Demonstration: Analyzing Presidential Speeches with Spark and Spark SQL

This tutorial was built for BDCS-CE version 17.4.1 as part of the New Data Lake User Journey: here (https://github.com/oracle/learning-library/tree/master/workshops/journey2-new-data-lake). Questions and feedback about the tutorial: david.bayard@oracle.com (mailto:david.bayard@oracle.com)

Contents

As a reminder, the documentation for BDCS-CE can be found: here (https://docs.oracle.com/cloud/latest/big-data-compute-cloud/index.html)

About Spark and Spark SQL

BDCS-CE comes with Spark version 2.1 and Scala version 2.11. This BDCS-CE version supplies Zeppelin interpreters for Spark(Scala) and Spark SQL.

This tutorial will give you examples using these.

The tutorial assumes you have a basic knowledge about Spark. To learn more about Spark, check out Spark Quick Start (https://spark.apache.org/docs/2.1.0/quick-start.html) and Spark SQL Programming Guide (https://spark.apache.org/docs/2.1.0/sql-programming-guide.html)

Setup - Downloading Presidental Speeches data

Next, we will download some sample data to experiment with. In this example, we are going to download the text of a handful of United States Presidential Inauguration Speeches. We will install the lynx browser to help us. Then we will download the speeches from the Yale Law School Avalon Project website: here (http://avalon.law.yale.edu/subject_menus/inaug.asp)

Script to download sample speeches

```
%sh

CONTAINER=journeyC
DIRECTORY=speeches

#setup for this tutorial
echo "running yum to install lynx. This may take 5 to 10 minutes to refresh yum cache. Please be patient"
sudo yum -y install lynx 2>&1
echo "after yum"

#let's make the zeppelin directory readable.  This is so that Spark jobs can read these files later...
#chmod ga+rx .

mkdir speeches
cd speeches

#download some text from the internet using the lynx browser
#we will grab inauguration speeches from here http://avalon.law.yale.edu/subject_menus/inaug.asp

lynx -dump -nolist http://avalon.law.yale.edu/18th_century/wash1.asp | head -n -23 | tail -n +17 > pres1789_wash1.txt
lynx -dump -nolist http://avalon.law.yale.edu/19th_century/lincoln1.asp | head -n -23 | tail -n +17 > pres1861_lincoln1.txt
lynx -dump -nolist http://avalon.law.yale.edu/20th_century/froos1.asp | head -n -23 | tail -n +17 > pres1933_fdr1.txt
lynx -dump -nolist http://avalon.law.yale.edu/20th_century/kennedy.asp | head -n -23 | tail -n +17 > pres1961_jfk.txt
lynx -dump -nolist http://avalon.law.yale.edu/20th_century/reagan1.asp | head -n -23 | tail -n +17 > pres1981_reagan1.txt

echo "files downloaded are:"
ls -l
```

```
running yum to install lynx. This may take 5 to 10 minutes to refresh yum cache. Please be patient
Loaded plugins: security
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package lynx.x86_64 0:2.8.6-27.el6 will be installed
--> Finished Dependency Resolution
Dependencies Resolved
================================================================================
 Package       Arch            Version                 Repository          Size
================================================================================
Installing:
 lynx          x86_64          2.8.6-27.el6            ol6_latest         1.3 M
Transaction Summary
================================================================================
Install       1 Package(s)
Total download size: 1.3 M
Installed size: 4.7 M
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing : lynx-2.8.6-27.el6.x86_64                                     1/1
  Verifying  : lynx-2.8.6-27.el6.x86_64                                     1/1
Installed:
  lynx.x86_64 0:2.8.6-27.el6
Complete!
after yum
files downloaded are:
total 72
```

```
-rw-rw-r-- 1 zeppelin zeppelin  9156 Nov 15 19:52 pres1789_wash1.txt
-rw-rw-r-- 1 zeppelin zeppelin 22083 Nov 15 19:52 pres1861_lincoln1.txt
-rw-rw-r-- 1 zeppelin zeppelin 11500 Nov 15 19:52 pres1933_fdr1.txt
-rw-rw-r-- 1 zeppelin zeppelin  8032 Nov 15 19:52 pres1961_jfk.txt
-rw-rw-r-- 1 zeppelin zeppelin 14496 Nov 15 19:52 pres1981_reagan1.txt
..
Copying files into Object Store
Found 5 items
-rw-rw-rw-   1       9156 2017-11-15 19:52 swift://journeyC.default/speeches/raw/pres1789_wash1.txt
-rw-rw-rw-   1      22083 2017-11-15 19:52 swift://journeyC.default/speeches/raw/pres1861_lincoln1.txt
-rw-rw-rw-   1      11500 2017-11-15 19:52 swift://journeyC.default/speeches/raw/pres1933_fdr1.txt
-rw-rw-rw-   1       8032 2017-11-15 19:52 swift://journeyC.default/speeches/raw/pres1961_jfk.txt
-rw-rw-rw-   1      14496 2017-11-15 19:52 swift://journeyC.default/speeches/raw/pres1981_reagan1.txt
..
done
```

Example: Spark(Scala) with Object Store data

Our first example will show a simple bit of Scala code accessing our object store data. Our example defines a Spark RDD (Resilient Distributed Dataset) against a text file stored in Object Store. Then it runs a few actions against the RDD, such as counting the # of lines, displaying the first line, and counting the number of lines matching a given term.

Scala example with Object Store data

```
%spark

val Container="journeyC"
val Directory="speeches"

println("Define a RDD against a Object Store textfile...")
//val textFile = sc.textFile("swift://"+Container+".default/"+Directory+"/raw/pres1981_reagan1.txt")
//We will use the bdfs (alluxio) cached file system to access our object store data...
val textFile = sc.textFile("bdfs://localhost:19998/"+Directory+"/raw/pres1961_jfk.txt")


println("..")

println("Count of # of lines: %s".format(
    textFile.count()
```

```
Container: String = journeyC
Directory: String = speeches
Define a RDD against a Object Store textfile...
textFile: org.apache.spark.rdd.RDD[String] = bdfs://localhost:19998/speeches/raw/pres1961_jfk.txt MapPartitionsRDD[68] at textFile at <co
nsole>:29
..
Count of # of lines: 155
..
The First Line:    Inaugural Address of John F. Kennedy
..
# Lines containing the word Constitution: 0
..
done
```

Example: Wordcount(Spark Scala)

Our next example runs the classic Wordcount algorithm.

The results of the Wordcount are an RDD named wordCounts that will also be used in the following example.

Wordcount Example (scala)

```
%spark
val Container="journeyC"
val Directory="speeches"
```

```
Container: String = journeyC
Directory: String = speeches
Define a RDD against a Object Store textfile...
textFile: org.apache.spark.rdd.RDD[String] = bdfs://localhost:19998/speeches/raw/pres1861_lincoln1.txt MapPartitionsRDD[71] at textFile a
t <console>:29
wordCounts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[74] at reduceByKey at <console>:31
..
Here is the output (partially shown)
res30: Array[(String, Int)] = Array((sovereign,1), (ills,2), (call,1), (country,3), (weary,1), (admitted,1), (humane,1), (greater,1), (in
tervals,1), (accession,1), (privileges,1), (contrary,2), (inadmissible,1), (maintained,1), (order,1), (recanted,1), (apprehension,2), (mo
dification,1), (national,9), (been,4), (pretext,2), (resident,1), (advantageous,1), (evil,1), (over,3), (executive,2), (parties,3), (any,
27), (make,6), (conflict,1), (instead,1), (consideration,1), (overthrow,1), (contract,2), (institution,1), (assumed,1), (me,6), (divorced
,1), (parts,2), (are,20), (sentiments,3), (fugitives,2), (existed,1), (brief,1), (element,1), (scarcely,1), (unmade,1), (hazard,1), (auda
city,1), (faithfully,2), (favor,1), (our,13), (bloodshed,1), (territories,2), (circumstances,3), (branch,1), (fo...
..
warning: there were two feature warnings; re-run with -feature for details
The first word is sovereign and the count of it is 1
```

Example: Converting wordCounts RDD to a DataFrame and registering it as table (Scala)

The next example continues from the previous example. Specifically, it takes the wordCounts RDD and registers it as Spark DataFrame. Then it registers the new data frame as a temporary Spark SQL table.

At this point, you might want to quickly review the Spark SQL programming guide for a refresher about Creating DataFrames from RDDs: here (https://spark.apache.org/docs/2.1.0/sql-programming-guide.html)

Converting a RDD to a DataFrame and Registering it as a temp view (scala)

```
%spark

// this example follows the "Inferring the Schema using Reflection" of converting an RDD to a DataFrame. See https://spark.apache.org/doc

// first define a case class that describes our Schema
case class Wordcount(word: String, wcount: Int)
println("..")

// second, map our wordCounts RDD into an RDD using our Wordcount class and convert that into a DataFrame
val wordcountsDF = wordCounts.map(p => Wordcount(p._1, p._2)).toDF()
println("..")

// now register our wordcountsDF data frame as a temporary Spark SQL table
wordcountsDF.createOrReplaceTempView("wordcounts")
println("..")

println("done")
```

```
defined class Wordcount
..
wordcountsDF: org.apache.spark.sql.DataFrame = [word: string, wcount: int]
..
..
done
```

Example: Spark SQL against our wordcounts table

This example continues from the previous example. Specifically, it provides two examples of running Spark SQL against our wordcounts table. It also demonstrates some of the features of Zeppelin's Spark SQL interpreter and display visualization capabilities.

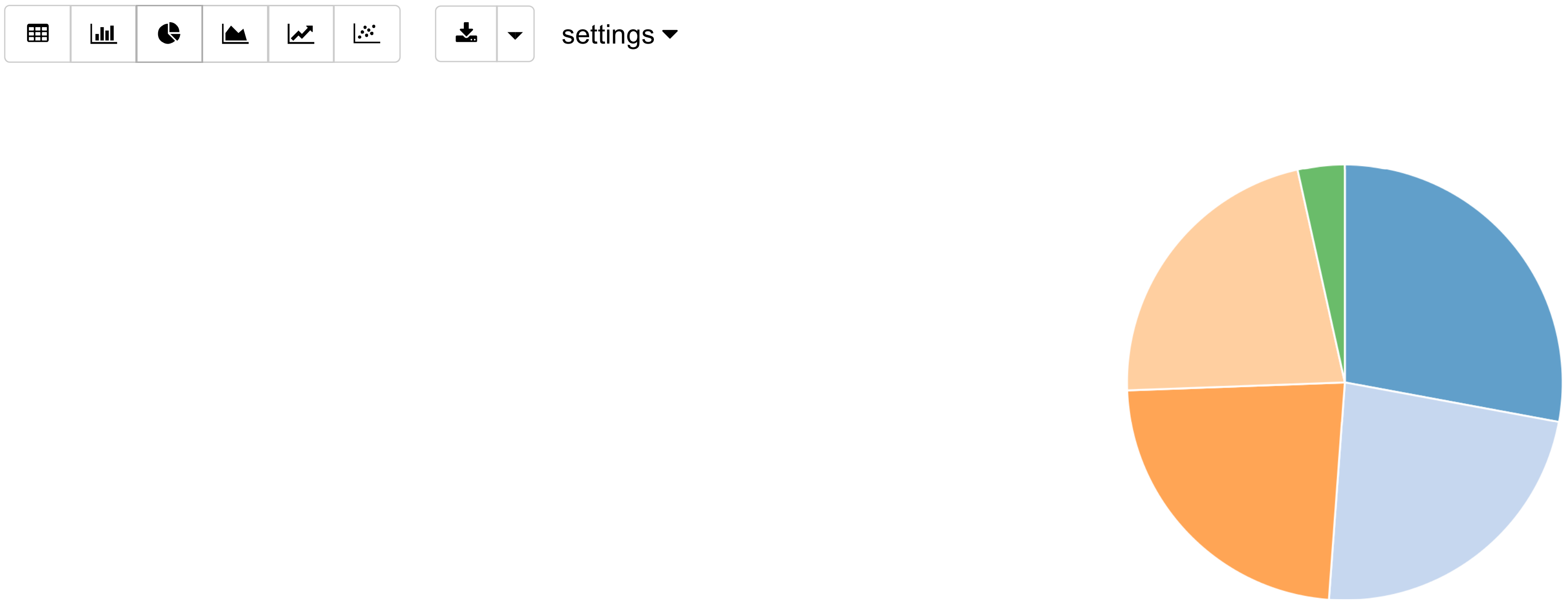
Spark SQL against our WordCounts table

```
%sql
select * from wordcounts
where length(word)>4
order by wcount desc limit 15
```

word	wcount
constitution	24
union	20
people	20
states	19
which	18
there	18
government	18
shall	16
their	14

More Spark SQL

```
%sql
select * from wordcounts
where word in ('god','america','constitution','nation','people','states','rights','freedom','union')
order by wcount desc
```



Example: More complex example- Speech Trends across time

Our next example continues from the previous example. Specifically, it builds an RDD against all of the speeches we stored in the Object Store. Then it performs a word count against the RDD and converts the result into a Data Frame that we register as a Spark SQL temporary table. Then we use some SparkSQL to prepare a couple of charts.

```
%spark
// more complex example.  Let's look for trends across presidents

val Container="journeyC"
val Directory="speeches"

val filebase = "swift://"+Container+".default/"+Directory+"/raw/pres"
// val filebase = "bdfs://localhost:19998/"+Directory+"/raw/pres"

val textFiles = sc.wholeTextFiles(filebase+"*.txt")

println("We found %s speeches".format(textFiles.count()))
```

```
Container: String = journeyC
Directory: String = speeches
filebase: String = swift://journeyC.default/speeches/raw/pres
textFiles: org.apache.spark.rdd.RDD[(String, String)] = swift://journeyC.default/speeches/raw/pres*.txt MapPartitionsRDD[88] at wholeText
Files at <console>:33
We found 5 speeches
..
words: org.apache.spark.rdd.RDD[(String, String)] = MapPartitionsRDD[89] at map at <console>:35
..
wordArray: org.apache.spark.rdd.RDD[(String, Array[String])] = MapPartitionsRDD[90] at map at <console>:37
..
defined class filewordsraw
..
filewordsrawDF: org.apache.spark.sql.DataFrame = [file: string, words: array<string>]
```

```
..
root
 |-- file: string (nullable = true)
 |-- words: array (nullable = true)
 |    |-- element: string (containsNull = true)
..
..
done
```

SQL Showing the # of words per Speech

```
%sql
select file, count(*) words from (select file, explode(words) word from filewordsraw) a
group by file
order by file
```

settings

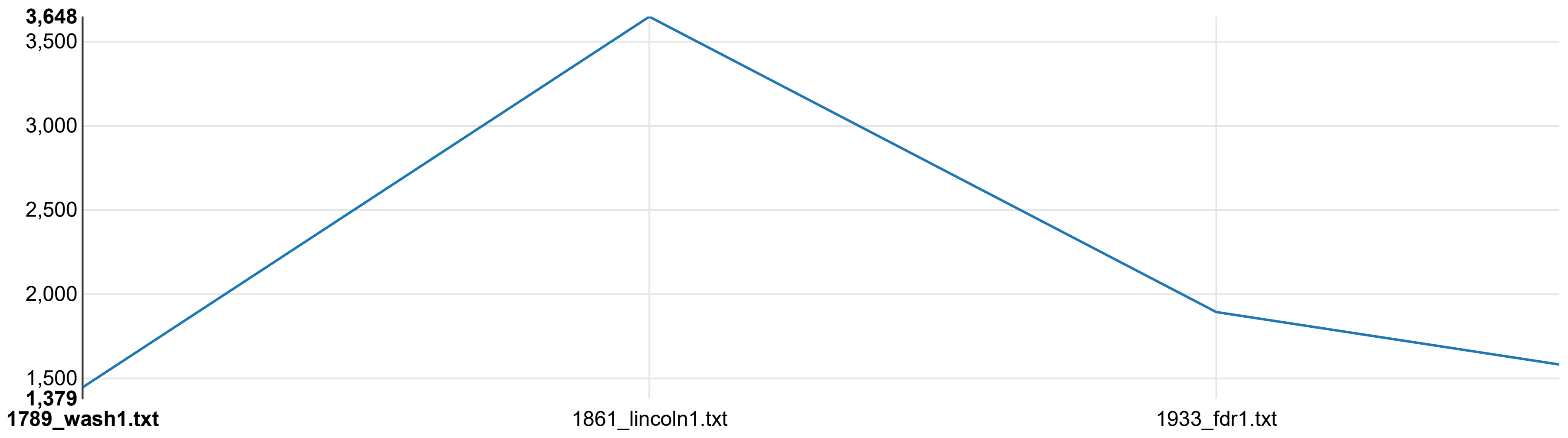
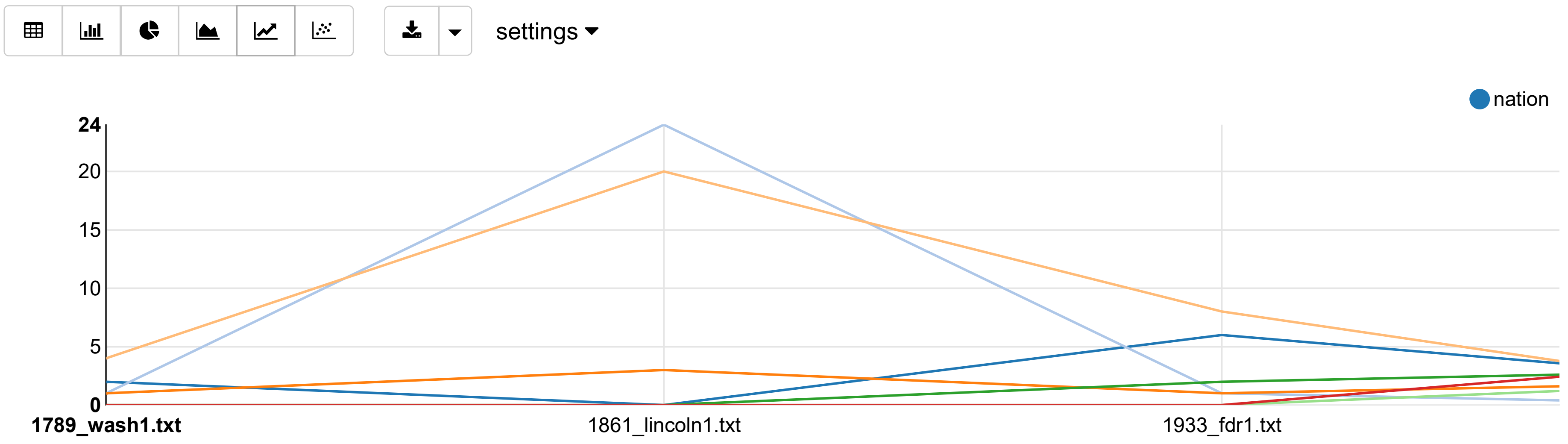


Chart showing the usage of certain words over time

```
%sql
select file, word, count(*) wc from (select file, explode(words) word from filewordsraw) a
```



Example: Saving a Data Frame back to the Object Store

Our final example continues from the previous example. Specifically, it defines a new data frame based off one of the sample SQL statements and writes that data frame back to the Object Store (as a json file). Then it reads the json data back from the Object Store into a new Data Frame.

```
%spark

val Container="journeyC"
val Directory="speeches"

val filebase = "swift://"+Container+".default/"+Directory+"/processed/wordcount_json"
// val filebase = "bdfs://localhost:19998/"+Directory+"/processed/wordcount_json"


//
val results = sqlContext.sql("select file, count(*) words from (select file, explode(words) word from filewordsraw) a group by file")
println("..")
```

```
Container: String = journeyC
Directory: String = speeches
filebase: String = swift://journeyC.default/speeches/processed/wordcount_json
results: org.apache.spark.sql.DataFrame = [file: string, words: bigint]
..
..
..
df: org.apache.spark.sql.DataFrame = [file: string, words: bigint]
..
res59: Array[org.apache.spark.sql.Row] = Array([1789_wash1.txt,1447], [1961_jfk.txt,1379], [1981_reagan1.txt,2463], [1861_lincoln1.txt,36
48], [1933_fdr1.txt,1895])
..
done
```

Optional: List the contents of your Object Store container to see the structure of the saved data frame.

```
%sh
CONTAINER=journeyC
DIRECTORY=speeches
hadoop fs -ls swift://$CONTAINER.default/$DIRECTORY/processed/wordcount_json
```

```
Found 2 items
```

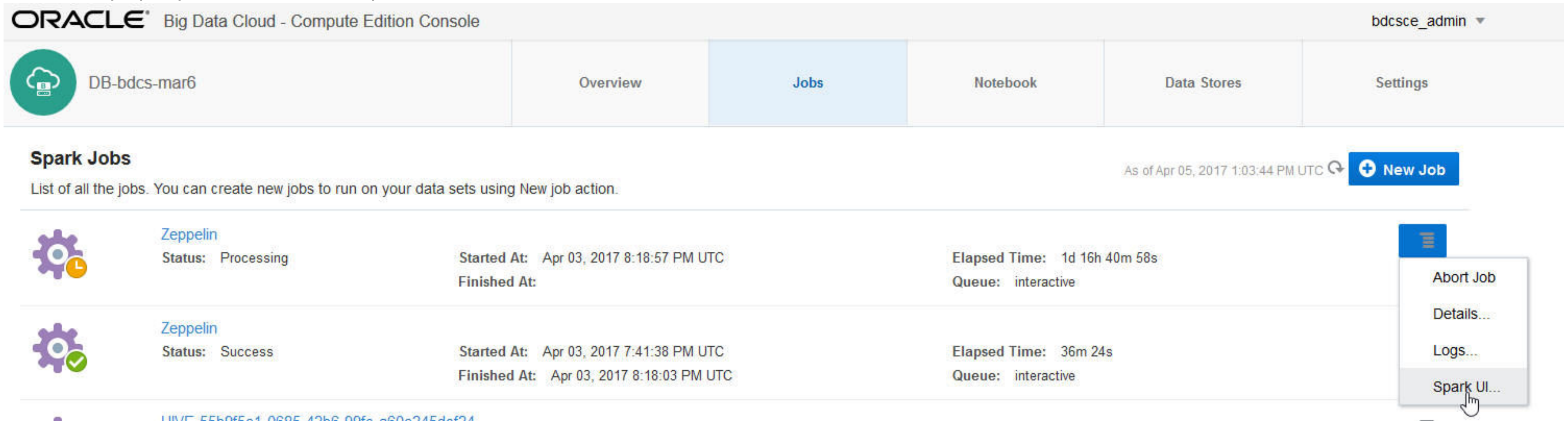
```
drw-rw-rw-   -          0 2017-11-15 19:54 swift://journeyC.default/speeches/processed/wordcount_json/_SUCCESS
-rw-rw-rw-   1        197 2017-11-15 19:54 swift://journeyC.default/speeches/processed/wordcount_json/part-00000-606db79b-b966-4f84-8a7f-
909672c136a7.json
```

Optional: Explore the Spark UI

When you use Spark (via Scala, Python, and/or SQL), you start a session with the Spark server. In many situations, it can be helpful to view the "Spark UI" for your session. BDCS-CE provides easy access to Spark UI for your Zeppelin session.

To view it, follow these steps...

- Click on the Jobs tab
- Find the running ("Processing") Zeppelin job
- From the pop-up menu, choose Spark UI



Spark Jobs

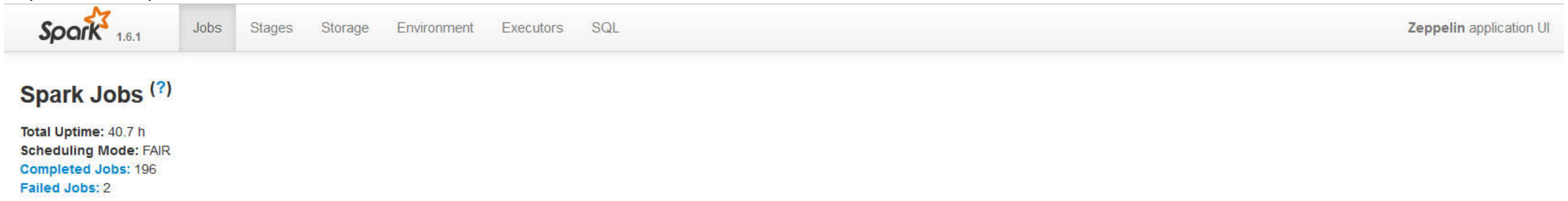
List of all the jobs. You can create new jobs to run on your data sets using New job action.

As of Apr 05, 2017 1:03:44 PM UTC

Zeppelin
Status: Processing
Started At: Apr 03, 2017 8:18:57 PM UTC
Finished At:
Elapsed Time: 1d 16h 40m 58s
Queue: interactive

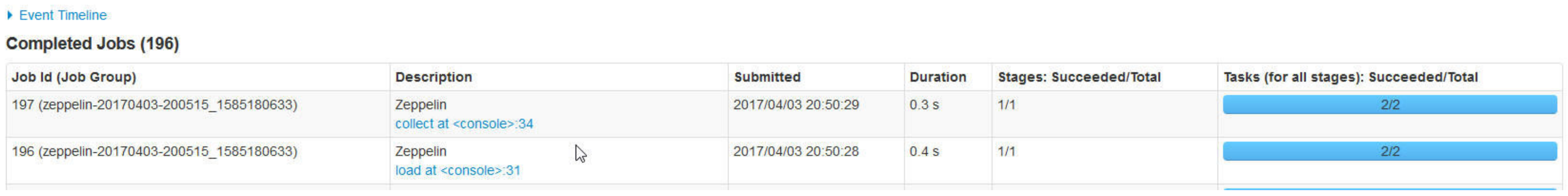
Zeppelin
Status: Success
Started At: Apr 03, 2017 7:41:38 PM UTC
Finished At: Apr 03, 2017 8:18:03 PM UTC
Elapsed Time: 36m 24s
Queue: interactive

- Click on Attempt_1
- Explore the Spark UI



Spark Jobs (?)

Total Uptime: 40.7 h
Scheduling Mode: FAIR
Completed Jobs: 196
Failed Jobs: 2



▸ Event Timeline

Completed Jobs (196)

Job Id (Job Group)	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
197 (zeppelin-20170403-200515_1585180633)	Zeppelin collect at <console>:34	2017/04/03 20:50:29	0.3 s	1/1	2/2
196 (zeppelin-20170403-200515_1585180633)	Zeppelin load at <console>:31	2017/04/03 20:50:28	0.4 s	1/1	2/2

Next Steps

- Review the Spark and Spark SQL documentation here: Spark Quick Start (https://spark.apache.org/docs/1.6.1/quick-start.html) and Spark SQL Programming Guide (https://spark.apache.org/docs/1.6.1/sql-programming-guide.html)
- Experiment with your own data sets
- Proceed to one of the other tutorials

Change Log

September 7. 2017 - Confirmed it works with 17.3.5-20. Partially switched to bdfs
August 13, 2017 - Confirmed it works with BDCSCE 17.3.3-20
August 11, 2017 - Journey v2. Confirmed it works with Spark 2.1
July 28, 2017 - Updated to work with BDCSCE 17.3.1-20

```
%md
```