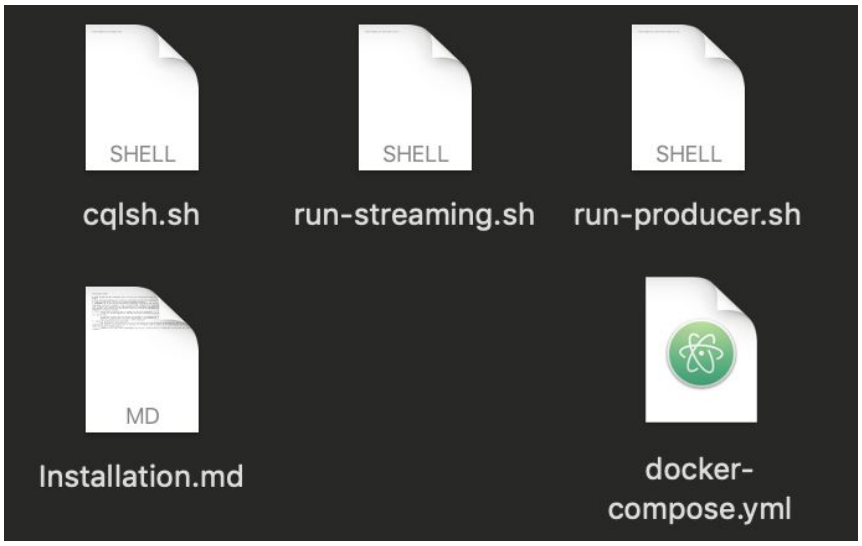
Marina Angelovska & Davide Di Matteo → Cevapcici

Lab2 Streaming - Part 1

Setup

In order to run the code and connect all the components (eg: Kafka, Cassandra) a Docker based environment has been used. Different containers for different scopes have been set-up. In the first terminal shell the `docker-compose up` command has been used, secondly in other three terminal shells we have run respectively: Cassandra, the Kafka Producer and our code. For more details on how to run please check the submission folder, you will find:



Note also that the `build.sbt` inside the `sparkstreaming` directory has been changed and it will be attached in the submission folder.

Code

The code can be divided into four main blocks: SparkStreaming setup, Spark-Cassandra interaction, Kafka-SparkInteraction, Stateful Transformation.

1. SparkStreaming setup

In this section we cover how our SparkStreaming application has been set-up. We have created aso `SparkConf`, a `StreamingContext` and a checkpoint. These elements are necessary for a SparkStreming application.

2. Spark-Cassandra Interaction

In this part of the code, we took care of the interaction between our Cassandra configuration and our SparkStreaming application. We have created the connection to listen to the cassandra host server `cassandra` (this is not localhost due to the Docker environment) on port `9042`. Subsequently, as specified in the guide, we have created a keyspace and a table with two columns to store our *(key, avg)* pair. The last line of the entire code is the data-storing process, achieved with the following: `stateDstream.saveToCassandra("avg_space", "avg", SomeColumns("word", "count"))`.

3. Spark-Kafka Interaction

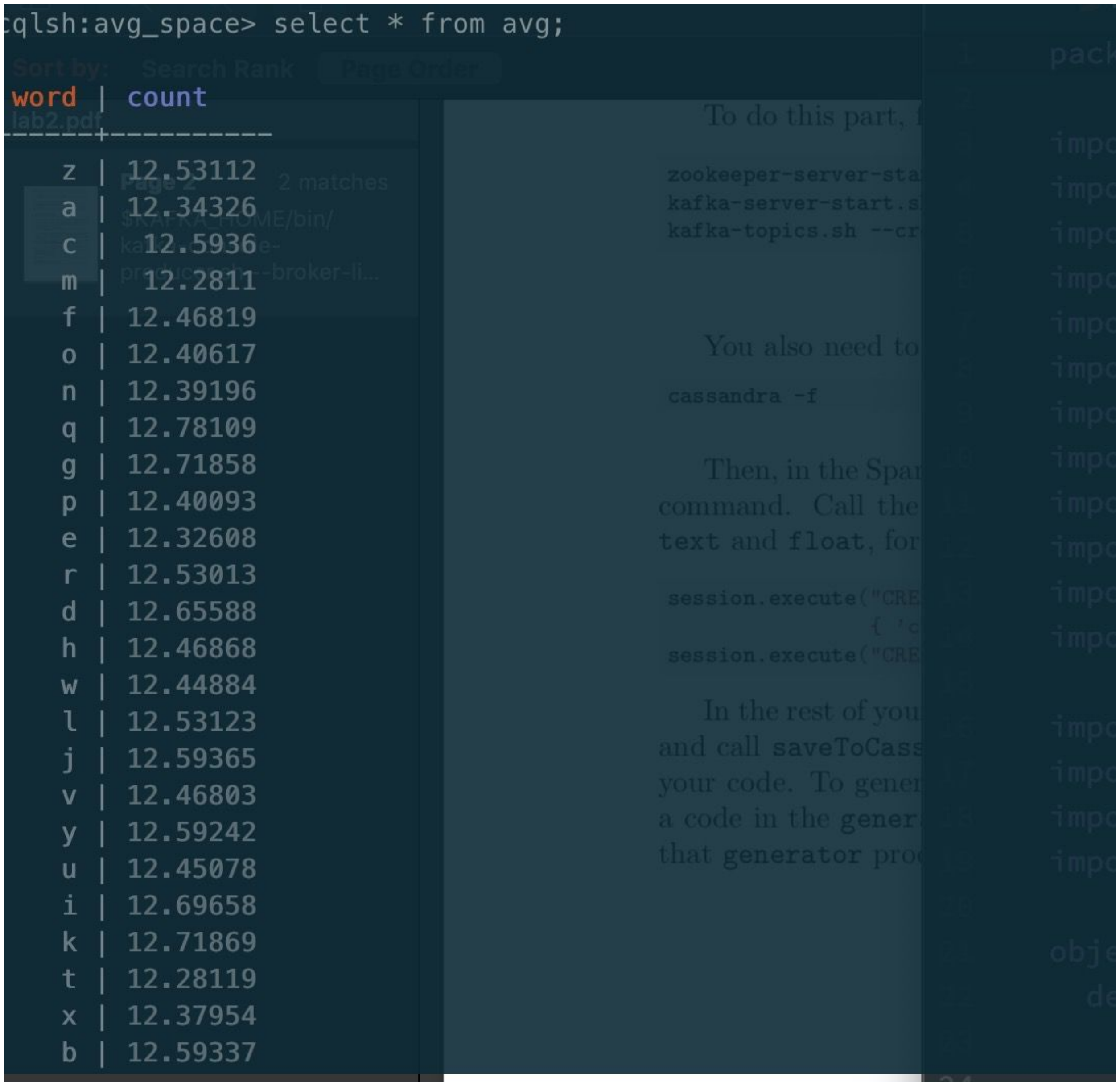
In this block, we have mapped various different configuration parameters needed by Kafka and we have created a `List("avg")` that represent our topic that we will listen from the Kafka producer through our application. After this, we passed these newly created entries and the *StreamingContext* `scc` to `kafkaUtils.createDirectStream` function in order to implement a receiver-less approach. In this approach, instead of using receivers to receive data, Spark periodically queries Kafka for the latest offsets in each topic+partition, and accordingly defines the offset ranges to process in each batch. Lastly, a parser function has been created to format and typecast the pairs of elements gathered via our application from the producer.

4. Stateful Transformation

In this last block, a stateful transformation has been implemented. In the specific, we will compute the average value by key. To achieve this, I have followed the example presented on the slides with some modification. Note that the *state* input definition in the skeleton has been changed to `state: State[(Double, Int)]` in order to record the sum and the counter.

5. Results

In this section, you can see a query to our cassandra database in the CQL shell. The following picture reports the table previously created with the word and their average occurrences.



```
cqlsh:avg_space> select * from avg;

 word | count
------+----------
    z | 12.53112
    a | 12.34326
    c |  12.5936
    m |  12.2811
    f | 12.46819
    o | 12.40617
    n | 12.39196
    q | 12.78109
    g | 12.71858
    p | 12.40093
    e | 12.32608
    r | 12.53013
    d | 12.65588
    h | 12.46868
    w | 12.44884
    l | 12.53123
    j | 12.59365
    v | 12.46803
    y | 12.59242
    u | 12.45078
    i | 12.69658
    k | 12.71869
    t | 12.28119
    x | 12.37954
    b | 12.59337
```