SAN JOSÉ STATE UNIVERSITY

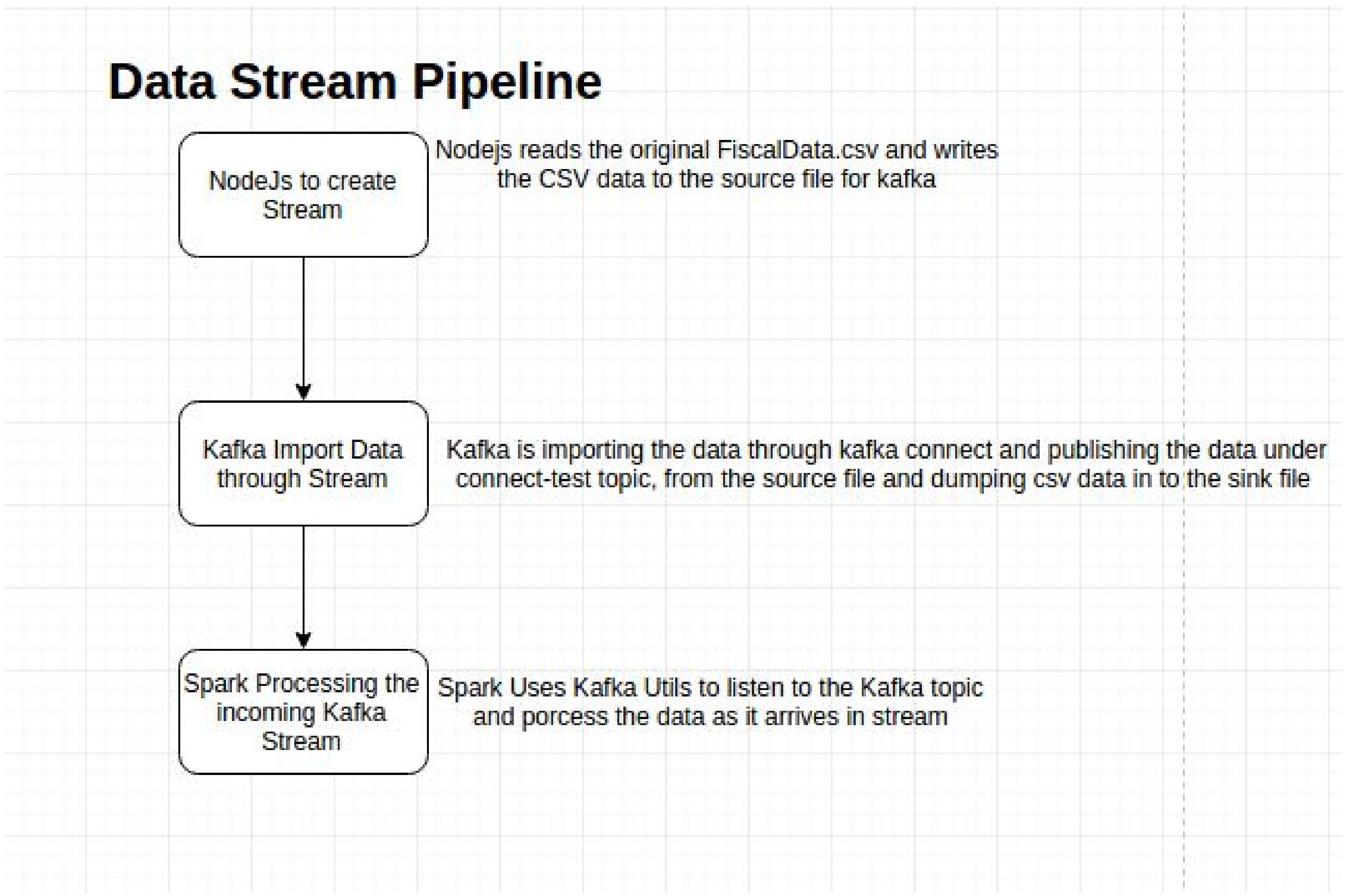
CMPE 272 - Extra Assignment

Country wise Government Spending Report

Name : Jayam Malviya

Student Id : 011435567

Github Link : https://github.com/midNight-jam/kafka_spark_streaming


Data Stream Pipeline
NodeJs to create Stream
Nodejs reads the original FiscalData.csv and writes the CSV data to the source file for kafka
Kafka Import Data through Stream
Kafka is importing the data through kafka connect and publishing the data under connect-test topic, from the source file and dumping csv data in to the sink file
Spark Processing the incoming Kafka Stream
Spark Uses Kafka Utils to listen to the Kafka topic and porcess the data as it arrives in stream

Installation

Kafka

1. Download kafka from https://kafka.apache.org/downloads.html

2. From the extracted location, Start the Zookeeper Server
   Using the below command from terminal

   bin/zookeeper-server-start.sh config/zookeeper.properties

3. Now, Start the Kafka Server. Using the below command

   bin/kafka-server-start.sh config/server.properties

4. Now before firing the import/export kafka command, we have to make changes in the below files to tell kafka about the source and destination of import.

   Go to the config folder, open “**connect-file-source.properties**” file and edit the file value to the file name from which we want import. In our case we will change this file to **fiscal.data.** We can also change the topic on which these import will be broadcasted, but for simplicity, let's keep it to default **connect-test** topic

   Now let’s also modify the “**connect-file-sink.properties**” file to let kafka know where to sink all the incoming data. For this, we will change the file value to **fiscal.sink.data.** This is a mandatory step, without its completion we will not be able to close the import/export loop.

5. With these configurations done we will now fire the import/export command for kafka from the terminal

   bin/connect-standalone.sh config/connect-standalone.properties config/connect-file-source.properties config/connect-file-sink.properties

6. In order to **test** if the configuration is working fine or not fire the below command to insert a test csv line in to our kafka source fiscal.data file.

   echo -e “This, shall, reflect, in, Fiscal.data, and, Fiscal.sink.data, ” >> fiscal.data

   If all the configs are correct this data should reflect in both fiscal.data and fiscal.sink.data files

Spark

1. To download spark in your system use this url

   http://spark.apache.org/downloads.html

2. After downloading and extracting spark, use below commands to setup spark for kafka streaming

```
export PATH=$PATH:/usr/local/spark/bin
```

3. Now we can submit the spark job, our python file to process the stream

   ```
   spark-submit  --packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.0.1
   Stream2.py --verbose >> sparklogs.txt
   ```

   If the spark Job is started successfully after submission and we can read the logs of job in sparklogs.txt file.

4. To be assure that spark is listening to the kafka stream, again use the same test command form kafka installation and insert a csv data in the kafka source file. If the spark stream is working expectedly then the newly added data should reflect in the sparklogs.txt file.

```
echo -e “This, shall, reflect, AGAIN  in, Fiscal.data, and,  Fiscal.sink.data, ” >>
fiscal.data
```

Nodejs

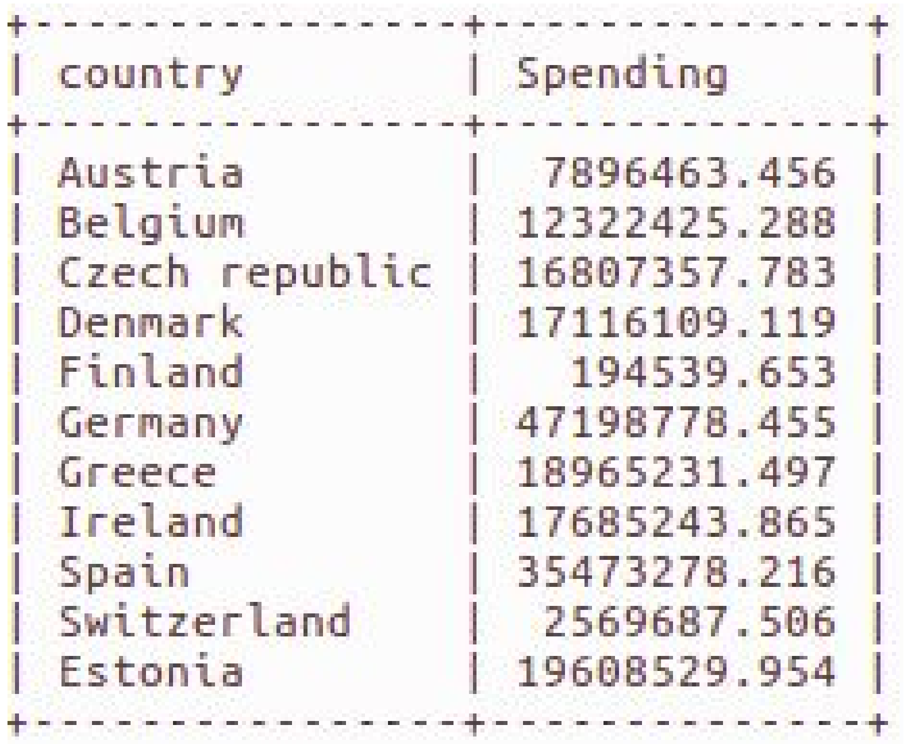
1. Now as the pipeline is tested for streaming, let fire our app.js file which read the actual fiscal.data csv file & inserts one line at a time in the kafka source file consequently creating a stream for kafka.

OpenSpendingApi

- We have use openspending api to get the fiscal data of few countries from Europe package it is a very large data set and contains about Hundred thousands transactions from several countries.

Data is processed as it arrives within the stream,in our case one transaction at a time. As a data is written in to fiscal.txt a kafka message is published under connect-test, this message is read and processed by spark using kafkaConnectStream . In code we keep printing the processed data till the time, that is for the data that has arrived till now. Below is the put when the stream has ended and there are no more writes in fiscal.data.

Output for countries with their spending as per the csv data


```
+-----------------+--------------+
| country         | Spending     |
+-----------------+--------------+
| Austria         |  7896463.456 |
| Belgium         | 12322425.288 |
| Czech republic  | 16807357.783 |
| Denmark         | 17116109.119 |
| Finland         |   194539.653 |
| Germany         | 47198778.455 |
| Greece          | 18965231.497 |
| Ireland         | 17685243.865 |
| Spain           | 35473278.216 |
| Switzerland     |  2569687.506 |
| Estonia         | 19608529.954 |
+-----------------+--------------+
```

Tools used : Pycharm, Gedit, python 2.7, kafka version 2.11, Spark Version 2.0

Code Listing : Stream2.py

```python
from __future__ import print_function
import sys
import json
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
from pyspark.streaming.kafka import KafkaUtils
from pyspark.sql import Row, SparkSession
from pyspark.sql.types import *
from pyspark.sql import SQLContext, Row



def getSparkSessionInstance(sparkConf):
  if ('sparkSessionSingletonInstance' not in globals()):
    globals()['sparkSessionSingletonInstance'] = SparkSession\
      .builder\
      .config(conf=sparkConf)\
      .getOrCreate()
  return globals()['sparkSessionSingletonInstance']




sc = SparkContext("local[2]","NetWordCount")
ssc = StreamingContext(sc,1)
sqlContext = SQLContext(sc)

topic  = "connect-test"
kvs = KafkaUtils.createStream(ssc,"localhost:2181","spark-streaming-consumer",{topic:1})
# words = kvs.map(lambda x:x[1])
parsed = kvs.map(lambda (key, value): json.loads(value))

# words = kvs.map(lambda line: line.split(","))
# words = kvs.flatMap(lambda line: line.split(" "))

# Convert RDDs of the words DStream to DataFrame and run SQL query
def process(time, rdd):
  print("========= %s =========" % str(time))

  try:
    # Get the singleton instance of SparkSession
    spark = getSparkSessionInstance(rdd.context.getConf())

    print(rdd.take(1))

    # Convert RDD[String] to RDD[Row] to DataFrame
    # parts = rdd.map(lambda line: json.load(line,encoding="UTF-8"))

    transactions = rdd.map(lambda p: p['payload'])

    records = transactions.map(lambda p: p.split(","))

    rowRecord = records.map(lambda p: Row(location=p[0],country=p[1],transact=p[2],transaction=p[3], activity=p[4], \
                        function=p[5], sector=p[6], sectorExp=p[7], measure=p[8], measureExp=p[9],\
                        time=p[10], year=p[11], unitCode=p[12], unit=p[13], powerCodeCode=p[14] \
                        , powerCode=p[15], referencePeriodCode=p[16], refercePeriod=p[17] \
                        , value=p[18], flagCodes=p[19], flags=p[20]))


    for x in records.collect():
      print(x)

    for y in rowRecord.collect():
      print(y)

    transactionsDataFrame = spark.createDataFrame(rowRecord)
    changeTypedDef = transactionsDataFrame.withColumn("valueDouble",transactionsDataFrame["value"].cast("double"))

    # transactionsDataFrame.createOrReplaceTempView("alltransactions")
    changeTypedDef.createOrReplaceTempView("alltransactions")

    results = spark.sql("SELECT * FROM alltransactions")
    results.show()

    resultsByCountry = spark.sql("SELECT country, SUM(valueDouble) as Spending FROM alltransactions group by country")
    resultsByCountry.show()

  except:
    pass

# parsed.pprint()
parsed.foreachRDD(process)

ssc.start()
ssc.awaitTermination()
```

==

App.js

```
var fs = require('fs');
var path = "/home/jayam/Downloads/kafka_2.11-0.10.0.0/fiscal.txt";

function writeToFile(data) {
   fs.appendFile(path, "\n"+data, function(err) {
       if(err) {
           return console.log(err);
       }
   });

}
//
// for(i=0;i<50;i++){
//     // writeToFile(' "some", "data", "from", "ZZZZ", "ZZZZ", "ZZZZ", "ZZZZ", "ZZZZ", "ZZZZ", "ZZZZ",
"ZZZZ"   ');
//     writeToFile('"KORZZZ","KoreaZZZ","D62_D631XXCGZZZ","Social benefits & transfers in kind -
purchased market production, payableZZZ","050ZZZ","Environment protectionZZZ","GS1312ZZZ","State
ZZZgovernmentZZZ","CZZZ","Current
pricesZZZ","2009ZZZ","2009ZZZ","KRWZZZ","WonZZZ","6ZZZ","MillionsZZZ",,,0,,');
// }

readline = require("readline");
```

```
var file = "dataShort.csv";
var cursorY =0;
var rl = readline.createInterface({
   input: fs.createReadStream(file),
   output: null,
   terminal: false
})

rl.on("line", function(line) {
   console.log(cursorY+": " + line);
   writeToFile(line);
   cursorY++;
});

rl.on("close", function() {
   console.log("All data processed, Lines Read "+cursorY);
});
```