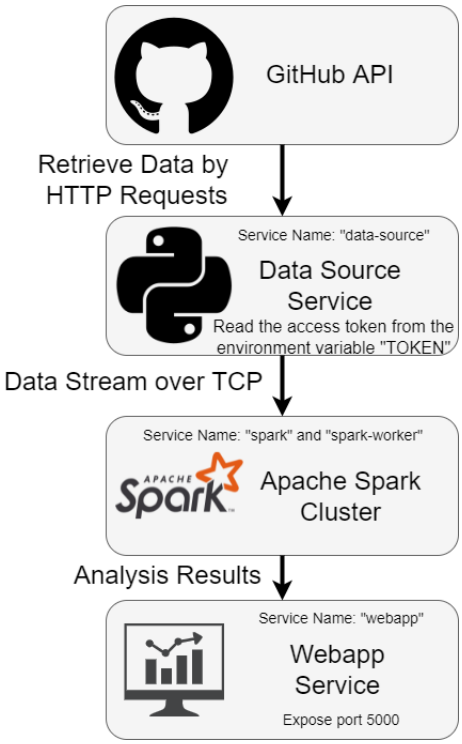
Hassan Mukhtar
215514797

Assignment 3 Report:

System Architecture:

data_source.py:

data_source.py calls the GitHub API URL and returns the response for a stream of GitHub data through HTTP requests. Next, we take this JSON object and extract the meaningful things to us, "language", “full_name”, “pushed_at”, “stargazers_count”, and “description”. After that it sends every useful data to our Spark Streaming instance through a TCP connection (spark_app.py). The main function makes the app host socket connections that spark will connect with. In our instance we configure our Ip to “0.0.0.0” (localhost) and this service to run on port 9999. We bind the connection to spark_app and wait for spark_app to form a TCP connection to send the data to spark_app.



spark_app.py:

First, we create SparkContext with sc then we create Streaming Context ssc from sc with a batch interval of 60 seconds that will do the transformation on all data received every 60 seconds. Then we define our “data” which will connect to the socket server we created in data_source.py on socket 9999 and read the data from that port. After the data is received, we split all the data by our delimiter (‘\t’) and store it in words. To remove duplicate “full_name”s, all indexes of the data besides “pushed_at” were re-mapped to one another. Then a reduceByKey was applied which would group everything with the same info besides “pushed_at”.

For 3i I first split words (the split data) into “LanguageName” (index[0]), 1. Then by executing a .reduceByKey on this operation, it groups all instances of each language name and their “1” value (adding their values when they group, e.g, (‘Python’, 1), (‘Python’, 1) when reduceByKey is operated is (‘Python’, 2)). Next updateStateByKey is applied to this operation which returns a new “state” where the state for each key is updated by applying the given function on the previous state of the key and the new values of the key. Lastly, this info is processed by a process_rdd function. In this function we do processing on the rdd in every batch in order to convert it into a table using Spark SQl Context and then perform a select statement in order to retrieve the Language with its Count and put them into new_results_df dataframe and send the information to send_df_dashboard which sends this to port 5000 for the webapp.

For 3ii I first run index of “full_name” through a function which checks for duplicates by appending all values to a list and then searching if they have been in a list before (this returns True or False at the index of “full_name”). With this information I repeat the process of 3i but with an if/else to check if it is a duplicate or not. I then repeat the same process_rdd step as 3i using process_rdd2.

Hassan Mukhtar
215514797

For 3iii: I use the split data to map the language name to its stargazers_count (e.g. ('Python', 35), ('Python', 12). Then I reduceByKey which would yield ('Python', 47). Next updateStateByKey is applied to this operation which returns a new "state" where the state for each key is updated by applying the given function on the previous state of the key and the new values of the key. Next I make a new map a new K, V pair which contains the language name and starsgazers_count from the operation that just occurred, and a another K, V pair by using the K, V pair from question 3i. I join these (K, V) pairs together since they both have the same keys, I join them at the keys and for their values I divide the stargazers_count by the total number of those language instances to get the average star count. I then repeat the same process_rdd step as 3i using process_rdd3.

For 3iv: I map the language with the description. However within this mapping I run a function called "splitter" on the description which strips the description per specifications, lower cases all the words and splits them at the ' ' (a space) to store them into a list. reduceByKey is used to group each languages list of words together from the description. Next, I remap the language name with this new list. On this new list I run a "counter_func" which uses "Counter" from collections to get the top 10 most occurrences of the list. Next updateStateByKey is applied to this operation which returns a new "state" where the state for each key is updated by applying the given function on the previous state of the key and the new values of the key. I then repeat the same process_rdd step as 3i using process_rdd4.

Send_df_to_dashboard is the last step which converts the data frame into a list and sends it through the REST API.

app.py (part of web_app service):

Checks to make sure there are no connection errors to the page.

flask_app.py(part of web_app service):

We create an updateData() function which will be called by Spark through the URL http://webapp:5000/updateData defined in spark_app.py. Also another function index() which gets all the data being sent by spark_app and loads the json data which we then apply basic operations on to index to our required information. A basic graph and chart are made using matplotlib and we return a value in index() which will have render a html page (index.html) and send it the updated labels.

index.html (part of web_app service):

Basic html page which displays our information that sent from flask_app.py in the html rendering command. Page is updated upon refresh.