "file:///usr/local/spark/stream_data/file_sink")
  .option("header", true)

```

```

        .option("checkpointLocation",
"file:///usr/local/spark/stream_data/checkpoint")
        .start()
        .awaitTermination()

```

Lưu ý, đảm bảo các thư mục đã tồn tại.

Mở thư mục lưu trữ tập tin để xem các tập tin được tạo mới.

### Gợi ý cho Kafka:

```

df.selectExpr("CAST(word AS STRING) AS key", "to_json(struct(*)) AS
value")
    .writeStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "localhost:9092")
    .option("topic", "resultTopic")
    .start()
    .awaitTermination()

```

Lưu ý, cần tạo thêm topic resultTopic trong Kafka để ghi lên.

Chạy consumer để đọc dữ liệu đã được viết lên Kafka:

```

kafka-console-consumer.sh --broker-list localhost:9092 --topic
resultTopic

```

## 5. Bài tập

**Bài tập 1:** Thực hiện nhận một JSON từ topic Kafka và ghi dữ liệu này lên một topic khác sử dụng chế độ append. Lược đồ JSON như sau:

```

val schema = new StructType()
    .add("id", IntegerType)
    .add("firstname", StringType)
    .add("middlename", StringType)
    .add("lastname", StringType)
    .add("dob_year", IntegerType)
    .add("dob_month", IntegerType)
    .add("gender", StringType)
    .add("salary", IntegerType)

```

```

ubuntu@namenode:~/kafka$ bin/kafka-console-producer.sh --broker-list localhost:9092 --topic json_topic
>{"id":1,"firstname":"James ","middlename":"","lastname":"Smith","dob_year":2018,"dob_month":1,"gender":"M","salary":3000}
>{"id":2,"firstname":"Michael ","middlename":"Rose","lastname":"","dob_year":2010,"dob_month":3,"gender":"M","salary":4000}
>{"id":3,"firstname":"Robert ","middlename":"","lastname":"Williams","dob_year":2010,"dob_month":3,"gender":"M","salary":4000}
>{"id":4,"firstname":"Maria ","middlename":"Anne","lastname":"Jones","dob_year":2005,"dob_month":5,"gender":"F","salary":4000}
>{"id":5,"firstname":"Jen","middlename":"Mary","lastname":"Brown","dob_year":2010,"dob_month":7,"gender":"","salary":-1}

```

**Bài tập 2:** Thực hiện streaming trên nguồn và đích Redis.

