

## Interacting with Apache Spark

In this document we are going to review:

- I. Spark cluster resources and Web UIs
- II. Login Spark Cluster
- III. HDFS
- IV. Interact with Spark interactively with a Shell
- V. Test Spark on a Self-Contained application
  - a. Local mode
  - b. Yarn Resource Manager
  - c. Standalone Resource Manager
- VI. Interact with Apache Spark by using Jupyter Notebooks

## Spark cluster resources

### Architecture

- node1 : HDFS NameNode + Spark Master + Anaconda + Jupyter
- node2 : YARN ResourceManager + JobHistoryServer + ProxyServer
- node3 : HDFS DataNode + YARN NodeManager + Spark Slave
- node4 : HDFS DataNode + YARN NodeManager + Spark Slave

**Web UIs:** You can check the following URLs to monitor your work

- NameNode (<http://10.211.55.101:50070/dfshealth.html>): Tells you information about hadoop filesystem
- ResourceManager (<http://10.211.55.102:8088/cluster>): Tells you information about the jobs submitted to the Hadoop Cluster by using Yarn
- Spark (<http://10.211.55.101:8080>): Tells you information about the jobs submitted to Spark in standalone mode.
- Spark History (<http://10.211.55.101:18080>)
- Jupyter Notebook (<http://10.211.55.101:8888>)

## Login into Spark Cluster

**Users will mainly log into node 1 as a root user:**

```
>> vagrant ssh node-1
>> sudo su
```

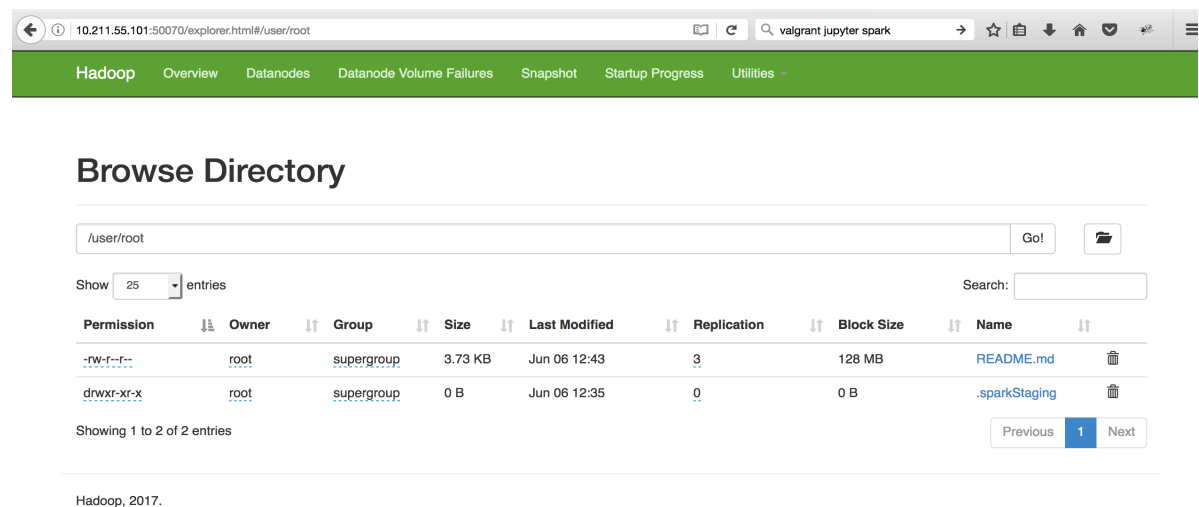
## HDFS

## Storing a file into HDFS file system

Type the following command to store a file ( /usr/local/README.md) file into HDFS file system. We will use this file later for testing spark.

```
>> hdfs dfs -put /usr/local/README.md README.md
```

Check the file in the **NameNode UI** → In (Utilities / Browse Directory) → you will be able to see this file now in your /user/root/ directory.



## Interact with Spark interactively with a Shell

Type the following command to start a PySpark Shell.

```
>> $SPARK_HOME/bin/pyspark
```

```
Welcome to  
      _/_/_/_/_/_/_/_/_/_  
     /_/_/_/_/_/_/_/_/_\ version 2.1.1  
    /\_/\_/\_/\_/\_/\_/  
   //__//__//__//__//__\\  
  //___//___//___//___//  
 /_____/_____/_____/_____\
```

Using Python version 3.6.1 (default, May 11 2017 13:09:58)  
SparkSession available as 'spark'.  
>>>

Type the following commands in your PySpark shell

```
>> textFile = sc.textFile("README.md")
>> textFile.first()
'# Apache Spark'
>> linesWithSpark = textFile.filter(lambda line: "Spark" in line)
>> textFile.filter(lambda line: "Spark" in line).count() # How many
lines contain "Spark"?
20
>> textFile.map(lambda line: len(line.split())).reduce(lambda a, b:
a if (a > b) else b)
22
>> wordCounts = textFile.flatMap(lambda line:
line.split()).map(lambda word: (word, 1)).reduceByKey(lambda a, b:
a+b)
>> wordCounts.collect()
[('#', 1), ('Apache', 1), ('Spark', 16), ('is', 6), ('a', 8),
('fast', 1), ('and', 9), ('general', 3), ('cluster', 2), .....
('contributing', 1), ('project.', 1)]
>> linesWithSpark.cache()
PythonRDD[10] at RDD at PythonRDD.scala:48
>> linesWithSpark.count()
20
>> exit()
```

## Test Spark on a Self-Contained application

Lets write our first self-contained application using the Spark API in Python.  
Copy the following lines into a script called SimpleApp.py

```
-----
"""SimpleApp.py"""
from pyspark import SparkContext

logFile = "README.md" # Should be some file on your system
sc = SparkContext("local", "Simple App")
textFile = sc.textFile(logFile)

wordCounts = textFile.flatMap(lambda line:
line.split()).map(lambda word: (word, 1)).reduceByKey(lambda
a, b: a+b)
wordCounts.collect()

sc.stop()
```

---

For submitting the application to the Spark Cluster, you will need to use the spark-submit command:

```
./bin/spark-submit \ --class <main-class> \ --master <master-url> \
\ --deploy-mode <deploy-mode> \ --conf <key>=<value> \ ... #
other options <application-jar> \ [application-arguments]
```

This command takes care of setting up the classpath with Spark and its dependencies, and can support different cluster managers and deploy modes that Spark supports:

- `--class`: The entry point for your application (e.g. `org.apache.spark.examples.SparkPi`)
- `--master`: The [master URL](#) for the cluster (e.g. `spark://node1:7077`, `yarn-cluster`, `local`)
- `--deploy-mode`: Whether to deploy your driver on the worker nodes (`cluster`) or locally as an external client (`client`)
- `--conf`: Arbitrary Spark configuration property in key=value format. For values that contain spaces wrap “key=value” in quotes (as shown).
- `application-jar`: Path to a bundled jar including your application and all dependencies. The URL must be globally visible inside of your cluster, for instance, an `hdfs://` path or a `file://` path that is present on all nodes.
- `application-arguments`: Arguments passed to the main method of your main class, if any

### Launching SimpleApp.py on local mode with 2 cores

```
>> $SPARK_HOME/bin/spark-submit --master local[2] SimpleApp.py
```

### Launching SimpleApp.py on Standalone mode

```
>> $SPARK_HOME/bin/spark-submit --master spark://node1:7077
SimpleApp.py
```

### Launching SimpleApp.py on YARN

```
>> $SPARK_HOME/bin/spark-submit --master yarn-cluster SimpleApp.py
```

A good Spark example to checkout your Spark cluster instance is the SparkPi example. (Note: the source code of this example is located in “/usr/local/spark/examples/src/main/scala/org/apache/spark/examples/SparkPi.scala”)

### Testing SparkPi on YARN

```
>> $SPARK_HOME/bin/spark-submit --class
org.apache.spark.examples.SparkPi --master yarn-cluster --num-
executors 10 --executor-cores 2 $SPARK_HOME/examples/jars/spark-
examples_2.11-2.1.1.jar 10
```

Check the **Resource Manager UI** to see if the job status: running/finished/failed

application_1496748187136_0004	root	org.apache.spark.examples.SparkPi	SPARK	default	0	Tue Jun 6 14:07:09 +0100 2017	Tue Jun 6 14:07:56 +0100 2017	FINISHED	SUCCEEDED	N
--------------------------------	------	-----------------------------------	-------	---------	---	-------------------------------	-------------------------------	----------	-----------	---

Check the **Spark History UI** and select the application that you just submitted. See the options available (e.g. Event TimeLine, DAG visualizations).

## Spark Jobs (?)

User: root  
Total Uptime: 37 s  
Scheduling Mode: FIFO  
Completed Jobs: 1

▶ Event Timeline

### Completed Jobs (1)

Job Id ▾	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	<a href="#">reduce at SparkPi.scala:38</a>	2017/06/15 10:34:25	4 s	1/1	10/10

## Test SparkPi on Standalone mode

```
>> $SPARK_HOME/bin/spark-submit --class
org.apache.spark.examples.SparkPi --master spark://node1:7077 --num-
executors 10 --executor-cores 1 $SPARK_HOME/examples/jars/spark-
examples_2.11-2.1.1.jar 10
```

If you check the **Resource Manager UI**, you will see we didn't get any new entry. This is because we didn't use the YARN-Hadoop cluster to submit the job. However, if you check the **Spark History UI**, you will see a new entry.

## Test SparkPi on local mode

```
>> $SPARK_HOME/bin/spark-submit --class
org.apache.spark.examples.SparkPi --master local --num-executors 10 -
-executor-cores 1 $SPARK_HOME/examples/jars/spark-examples_2.11-
2.1.1.jar 10
```

## Check the Spark History UI

<div> <div>10.211.55.101:18080</div> <div>spark anaconda vagrant</div> </div>							
<div> <div> <div>spark</div> <div>2.1.1</div> </div> <div>History Server</div> </div>							
<div> <div>Event log directory: hdfs://node1/tmp/spark-events</div> <div>Last updated: 06/06/2017, 13:50:29</div> </div>							
<div> <div>Search:</div> <div></div> </div>							
App ID	App Name	Started	Completed	Duration	Spark User	Last Updated	Event Log
local-1496753398812	PySparkShell	2017-06-06 12:49:56	2017-06-06 12:50:29	33 s	root	2017-06-06 12:50:29	<a href="#">Download</a>
app-20170606124543-0005	PySparkShell	2017-06-06 12:45:41	2017-06-06 12:47:03	1.4 min	root	2017-06-06 12:47:03	<a href="#">Download</a>
app-20170606123511-0004	PySparkShell	2017-06-06 12:35:09	2017-06-06 12:36:36	1.4 min	root	2017-06-06 12:36:36	<a href="#">Download</a>
app-20170606122347-0003	PySparkShell	2017-06-06 12:23:44	2017-06-06 12:34:18	11 min	root	2017-06-06 12:34:18	<a href="#">Download</a>
app-20170606120721-0002	PySparkShell	2017-06-06 12:07:19	2017-06-06 12:12:10	4.8 min	root	2017-06-06 12:12:10	<a href="#">Download</a>
local-1496749732745	PySparkShell	2017-06-06 11:48:51	2017-06-06 11:54:08	5.3 min	root	2017-06-06 11:54:08	<a href="#">Download</a>
local-1496749577058	PySparkShell	2017-06-06 11:46:15	2017-06-06 11:46:39	24 s	root	2017-06-06 11:46:39	<a href="#">Download</a>
local-1496749368028	PySparkShell	2017-06-06 11:42:46	2017-06-06 11:43:14	28 s	root	2017-06-06 11:43:14	<a href="#">Download</a>
local-1496749298022	PySparkShell	2017-06-06 11:41:36	2017-06-06 11:42:00	24 s	root	2017-06-06 11:42:00	<a href="#">Download</a>
app-20170606114015-0001	PySparkShell	2017-06-06 11:40:13	2017-06-06 11:41:20	1.1 min	root	2017-06-06 11:41:20	<a href="#">Download</a>
app-20170606113623-0000	Spark Pi	2017-06-06 11:36:20	2017-06-06 11:36:32	12 s	root	2017-06-06 11:36:32	<a href="#">Download</a>
application_1496748187136_0002	Spark Pi	2017-06-06 11:34:56	2017-06-06 11:35:30	35 s	root	2017-06-06 11:35:30	<a href="#">Download</a>
application_1496748187136_0001	Spark Pi	2017-06-06 11:33:08	2017-06-06 11:33:57	48 s	root	2017-06-06 11:33:57	<a href="#">Download</a>

### Curiosities:

local-XXXX → Application submitted locally

app-XXXX → Application submitted with the standalone cluster

application\_xxxxxx → Application submitted with YARN

Select a job, and explore the different information available (e.g DAG, Event TimeLine, Stages, etc.)

## Test Spark on a Jupyter Notebooks

Three environment variables need particular values to be able to work with Jupyter and Notebooks:

```
PYSPARK_PYTHON=/usr/local/anaconda/bin/python3.6
```

```
PYSPARK_DRIVER_PYTHON=jupyter
```

```
PYSPARK_DRIVER_PYTHON_OPTS="notebook --ip=0.0.0.0 --allow-root"  
/usr/local/spark/bin/pyspark
```

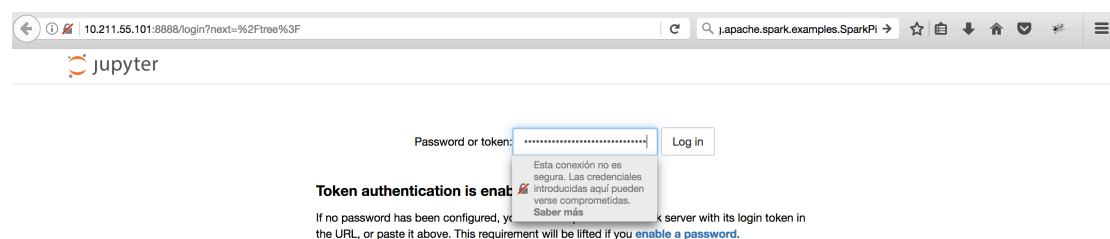
We have already a script called **start\_Jupyter\_local.sh** in  
/home/vagrant/notebooks

```
>> cd notebooks
```

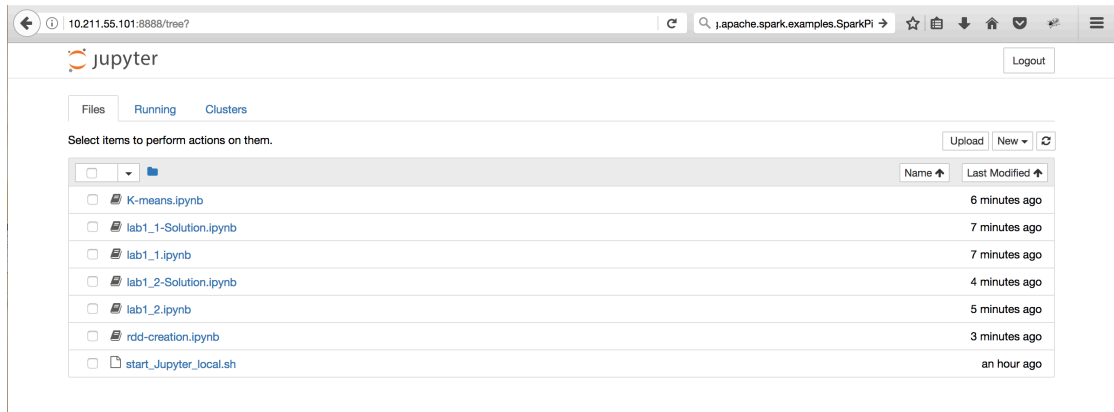
```
>> ./start_jupyter_local.sh
```

```
[root@node1 notebooks]# ./start_Jupyter_local.sh  
[I 12:15:10.733 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret  
[I 12:15:10.782 NotebookApp] Serving notebooks from local directory: /home/vagrant/notebooks  
[I 12:15:10.782 NotebookApp] 0 active kernels  
[I 12:15:10.782 NotebookApp] The Jupyter Notebook is running at: http://0.0.0.0:8888/?token=9e3ea03d3bce3871924e287af665ba19f2d1e2372e507  
[I 12:15:10.782 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).  
[W 12:15:10.782 NotebookApp] No web browser found: could not locate runnable browser.  
[C 12:15:10.782 NotebookApp]  
  
Copy/paste this URL into your browser when you connect for the first time,  
to login with a token:  
http://0.0.0.0:8888/?token=9e3ea03d3bce3871924e287af665ba19f2d1e2372e507
```

Go to your browser and type: <http://10.211.55.101:8888>



Copy/Paste the token (this is only needed to do it the first time) that the shell gave into the browser (for this example the token is:  
9e3ea03d3bce3871924e287af665ba19f2d1e2372e507)



Select lab1\_1 and start the exercise.

The screenshot shows the JupyterLab notebook interface for the file 'lab1\_1'. The top bar indicates 'Last Checkpoint: 2 minutes ago (unsaved changes)'. The notebook contains the following content:

Next, try executing the code in the code cell below using one of the described methods.

```
In [1]: # Assign the value 43 to the variable a
a = 42
print(a)
42
```

The typical life cycle of a Spark program is –

- Create RDDs from some external data source or parallelize a collection in your driver program.
- Lazily transform the base RDDs into new RDDs using transformations.
- Cache some of those RDDs for future reuse.
- Perform actions to execute parallel computation and to produce results.

**Part 2: Creating RDDs**

The typical life cycle of a Spark program is:

- Create RDDs from some external data source or parallelize a collection.
- Lazily transform the base RDDs into new RDDs using transformations.
- Cache some of those RDDs for future reuse.
- Perform actions to execute parallel computation and to produce results.