# Streaming Data Processing TD 4

Minh NGUYEN

November 13th, 2023

**Abstract**

In this programming assignment, we will continue to study the streaming data processing with Spark Structured Streaming and Kafka source.

## 1 Summary

## 2 Introduction

### 2.1 Prerequisites

- Programming language and SDK:
  - Python 3
  - Java 11/13/15/17
- IDE such as IntelliJ (Java) or VS Code / PyCharm with Python.
- Spark 3.5.0

## 3 Install Spark

### 3.1 Windows 11:

Follow  this tutorial .

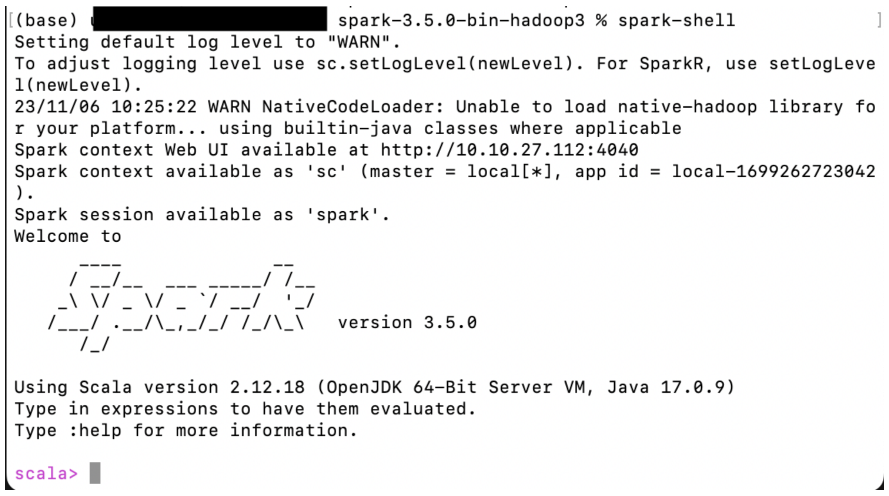
### 3.2 Other than Windows:

- Download Spark from this link: spark-3.5.0-bin-hadoop3.tgz.
- Extract the downloaded compressed file to a folder as you like.
- Config the PATH environment:
  - If you're using MacOS: do the following steps:
    1. Open ~/*.zshrc* by an editor (can use Vim)
    2. Add the following lines in the end of the file:

```
export SPARK_HOME="<your-path-to-spark-folder>/spark-3.5.0-bin-hadoop3"
export PATH="$SPARK_HOME:$SPARK_HOME/bin:$PATH"
```

    3. Apply this change by running this command in the terminal:

```
source ~/.zshrc
```



```
(base) [redacted] spark-3.5.0-bin-hadoop3 % spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLeve
l(newLevel).
23/11/06 10:25:22 WARN NativeCodeLoader: Unable to load native-hadoop library fo
r your platform... using builtin-java classes where applicable
Spark context Web UI available at http://10.10.27.112:4040
Spark context available as 'sc' (master = local[*], app id = local-1699262723042
).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.0
      /_/

Using Scala version 2.12.18 (OpenJDK 64-Bit Server VM, Java 17.0.9)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

Figure 1: Spark shell interface

4. Testing by opening another terminal and running this command. If Spark shell appears, then the installation can be said as done.

```
spark-shell
```

## 4 Setup environment for Python

A virtual environment is a directory that contains a specific collection of Python packages that you have installed. For example, you may have one environment with NumPy 1.7 and its dependencies, and another environment with NumPy 1.6 for legacy testing. If you change one environment, your other environments are not affected. You can easily activate or deactivate environments, which is how you switch between them. Working with separate environments help you:

- Profesionalize your workflow, in which Python dependencies can be fulfilled inside an environement
- Separate one environement from another, to avoid the dependencies conflict.

To create virtual environments, you can use either **venv** or **conda**. In this LAB, we will use conda.

### 4.1 Install Conda

- Install Miniconda from this link: https://docs.conda.io/projects/miniconda/en/latest/

### 4.2 Managing environments

Full list of tutorial can be found here :

Creating an environment with required packages (pyspark):

```
conda create -n <environment_name> python=3.10 pyspark
```

Activate the environment:

```
conda activate <environment_name>
```

## 5 LAB 4

### 5.1 Context

In this LAB we will try to connect Spark Structured Streaming with your previous Kafka topic (from generated data in LAB 2) and do basic task: WordCount. Then we will do some analysis on the data stream.

### 5.2 Lab report

You are required to **submit a Lab report** for the following assignments:

**EXERCISE 1:** Use the code sample from here to connect with your Kafka cluster and Kafka topic in the previous LAB, and do the wordcount.
Do the following steps and write down your observation in Lab report:

- Add new message to your Kafka topic manually. What will happen in the terminal of Spark application?
- [**Code**] Write a program with **Kafka Producer** to publish to your Kafka topic at various rate from 1-10 message per second. The program should publish a pseudo e-commerce sales data with the following information:
  - order_id
  - order_product_name
  - order_card_type
  - order_amount
  - order_datetime
  - order_country_name
  - order_city_name
  - order_ecommerce_website_name

  Attach the program code to your lab report. Now run this program to get Kafka topic updated regularly, and answer the next question.
- What does the *trigger* means? What will happen if you change the processingTime in trigger to 5, 10 and 30 seconds?
- Do the following analysis:
  - Find total order amount by country and city

**EXERCISE 2:** Mimic a stream of data by reading from a file, and do streamed aggregation report using Spark Structured Streaming.
Let's have some warm-up:

- Open TD_samples_count.py and run the application. Is the program functionning?
- Replace outputMode '*append*' by '*complete*'. Is it working now?
- Write in the report your observation and the difference between append / complete mode.

Now do some analysis and write your observation/code into your report: Data is already provided in *TD4/data/adidas/adidas_stream*. To mimic the stream, we will read from another folder, named *TD4/data/adidas/stream/*, which has only one file at the moment (file **aaa**). Every coming stream will correspond to copying a file from **adidas_stream/** to **stream/**, using the following command:

```
cp adidas_stream/aab stream/
```

- Write for each query a streaming application, to answer the following demand:
  - List top 5 products having highest reviews and least expensive in unit price.
  - List top 5 products having the biggest percentage of discount (selling_price vs original_price)
- We'll copy a find to **stream/** folder once per second. What will you observe if you change the processingTime in trigger to 5, 10 and 30 seconds?

**Notes:**

```
"""
Consumes messages from one or more topics in Kafka and does wordcount.
 Usage:

python structured_kafka_wordcount.py <bootstrap-servers> <subscribe-type> <topics>
   <bootstrap-servers> The Kafka "bootstrap.servers" configuration. A
   comma-separated list of host:port.
   <subscribe-type> There are three kinds of type, i.e. 'assign', 'subscribe',
   'subscribePattern'.
   |- <assign> Specific TopicPartitions to consume. Json string
   |  {"topicA":[0,1],"topicB":[2,4]}.
   |- <subscribe> The topic list to subscribe. A comma-separated list of
   |  topics.
   |- <subscribePattern> The pattern used to subscribe to topic(s).
   |  Java regex string.
   |- Only one of "assign, "subscribe" or "subscribePattern" options can be
   |  specified for Kafka source.
   <topics> Different value format depends on the value of 'subscribe-type'.
"""
```