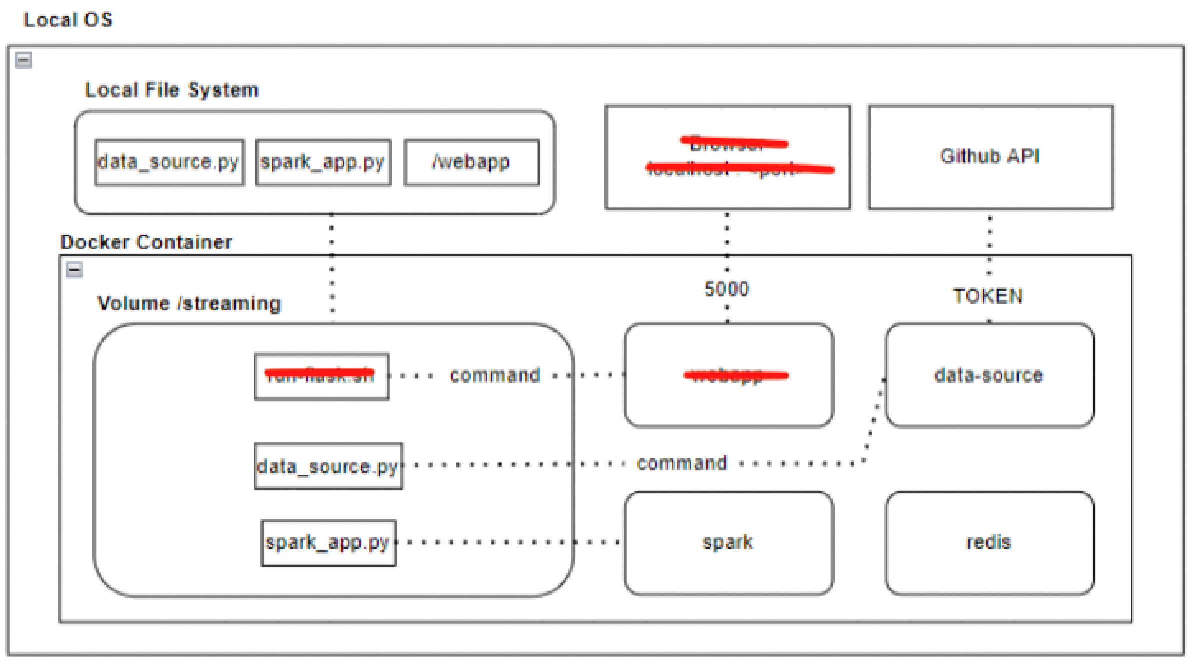
EECS 4415: Project 3
Real-Time Streaming Analytics with Apache Spark and Python

Nabi Khalid   216441677
10-04-2023

System Architecture

Below, the system architecture of a data streaming application can be seen. However *in my submission the flask web app is not implemented, everything else is fully functional*. The outer layer consists of the operating system being used. Within this, a Docker container is launched, which initiates multiple containers for Spark, Redis, and the github data source. Additionally, the container mounts necessary files from the local file system into a streaming folder using volumes. Docker then carries out commands from the docker-compose file to initiate the data source which employs a predefined token variable and awaits the Spark application to begin retrieving data from the Github API.



The system is composed of two main components (webapp service not implemented), a Python data source script and a Spark streaming application (pyspark). The data source script collects information about the most recently pushed repositories from GitHub API for three programming languages: Python, Java, and C. It then pushes the collected data to the Spark application in JSON format every 15 seconds.

The Spark streaming application receives the data sent by the data source script, processes it in batches at an interval of 60 seconds, and performs the following four analysis tasks then prints the batch results in the spark application streaming logs after every iteration:

1. Compute the total number of the collected repositories since the start of the streaming application for each of the three programming languages.

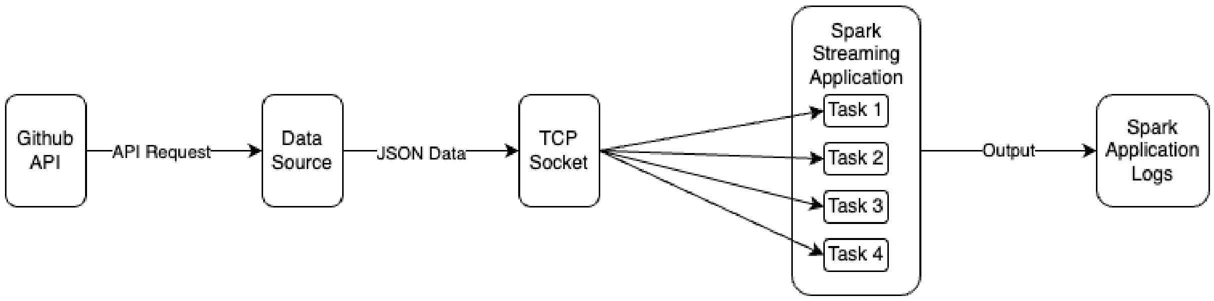
2. Compute the number of the collected repositories with changes pushed during the last 60 seconds.
3. Compute the average number of stars of all the collected repositories since the start of the streaming application for each of the three programming languages.
4. Find the top 10 most frequent words in the description of all the collected repositories since the start of the streaming application for each of the three programming languages.

Interaction between Services

The entire system begins running after the execution of the following commands in the spark environment specified in Lab 7:

```
$ docker-compose up
$ docker exec streaming_spark_1 spark-submit /streaming/spark_app.py
```

The data source script runs in the background after the execution of the above commands as specified in the docker-compose.yaml file and waits to secure a TCP connection with the Spark streaming application before it begins pulling data from the Github API. When the data is pulled from the Github API (every 15 seconds) it is sent as JSON to the Spark streaming application using a TCP socket. Once received, the spark application then processes the data and produces an output batch every 60 seconds which is printed in the logs. Details regarding the processing within the Spark streaming application can be found in the following section.



Spark Application

The Spark application processes the streaming data using PySpark Streaming. The data stream is connected to the data source via a *socketTextStream*, which reads the JSON data sent by the data source script. Data is processed in batches of 60 seconds and uses stateful transformations to compute and update the required metrics. Checkpointing is an important feature utilized to ensure fault tolerance and enable recovery in case of failures.

The application completes the following four analysis tasks and prints them to the logs as follows:

Task 1: Compute the total number of collected repositories for each language since the start of the streaming application. This task uses the *updateStateByKey* transformation to maintain a running count of repositories for each language.

Task 2: Compute the number of collected repositories with changes pushed during the last 60 seconds. This task processes each RDD in the DStream and filters the repositories based on the "pushed_at" timestamp.

Task 3: Compute the average number of stars of all collected repositories for each language since the start of the streaming application. This task uses the *updateStateByKey* transformation to maintain a running sum and count of stars for each language, and then computes the average by dividing the sum by the count.

Task 4: Find the top 10 most frequent words in the description of all collected repositories for each language since the start of the streaming application. This task tokenizes the repository descriptions, maps the words to their respective languages, and uses the *updateStateByKey* transformation to maintain a running count of word frequencies. It then groups the words by language and sorts them in descending order to obtain the top 10 most frequent words for each language.

The results of the analysis tasks are printed as pandas DataFrames for each batch interval in the logs of the Spark application. Below are the results of 2 hours of streaming (start timestamp = 2023-04-08 20:52:00):

```
-------------------------------------------
Time: 2023-04-08 22:52:00
-------------------------------------------
Total number of repositories by language (since the start):
Language  Value
  Python  10663
    Java   8699
       C   2087
Number of repositories with changes pushed during the last 60 seconds:  150
-------------------------------------------
Time: 2023-04-08 22:52:00
-------------------------------------------
Average stars by language (since the start):
Language    Value
  Python 0.415080
    Java 0.275434
       C 0.593675
-------------------------------------------
Time: 2023-04-08 22:52:00
```

```
-------------------------------------------
Top 10 most frequent words by language (since the start):
Language        Word  Count
  Python           a   1297
  Python          to   1270
  Python         the   1244
  Python         and   1050
  Python         for   1037
  Python          of    963
  Python      python    918
  Python          in    628
  Python        with    574
  Python        this    462
    Java        java    796
    Java          de    605
    Java         for    485
    Java           a    425
    Java          by    339
    Java     created    327
    Java      github    317
    Java   classroom    313
    Java         the    302
    Java         and    284
       C           a    475
       C           c    335
       C         for    191
       C         the    181
       C          of    163
       C          to    162
       C        with    160
       C      binary    143
       C         and    134
       C programming    131

2023-04-08 20:52:00:2023-04-08 22:52:00
Python:10663:0.415080
Java:8699:0.275434
C:2087:0.593675
Python:(a,1297),(to,1270),(the,1244),(and,1050),(for,1037),(of,963),(python,918),(in,
628),(with,574),(this,462)
Java:(java,796),(de,605),(for,485),(a,425),(by,339),(created,327),(github,317),(classr
oom,313),(the,302),(and,284)
C:(a,475),(c,335),(for,191),(the,181),(of,163),(to,162),(with,160),(binary,143),(and,13
4),(programming,131)
```