SPARK (SCALA) DEVELOPER TRAINING – LAB GUIDE

Version 2.2

Abstract

This is the lab guide for the participants to learn and complete all lab exercises as part of the Apache Spark Developer training.

Manaranjan Pradhan

manaranjan@enablecloud.com

https://www.linkedin.com/in/manaranjanpradhan

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

1) Access Spark on the Cluster: Configuring Spark Client Machine

If users using common servers,

Participant needs to Login to anyone of the following hosts using their Kerberos id.

- d159101-001.dc.gs.com
- d159101-002.dc.gs.com

After login, execute the script to get hadoop cluster config files:

/gns/software/infra/big-data/hadoop/client-latest/get_client_config.ksh 179663

Give cluster name and directory where the config files need to be stored.

kinit <Kerberos id>

source <directory chosen>/hadoop/conf/hadoop.client.profile

User should now be able to access the cluster.

Note:
GNS Path :
/gns/software/infra/big-data/spark/spark-1.5.1

If users have their own DC boxes,

After login, execute the script to get hadoop cluster config files:

/gns/software/infra/big-data/hadoop/client-latest/get_client_config.ksh

Give cluster name and directory where the config files need to be stored.

kinit <Kerberos id>

source <directory chosen>/hadoop/conf/hadoop.client.profile

User should now be able to access the cluster.

Users need to install the gns package group to get the hadoop software on their own DC host via canvas

/package-groups/infra/big-data/hdp-2.2

The following steps need to be done by all users,

1. **Open the file <install_path>/hadoop/conf/spark-defaults.conf**
2. Comment out all conf except first four lines in **<install_path>/hadoop/conf/spark-defaults.conf** files.
3. From within the HADOOP_CONF_DIR directory, create a symbolic link to hive-site.xml (ln -s <install_path>/hive/conf/hive-site.xml). echo $SPARK_CONF_DIR and confirm it points to <install_path>/hadoop/conf. Another alternative is to copy hive-site.xml to $HADOOP_CONF_DIR

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

2) Download the file lab guides and data from github

Download the following project onto your desktop and untar.

https://github.com/manaranjanp/spark-using-scala

Unzip the file.

The data files that will be used for the lab sessions are available under ***data*** directory of the zip file.

Create a directory called ***sparklab*** under your /home/<kerbores id> directory of the spark-client machine configured in step 1.

Transfer the data files onto this directory using winscp or sftp.

3) Copy data files to HDFS

Create a hdfs directory

hdfs dfs -mkdir /user/kerbores id>/sparklab

Copy files from local file system to HDFS

hdfs dfs -copyFromLocal /home/<kerbores id>/sparklab/* /user/kerbores id>/sparklab

Check if files have been copied

hdfs dfs -ls /user/kerbores id>/**sparklab/**

Now all data files have been copied to HDFS

4) Starting Spark

A. Start Spark Shell

Enter the command at linux prompt

spark-shell --master yarn-client

The spark console should start as shown in the figure below along with Spark Version.

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

```
16/07/11 23:29:29 INFO spark.HttpServer: Starting HTTP Server
16/07/11 23:29:29 INFO server.Server: jetty-8.y.z-SNAPSHOT
16/07/11 23:29:29 INFO server.AbstractConnector: Started SocketConnector@0.0.0.0:37048
16/07/11 23:29:29 INFO util.Utils: Successfully started service 'HTTP class server' on port 37048.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 1.6.0
      /_/

Using Scala version 2.10.5 (OpenJDK 64-Bit Server VM, Java 1.7.0_55)
Type in expressions to have them evaluated.
Type :help for more information.
16/07/11 23:29:33 INFO spark.SparkContext: Running Spark version 1.6.0
16/07/11 23:29:33 INFO spark.SecurityManager: Changing view acls to: hadoop
16/07/11 23:29:33 INFO spark.SecurityManager: Changing modify acls to: hadoop
```

B. Check Spark Version

Type the following commands at the spark prompt to verify some more information.

>>> sc.version

'1.6.0'

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

Note:

File locations provided in the samples below are only for demo purpose. Data may not necessarily be available at the same location. Please change the file locations as per your lab cluster environment.

5) Running first spark program on command line

The first program will be a word count problem.

Enter the following lines at ***spark-shell*** prompt

```
var wordfile = sc.textFile( "/user/<kerbores id>/sparklab/words")
var words = wordfile.flatMap( line => line.split( " " ) )
words.take( 10 ).foreach( println )
var word_one = words.map( word => ( word, 1 ) )
var word_counts = word_one.reduceByKey( _+_ )
word_counts.take( 10 ).foreach( println )
```

6) Scala Overview

Download the install Eclipse IDE or Intellij IDE for Scala.

To learn Scala basics follow the ***Scala Introduction Ver 1.0.pdf*** tutorial.

7) Working with Spark APIs – using spark-shell (Interactive)

Follow the ***RDD APIs using Spark Scala - Top Captains.pdf*** tutorial.

8) Install Scala IDE

Install Eclipse or Intellij IDE for SCALA.

If you already have an IDE, you can install the SCALA plug-ins for eclipse or Intellij IDE.

For IntelliJ IDE:

For Eclipse IDE:

9) Writing a Scala Spark Program for running on Batch Mode

- Start Scala IDE (Eclipse or IntelliJ)
- Create a new Scala Maven Project
- Add the pom.xml file from shared location
- Select ***src*** on the project and right click and click new scala class and name is "***TopCaptains***".
- Write the following code in the class

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

```
import org.apache.spark.{SparkConf, SparkContext}

object TopCaptains {

 def parse( line:String ) = {

  val pieces = line.split(",")
  val name = pieces(0)
  val country = pieces(1)
  val career = pieces(2)
  val matches = pieces(3).toInt
  val won = pieces(4).toInt
  val lost = pieces(5).toInt
  val ties = pieces(6).toInt
  val toss = pieces(7).toInt

  Captain( name, country, career, matches, won, lost, ties, toss )

 }

 case class Captain( name:String,
   country:String,
   career: String,
   matches: Int,
   won: Int,
   lost: Int,
   ties: Int,
   toss: Int )

 def main(args: Array[String]) = {

        val conf = new SparkConf()
             .setAppName("Top Captains")

        val sc = new SparkContext(conf)

         var captains_odis = sc.textFile( "/user/<kerobores id>/sparklab/captains_ODI.csv" )
                                .map( line => parse( line ) )
                                .filter( rec => rec.matches > 100 )
                                .map( rec => ( rec.name, ( rec.won.toFloat / rec.matches.toFloat ) )
                                )

        var captains_tests = sc.textFile( "/user/<kerobores id>/sparklab/captains_Test.csv" )
                                .map( line => parse( line ) )
                                .filter( rec => rec.matches > 50 )
                                .map( rec => ( rec.name, ( rec.won.toFloat / rec.matches.toFloat) )
                        )
                                .sortBy( rec => rec._2, ascending = false )

        var all_time_best_captains = captains_odis.join( captains_tests )
                        .map( rec => ( rec._1, ( rec._2._1 * 0.4 + rec._2._2 * .6 ) ) )
                        .saveAsTextFile("/user/<kerobores id>/sparklab/captains")
 }

}
```


Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

- Do a maven build and create the jar file
- Transfer the jar file to VM using WinSCP under the directory ***/home/<kerbores id>/sparklab/***
- Traverse to the above directory in putty terminal

  cd /home/<kerbores id>/sparklab/

- Submit the program for execution

  spark-submit --class TopCaptains --master yarn-client <jar filename>

- Go to /home/hadoop/lab/results directory. The program should have created a directory called ***captains***.

  /user/<kerobores id>/sparklab/captains

- Go to topCaptains directory and list the files

  hdfs dfs -ls /user/<kerobores id>/sparklab/captains

  -rw-r--r--. 1 hadoop root 224 Feb 13 22:39 part-00000

  -rw-r--r--. 1 hadoop root 0 Feb 13 22:39 _SUCCESS

- Print the content of the file part-00000

  hdfs dfs -ls /user/<kerobores id>/sparklab/captains/part-*

  ('Smith G C', 0.61, 0.49)
  ('Fleming S P', 0.45, 0.35)
  ('Border A R', 0.6, 0.34)
  ('Dhoni M S*', 0.55, 0.45)
  ('Waugh S R', 0.63, 0.72)
  ('Cronje W J', 0.71, 0.51)
  ('Ranatunga A', 0.46, 0.21)
  ('Ponting R T', 0.72, 0.62)

10) Working with Spark DataFrames

Copy the jar files onto your local file system

hdfs dfs -copyToLocal /tmp/commons-csv-1.1.jar /home/<kerobores id>

hdfs dfs -copyToLocal /tmp/spark-csv_2.11-1.3.0.jar /home/<kerobores id>

Note: If the files are not available in the above /tmp location, confirm from the instructor the location.

Add the jar files while starting the spark-shell

spark-shell --master yarn-client --jars /home/<kerobores id>/commons-csv-1.1.jar,/home/<kerobores id>/spark-csv_2.11-1.3.0.jar

For next steps, follow ***DataFrames using Spark Scala – MovieLens.pdf*** tutorial

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

11) Working with Hadoop: HDFS, YARN & Spark SQL

Follow ***Working with HDFS and YARN - Retail Analysis.pdf*** tutorial.

12) Integrate with Hive

- Start beeline interface

  beeline -u 'jdbc:hive2://d179663-001.dc.gs.com,d179663-002.dc.gs.com,d179663-003.dc.gs.com:2181/default;principal=dchive/_HOST@GS.COM;serviceDiscoveryMode=zooKeeper;zooKeeperNamespace=hiveserver2

- Create hive database and table

  Create a database with your name (use your short name).

  create database <your name>;

  use <your name>;

  CREATE TABLE flights2008 (Year STRING,
  Month STRING,
  DayofMonth INT,
  DayOfWeek INT,
  DepTime INT,
  CRSDepTime INT,
  ArrTime INT,
  CRSArrTime INT,
  UniqueCarrier STRING,
  FlightNum STRING,
  TailNum STRING,
  ActualElapsedTime FLOAT,
  CRSElapsedTime FLOAT,
  AirTime FLOAT,
  ArrDelay FLOAT,
  DepDelay FLOAT,
  Origin STRING,
  Dest STRING,
  Distance FLOAT,
  TaxiIn INT,
  TaxiOut INT,
  Cancelled STRING,
  CancellationCode STRING,
  Diverted INT,
  CarrierDelay FLOAT,
  WeatherDelay FLOAT,
  NASDelay FLOAT,

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

SecurityDelay FLOAT,
LateAircraftDelay FLOAT)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ','
LINES TERMINATED BY '\n'
STORED AS TEXTFILE;

- Transfer the dataset from Desktop to VM under /home/hadoop/lab/data/ directory.
- Load data

LOAD DATA LOCAL INPATH '/home/hadoop/lab/data/2008.csv.bz2' INTO TABLE flights2008;

13) Working with Hive

Follow the steps in ***Working with Hive - Flight Delay Analysis.pdf*** tutorial.

14) Monitoring & Debugging

The guide for this will be shared before the workshop.

15) Working with Unstructured data

Follow ***Working with logs & SQL Functions - NASA_logs.pdf*** tutorial

16) Using Spark Streaming

Transfer files to server

Transfer the following files to the spark client (linux) machine

- TweetServer.jar (the jar file contains the Tweet Server code)
- Tweets (contains the tweets messages)

Start putty terminals

Two run spark streaming example below, your need to start two putty terminals

- On one terminal you will run Tweet Server
- On the other terminal the spark streaming will run

Start Twitter Server

It takes two parameters the server socket port and the tweets file which contains the tweets

java -classpath tweetserver.jar TweetServer 5656 tweets

Start Spark Shell

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

The spark application should be started on local or yarn-client mode. If you are running on local mode, then run two executors.

spark-shell –master yarn-client

Write Spark streaming application

Enter the following streaming code in spark console

//import the following libraries

import org.apache.spark.streaming._
import scala.io.Source
import org.apache.spark.storage.StorageLevel
import scala.collection.mutable.HashMap
import java.io.File
import org.apache.log4j.Logger
import org.apache.log4j.Level
import sys.process.stringSeqToProcess

//provide the streaming batch duration

val ssc = new StreamingContext(sc, Seconds(5))

val rootLogger = Logger.getRootLogger()
rootLogger.setLevel(Level.ERROR)

// provide the hostname or ip address and port number where the Tweet Server is running

val tweets = ssc.socketTextStream("localhost", 5656)

// provide the window of processing streaming inputs

val twt = tweets.window(Seconds(10))

// create a case class for storing tweet tags and their counts

case class Tweet(hashtag:String, counts:Int)

// It receives tweets, splits the messages into words,
// filter all words by hashtag (#), converts the words to lowercase, maps
// each word to (word, 1), then counts occurrences of each hash tag.

var tweets_count_rdd = twt.flatMap(text => text.split(" ")).filter(word =>
word.toLowerCase().startsWith("#")).map(word => (word.toLowerCase(), 1)
).reduceByKey(_+_).map(rec => Tweet(rec._1, rec._2))

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

val sqlContext= new org.apache.spark.sql.SQLContext(sc)
import sqlContext.implicits._

// finally the it converts the final counts into a data frame, sorts them in descending order //and takes only the first 10 records and registers it as a temp table. Then prints the table rows

tweets_count_rdd.foreachRDD{ rdd =>

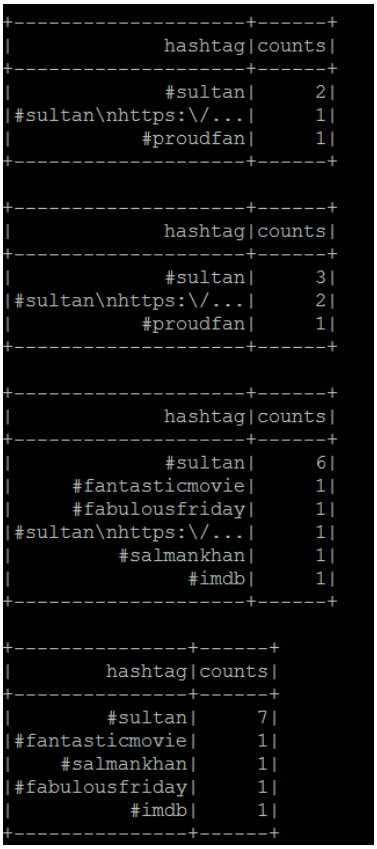
var tweets_df = rdd.toDF().sort(desc("counts")).limit(10).registerTempTable("tweets")
*var tweet_top_10 = sqlContext.sql("select * from tweets")*
tweet_top_10.show(10)

}

// start the streaming
ssc.start()

It starts the streaming. The terminal running the Tweet Server should print a message "Sending Messages"

// It prints the following tags and counts every 5 seconds on the terminal which runs spark shell



```
+--------------------+------+
|             hashtag|counts|
+--------------------+------+
|             #sultan|     2|
|#sultan\nhttps:\/...|     1|
|           #proudfan|     1|
+--------------------+------+

+--------------------+------+
|             hashtag|counts|
+--------------------+------+
|             #sultan|     3|
|#sultan\nhttps:\/...|     2|
|           #proudfan|     1|
+--------------------+------+

+--------------------+------+
|             hashtag|counts|
+--------------------+------+
|             #sultan|     6|
|     #fantasticmovie|     1|
|     #fabulousfriday|     1|
|#sultan\nhttps:\/...|     1|
|          #salmankhan|     1|
|                #imdb|     1|
+--------------------+------+

+---------------+------+
|        hashtag|counts|
+---------------+------+
|         #sultan|     7|
| #fantasticmovie|     1|
|     #salmankhan|     1|
| #fabulousfriday|     1|
|           #imdb|     1|
+---------------+------+
```


Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan

Stop the streaming example

Stop the spark shell and Tweet Server process.

17) Assignment: Visualization, Statistics & Machine Learning Library

The guide for this will be shared before the workshop.

Manaranjan Pradhan
manaranjan@enablecloud.com
https://www.linkedin.com/in/manaranjanpradhan