1)

Flume Configuration file for Extracting Twitter Data About covid-19

Save this file as twitter-to-hdfs.properties

agent1.sources =source1

agent1.sinks = sink1

agent1.channels = channel1

agent1.sources.source1.channels = channel1

agent1.sinks.sink1.channel = channel1

agent1.sources.source1.type = org.apache.flume.source.twitter.TwitterSource

agent1.sources.source1.consumerKey = Br7dImahUyebJ79adj11

agent1.sources.source1.consumerSecret = Kjsin23UI7ghTY8sb

agent1.sources.source1.accessToken = G4Hnudo67Gheijnll8Jil0

agent1.sources.source1.accessTokenSecret =hxeyABkOwh23Ah6Etgh77

agent1.sources.source1.keywords = covid-19

agent1.sinks.sink1.type = hdfs

agent1.sinks.sink1.hdfs.path = /flume/twitter

agent1.sinks.sink1.hdfs.filePrefix = events

agent1.sinks.sink1.hdfs.fileSuffix = .log

agent1.sinks.sink1.hdfs.inUsePrefix = \_

agent1.sinks.sink1.hdfs.fileType = DataStream

agent1.channels.channel1.type = memory

agent1.channels.channel1.capacity = 1000

2)

Sqoop job for extracting data from mysql and put In Hive Warehouse

sqoop import \

--connect jdbc:mysql://localhost:8080/FoodDb \

--username=akshay \

--password=password \

--table=food \

--hive-import \

--hive-table=food\_items \

--target-dir /mysql/table/food\_items \

--m 1

3)

--Consider a file named source.csv having data on airlines

🡪Loading the file

airlinePath= "hdfs:///spark/rdd/source.csv"

🡪Reading the file by creating an RDD

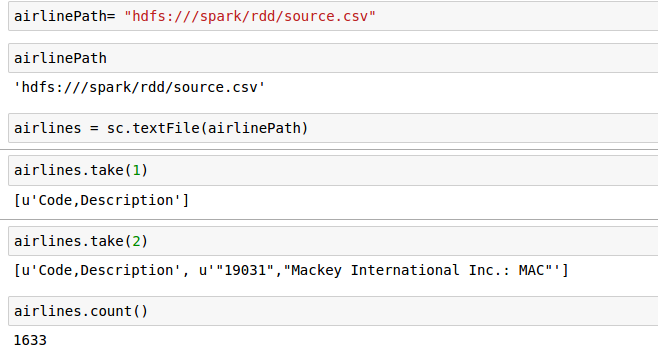
airlines = sc.textFile(airlinePath)

🡪The "take()" function

airlines.take(5) -- the 1st 5 rows r printed

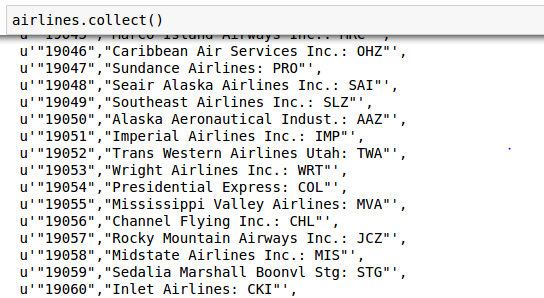
The count function

airlines.count() - Returns the total no. of rows.



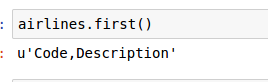
🡪The collect function

airlines.collect() - returns all the elements in the RDD.



🡪The first() function

airlines.first() - The 1st row is printed



4)

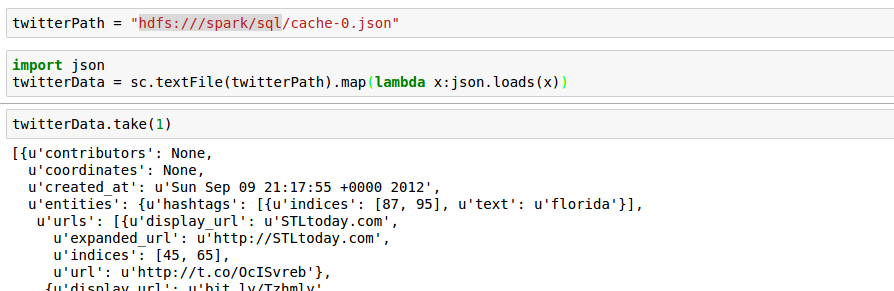
🡪 Loading the file

twitterPath = "hdfs:///spark/sql/source.json"

🡪 Loading data from json into RDD

import json

twitterData = sc.textFile(twitterPath).map(lambda x:json.loads(x))



🡪 SQLContext has a method to directly read JSON files

from pyspark.sql import SQLContext,Row

sqlC = SQLContext(sc)

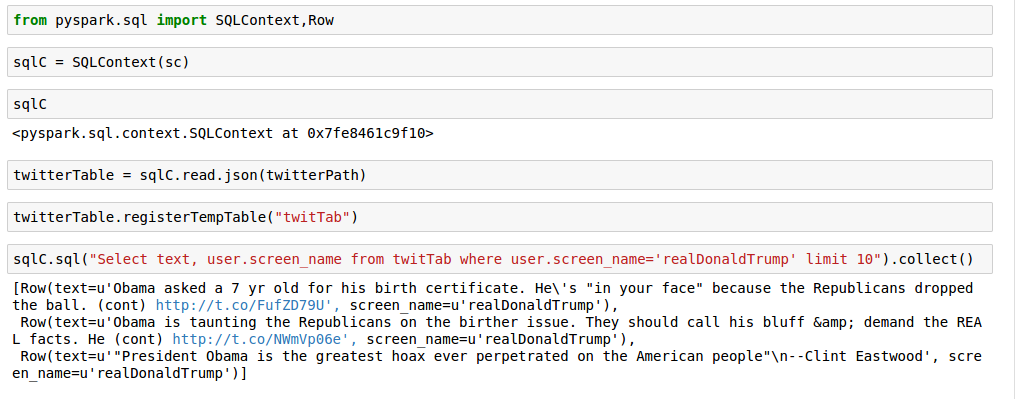
🡪 Register this DataFrame as a table with the SQLContext

twitterTable = sqlC.read.json(twitterPath)

twitterTable.registerTemplate("twitTab")

🡪 The SQLContext has an SQL method to which you’ll give your SQL query

sqlC.sql("Select text, user.screen\_name from twitTab where user.screen\_name='realDonaldTrump' limit 10").collect()



5)

🡪Spark Streaming Python Program to Count Error in real Time

#Importing required functions from libraries

from pyspark import SparkContext

from pyspark.streaming import StreamingContext

sc = SparkContext("local[2]", "StreamingErrorCount")

ssc = StreamingContext(sc,10)

ssc.checkpoint("hdfs:///spark/streaming")

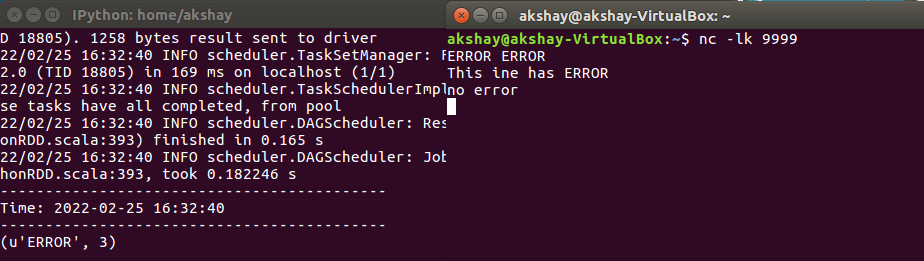
ds1 = ssc.socketTextStream("localhost",9999)

count = ds1.flatMap(lambda x:x.split(" ")).filter(lambda word:"ERROR" in word).map(lambda word:(word,1)).reduceByKey(lambda x,y:x+y)

count.pprint()

ssc.start()

ssc.awaitTermination()



6)

🡪 loading data into RDD

flightsPath = "hdfs:///spark/rdd/flights.csv"

flightsData = sc.textFile(flightsPath)

🡪Taking care of the missing values

blanks = flightData.map(lambda x:’,’.join(x or ’00.00’ for x in x.split(‘,’)))

blanktime = blanks.map(lambda x:x.replace(‘,””’,’,”0000”’))

finalF = blanktime

f = finalF.filter(notHeader).map(split)

🡪Parsing data using python function “parse”

from datetime import datetime

from collections import namedtuple

fields = ('date','airline','flightnum','origin','dest','dep','dep\_delay','arv','arv\_delay','airtime','distance')

Flight = namedtuple('Flight',fields, verbose=False)

DATE\_FMT = '%Y-%m-%d'

TIME\_FMT = '%H%M%S'

def parse(row):

row[0] = datetime.strptime(row[0], DATE\_FMT).date()

row[5] = datetime.strptime(row[5], TIME\_FMT).time()

row[6] = float(row[6])

row[7] = datetime.strptime(row[7], TIME\_FMT).time()

row[8] = float(row[8])

row[9] = float(row[9])

row[10] = float(row[10])

return Flight(\*row[:11])

fp = f.map(parse)

fp.filter(lambda x:x.dep\_delay>0).count()/float(fp.count())

sumCount = fp.map(lambda x:x.dep\_delay).aggregate((0,0),(lambda acc,value: (acc[0]+value, acc[1]+1)),(lambda acc1,acc2:(acc1[0]+acc2[0],acc1[1]+acc2[1])))

print "The average delay is "+str(sumCount[0]/float(sumCount[1]))

