MIDS W205

|  |  |  |  |
| --- | --- | --- | --- |
| **Lab #** | 6 | **Lab Title** | Apache Spark SQL - a few features |
| **Related Module(s)** | 6 | **Goal** | Get you started on spark |
| **Last Updated** | 9/24/15 | **Expected duration** | 20 to 30 minutes |

Todos:

* Improve introduction
* Explain what they will learn.
* For each step explain what and why and how to verify
* Include references to useful resources.

# This Lab

In this lab you will get an hands on introduction to Sparks and specifically to Spark SQL. After this lab you should be able to launch Spark, create tables, load data and run simple queries using Spark SQL. We will go over Apache Spark features and common commands as below:

1. Accessing Spark SQL - CLI
2. Accessing through Java database connectivity (JDBC)
3. Accessing through Java database connectivity - Beenline
4. Let’s create a table and load data using CSV file
5. Accessing Spark-SQL in Python Code
6. Caching tables and Un-caching tables

# Intro

## Apache Spark

Apache Spark is an open-source distributed computing. This was developed in the AMPLab at UC Berkley. Spark uses in-memory processing and can sometime be up to 100 times faster than regular Hadoop Map Reduce on certain types of jobs.

Apache Spark has a basic computing substrate and several frameworks built on top of it. There is a framework for querying structured data (Spark SQL), a streaming analytics framework, a machine learning framework etc.



Spark SQL is a Spark module in which uses structured data processing. It provides a programming abstraction called DataFrames. Spark SQL acts as a distributed SQL query engine.

## Here are some useful resources:

|  |  |
| --- | --- |
| **Resource** | **What** |
| http://spark.apache.org/docs/latest/programming-guide.html | This guide shows each of these features in each of Spark’s supported languages. It is easiest to follow along with if you launch Spark’s interactive shell. |
| https://spark.apache.org/docs/1.1.0/sql-programming-guide.html | Guide for using Spark SQL. |

In Spark, a DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs.

## Here is an example which shows how to construct DataFrames in Python programming language :

//Constructs a DataFrame from the users table in Hive.

users = context.table(“users”)

//Constructs a DataFrame from from JSON files in S3

logs = context.load(“s3n://path/to/data.json”, “json”)

## Here are a few supported Data Formats and Sources :

1. JSON files, Parquet files, Hive tables  
 2. Local file systems  
 3. Distributed file systems (HDFS)  
 4. Cloud storage (S3)   
 5. External relational database systems via JDBC  
 6. Extensions include Avro, CSV, ElasticSearch, and Cassandra

# Let’s go!

# Step-1.Accessing Apache Spark SQL (CLI)

## Using the command line interface (CLI)

From Spark Installation folder, let’s go to spark-sql prompt. Please check where is your spark application is installed.

cd /usr/lib/spark

./bin/spark-sql

Once you are in spark prompt (spark-sql) , to list the tables, you can run as follows:

spark-sql> show tables;

##### If you want to connect via spark-shell, here is a very useful link:

<https://docs.sigmoidanalytics.com/index.php/Interactive_Analysis_with_the_Spark_Shell>

## You should see the following result

# Step-2. Accessing through Java database connectivity (JDBC)

## Starting a JDBC server

Spark SQL provides JDBC connectivity, which is useful for connecting business intelligence (BI) tools to a spark cluster and other users or applications.

**Here are the steps to launch a the Spark SQL JDBC server**

From Spark Installation folder

cd /usr/lib/spark

./sbin/start-thriftserver.sh –master sparkMaster

## You should see the following result

# 

# Step-3. Let’s connect to the above server(JDBC) from Step-2 through Beenline:

## Connecting to a Database

From Spark Installation folder

cd /usr/lib/spark

//10000 will be the default thrift server port

./bin/beenline -u jdbc:hive2://localhost:10000

0: jdbc:hive2://localhost:10000> show tables;

## You should see the following result

# Step-4. Let’s create a table and load data using CSV file:

## Creating a table

You can create a Spark SQL table using the following format.

create table Web\_Session\_Log

(DATETIME varchar(500),

USERID varchar(500),

SESSIONID varchar(500),

PRODUCTID varchar(500),

REFERERURL varchar(500))

row format delimited fields terminated by ‘\t’

stored as textfile;

Now let’s load data to this table from HDFS and Local:

HDFS :

LOAD DATA INFILE ‘/mnt/weblog.csv’ INTO TABLE Web\_Session\_Log;

Local file system :

LOAD DATA LOCAL INFILE ‘/mnt/weblog.csv’ INTO TABLE Web\_Session\_Log;

You, please create a datafile of your own choice. Also, create a table using Spark-sql and load the data.

## You should see the following result

# 

# Step-5. Accessing Spark-SQL in Python Code:

## Programming Spark SQL with Python

Here is a sample code:

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

Running SQL Queries Programmatically :

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

df = sqlContext.sql(“**SELECT \* FROM Web\_Session\_Log limit 10**”)

Programmatically Specifying the Schema :

from pyspark.sql import \*

sqlContext = SQLContext(sc)

lines = sc.textFile(“/mnt/weblog.csv”)

parts = lines.map(lambda l: l.split(“\t”))

Web\_Session\_Log = parts.map(lambda p: (p[0], p[1],p[2], p[3],p[4]))

schemaString = “DATETIME USERID SESSIONID PRODUCTID REFERERURL”

fields = [StructField(field\_name, StringType(), True) for field\_name in schemaString.split()]

schema = StructType(fields)

schemaPeople = sqlContext.createDataFrame(Web\_Session\_Log, schema)

schemaPeople.registerTempTable(“Web\_Session\_Log”)

results = sqlContext.sql(“SELECT USERID FROM Web\_Session\_Log”)

names = results.map(lambda p: “Name: ” + p.USERID)

for name in names.collect():

print name

How to Run This Python Code:

First, go to pyspark prompt by running ./pyspark from your installation directory.

[root@ip-10-229-95-146 bin]# ./pyspark

You will see the following prompt:

Using Python version 2.6.6 (r266:84292, Jan 22 2014 09:42:36)

SparkContext available as sc, HiveContext available as sqlContext.

>>>

You can just launch pyspark from Spark installation bin directory and paste above code. Or, you can create a file with the above content as say mypyspark.py and run as

./bin/pyspark /yourdirectory/myspark.py

Analyze the result. Notice that the table used in this step was created in step 4. If you have created your own table, refer to that. If you run into any issue, you will need to debug. This will help you to learn what each line of code is doing.

## You should see the following result

# 

# Step-6. Caching tables and Un-caching tables;

## Caching tables

This is extremely useful when you are joining tiny dataset with huge dataset.

CACHE TABLE and UNCACHE TABLE statements are available to do the above making it very easy.

To Cache a table :

CACHE TABLE logs\_last\_month;

To UnCache a table :

UNCACHE TABLE logs\_last\_month;

Once as table is cached, you can use in your spark queries.

## You should see the following result

# What you should have learnt

1. What are the various components of Apache Spark?
2. What is the difference between Spark-shell and Spark-sql?
3. What is a RDD?
4. Why Spark is faster than Map-reduce FW?
5. How will you use Spark from your visualization tools?