SPARK

wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

wget https://centos7.iuscommunity.org/ius-release.rpm

rpm -ivh epel-release-latest-7.noarch.rpm

rpm -ivh ius-release.rpm

yum install python36u.x86_64

yum install python36u-pip

pip3.6 install jupyter

jupyter notebook --allow-root --ip=0.0.0.0

rpm -ivh jdk-8u144-linux-x64.rpm

wget https://downloads.lightbend.com/scala/2.12.3/scala-2.12.3.rpm

rpm -ivh scala-2.12.3.rpm

scala -version

Connect python with scala

pip3.6 install py4j

tar -xvzf spark-2.2.0-bin-hadoop2.7.tgz

mv spark-2.2.0-bin-hadoop2.7 /spark

export SPARK_HOME=/spark/

export PATH=/spark/bin/:/spark/sbin/:$PATH

export PYTHONPATH=/spark/python/:$PYTHONPATH

export PYSPARK_DRIVER_PYTHON="jupyter"

export PYSPARK_DRIVER_PYTHON_OPTS="notebook"

export PYSPARK_PYTHON=python3.6

export SPARK_LOCAL_IP="0.0.0.0"

cd /spark/python

jupyter notebook --allow-root --ip=0.0.0.0

pip3.6 install findspark

##

 # export SPARK_LOCAL_IP="0.0.0.0"

[root@node1 python]# pwd

/spark/python

[root@node1 python]# cat my.json

{ "age": 26, "name": "vimal" }

{ "age": 24, "name": "rahul" }

{ "name": "krish" }

[root@node1 python]# vim basic.py

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Basics').getOrCreate()

df = spark.read.json('my.json')

df.show()

df.printSchema()

df.columns

df.describe()

df.describe().show()
```

[root@node1 python]# cat basic.py

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Basics').getOrCreate()

df = spark.read.json('my.json')

df.show()

type(df['age'])

df.select('age').show()
```

type(df.select('age'))

df.head(2)

df.select(['age', 'name']).show()

df.withColumn('newage' , df['age']*2).show()

df.withColumnRenamed('age', 'my_new_age').show()

---

from pyspark.sql.types import StructField, StringType, IntegerType, StructType

data_schema = [StructField('age, IntegerType(),True'), StructField('name',StringType(), True)]

final_struc = StructType(fields=data_schema)

df = spark.read.json('my.json', schema=final_struc)

df.printSchema()

https://spark.apache.org/docs/latest/sql-programming-guide.html

---

[root@node1 python]# cat sql.py

from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('Basics').getOrCreate()

df = spark.read.json('my.json')

df.createOrReplaceTempView('people')

results = spark.sql("SELECT * FROM people")

results.show()

results = spark.sql("SELECT * FROM people WHERE age=26")

results.show()

Cd /spark/bin

./pyspark

```
>>> a= sc.textFile('../../spark/examples/src/main/resources/people.json')
>>> a
>>> a.take(10)
>>> a.count()
>>> a.first()
>>> a.collect()
----------
>>> user=sc.textFile('/etc/passwd')
>>> user.first()
u'root:x:0:0:root:/root:/bin/bash'
>>>
>>>
>>> bashuser=user.filter(lambda x: "bash" in x)
>>> bashuser
PythonRDD[41] at RDD at PythonRDD.scala:48
>>> bashuser.first()
u'root:x:0:0:root:/root:/bin/bash'
>>> bashuser.collect()
[u'root:x:0:0:root:/root:/bin/bash']
>>> bashuser.collect()
[u'root:x:0:0:root:/root:/bin/bash', u'harry:x:1000:1000::/home/harry:/bin/bash']
>>> numbersRDD = sc.parallelize([1,2,3,4])
>>> squareRDD = numbersRDD.map(lambda x:x*x).collect()
>>> for i in squareRDD:
```

```
... print i

...

1

4

9

16

>>> numbersRDD = sc.parallelize([1,2,3,4])

>>> filterRDD = numbersRDD.filter(lambda x:(x !=1)).collect()

>>> for i in filterRDD:

... print i

...

2

3

4

>>> linesRDD = sc.parallelize(["hellow world", "how are you"])

>>> wordsRDD = linesRDD.flatMap(lambda x:x.split(" ")).collect()

>>> for w in wordsRDD:

... print w

...

hellow

world

how

are

you

[root@node1 ~]# /spark/bin/pyspark --master local[2]

>>> path = "file:///root/nyc.csv"
```

```
>>> data = sc.textFile(path)

>>> data

file:///root/nyc.csv MapPartitionsRDD[3] at textFile at NativeMethodAccessorImpl.java:0

>>> data.take(2)

[u'OBJECTID,Identifier,Occurrence Date,Day of Week,Occurrence Month,Occurrence

Day,Occurrence Year,Occurrence Hour,CompStat Month,CompStat Day,CompStat

Year,Offense,Offense

Classification,Sector,Precinct,Borough,Jurisdiction,XCoordinate,YCoordinate,Location 1',

u'1,f070032d,09/06/1940 07:30:00

PM,Friday,Sep,6,1940,19,9,7,2010,BURGLARY,FELONY,D,66,BROOKLYN,N.Y. POLICE

DEPT,987478,166141,"(40.6227027620001, -73.9883732929999)"']

>>> header = data.first()

>>> header

u'OBJECTID,Identifier,Occurrence Date,Day of Week,Occurrence Month,Occurrence

Day,Occurrence Year,Occurrence Hour,CompStat Month,CompStat Day,CompStat

Year,Offense,Offense

Classification,Sector,Precinct,Borough,Jurisdiction,XCoordinate,YCoordinate,Location 1'

>>> dataWithoutHeader = data.filter(lambda x: x <> header)

>>> dataWithoutHeader.first()

>>> dataWithoutHeader.map(lambda x:x.split(",")).take(10)

>>> import csv

>>> from StringIO import StringIO

>>> from collections import namedtuple

>>> fields = header.replace(" " , "_").replace("/","_").split(",")

>>> fields

>>> Crime = namedtuple('Crime', fields, verbose=True)
```

```
>>> def parse(row):
... reader = csv.reader(StringIO(row))
... row=reader.next()
... return Crime(*row)
>>> crimes = dataWithoutHeader.map(parse)
>>> crimes
PythonRDD[3] at RDD at PythonRDD.scala:48
>>> crimes.first()
Crime(OBJECTID='1', Identifier='f070032d', Occurrence_Date='09/06/1940 07:30:00 PM',
Day_of_Week='Friday', Occurrence_Month='Sep', Occurrence_Day='6',
Occurrence_Year='1940', Occurrence_Hour='19', CompStat_Month='9', CompStat_Day='7',
CompStat_Year='2010', Offense='BURGLARY', Offense_Classification='FELONY', Sector='D',
Precinct='66', Borough='BROOKLYN', Jurisdiction='N.Y. POLICE DEPT',
XCoordinate='987478', YCoordinate='166141', Location_1='(40.6227027620001,
-73.9883732929999)')
>>> crimes.first().Offense
'BURGLARY'
>>> crimes.map(lambda x:x.Offense).countByValue()
>>> crimes.map(lambda x:x.Occurrence_Year).countByValue()
>>> crimesFiltered=crimes.filter(lambda x: not (x.Offense=="NA" or
x.Occurrence_Year="")).filter(lambda x: int(x.Occurrence_Year)>=2006)
>>> crimesFiltered.map(lambda x:x.Occurrence_Year).countByValue()
>>> crimes.filter(lambda x : x.Offense=="BURGLARY").map(lambda
x:x.Occurrence_Year).countByValue()
```

---

https://archive.ics.uci.edu/ml/machine-learning-databases/event-detection/

```
>>> trafficPath="file:///root/Dodgers.data"
>>> gamesPath="file:///root/Dodgers.events"
>>>
>>> traffic = sc.textFile(trafficPath)
>>> traffic.take(2)
[u'4/10/2005 0:00,-1', u'4/10/2005 0:05,-1']
>>> games= sc.textFile(gamesPath)
>>> games.take(2)
[u'04/12/05,13:10:00,16:23:00,55892,San Francisco,W 9-8\ufffd',
u'04/13/05,19:10:00,21:48:00,46514,San Francisco,W 4-1\ufffd']
>>> def parseTraffic(row):
... DATE_FMT = "%m/%d/%Y %H:%M"
... row = row.split(',')
... row[0] = datetime.strptime(row[0], DATE_FMT)
... row[1] = int(row[1])
... return (row[0],row[1])
```

Create Pair RDD :

```
>>> trafficParsed = traffic.map(parseTraffic)
>>> trafficParsed.take(2)
[(datetime.datetime(2005, 4, 10, 0, 0), -1), (datetime.datetime(2005, 4, 10, 0, 5), -1)]
>>> dailyTrend = trafficParsed.map(lambda x: (x[0].date(), x[1])).reduceByKey(lambda x,y:x+y)
>>> dailyTrend.take(2)
[(datetime.date(2005, 8, 9), 5958), (datetime.date(2005, 7, 7), 6301)]
>>> dailyTrend.sortBy(lambda x:-x[1]).take(2)
[(datetime.date(2005, 7, 28), 7661), (datetime.date(2005, 7, 29), 7499)]
```

Join :

```
>>> def parseGames(row):
... DATE_FMT = "%m/%d/%y"
... row = row.split(",")
... row[0] = datetime.strptime(row[0], DATE_FMT).date()
... return (row[0],row[4])
...
>>> gamesParsed = games.map(parseGames)
>>> gamesParsed.take(2)
[(datetime.date(2005, 4, 12), u'San Francisco'), (datetime.date(2005, 4, 13), u'San Francisco')]
>>> dailyTrendCombined.take(2)
[(datetime.date(2005, 8, 9), (5958, u'Philadelphia')), (datetime.date(2005, 6, 29), (5437, u'San
Diego'))]
>>> dailyTrendCombined.take(10)
>>> def checkGameDay(row):
... if row[1][1] == None:
... return (row[0],row[1][1], "Regular Day", row[1][0])
... else:
... return (row[0], row[1][1], "Game Day", row[1][0])
...
>>> dailyTrendbyGames = dailyTrendCombined.map(checkGameDay)
>>> dailyTrendbyGames.take(2)
[(datetime.date(2005, 8, 9), u'Philadelphia', 'Game Day', 5958), (datetime.date(2005, 6, 29),
u'San Diego', 'Game Day', 5437)]
>>> dailyTrendbyGames.sortBy(lambda x:-x[3]).take(2)
[(datetime.date(2005, 7, 28), u'Cincinnati', 'Game Day', 7661), (datetime.date(2005, 7, 29), u'St.
Louis', 'Game Day', 7499)]
```

Average on game day vs non game day :

```
>>> dailyTrendbyGames.map(lambda x:(x[2],x[3])).combineByKey(lambda value: (value,1) ,
lambda acc,value:(acc[0] + value, acc[1]+1), lambda acc1,acc2: (acc1[0]+ acc2[0], acc1[1] +
acc2[1])).mapValues(lambda x:x[0]/x[1]).collect()
[('Game Day', 5948), ('Regular Day', 5411)]
```

---

set up spark context in python prog :

```
[root@node1 ~]# yum install python-pip
[root@node1 ~]# pip install py4j
[root@node1 ~]# cat sparksetup.py
#!/usr/bin/python
import os
import sys
if 'SPARK_HOME' not in os.environ:
os.environ['SPARK_HOME'] = '/spark'
SPARK_HOME = os.environ['SPARK_HOME']
os.environ['SPARK_LOCAL_IP']="0.0.0.0"
sys.path.insert(0,os.path.join(SPARK_HOME,"python"))
sys.path.insert(0,os.path.join(SPARK_HOME,"python","lib"))
sys.path.insert(0,os.path.join(SPARK_HOME,"python","lib","pyspark"))
from pyspark import SparkContext
from pyspark import SparkConf
conf=SparkConf()
conf.set("spark.executor.memory", "1g")
conf.set("spark.cores.max", "2")
conf.setAppName("myapp")
```

```
sc = SparkContext('local', conf=conf)

mylist=[1,2,3,4]

print sc.parallelize(mylist).collect()

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

empDf = sqlContext.read.json("customer.json")

empDf.show()

"""

empDf.printSchema()

empDf.select("name").show()

empDf.filter(empDf["age"] == 28).show()

empDf.groupBy("gender").count().show()

empDf.groupBy("deptid").agg({"salary": "avg", "age": "max"}).show()

# create data frame from list

deptList = [{'name': 'sales', 'id': '100'}, {'name': 'engineer', 'id': '200'}]

deptDf = sqlContext.createDataFrame(deptList)

deptDf.show()

empDf.join(deptDf, empDf.deptid == deptDf.id).show()

empDf.registerTempTable("employees")

sqlContext.sql("select * from employees where salary > 3000").show()

"""
```

Spark Streaming

```
[root@node1 ~]# cat sparkstream.py
#!/usr/bin/python
import os
import sys
if 'SPARK_HOME' not in os.environ:
os.environ['SPARK_HOME'] = '/spark'
SPARK_HOME = os.environ['SPARK_HOME']
os.environ['SPARK_LOCAL_IP']="0.0.0.0"
sys.path.insert(0,os.path.join(SPARK_HOME,"python"))
sys.path.insert(0,os.path.join(SPARK_HOME,"python","lib"))
sys.path.insert(0,os.path.join(SPARK_HOME,"python","lib","pyspark"))
from pyspark import SparkContext
from pyspark import SparkConf
conf=SparkConf()
conf.set("spark.executor.memory", "1g")
conf.set("spark.cores.max", "2")
conf.setAppName("myapp")
sc = SparkContext('local[2]', conf=conf)
from pyspark.streaming import StreamingContext
ssc = StreamingContext(sc, 2)
ssc.checkpoint("file:///root/myspark")
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))
counts = lines.flatMap(lambda line: line.split(" ")).filter(lambda word: "ERROR" in
word).map(lambda word: (word,1)).reduceByKey(lambda a,b: a+b)
counts.pprint()
```

```
ssc.start()

ssc.awaitTermination()

[root@node1 ~]# nc -l 9999

[root@node1 ~]# python sparkstream.py localhost 9999

------------------------

[root@node1 ~]# cat sparkstreamwordcount.py

#!/usr/bin/python

import os

import sys

if 'SPARK_HOME' not in os.environ:

os.environ['SPARK_HOME'] = '/spark'

SPARK_HOME = os.environ['SPARK_HOME']

os.environ['SPARK_LOCAL_IP']="0.0.0.0"

sys.path.insert(0,os.path.join(SPARK_HOME,"python"))

sys.path.insert(0,os.path.join(SPARK_HOME,"python","lib"))

sys.path.insert(0,os.path.join(SPARK_HOME,"python","lib","pyspark"))

from pyspark import SparkContext

from pyspark import SparkConf

conf=SparkConf()

conf.set("spark.executor.memory", "1g")

conf.set("spark.cores.max", "2")

conf.setAppName("myapp")

sc = SparkContext('local[2]', conf=conf)

from pyspark.streaming import StreamingContext

ssc = StreamingContext(sc, 2)

ssc.checkpoint("file:///root/myspark")
```

```
lines = ssc.socketTextStream(sys.argv[1], int(sys.argv[2]))

def countWords(newValues, lastSum):

if lastSum is None:

lastSum = 0

return sum(newValues, lastSum)

counts = lines.flatMap(lambda line: line.split(" ")).map(lambda word:

(word,1)).updateStateByKey(countWords)

counts.pprint()

ssc.start()

ssc.awaitTermination()
```

HIve:

```
CREATE EXTERNAL TABLE person ( name String, age Int, Sex String)

STORED as PARQUET

LOCATION '/tmp/person'

Import org.apache.spark.sql.SaveMode

df.select("name",

"age","sex").write.mode(saveMode.Append).format("parquet").save("/tmp/person")

>>> from pyspark.sql import HiveContext

>>> hc = HiveContext(sc)

>>> hc.table("students"
```