Spark In R

Lab 02: Coding To The SparkR API

The goal of this lab is to introduce you to the SparkR API for R.

Setup

If you haven't done so already, connect to RStudio Server on your cluster. You can find the process for doing so in Lab 01.

You can find the code for this walkthrough in `02-SparkR-API.R`.

Initializing R For Spark

For SparkR to run properly under RStudio, it requires several environment variables to be properly set. Specifically, SparkR needs to know:

- SPARK_HOME: Where Spark is installed.
- HADOOP_CONF_DIR: Where the Hadoop configuration files lives.
- YARN_CONF_DIR: The configuration directory for YARN.

We'll set those up before we initialize Spark using the following code:

```
if (nchar(Sys.getenv("SPARK_HOME")) < 1) {
  Sys.setenv(SPARK_HOME = "/usr/lib/spark") # or whereever your Spark install lives
}
if (nchar(Sys.getenv("HADOOP_CONF_DIR")) < 1) {
  Sys.setenv(HADOOP_CONF_DIR = "/etc/hadoop/conf")  # or wherever your Hadoop lives
}
if (nchar(Sys.getenv("YARN_CONF_DIR")) < 1) {
  Sys.setenv(YARN_CONF_DIR = "/etc/hadoop/conf")    # or wherever your YARN config lives
}
```

Now, we'll load the `magrittr` package for data pipelining and then initialize Spark. We'll request 3GB of memory for the driver program in our initialization. This typically takes about 10-15 seconds to complete because it must spin up the Spark executors.

For those R programmers who typically load the `dplyr` library on startup, you don't want to do that when working with Spark because some of the `dplyr` function names conflict with the SparkR API functions, which can result in very confusing errors.

```
library(magrittr)
# please note -- do not load dplyr or you will have function name conflicts

library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
sparkR.session(master = "yarn",   # execution type
   sparkConfig = list(spark.driver.memory = "3g"))s # configure driver and executors
```

Getting Started With SparkR DataFrames

R natively supports a data.frame type, which is implemented as a list of columns. SparkR supports a different dataframe type called a DataFrame, which is implemented within the Spark Cluster as an RDD of Rows. You must use different functions for the two different types! You also must be careful to spell the dataframe you want appropriately.

Here's an example of the built-in data.frame `faithful` being converted to a DataFrame.

```
head(faithful)
##   eruptions waiting
## 1     3.600      79
## 2     1.800      54
## 3     3.333      74
## 4     2.283      62
## 5     4.533      85
## 6     2.883      55

str(faithful)
## 'data.frame':    272 obs. of  2 variables:
##  $ eruptions: num  3.6 1.8 3.33 2.28 4.53 ...
##  $ waiting  : num  79 54 74 62 85 55 88 85 51 85 ...

df <- as.DataFrame(faithful)            # note this is as.Dataframe, not as.data.frame
# Displays the first part of the SparkDataFrame
head(df)
## you see the same numbers but it takes longer.
##  eruptions waiting
##1     3.600      79
##2     1.800      54
##3     3.333      74
```

Throughout this lab, we'll largely focus on SparkR DataFrames, not data.frames.

Simple DataFrame Operations

We'll walk through the `people.json` data set just as we did in Scala and Python. However, the syntax for R is quite different because R allows the dot character in variable names. As a result, we will use the `magrittr` pipe operator `%>%` when we want to chain the result of one DataFrame operation into the input of another.

So let's read in `/data/spark-resources-data/people.json` as a DataFrame and perform simple DataFrame operations on it.

```
df <- read.df("/data/spark-resources-data/people.json", "json")
# Show the content of the DataFrame
showDF(df)
##   age    name
## 1  NA Michael
## 2  30    Andy
```

```
## 3  19  Justin

printSchema(df)
## root
##  |-- age: long (nullable = true)
##  |-- name: string (nullable = true)

names(df)
## [1] "age"  "name"

df %>% select("name") %>% showDF()
## +-------+
## |   name|
## +-------+
## |Michael|
## |   Andy|
## | Justin|
## +-------+

## If you don't like pipelines, you can also express this as:
showDF(select(df, "name"))
## +-------+
## |   name|
## +-------+
## |Michael|
## |   Andy|
## | Justin|
## +-------+
##
## However, this gets messy for more complicated pipelines

df %>% select(df$name, df$age + 1) %>% showDF()
## +-------+-----------+
## |   name|(age + 1.0)|
## +-------+-----------+
## |Michael|       null|
## |   Andy|       31.0|
## | Justin|       20.0|
## +-------+-----------+

df %>% filter(df$age > 21) %>% showDF()
## +---+----+
## |age|name|
## +---+----+
## | 30|Andy|
```

```
## +---+----+

df %>% groupBy("age") %>% count() %>% showDF()
## +----+-----+
## | age|count|
## +----+-----+
## |  19|    1|
## |null|    1|
## |  30|    1|
## +----+-----+

## Let's save this as a table
createOrReplaceTempView(df, "people")
sql("show tables") %>% showDF()
## +---------+-----------+
## |tableName|isTemporary|
## +---------+-----------+
## |   people|      false|
## +---------+-----------+

teenagers <- sql("select name from people where age >= 13 and age <= 19")
showDF(teenagers)
## +------+
## |  name|
## +------+
## |Justin|
## +------+
```

Here's a quick rundown of what's different when we do DataFrame operations in R instead of Scala or Python:

- The pipeline operator is `%>%, not .`.
- We show DataFrames using the R `showDF` function instead of `show`
- We select columns in R using the `$` operator.
- We don't explicitly reference the Spark session (i.e., we use `read.DF` or `loadDF`, not `spark.read` or `spark.load`). Similarly, we use `sql` to invoke a SQL statement, not `spark.sql` as in the other languages.

Working With The Stocks Table

As with Scala nd Python, we'll want to be able to use Hive tables in our analyses.

You probably have already created a `stocks` table in prior labs. However, if you haven't, don't worry; we can

recreate it here from the original .csv file quite easily using `read.df`. While versions of Spark earlier than 2.0 didn't include a .csv file reader, it's now a built-in format in the `read.df` function (and in the `spark.read` functions in Scala and Python).

```
stocks <- read.df("/data/stocks-flat/input", "csv")
names(stocks) <- c("exchg", "symbol", "ymd", "price_open",
   "price_high", "price_low", "price_close",
   "volume", "price_adj_close")
createOrReplaceTempView(stocks, "stocks")
```

Please note that we didn't specify a schema, so Spark is going to read all those fields as strings.

Check to make sure you read the DataFrame correctly. We'll look at the first few rows and then count the total number of rows and those rows that reference AAPL.

```
first10 <- sql("select * from stocks limit 10")
showDF(first10)
## +------+------+----------+----------+----------+---------+-----------+------+--------------
## | exchg|symbol|       ymd|price_open|price_high|price_low|price_close|volume|price_adj_clos
## +------+------+----------+----------+----------+---------+-----------+------+--------------
## |NASDAQ|  AAIT|2015-06-22| 35.299999| 35.299999|35.299999|  35.299999|   300|      35.29999
## |NASDAQ|  AAIT|2015-06-19| 35.259998| 35.259998|35.259998|  35.259998|   400|      35.25999
## |NASDAQ|  AAIT|2015-06-18|     34.52| 34.830002|    34.52|  34.830002|   300|      34.83000
## |NASDAQ|  AAIT|2015-06-17| 34.650002| 34.650002|34.650002|  34.650002|   200|      34.65000
## |NASDAQ|  AAIT|2015-06-16| 34.799999| 34.799999|34.709999|      34.77|   700|          34.7
## |NASDAQ|  AAIT|2015-06-15| 35.009998| 35.009998|35.009998|  35.009998|     0|      35.00999
## |NASDAQ|  AAIT|2015-06-12| 34.639999| 35.009998|34.639999|  35.009998|   300|      35.00999
## |NASDAQ|  AAIT|2015-06-11|     34.68|     34.68|    34.68|      34.68|   200|          34.6
## |NASDAQ|  AAIT|2015-06-10| 34.689999| 34.689999|34.689999|  34.689999|     0|      34.68999
## |NASDAQ|  AAIT|2015-06-09| 34.189999| 34.689999|34.189999|  34.689999|  3500|      34.68999
## +------+------+----------+----------+----------+---------+-----------+------+--------------

count(stocks)
## Answer should be 2131092

stocks %>% filter(stocks$symbol == "AAPL") %>% count()
## Answer should be 8706
```

Finally, we'll stop our R Spark Session. While this isn't strictly required because our Spark session times out within a few minutes if you don't use it, explicitly stopping the Spark session prevents complaints from the R interpreter.

```
sparkR.session.stop()
```

This step concludes the lab.