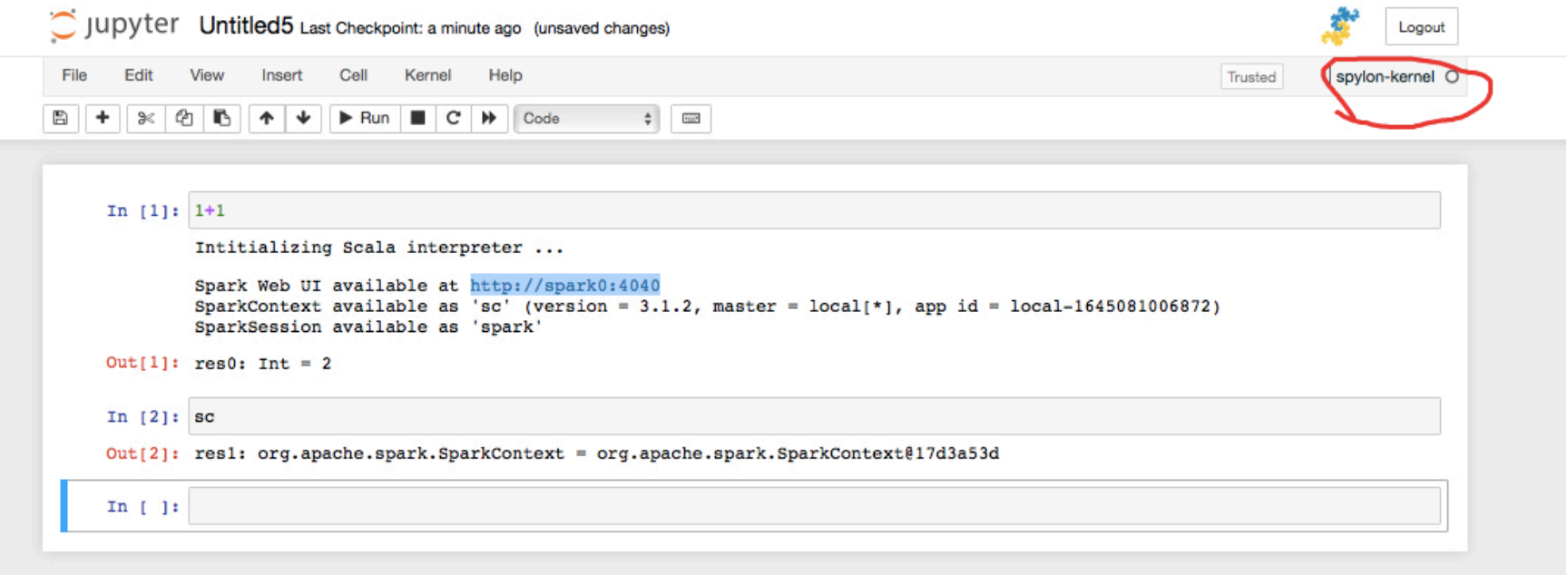
Configure scala on Jupyter.

Option I: Using spylon – Execute Scala spark program.

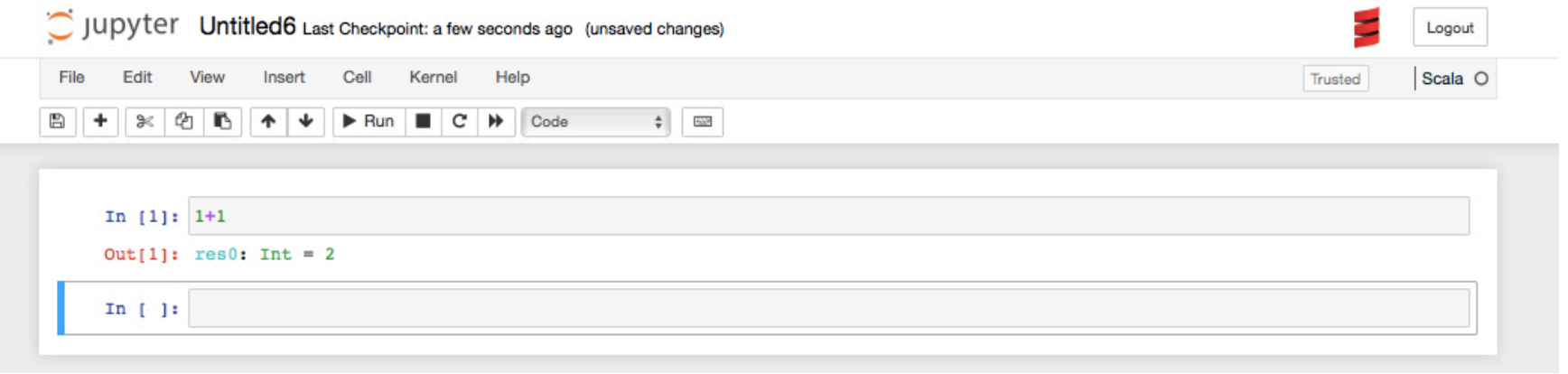
```
#pip3 install spylon-kernel
#python3 -m spylon_kernel install
#export SPARK_HOME=/opt/spark
#jupyter notebook
```

In the web console, select New -> Spylon-kernel and enter the following commands.



Option 2: using Scala in Jupyter

https://almond.sh/docs/quick-start-install




Or using Anaconda: (Optional)

Anaconda 4.2.0
For Linux
Anaconda is BSD licensed which gives you permission to use Anaconda commercially and for redistribution.
Changelog
Download the installer
Optional: Verify data integrity with MD5 or SHA-256 _ More info
In your terminal window type one of the below and follow the instructions:
Python 3.5 version
bash Anaconda3-4.2.0-Linux-x86_64.sh
Python 2.7 version

bash Anaconda2-4.2.0-Linux-x86_64.sh

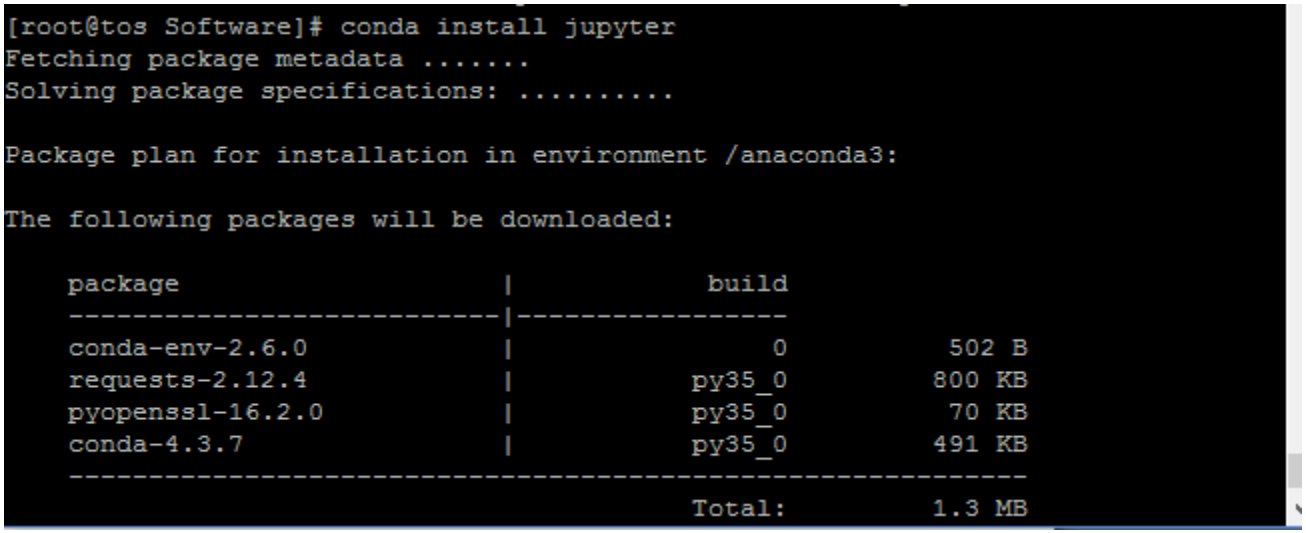
NOTE: Include the "bash" command even if you are not using the bash shell.

```
[root@tos Software]# bash
[root@tos Software]# sh Anaconda3-4.2.0-Linux-x86_64.sh

Welcome to Anaconda3 4.2.0 (by Continuum Analytics, Inc.)

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>>
```

```
Anaconda3 will now be installed into this location:
/root/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/root/anaconda3] >>> /anaconda3
```

```
Do you wish the installer to prepend the Anaconda3 install location
to PATH in your /root/.bashrc ? [yes|no]
[no] >>> yes

Prepending PATH=/anaconda3/bin to PATH in /root/.bashrc
A backup will be made to: /root/.bashrc-anaconda3.bak


For this change to become active, you have to open a new terminal.

Thank you for installing Anaconda3!

Share your notebooks and packages on Anaconda Cloud!
Sign up for free: https://anaconda.org

[root@tos Software]#
```

#bash
Run the following command to install Jupyter.
#conda install jupyter



```
The following packages will be UPDATED:

    conda:     4.2.9-py35_0  --> 4.3.7-py35_0
    pyopenssl: 16.0.0-py35_0 --> 16.2.0-py35_0
    requests:  2.11.1-py35_0 --> 2.12.4-py35_0

Proceed ([y]/n)? y

Fetching packages ...
conda-env-2.6. 100% |################################| Time: 0:00:00 129.08 kB/s
requests-2.12. 100% |################################| Time: 0:00:11  70.63 kB/s
pyopenssl-16.2 100% |################################| Time: 0:00:01  53.66 kB/s
conda-4.3.7-py 100% |################################| Time: 0:00:04 104.02 kB/s
Extracting packages ...
[      COMPLETE      ]|##################################################| 100%
Unlinking packages ...
[      COMPLETE      ]|##################################################| 100%
Linking packages ...
[      COMPLETE      ]|##################################################| 100%
dbus post-link :: /etc/machine-id not found ..
dbus post-link :: .. using /proc/sys/kernel/random/boot_id
[root@tos Software]#
```

**Note:**
Configure default Python version, If you have multiples python version

```
#alternatives --config python3
or
#update-alternatives --config python3
```

Reference:
https://www.sicara.ai/blog/2017-05-02-get-started-pyspark-jupyter-notebook-3-minutes
https://community.hortonworks.com/articles/75551/installing-and-exploring-spark-20-with-jupyter-not.html
https://medium.com/@bogdan.cojocar/how-to-run-scala-and-spark-in-the-jupyter-notebook-328a80090b3b

----------Lab Ends Here ----------

## 15. Explore Spark Two Nodes Cluster – 90 Minutes

In this lab, we will configure 2 nodes spark standalone cluster and deploy application on the cluster.

Prerequisites:

- You required 2 instances of node.
  - Spark0
  - Spark1

Architecture

| Roles | Hostname | Remarks |
|---|---|---|
| **master** | Sparko | Install Spark |
| **slave/worker** | Spark1 | Install Spark |

Let us start the Spark master – sparko.

Install spark on the master node.  You can refer the first lab or skip already done.

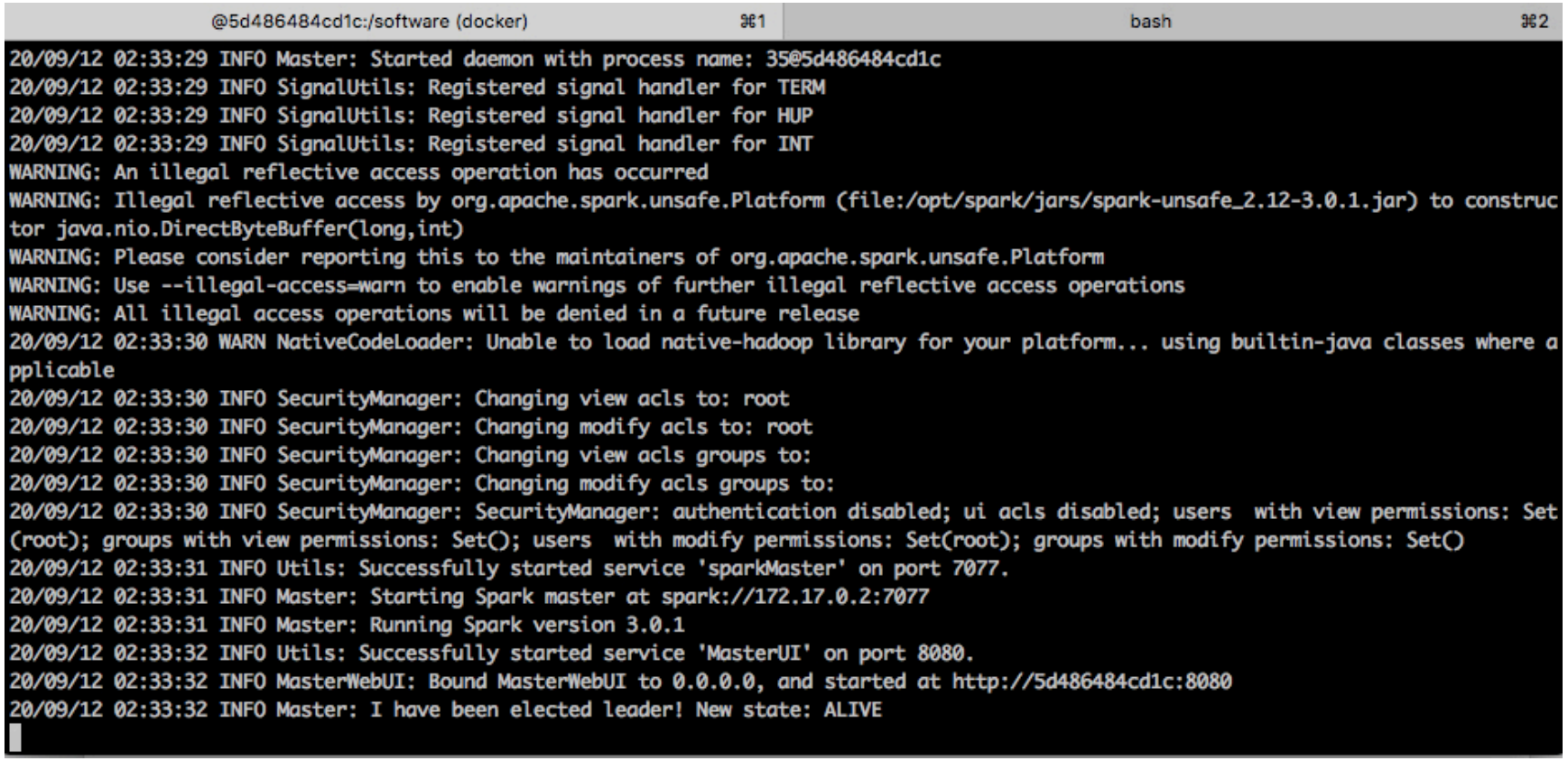
Execute the following in Sparko

Setup a Spark master node

Change the Master Port to 8090, there are some issues when it is executed in 7077 port in a docker environment.

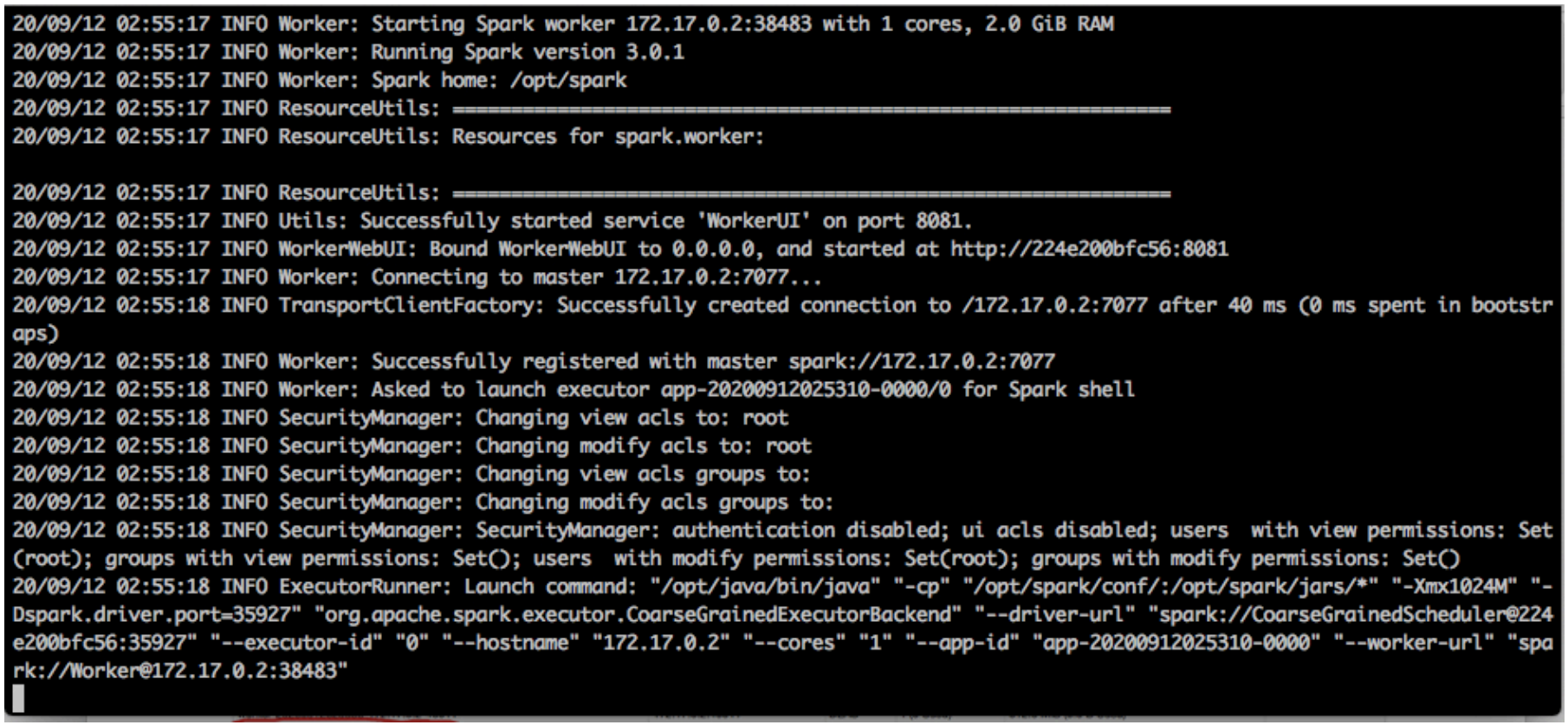
Execute on the sparko – Terminal.

```
#export PATH=$PATH:/opt/spark/bin
#export SPARK_MASTER_PORT=8090
#spark-class org.apache.spark.deploy.master.Master
```


@5d486484cd1c:/software (docker) ⌘1
bash ⌘2
20/09/12 02:33:29 INFO Master: Started daemon with process name: 35@5d486484cd1c
20/09/12 02:33:29 INFO SignalUtils: Registered signal handler for TERM
20/09/12 02:33:29 INFO SignalUtils: Registered signal handler for HUP
20/09/12 02:33:29 INFO SignalUtils: Registered signal handler for INT
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.apache.spark.unsafe.Platform (file:/opt/spark/jars/spark-unsafe_2.12-3.0.1.jar) to constructor java.nio.DirectByteBuffer(long,int)
WARNING: Please consider reporting this to the maintainers of org.apache.spark.unsafe.Platform
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
20/09/12 02:33:30 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
20/09/12 02:33:30 INFO SecurityManager: Changing view acls to: root
20/09/12 02:33:30 INFO SecurityManager: Changing modify acls to: root
20/09/12 02:33:30 INFO SecurityManager: Changing view acls groups to:
20/09/12 02:33:30 INFO SecurityManager: Changing modify acls groups to:
20/09/12 02:33:30 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions: Set(root); groups with view permissions: Set(); users with modify permissions: Set(root); groups with modify permissions: Set()
20/09/12 02:33:31 INFO Utils: Successfully started service 'sparkMaster' on port 7077.
20/09/12 02:33:31 INFO Master: Starting Spark master at spark://172.17.0.2:7077
20/09/12 02:33:31 INFO Master: Running Spark version 3.0.1
20/09/12 02:33:32 INFO Utils: Successfully started service 'MasterUI' on port 8080.
20/09/12 02:33:32 INFO MasterWebUI: Bound MasterWebUI to 0.0.0.0, and started at http://5d486484cd1c:8080
20/09/12 02:33:32 INFO Master: I have been elected leader! New state: ALIVE

Start worker process On sparko: In a separate terminal.
Attach a worker node to the cluster, execute the following in the sparko container.

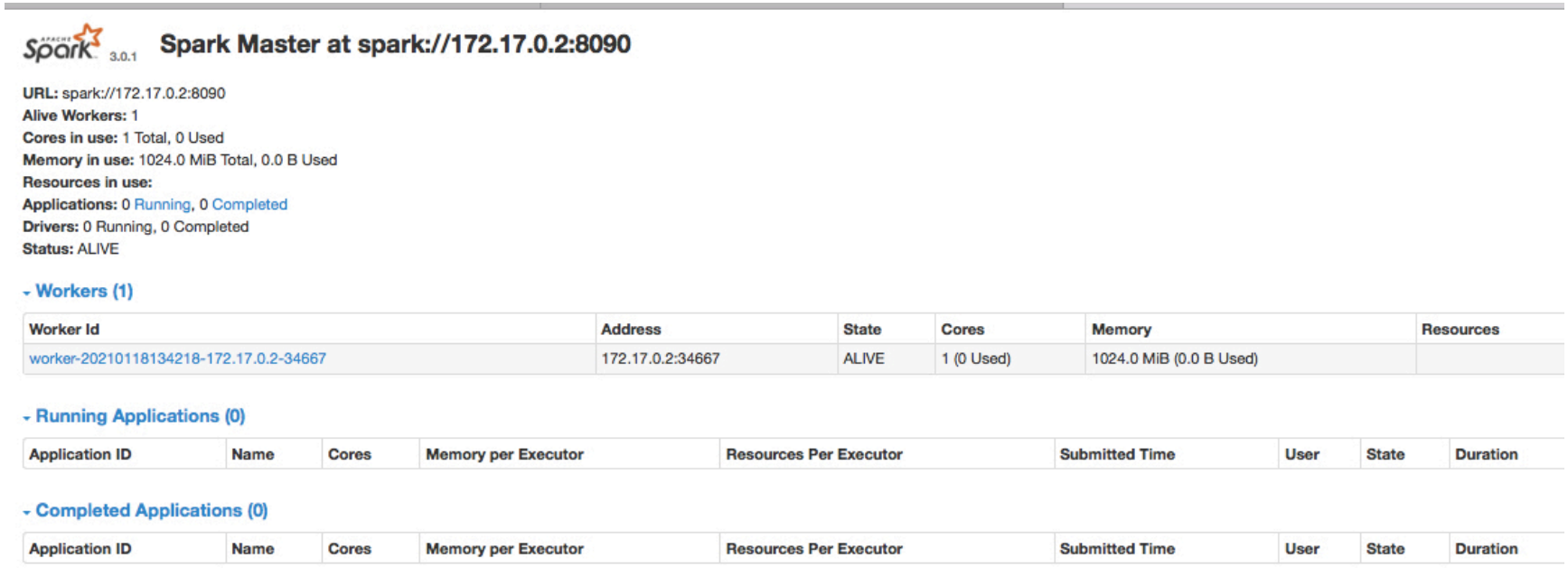
```
#export PATH=$PATH:/opt/spark/bin
#spark-class org.apache.spark.deploy.worker.Worker -c 1 -m 1G spark://sparko:8090
```




If unable to connect to sparko replace it with the container IP.

Refresh the web UI, ensure that you can see a worker as shown below.

http://127.0.0.1:8080




IP against the worker node will depends on the environment or your system.

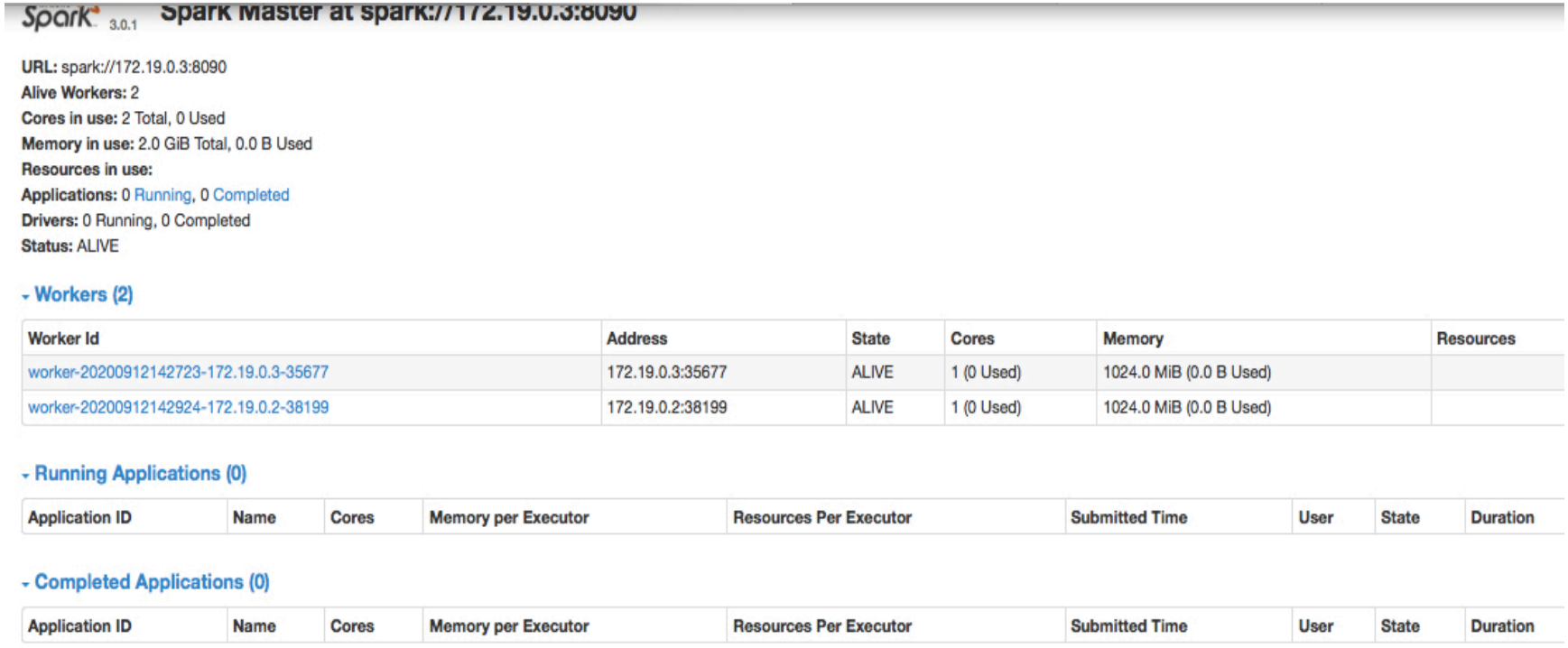
Start the Node on spark1

Ensure that you have install spark in the Spark1 too.

On the spark1 terminal, it will register to the spark master – spark0

```
# spark-class org.apache.spark.deploy.worker.Worker -c 1 -m 1G spark://spark0:8090
```

At the end of this step, you should have 2 worker nodes as shown below: Refresh the Spark Web UI.



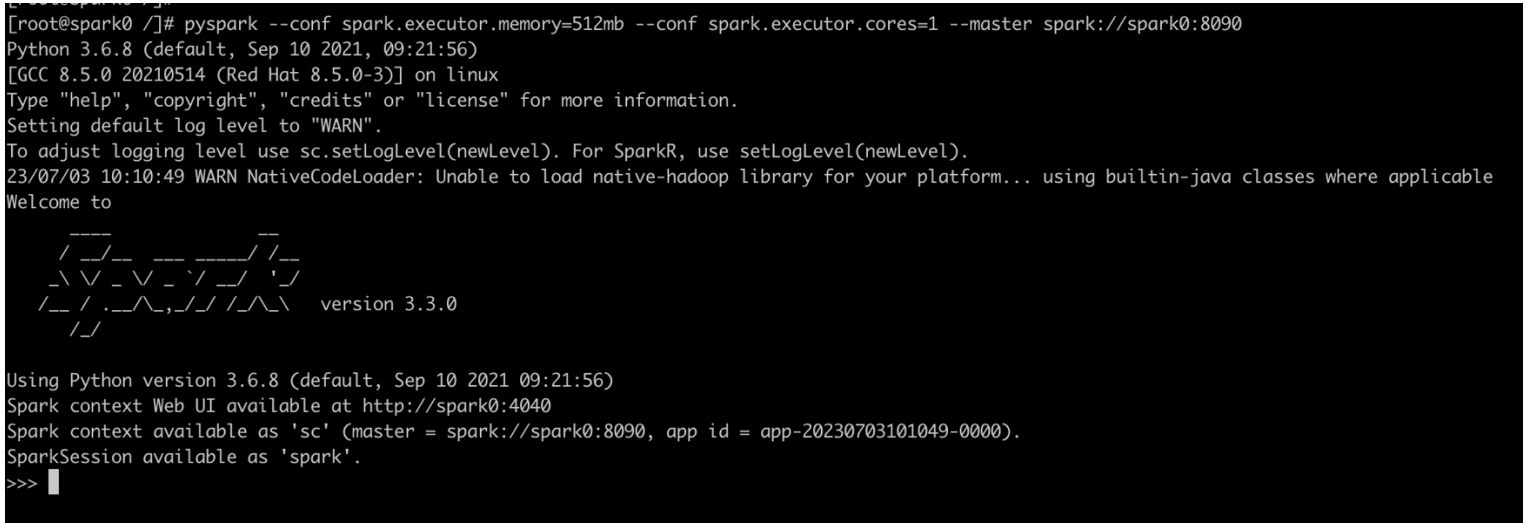
Open a pyspark shell and connect to the Spark cluster

Let us execute Spark shell using Cluster mode:

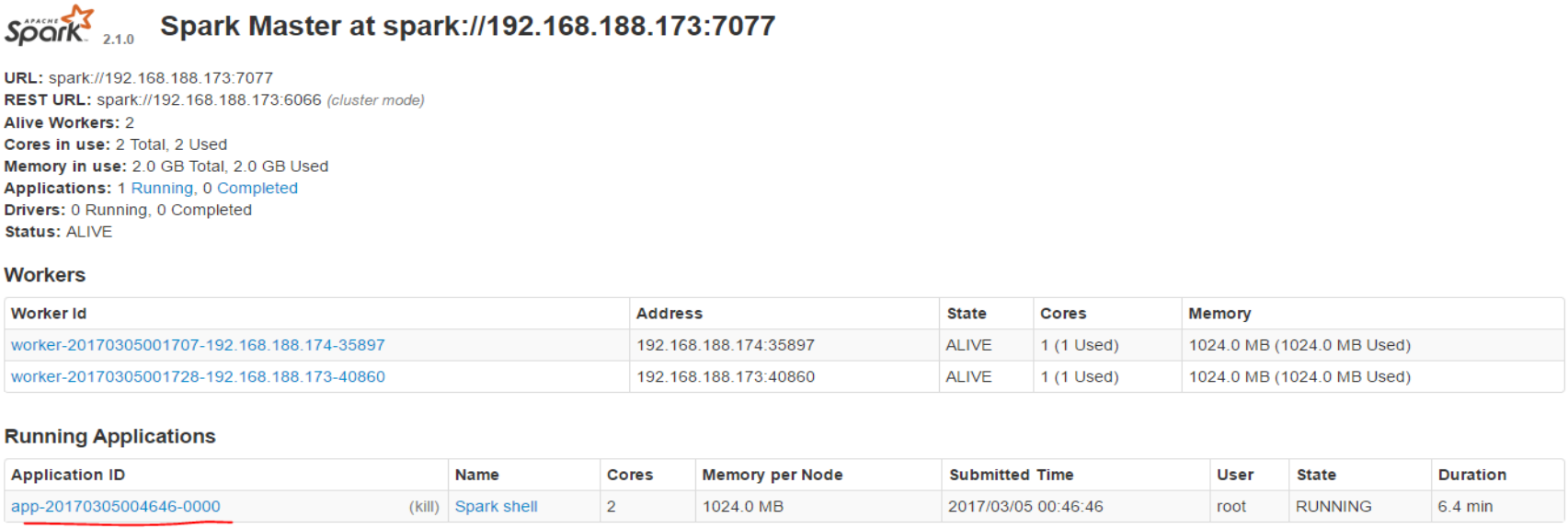
You can perform from any client/server node i.e windows desktop client.

Connecting an Application to the Cluster

To run an application on the Spark cluster, simply pass the spark://IP:PORT URL of the master as to the SparkContext constructor.
To run an interactive Spark shell against the cluster(Specify the master IP), run the following command from the bin folder:
We are executing from the slave node – spark1 console.
#pyspark --conf spark.executor.memory=512mb --conf spark.executor.cores=1 --master spark://spark0:8090



You can verify the application execution from the web UI. There should be an entry in the Applications section.



Spark 2.1.0 **Spark Master at spark://192.168.188.173:7077**

**URL:** spark://192.168.188.173:7077
**REST URL:** spark://192.168.188.173:6066 *(cluster mode)*
**Alive Workers:** 2
**Cores in use:** 2 Total, 2 Used
**Memory in use:** 2.0 GB Total, 2.0 GB Used
**Applications:** 1 Running, 0 Completed
**Drivers:** 0 Running, 0 Completed
**Status:** ALIVE

**Workers**

| Worker Id | Address | State | Cores | Memory |
|---|---|---|---|---|
| worker-20170305001707-192.168.188.174-35897 | 192.168.188.174:35897 | ALIVE | 1 (1 Used) | 1024.0 MB (1024.0 MB Used) |
| worker-20170305001728-192.168.188.173-40860 | 192.168.188.173:40860 | ALIVE | 1 (1 Used) | 1024.0 MB (1024.0 MB Used) |

**Running Applications**

| Application ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|
| app-20170305004646-0000 (kill) | Spark shell | 2 | 1024.0 MB | 2017/03/05 00:46:46 | root | RUNNING | 6.4 min |

Let's make a new RDD from the text of the README file in the Spark source directory:

```
textFile = sc.textFile("/opt/spark/README.md")
```

**Let's start with a few actions:**

```
textFile.count() #Number of items in this RDD
textFile.first() # First item in this RDD
```

```
>>> textFile = sc.textFile("/opt/spark/README.md")
>>> textFile.count() #Number of items in this RDD
124
>>> textFile.first() # First item in this RDD
'# Apache Spark'
>>>
```

Now let's use a transformation. We will use the filter transformation to return a new RDD with a subset of the items in the file. Use collect to display the output.

```
linesWithSpark = textFile.filter(lambda line :  "Spark" in line)
linesWithSpark.collect()
```

```
>>> linesWithSpark = textFile.filter(lambda line :  "Spark" in line)
>>> linesWithSpark.collect()
['# Apache Spark', 'Spark is a unified analytics engine for large-scale data processing. It provides', 'rich set of higher-level tools including Spark SQL
for SQL and DataFrames,', 'pandas API on Spark for pandas workloads, MLlib for machine learning, GraphX for graph processing,', '[![PySpark Coverage](https
://codecov.io/gh/apache/spark/branch/master/graph/badge.svg)](https://codecov.io/gh/apache/spark)', 'You can find the latest Spark documentation, including
 a programming', '## Building Spark', 'Spark is built using [Apache Maven](https://maven.apache.org/).', 'To build Spark and its example programs, run:', '
["Building Spark"](https://spark.apache.org/docs/latest/building-spark.html).', 'For general development tips, including info on developing Spark using an
IDE, see ["Useful Developer Tools"](https://spark.apache.org/developer-tools.html).', 'The easiest way to start using Spark is through the Scala shell:', '
Spark also comes with several sample programs in the `examples` directory.', './bin/run-example SparkPi', 'MASTER=spark://host:7077 ./bin/run-example Spark
Pi', 'Testing first requires [building Spark](#building-spark). Once Spark is built, tests', 'Spark uses the Hadoop core library to talk to HDFS and other
Hadoop-supported', 'Hadoop, you must build Spark against the same version that your cluster runs.', 'in the online documentation for an overview on how to
configure Spark.', 'Please review the [Contribution to Spark guide](https://spark.apache.org/contributing.html)']
>>>
```

We can chain together transformations and actions:

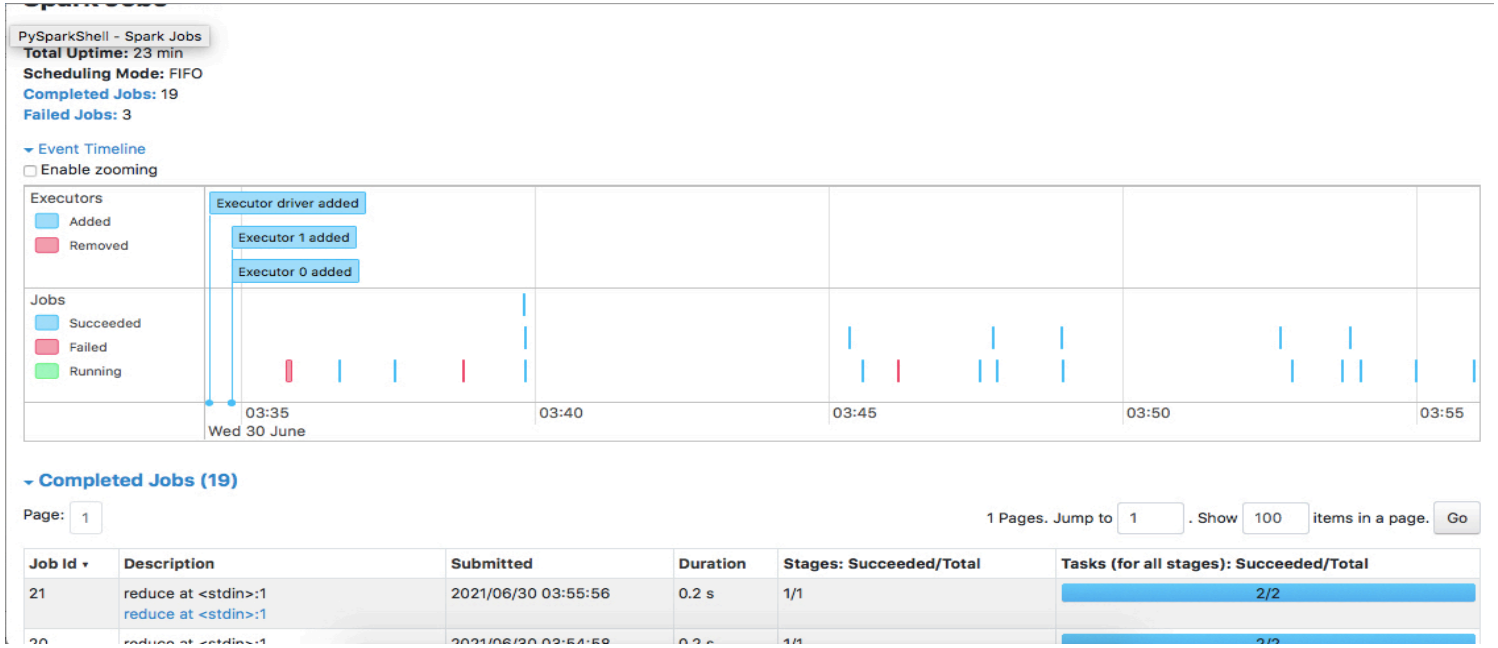
```
textFile.filter(lambda line : "Spark" in line).count()  # How many lines contain "Spark"?
```

Let's say we want to find the line with the most words

```
textFile.map(lambda line : len(line.split(" "))). reduce(lambda a, b: a if (a > b) else b)
```

```
>>> 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)
16
>>>
```

Refer the web UI for the Job details as shown below.




The console should display the result as shown above.

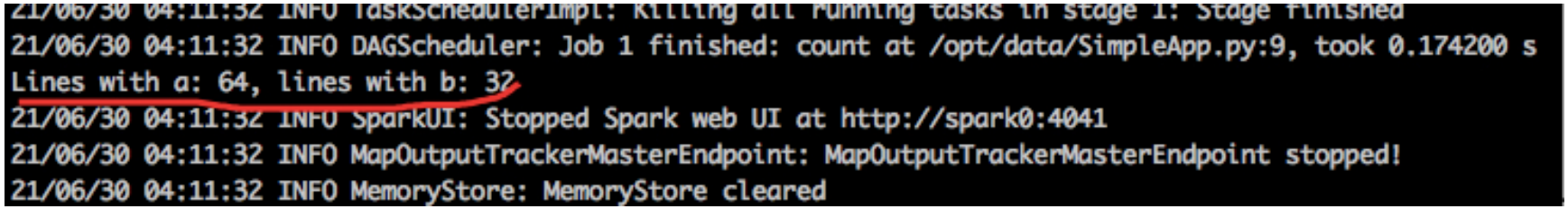
Hints: (For submitting job in cluster)

You need to complete the **Lab - Explore PySpark Application** before going ahead. Submit from the /software folder, since the input file exist in that folder.

You need to copy the data files in both the nodes. However, for our lab its already present in the spark folder. Execute the command from the folder where the python code is.

#spark-submit --master spark://spark0:8090 SimpleApp.py --conf spark.executor.memory=256mb

Verify the output.




---------- Lab Ends Here ----------

https://towardsdatascience.com/diy-apache-spark-docker-bb4f11c10d24

## 16. Explore Spark on Hadoop YARN – 150 Minutes

In this lab, you will deploy Spark Application on Hadoop Cluster.

Steps:

- Configure and install YARN on hadoop0.
- One Node will be exclusively running YARN cluster
- Existing spark0 node to be used to submit spark job on YARN cluster.

Install YARN – On the New Node.

By now, you should have two nodes or servers:

Ensure to follow the naming conditions to avoid confusion. Its very important.

| | Container | Remarks |
|---|---|---|
| **New Node** | Hadoop0 | YARN Services |
| **Existing Node** | Spark0 | Spark Client or Spark. |

All the below commands should be executed on (Hadoop Node – hadoop0) unless specify. We are configuring YARN cluster now.

Change directory to the location where you have the software.

Extract the hadoop & jdk compressed files as shown below:

```
tar -xvf hadoop-* -C /opt
tar -xvf jdk-11* -C /opt
```

Rename the folder for better readability:

```
#mv hadoo* hadoop
#mv jdk* jdk
```

--- Install JDK and set Java Home (Copy the bin file in the YARN folder) , To include JAVA_HOME for all bash users , make an entry in /etc/profile.d as follows:

```
echo "export JAVA_HOME=/opt/jdk/" > /etc/profile.d/java.sh
```

After Unpack the Hadoop distribution. In the distribution, edit the file /opt/hadoop/etc/hadoop/hadoop-env.sh to define some parameters as follows: Look for the below line and update.

```
 # set to the root of your Java installation
 export JAVA_HOME=/opt/jdk

# cd /opt/hadoop
```

Try the following command:

```
 $ bin/hadoop
```

It should list commands as shown below, if everything is configured properly

```
kerbname      show auth_to_local principal conversion
key           manage keys via the KeyProvider
rumenfolder   scale a rumen input trace
rumentrace    convert logs into a rumen trace
s3guard       manage metadata on S3
trace         view and modify Hadoop tracing settings
version       print the version

    Daemon Commands:

kms           run KMS, the Key Management Server

SUBCOMMAND may print help when invoked w/o parameters or with -h.
[root@hadoop0 hadoop]#
```

You need to modify some setting as follows: Replace with your hostname accordingly.

Use and Update the following:

Specify the Namenode hostname and its port.
#vi /opt/hadoop/etc/hadoop/core-site.xml

```
<configuration>
  <property>
    <name>fs.defaultFS</name>
    <value>hdfs://hadoop0:8020</value>
```

```
  </property>
<property>
        <name>io.file.buffer.size</name>
        <value>131072</value>
        <description>Buffer size</description>
</property>
</configuration>
```

Configure its replication.

```
#vi /opt/hadoop/etc/hadoop/hdfs-site.xml

<configuration>
  <property>
    <name>dfs.replication</name>
    <value>1</value>
  </property>
</configuration>
```

Setup passphraseless ssh for the host to logon itself for hadoop services.

Now check that you can ssh to the hadoopo without a passphrase:

```
yum -y install openssh-server openssh-clients
systemctl start sshd
```

Change the password of root using passwd command.

```
#passwd
```

Enter the password.

```
$ ssh hadoop0
```

If you cannot ssh to hadoop0 without a passphrase, execute the following commands:

```
$ ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ chmod 0600 ~/.ssh/authorized_keys
```

Execute to initialize the required parameters on hadoop0 terminal.

```
export HDFS_NAMENODE_USER="root"
export HDFS_DATANODE_USER="root"
export HDFS_SECONDARYNAMENODE_USER="root"
export YARN_RESOURCEMANAGER_USER="root"
export YARN_NODEMANAGER_USER="root"
```

Format the hdfs filesystem:

```
$ hdfs namenode -format
```

```
2023-07-04 21:02:03,839 INFO common.Storage: Storage directory /opt/hdfs/namenode has been successfully formatted.
2023-07-04 21:02:04,017 INFO namenode.FSImageFormatProtobuf: Saving image file /opt/hdfs/namenode/current/fsimage.ckpt_0000000000
0000000000 using no compression
2023-07-04 21:02:04,565 INFO namenode.FSImageFormatProtobuf: Image file /opt/hdfs/namenode/current/fsimage.ckpt_0000000000000000
000 of size 391 bytes saved in 0 seconds .
2023-07-04 21:02:04,647 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
2023-07-04 21:02:04,696 INFO namenode.NameNode: SHUTDOWN_MSG:
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop0/172.18.0.2
************************************************************/
[root@hadoop0 hadoop]#
```

Start NameNode daemon and DataNode daemon:

#cd /opt/hadoop

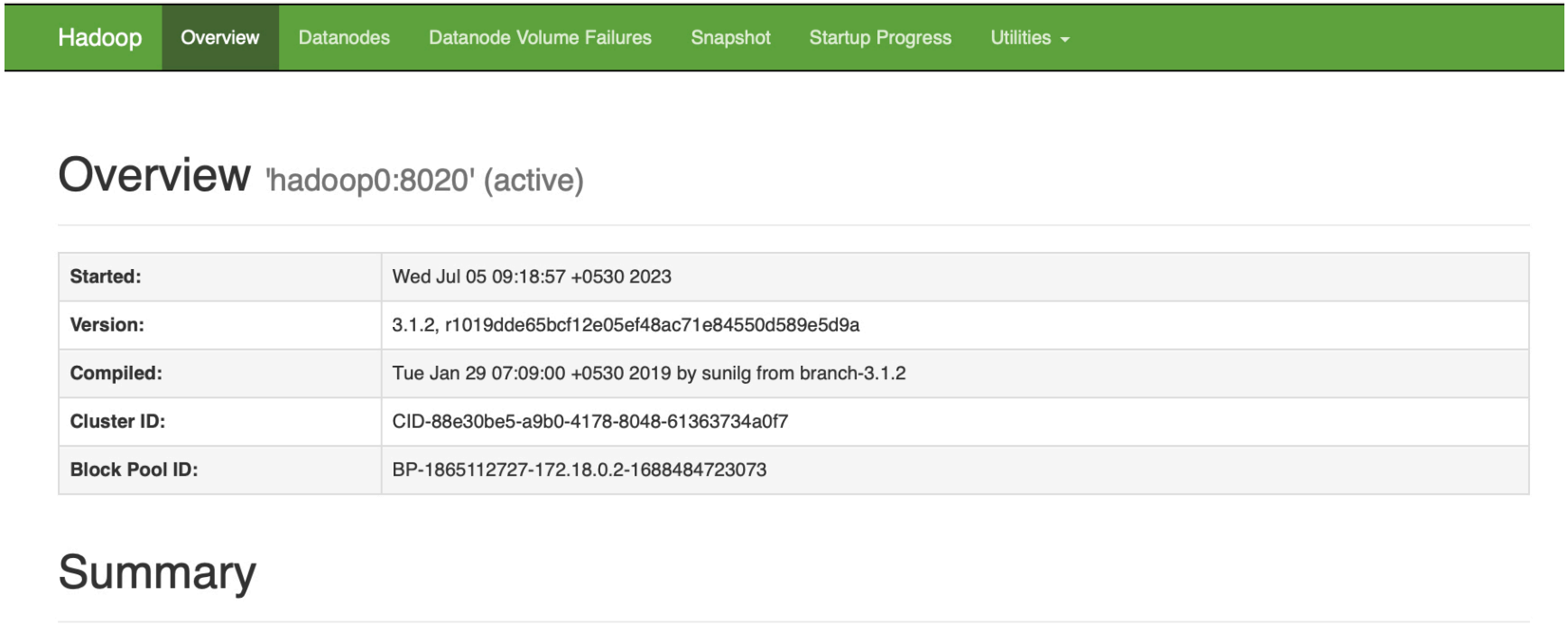
$ sbin/start-dfs.sh

```
[root@hadoop0 hadoop]# sbin/start-dfs.sh
Starting namenodes on [localhost]
Last login: Thu Jan 21 08:00:44 UTC 2021 from localhost on pts/2
Starting datanodes
Last login: Thu Jan 21 08:06:21 UTC 2021 on pts/2
Starting secondary namenodes [hadoop0]
Last login: Thu Jan 21 08:06:23 UTC 2021 on pts/2
hadoop0: Warning: Permanently added 'hadoop0,172.17.0.2' (ECDSA) to the list of known hosts.
[root@hadoop0 hadoop]#
```

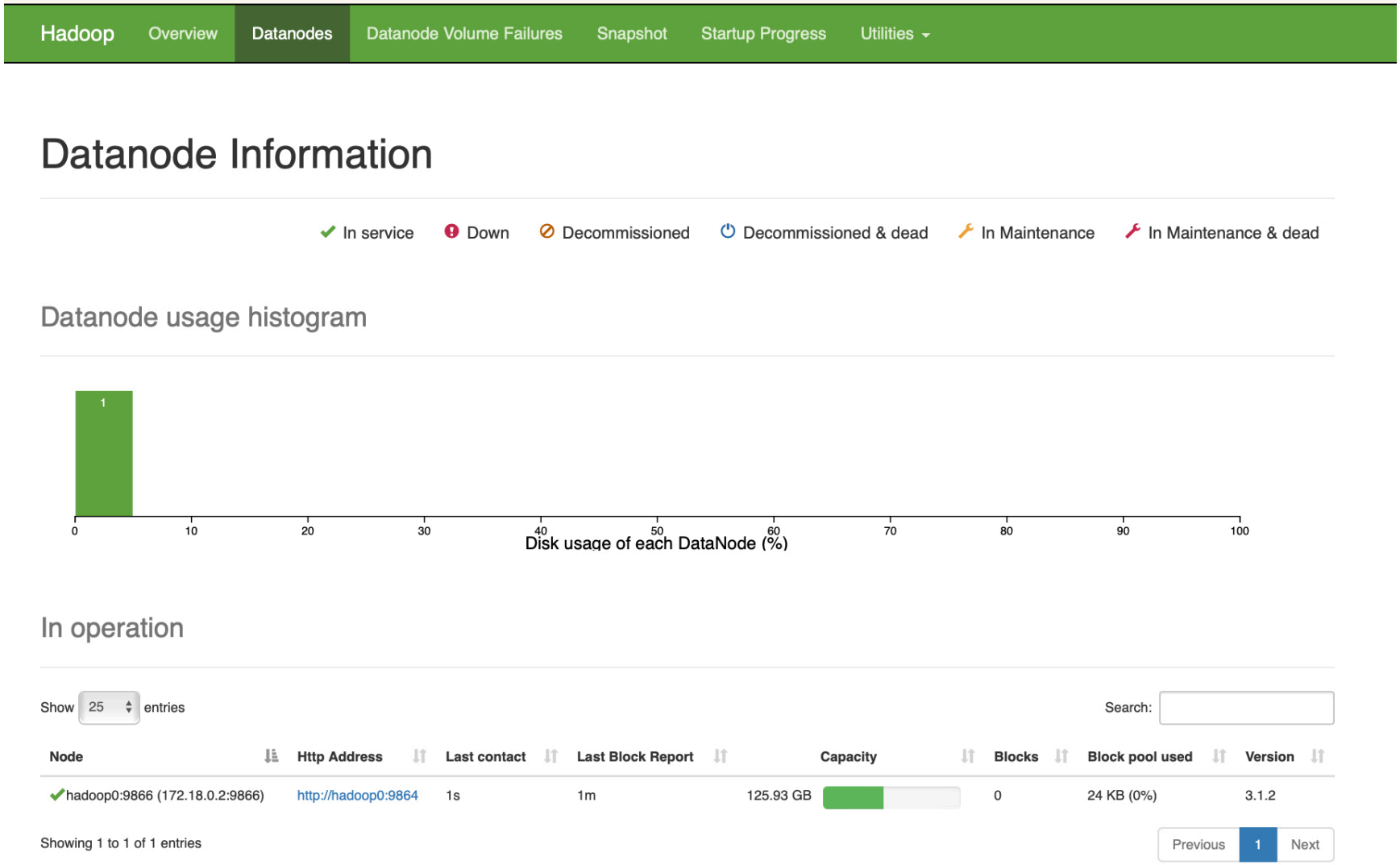
Browse the web interface for the NameNode; by default, it is available at:
NameNode –

http://localhost:9870/

Verify the Node as shown below.



Click on **Datanodes.**



You should have one data node as shown above.

Make the HDFS directories required to execute MapReduce jobs:

```
$ hdfs dfs -mkdir /user
$ hdfs dfs -mkdir /user/root
```

You can run a MapReduce job on YARN in a pseudo-distributed mode

Start HDFS and YARN on the YARN Node: hadoop1

Configure parameters as follows:

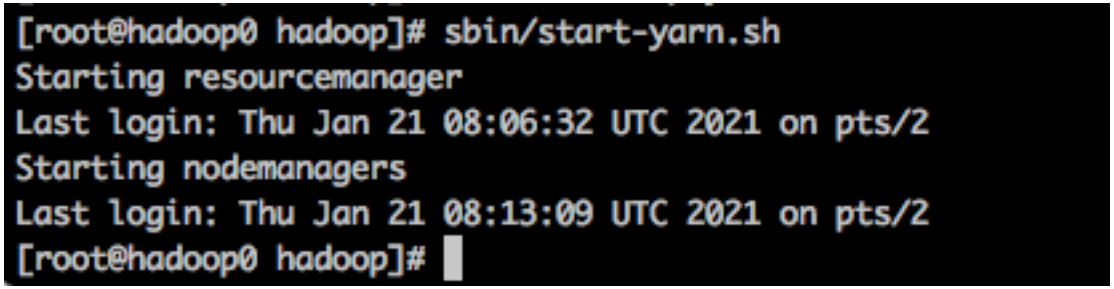
```
#vi /opt/hadoop/etc/hadoop/mapred-site.xml

<configuration>
  <property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
  </property>
  <property>
    <name>mapreduce.application.classpath</name>
<value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HOME/sh
are/hadoop/mapreduce/lib/*</value>
  </property>
</configuration>
```

```
#vi /opt/hadoop/etc/hadoop/yarn-site.xml

<configuration>
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>hadoop0</value>
      <description>The hostname of the RM.</description>
   </property>
<property>
      <name>yarn.nodemanager.aux-services</name>
      <value>mapreduce_shuffle</value>
   </property>
   <property>
      <name>yarn.nodemanager.env-whitelist</name>
<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,
CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_MAPRED_HOME</value
>
   </property>
<property>
<name>yarn.nodemanager.vmem-check-enabled</name>
<value>false</value>
<description>Whether virtual memory limits will be enforced for containers</description>
</property>
<property>
<name>yarn.nodemanager.vmem-pmem-ratio</name>
<value>4</value>
<description>Ratio between virtual memory to physical memory when setting memory limits
forcontainers</description>
</property>
```

```
</configuration>
```

Start ResourceManager daemon and NodeManager daemon
Open a terminal to hadoopo

```
#cd /opt/hadoop
$ sbin/start-yarn.sh
```




Browse the web interface for the ResourceManager; by default it is available at:
ResourceManager - http://localhost:8088/

#verify with jps , all the below services should be there.

```
jps
```

```
[root@hadoop0 hadoop]# export PATH=$PATH:/opt/jdk/bin
[root@hadoop0 hadoop]# jps
1826 NodeManager
919 NameNode
1034 DataNode
1210 SecondaryNameNode
1708 ResourceManager
2174 Jps
[root@hadoop0 hadoop]#
```

Hadoop Node - Let us set some environment and path variable as follows: ( vi ~/.bashrc)

```
export HADOOP_HOME=/opt/hadoop
export PATH=$HADOOP_HOME/bin:$PATH
```

Let us create a temporary folder, that will be used for working space for the YARN cluster.

```
hdfs dfs -mkdir /tmp
hdfs dfs -chmod -R 1777 /tmp
hdfs dfs -mkdir /tmp/in
hdfs dfs -ls /tmp
```

You can get the README.md file from the Software folder, which is provided along with the training.

```
hdfs dfs -copyFromLocal /opt/hadoop/README.txt /tmp/in
hdfs dfs -ls /tmp/in
```

```
hdfs dfs -mkdir  /tmp/spark-events
```

Congrats! You have successfully configured Yarn Cluster.

Let us configure hadoop client setting on the Spark node -> sparko.

Compress the Hadoop folder on the Hadoopo instances or Hadoop node.
#cd /opt
#tar -cvzf hadoop.tar hadoop/

Copy the compressed hadoop folder to the sparko or spark node and uncompressed in /opt folder.
[Docker:
copy from hadoop to host : docker cp hadoopo:/opt/hadoop.tar .
copy from host to spark : docker cp hadoop.tar sparko:/opt/
docker cp hadoopo:/opt/hadoop.tar sparko:/opt/
docker cp hadoop.tar hadoop1:/opt/hadoop.tar
]

On the **sparko** or spark instance.
#tar -xvf hadoop.tar

Update **yarn-site.xml** on the Spark Node.
#vi /opt/spark/etc/hadoop/yarn-site.xml

```
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>hadoopo</value>
      <description>The hostname of the RM.</description>
   </property>
<property>
```

```
    <name>yarn.resourcemanager.address</name>
    <value>hadoop0:8032</value>
    <description>The hostname of the RM.</description>
  </property>
```

Create the following folder on HDFS cluster and copy the file in the folder.

```
# hdfs dfs -mkdir -p /opt/spark
# hdfs dfs -copyFromLocal /opt/spark/README.md /opt/spark/
```

You can verify the file with the following command. This command will display the content of the file in the HDFS cluster.

```
# hdfs dfs -cat /opt/spark/README.md
```

Open a command console to execute the Spark jobs to submit on YARN.

Ensure that you install python on Hadoop node:

```
# yum install -y python3
```

On Spark Node
Export Hadoop and Yarn environment variable pointing to the Hadoop Node – Hadoop conf file.

```
export JAVA_HOME=/opt/jdk
export HADOOP_HOME=/opt/hadoop
export PATH=$PATH:$JAVA_HOME/bin:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

Create a python app, **SimpleApp.py** with the following code:

```
# vi /opt/SimpleApp.py
// ---------------------------------- Code Begin ----------------------------------------//

"""SimpleApp.py"""
from pyspark.sql import SparkSession

logFile = "/opt/spark/README.md"  # Should be some file on your system

spark   = SparkSession.builder.appName("Python App").getOrCreate()
logData = spark.read.text(logFile).cache()

numAs = logData.filter(logData.value.contains('a')).count()
numBs = logData.filter(logData.value.contains('b')).count()

print("Lines with a: %i, lines with b: %i" % (numAs, numBs))

spark.stop()

// ---------------------------------- Code Ends ----------------------------------------//
```

The above code reads the README.md file from HDFS cluster and counts the line containing ‘a’ and ‘b’ respectively.

On **sparko** terminal

```
#export HADOOP_CONF_DIR=/opt/hadoop/etc/hadoop
#export YARN_CONF_DIR=/opt/hadoop/etc/hadoop
#cd /opt/spark/bin
#./spark-submit  --master yarn --deploy-mode cluster --executor-memory 512m --num-executors 2
/opt/SimpleApp.py
```

or client side - Optional.

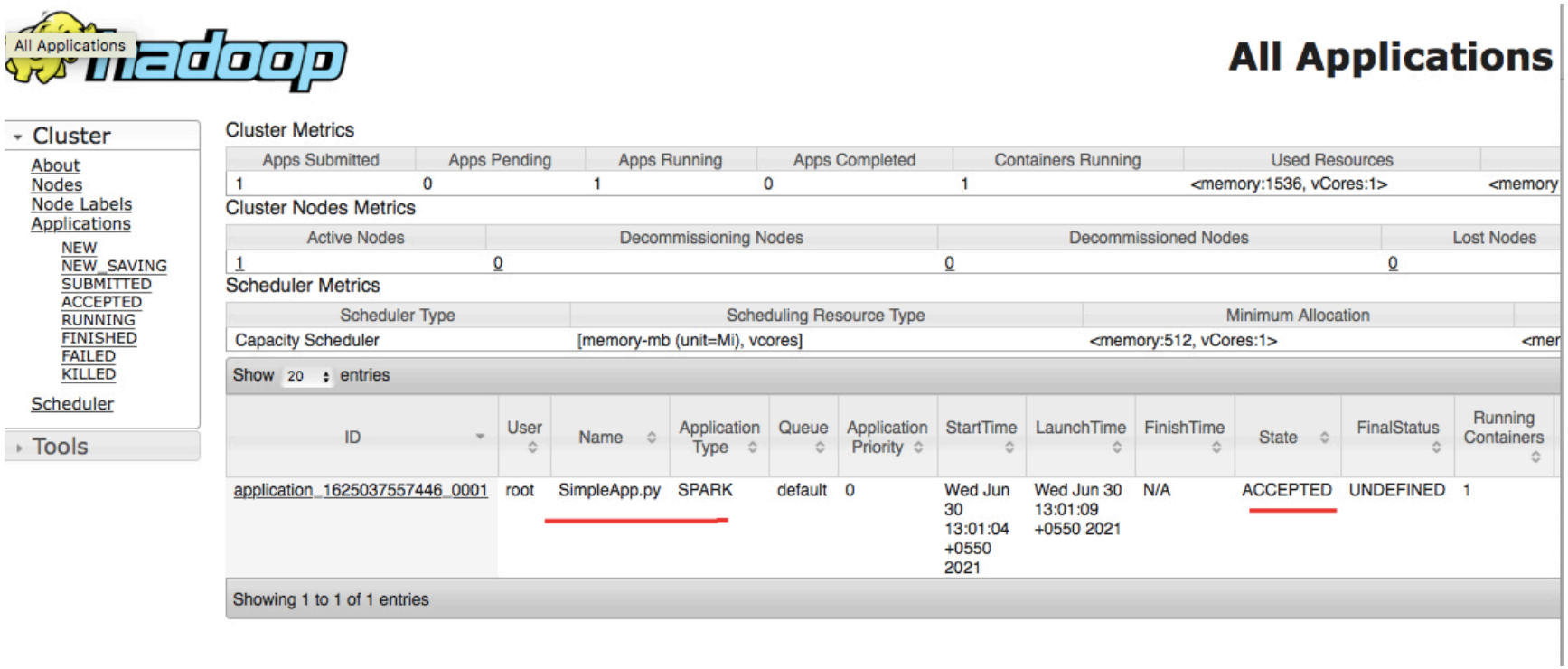
```
#./spark-submit  --master yarn --deploy-mode client /opt/SimpleAppP.py
```

Ensure that `HADOOP_CONF_DIR` or `YARN_CONF_DIR` points to the directory which contains the (client side) configuration files for the Hadoop cluster.

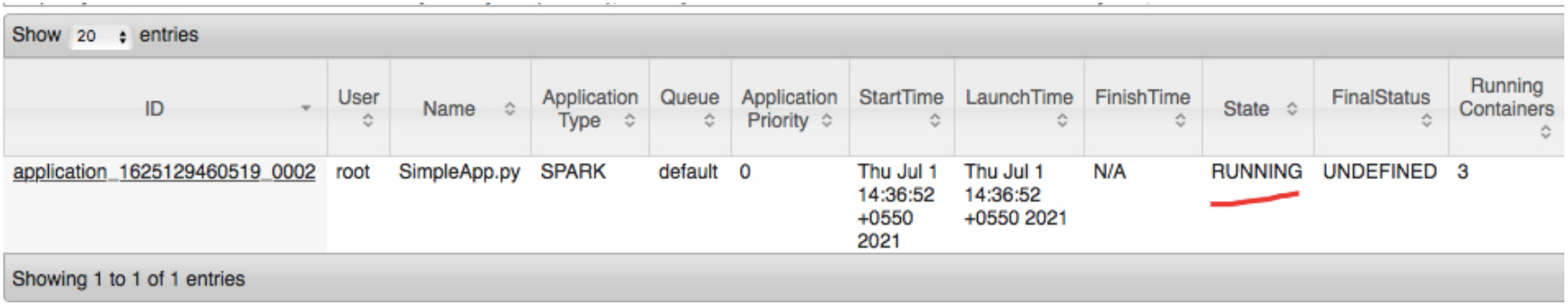
Note --> Application py/Jar should be in the local file system, In folder is in HDFS and Out should not be present in the HDFS,it will be automatically created.

You can verify the progress of the application using http://lcoalhost:8088/cluster

Click on Cluster-- >Application --> accepted



Wait for few moment and verify on Running tab.

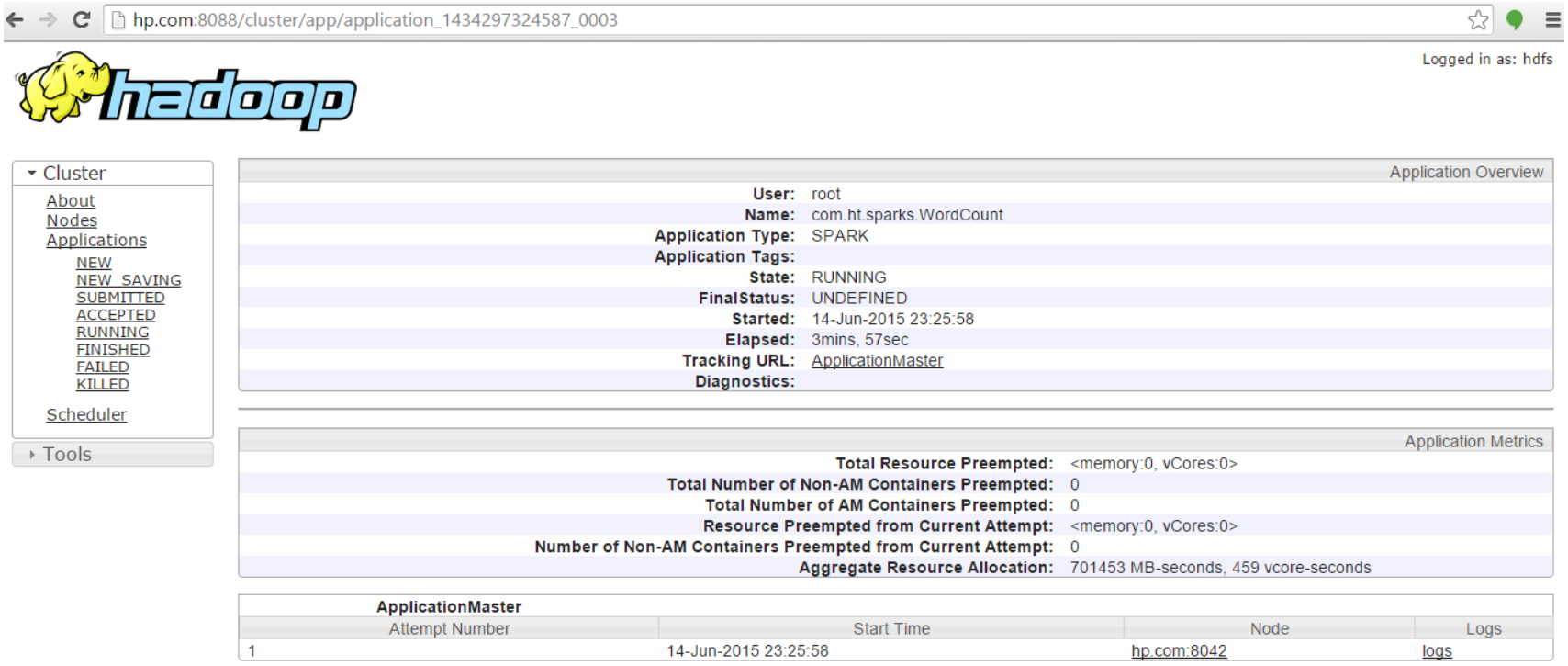


Show 20 entries

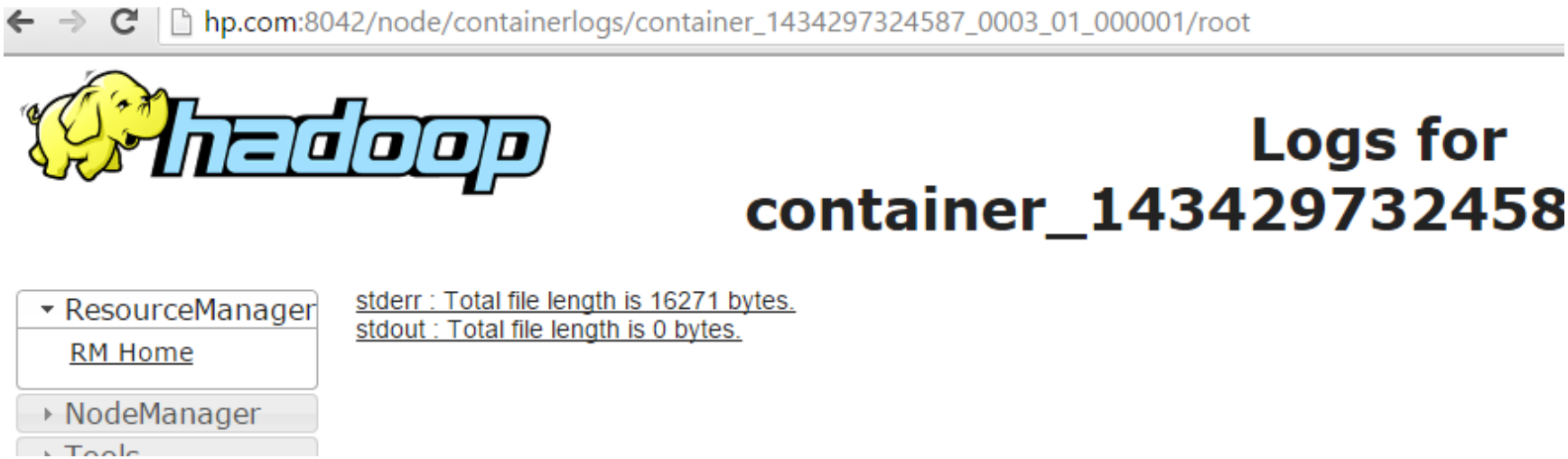
| ID | User | Name | Application Type | Queue | Application Priority | StartTime | LaunchTime | FinishTime | State | FinalStatus | Running Containers |
|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1625129460519_0002 | root | SimpleApp.py | SPARK | default | 0 | Thu Jul 1 14:36:52 +0550 2021 | Thu Jul 1 14:36:52 +0550 2021 | N/A | RUNNING | UNDEFINED | 3 |

Showing 1 to 1 of 1 entries

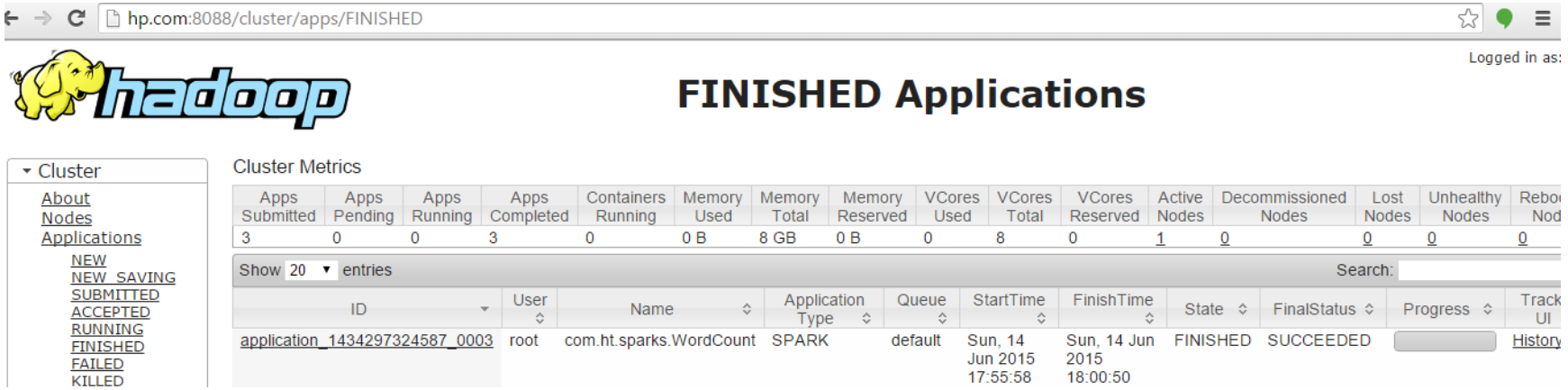
Click on the Application ID --> Logs



Logs --> Stderr to get details as follows:




**After sometimes click on Finished, after the console exit the program.**



On the job submit console.

```
2021-01-22 16:36:11,589 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:12,662 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:13,765 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:14,896 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:16,011 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:17,296 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:18,725 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:19,914 INFO yarn.Client: Application report for application_1611331123305_0002 (state: RUNNING)
2021-01-22 16:36:21,041 INFO yarn.Client: Application report for application_1611331123305_0002 (state: FINISHED)
2021-01-22 16:36:21,401 INFO yarn.Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: hadoop0
         ApplicationMaster RPC port: 38603
         queue: default
         start time: 1611333080086
         final status: SUCCEEDED
         tracking URL: http://hadoop0:8088/proxy/application_1611331123305_0002/
         user: root
2021-01-22 16:36:23,265 INFO util.ShutdownHookManager: Shutdown hook called
2021-01-22 16:36:23,442 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-7b20b446-2b75-4550-9e2d-c84ad6
7c9b3b
2021-01-22 16:36:23,532 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-89770943-febc-4bdb-bf6e-404103
88026d
[root@303b68143644 bin]#
```

Since we have printed the output. You can verify from the log fie only.

Go to the User Log folders of the hadoop installation. (Job ID and the container ID will be different in your machine, update accordingly)

#cd /opt/hadoop/logs/userlogs
#more application_1625129460519_0004/container_1625129460519_0004_01_000001/stdout

```
[root@hadoop0 userlogs]# pwd
/opt/hadoop/logs/userlogs
[root@hadoop0 userlogs]# more application_1625129460519_0004/container_1625129460519_0004_01_000001/stdout
Lines with a: 64, lines with b: 32
[root@hadoop0 userlogs]#
```

You can also view the log from Web UI

Directory →Utilities -> Logs → userlogs -> application id (Your application id) -> Container Id(browse subsequently all the logs) --> stdout.

```
23/07/07 10:16:49 INFO YarnClusterScheduler: Killing all running tasks in stage 5: Stage finished
23/07/07 10:16:49 INFO DAGScheduler: Job 3 finished: count at NativeMethodAccessorImpl.java:0, took 7.896622 s
Lines with a: 71, lines with b: 38
23/07/07 10:16:49 WARN AbstractConnector:
```

Task -> Let us write the output into file instead of writing in console.

Create a file SimpleAppP.py and update with the following code. It only writes the dataframe which has the line having ‘a’ in it.

//------------------------ Code Begin --------------------

```
"""SimpleApp.py"""
from pyspark.sql import SparkSession

logFile = "/opt/spark/README.md"  # Should be some file on your system

spark   = SparkSession.builder.appName("Python App").getOrCreate()
logData = spark.read.text(logFile).cache()

numAs = logData.filter(logData.value.contains('a')).count()
numBs = logData.filter(logData.value.contains('b')).count()

#print("Lines with a: %i, lines with b: %i" % (numAs, numBs))

myDF = logData.filter(logData.value.contains('a'))
myDF.write.save("mydatap")
spark.stop()
```

//------------------------ Code Ends --------------------

Execute the program again.

```
#spark-submit  --master yarn --deploy-mode cluster --executor-memory 512m --num-executors 2
SimpleAppP.py
```

At the end of the execution, you should have the following output in the console.

```
2021-07-01 10:10:44,466 INFO yarn.Client: Application report for application_1625129460519_0005 (state: FINISHED)
2021-07-01 10:10:44,467 INFO yarn.Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: hadoop0
         ApplicationMaster RPC port: 41043
         queue: default
         start time: 1625133769650
         final status: SUCCEEDED
         tracking URL: http://hadoop0:8088/proxy/application_1625129460519_0005/
         user: root
2021-07-01 10:10:44,535 INFO util.ShutdownHookManager: Shutdown hook called
2021-07-01 10:10:44,550 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-85f5b09e-f56b-49a9-9ca7-fad964848b7f
2021-07-01 10:10:44,835 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-009ed528-fe7b-4f7e-acda-14c34a265a3a
[root@spark0 data]#
```

You can verify the output file in the hdfs as shown below:

```
#hdfs dfs -ls -R /user/root/mydatap
```

```
23/07/07 12:26:35 INFO ShutdownHookManager: Deleting directory /tmp/spark-fa9f16f5-b4a8-405d-a39f-1079a24a109e
23/07/07 12:26:35 INFO ShutdownHookManager: Deleting directory /tmp/spark-4183b6bb-c10a-4bc7-9553-cfd6d87ac93d
[root@hadoop1 opt]# hdfs dfs -ls -R /user/root/mydatap
-rw-r--r--   2 root supergroup          0 2023-07-07 12:25 /user/root/mydatap/_SUCCESS
-rw-r--r--   2 root supergroup       3464 2023-07-07 12:25 /user/root/mydatap/part-00000-713d9b32-24cb-47ed-b6e1-7044ca7ccaf9-c000.snappy.parquet
[root@hadoop1 opt]#
```

Substitute the path of the file listed by above command.

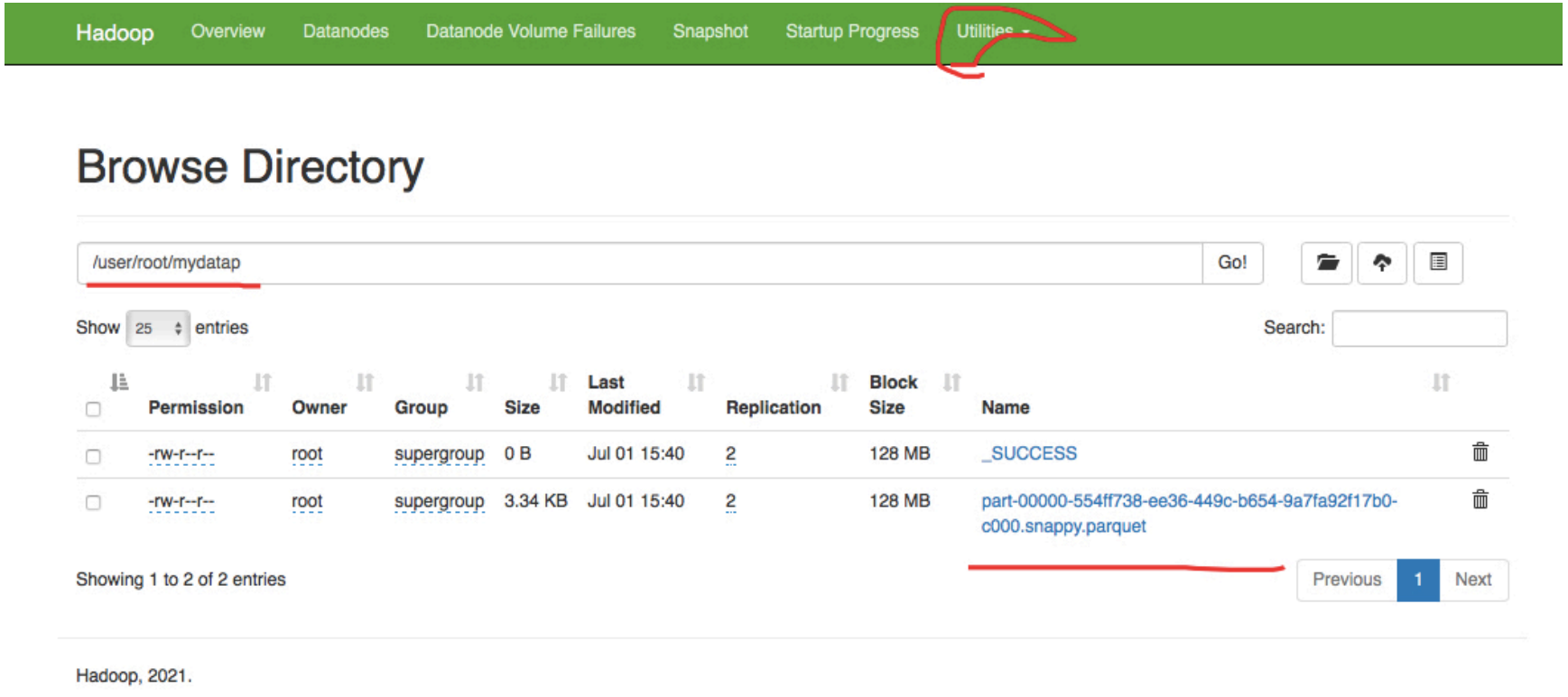
```
# hdfs dfs -cat /user/root/mydatap/part-00000-713d9b32-24cb-47ed-b6e1-7044ca7ccaf9-c000.snappy.parquet
```


It's a binary parquet file.

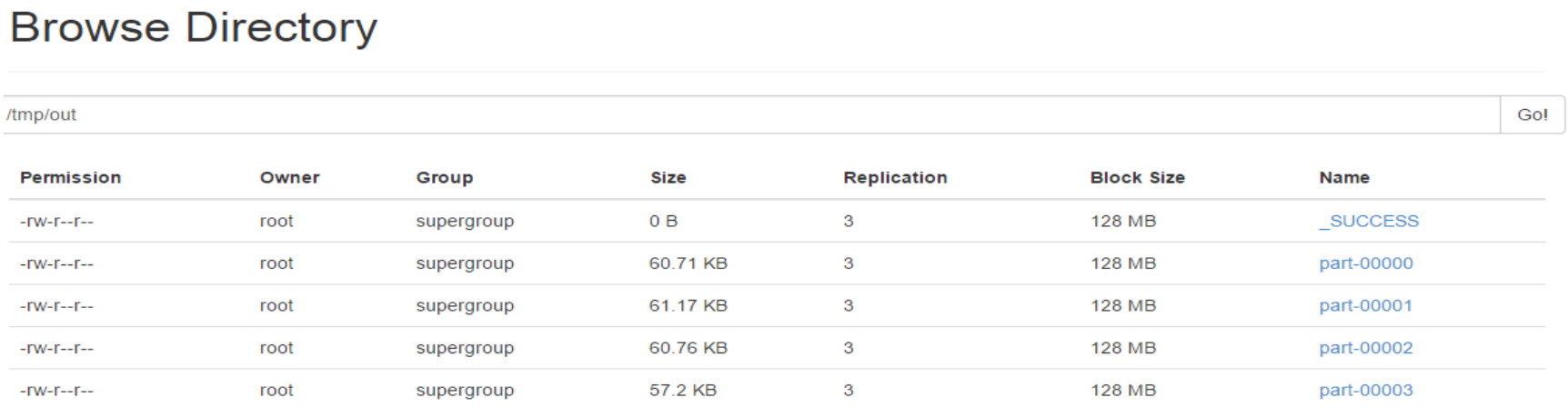
Great! You have successfully executed spark job in YARN cluster.
Let us verify the output now using HDFS browser.

http://localhost:9870/explorer.html#/

Click Utilities --> Browse the file system → /user/root/output (Enter in the Directory ) - Go




Click on the above mark file.




Let us view one of the output. click on part-0001 --> Download.

When you're done, you can stop the daemons with: Optional.
On the hadoop0 terminal

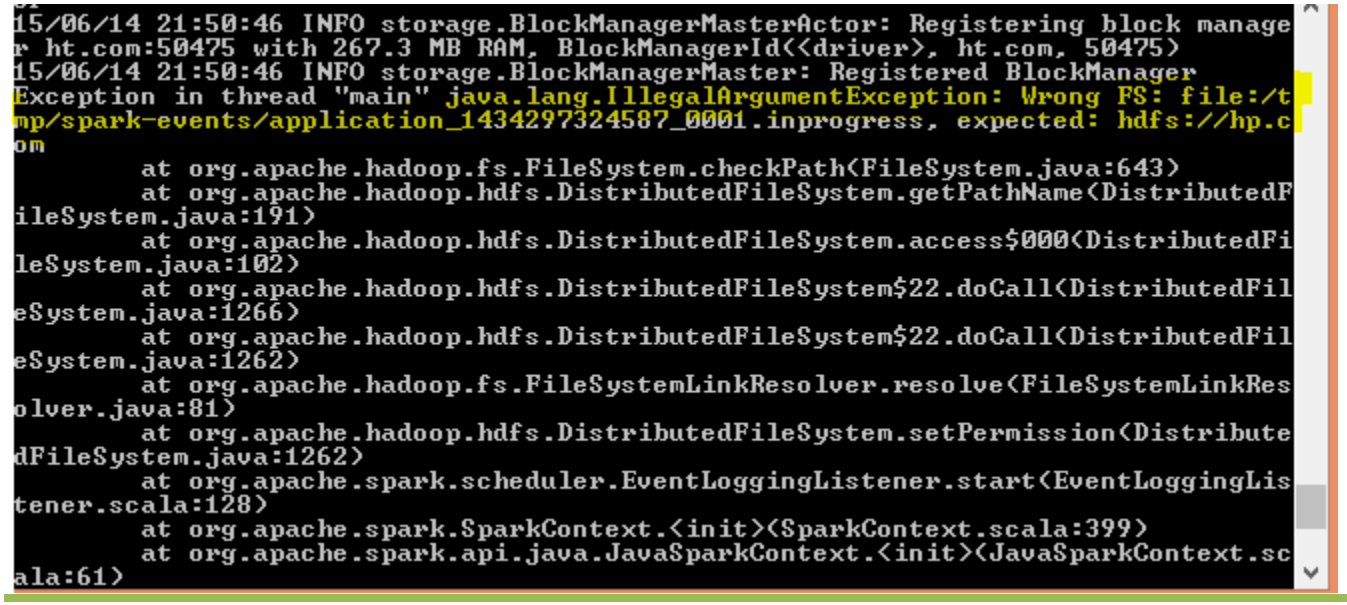
```
 # stop-yarn.sh
#stop-hdfs.sh
```

Congrats!.

**---------- End of Lab----------**

**Errata:**

1) 15/06/14 21:50:46 INFO storage.BlockManagerMasterActor: Registering block manager ht.com:50475 with 267.3 MB RAM, BlockManagerId(<driver>, ht.com, 50475)
15/06/14 21:50:46 INFO storage.BlockManagerMaster: Registered BlockManager
Exception in thread "main" java.lang.IllegalArgumentException: Wrong FS: file:/tmp/spark-events/application_1434297324587_0001.inprogress, expected: hdfs://hp.com at org.apache.hadoop.fs.FileSystem.checkPath(FileSystem.java:643)
at org.apache.hadoop.hdfs.DistributedFileSystem.getPathName(DistributedFileSystem.java:191)
at org.apache.hadoop.hdfs.DistributedFileSystem.access$000(DistributedFileSystem.java:102)
at org.apache.hadoop.hdfs.DistributedFileSystem$22.doCall(DistributedFileSystem.java:1266)
at org.apache.hadoop.hdfs.DistributedFileSystem$22.doCall(DistributedFileSystem.java:1262)
at org.apache.hadoop.fs.FileSystemLinkResolver.resolve(FileSystemLinkResolver.java:81)
at org.apache.hadoop.hdfs.DistributedFileSystem.setPermission(DistributedFileSystem.java:1262)
at org.apache.spark.scheduler.EventLoggingListener.start(EventLoggingListener.scala:128)



Solution: Verify default configuration of the spark installation. /spark/spark-1.3.0-bin-hadoop2.4/conf/spark-defaults.conf


Check that spark.eventLog.dir is begin with hdfs:// and the /tmp/spark-events is already created in HDFS system
You can verify on the YARN cluster only.

Verify using : hadoop fs -ls -R /tmp/

```
[root@hp ~]# hadoop fs -ls -R /tmp/
Java HotSpot(TM) Client VM warning: You have loaded library /YARN/hadoop-2.6.0/l
ib/native/libhadoop.so.1.0.0 which might have disabled stack guard. The VM will
try to fix the stack guard now.
It's highly recommended that you fix the library with 'execstack -c <libfile>',
or link it with '-z noexecstack'.
15/06/14 22:04:11 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
drwxr-xr-x   - root supergroup          0 2015-06-14 16:25 /tmp/in
-rw-r--r--   1 root supergroup       3629 2015-06-14 16:25 /tmp/in/README.md
drwxr-xr-x   - root supergroup          0 2015-06-14 17:47 /tmp/out
-rw-r--r--   3 root supergroup          0 2015-06-14 17:47 /tmp/out/_SUCCESS
-rw-r--r--   3 root supergroup       1956 2015-06-14 17:47 /tmp/out/part-00000
-rw-r--r--   3 root supergroup       1717 2015-06-14 17:47 /tmp/out/part-00001
drwxr-xr-x   - root supergroup          0 2015-06-14 17:50 /tmp/out1
-rw-r--r--   3 root supergroup          0 2015-06-14 17:50 /tmp/out1/_SUCCESS
-rw-r--r--   3 root supergroup       1956 2015-06-14 17:50 /tmp/out1/part-00000
-rw-r--r--   3 root supergroup       1717 2015-06-14 17:50 /tmp/out1/part-00001
drwxr-xr-x   - root supergroup          0 2015-06-14 17:52 /tmp/out3
-rw-r--r--   3 root supergroup          0 2015-06-14 17:52 /tmp/out3/_SUCCESS
-rw-r--r--   3 root supergroup       1956 2015-06-14 17:52 /tmp/out3/part-00000
-rw-r--r--   3 root supergroup       1717 2015-06-14 17:52 /tmp/out3/part-00001
drwxr-xr-x   - root supergroup          0 2015-06-14 21:59 /tmp/out5
-rw-r--r--   3 root supergroup          0 2015-06-14 21:59 /tmp/out5/_SUCCESS
-rw-r--r--   3 root supergroup       1956 2015-06-14 21:59 /tmp/out5/part-00000
-rw-r--r--   3 root supergroup       1717 2015-06-14 21:59 /tmp/out5/part-00001
drwxr-xr-x   - root supergroup          0 2015-06-14 21:58 /tmp/spark-events
-rwxrwx---   3 root supergroup      20287 2015-06-14 21:59 /tmp/spark-events/app
lication_1434297324587_0002.inprogress
```

2) 15/06/14 00:41:20 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
15/06/14 21:58:27 INFO yarn.Client: Application report for application_1434297324587_0002 (state: ACCEPTED)
Indefinite loop of above
**Solution**: Ensure that all configuration file are map to appropriate hostname and hostname to ip details are modified in the hosts file , Increase the resources and wait for few minutes to convert the status to running.

```
Yarn-site.xml
yarn.scheduler.minimum-allocation-mb: 256m
yarn.scheduler.increment-allocation-mb: 256m
```

```
<property>
<name>yarn.nodemanager.resource.memory-mb</name>
<value>4096</value> <!-- 4 GB -->
</property>
<property>
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>512</value> <!-- 1 GB -->
</property>
<property>
<name>yarn.scheduler.increment-allocation-mb</name>
<value>256</value> <!-- 1 GB -->
</property>
```

**yarn-site.xml** – Spark Node. (Specify the hostname of the Hadoop RM Node as shown below)

```
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>hadoop0</value>
      <description>The hostname of the RM.</description>
   </property>
<property>
      <name>yarn.resourcemanager.address</name>
      <value>hadoop0:8032</value>
      <description>The hostname of the RM.</description>
   </property>
```

3) If unhealthy nodes are being displayed in the cluster URL with 1/1 local-dirs are bad: /tmp/hadoop-yarn/nm-local-dir;

**Solution**: delete the folder using the following command and ensure that nodes are healthy before proceeding forward, you can restart if require

hadoop fs -rm -fr /tmp/hadoop-yarn/*

Try the following command:
yarn application -list
yarn application -kill [application name]

You can view jps in YARN cluster when spark job is executing. It will start Coarse* processes.


Even, cluster mode works

```
15/06/14 23:43:46 INFO yarn.Client: Application report for application_143429732
4587_0004 (state: FINISHED)
15/06/14 23:43:46 INFO yarn.Client:
         client token: N/A
         diagnostics: N/A
         ApplicationMaster host: hp.com
         ApplicationMaster RPC port: 0
         queue: default
         start time: 1434305512588
         final status: SUCCEEDED
         tracking URL: http://hp.com:8088/proxy/application_1434297324587_0004/A
         user: root
[root@ht spark-1.3.0-bin-hadoop2.4]# ./bin/spark-submit --master yarn-cluster -
-class com.ht.sparks.WordCount  --num-executors 3 --executor-cores 1  /spark/Sp
arkWC-0.0.1-SNAPSHOT.jar /tmp/in /tmp/out7
```

Issue:

```
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:08 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 2 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:09 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 3 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:10 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 4 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:11 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 5 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:12 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 6 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:13 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 7 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:14 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 8 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:15 INFO Client: Retrying connect to server: 0.0.0.0/0.0.0.0:8032. Already tried 9 time(s); retry policy is Retry
UpToMaximumCountWithFixedSleep(maxRetries=10, sleepTime=1000 MILLISECONDS)
21/01/23 05:38:15 INFO RetryInvocationHandler: java.net.ConnectException: Your endpoint configuration is wrong; For more detail
s see:  http://wiki.apache.org/hadoop/UnsetHostnameOrPort, while invoking ApplicationClientProtocolPBClientImpl.getClusterMetri
cs over null after 1 failover attempts. Trying to failover after sleeping for 42760ms.
```

Verify the ip of the resource manager and set the home directory appropriately.
Try updating the following setting in yarn-site.xml of Hadoop and spark node too..

```
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>hadoop0</value>
      <description>The hostname of the RM.</description>
   </property>
<property>
      <name>yarn.resourcemanager.address</name>
      <value>hadoop0:8032</value>
      <description>The hostname of the RM.</description>
   </property>
```

---

## 17. Jobs Monitoring : Using Web UI – 45 Minutes

Execute the Job as specified in the **Spark Two Nodes** cluster before proceeding ahead.

- Start the Two Nodes Spark Cluster
- Copy the input file in both the nodes, if you are executing on the Standalone cluster.

Execute the following job:

Update File, /Software/data/names.json with the following content:

```
{"firstName":"Grace","lastName":"Hopper"}
{"firstName":"Alan","lastName":"Turing"}
{"firstName":"Ada","lastName":"Lovelace"}
{"firstName":"Charles","lastName":"Babbage"}
```

Enter the following commands one at a time.

```
#pyspark --master spark://spark0:8090
```

```
[root@e5217f917ab5 /]# pyspark --master spark://spark0:8090
Python 3.8.8 (default, Aug 25 2021, 16:13:02)
[GCC 8.5.0 20210514 (Red Hat 8.5.0-3)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/07/10 09:00:31 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java
classes where applicable
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.0
      /_/

Using Python version 3.8.8 (default, Aug 25 2021 16:13:02)
Spark context Web UI available at http://e5217f917ab5:4040
Spark context available as 'sc' (master = spark://spark0:8090, app id = app-20230710143039-0000).
SparkSession available as 'spark'.
>>>
```

```
ip="/Software/data/names.json"
op="/Software/data/nameop"
spark.sparkContext.setLogLevel("WARN")
peopleDF = spark.read.json(ip)      # One Job
namesDF = peopleDF.select("firstName","lastName")
namesDF.write.option("header","true").csv(op)     # Second Job
```

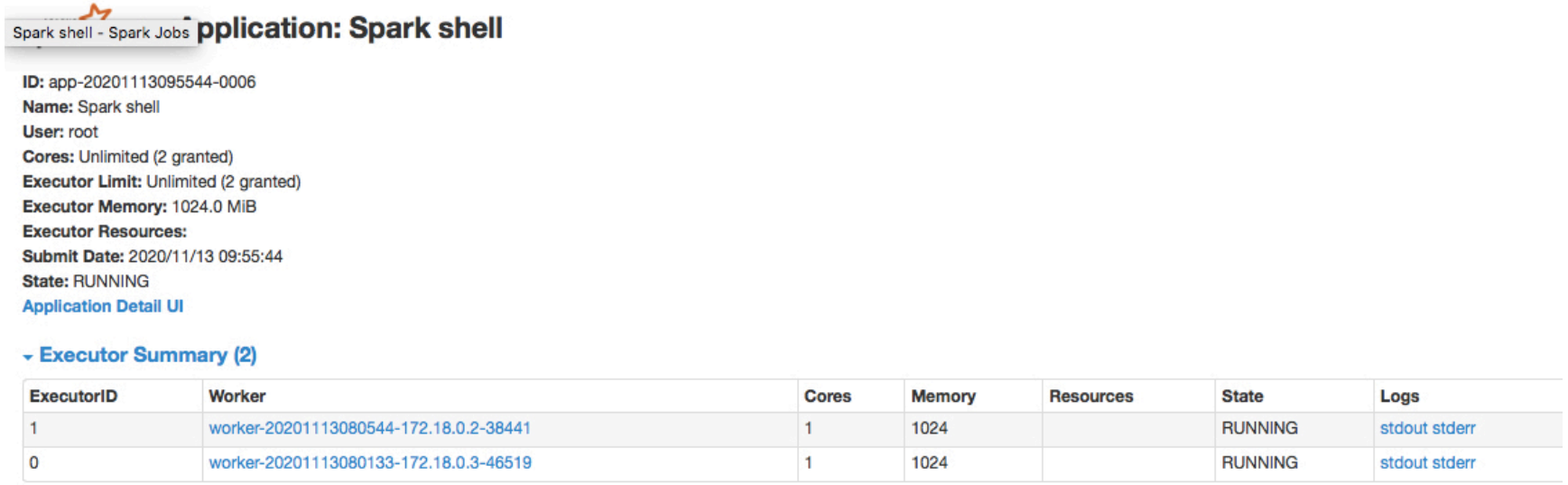
```
Spark context available as 'sc' (master = spark://spark0:8090, app id = app-20230710143039-0000).
SparkSession available as 'spark'.
>>> ip="/Software/data/names.json"
>>> op="/Software/data/nameop"
>>> spark.sparkContext.setLogLevel("WARN")
>>> peopleDF = spark.read.json(ip)     # One Job
>>> namesDF = peopleDF.select("firstName","lastName")
>>> namesDF.write.option("header","true").csv(op)    # Second Job
>>>
```

Access the Spark Cluster web console.

http://127.0.0.1:8080

You should  have two worker nodes as shown below.

Click on Application ID → Application Detail UI

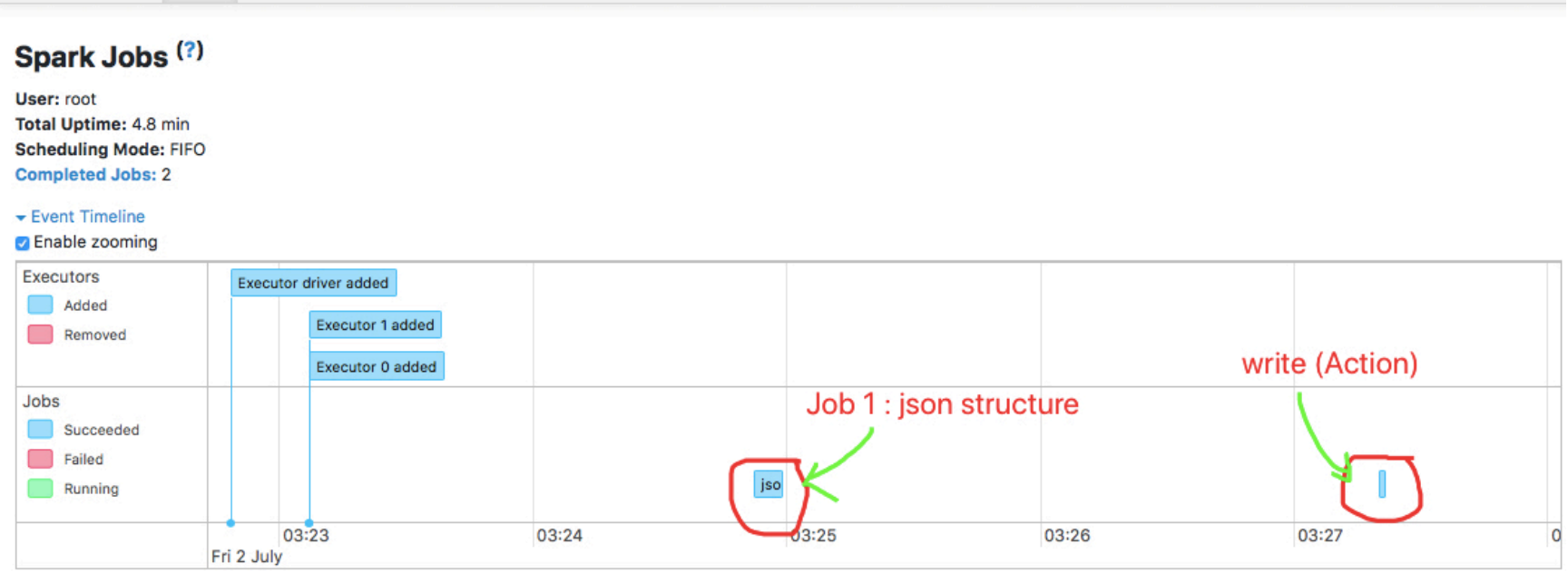

Spark shell - Spark Jobs
pplication: Spark shell
ID: app-20201113095544-0006
Name: Spark shell
User: root
Cores: Unlimited (2 granted)
Executor Limit: Unlimited (2 granted)
Executor Memory: 1024.0 MiB
Executor Resources:
Submit Date: 2020/11/13 09:55:44
State: RUNNING
Application Detail UI
Executor Summary (2)
ExecutorID Worker Cores Memory Resources State Logs
1 worker-20201113080544-172.18.0.2-38441 1 1024 RUNNING stdout stderr
0 worker-20201113080133-172.18.0.3-46519 1 1024 RUNNING stdout stderr

You can verify the Jobs details Using Spark UI: http://localhost:4040/jobs/

Web UI comes with the following tabs (which may not all be visible at once as they are lazily created on demand, e.g. Streaming tab):

- Jobs
- Stages
- Storage with RDD size and memory use
- Environment
- Executors
- SQL/Dataframe

The **Jobs Tab** shows status of all Spark jobs in a Spark application. Expand Eventtime line

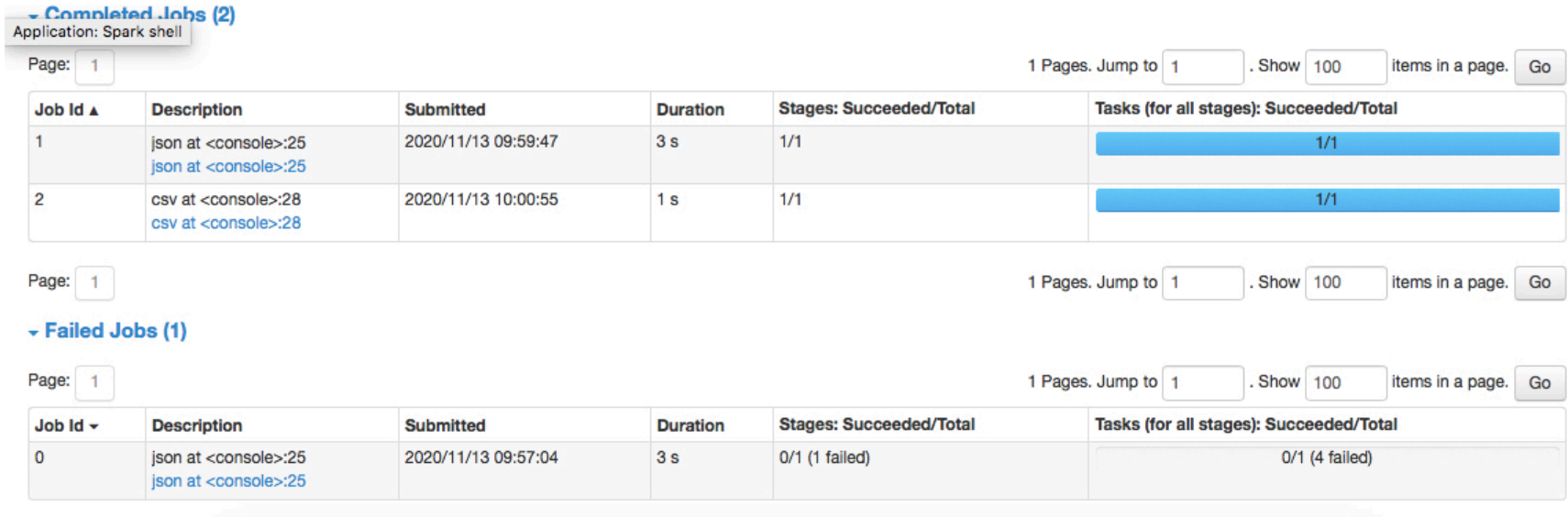


Display the timeline of Executor being added in the Job. When the Job Get completed etc.
In the above, you can verify that 2 executors have been added.

1 Job Succeded i.e (Reading Json file)
2 Job Succeeded i.e write action.

When you hover over a job in Event Timeline not only you see the job legend but also the job is highlighted in the Summary section.

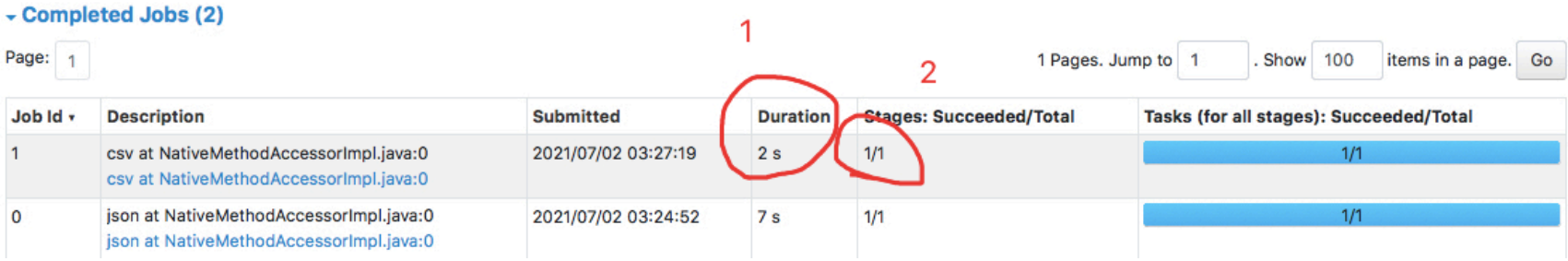
The Event Timeline section shows not only jobs but also executors.
You can verify the completed and failed jobs too from this console:



You can verify some of the following Queries from this console?

How long the Job takes to execute? – (Job ID -> 1: Here Duration is 2 s.)

How Many States and Tasks created per Job? ( 2: 1 stages and only one task)



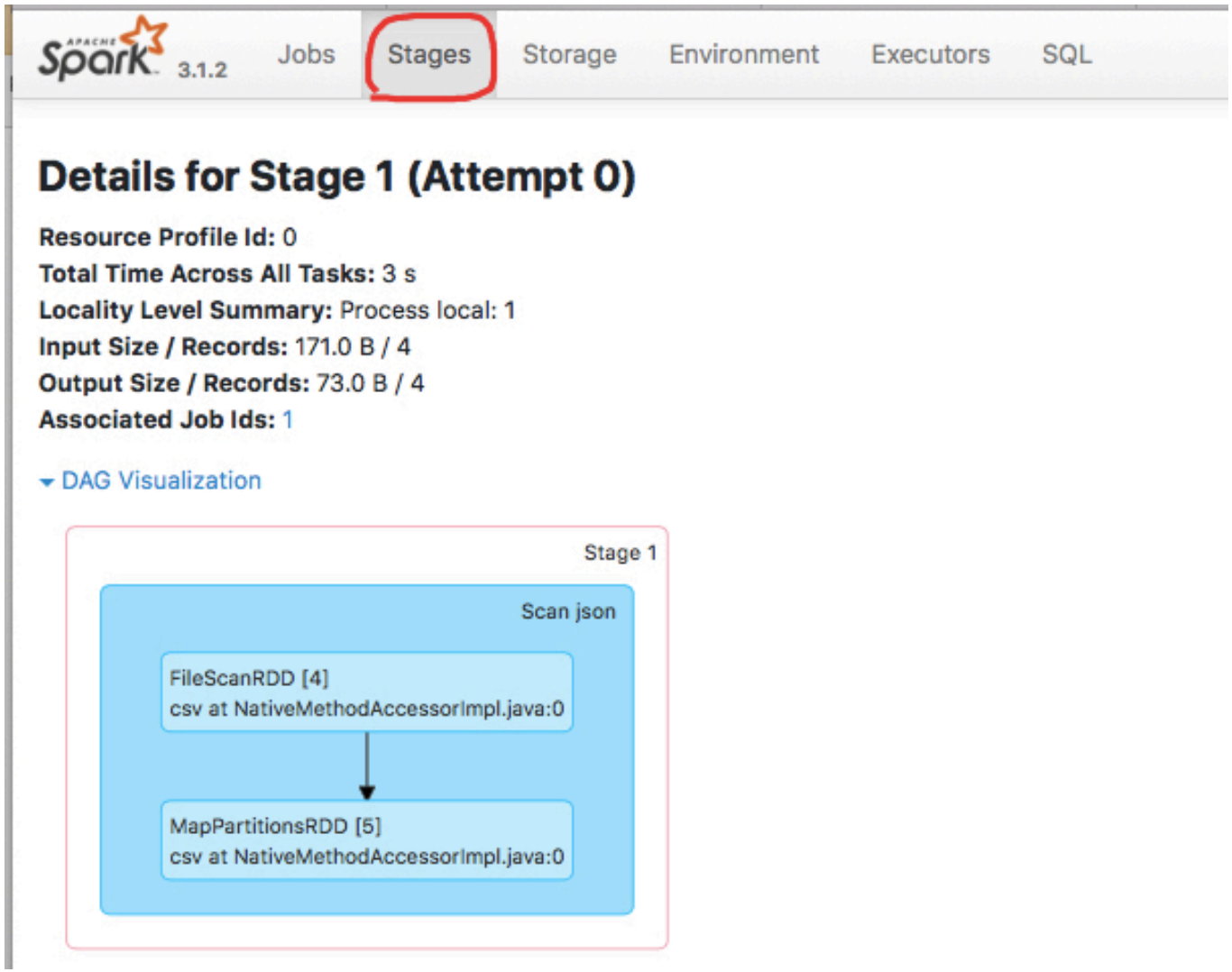
Completed Jobs (2)

Page: 1

1 Pages. Jump to 1 . Show 100 items in a page. Go

| Job Id | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 1 | csv at NativeMethodAccessorImpl.java:0<br>csv at NativeMethodAccessorImpl.java:0 | 2021/07/02 03:27:19 | 2 s | 1/1 | 1/1 |
| 0 | json at NativeMethodAccessorImpl.java:0<br>json at NativeMethodAccessorImpl.java:0 | 2021/07/02 03:24:52 | 7 s | 1/1 | 1/1 |

Click on job (1 – write action or job ) -> for details on Description → stages to view DAG - When you click a job in All Jobs Page page, you see the **Details for Job** page.



As seen above, there is 1 stage in the Job. It scans the file and create a map partition RDD.

It shows the DAG graph and the stage details.

Scroll down to view the Event timeline.


**Summary Metrics for 1 Completed Tasks**

| Metric | Min | 25th percentile | Median | 75th percentile | Max |
|---|---|---|---|---|---|
| Duration | 2 s | 2 s | 2 s | 2 s | 2 s |
| GC Time | 68.0 ms | 68.0 ms | 68.0 ms | 68.0 ms | 68.0 ms |
| Input Size / Records | 171 B / 5 | 171 B / 5 | 171 B / 5 | 171 B / 5 | 171 B / 5 |

Showing 1 to 3 of 3 entries

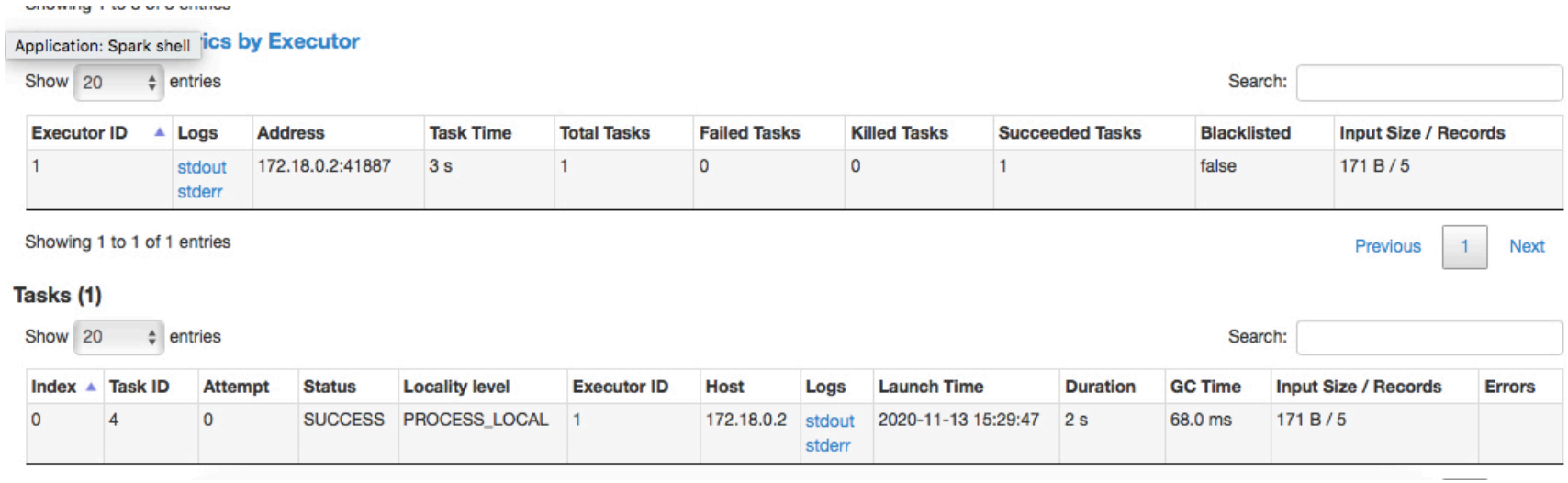
Questions:

How long does the scheduler take to schedule the task? (Check the Light Blue Block)
How long does the task deserialization take? (Check the Orange Block)
What is the actual computation time? (Verify the Green Color block)

In ideal scenario, Green should take most of the computing resources.

You can also verify how much each task took to execute. It will provide some information about the health of the computing task.

Scroll down to view the task metrics: which task take majority of the time etc. How much byte consume or output by each task and its statistics.




You can verify from the above, Locality_Level column – the data is process locally. i.e data locality on the same process.

How much the input size for the task above? Hints -> Look for Input Size/ Records column.

## Executors Tab

Stats of each executor, how much time it spent on GC etc. How much data get shuffle?
Executors tab in web UI shows

**Summary**

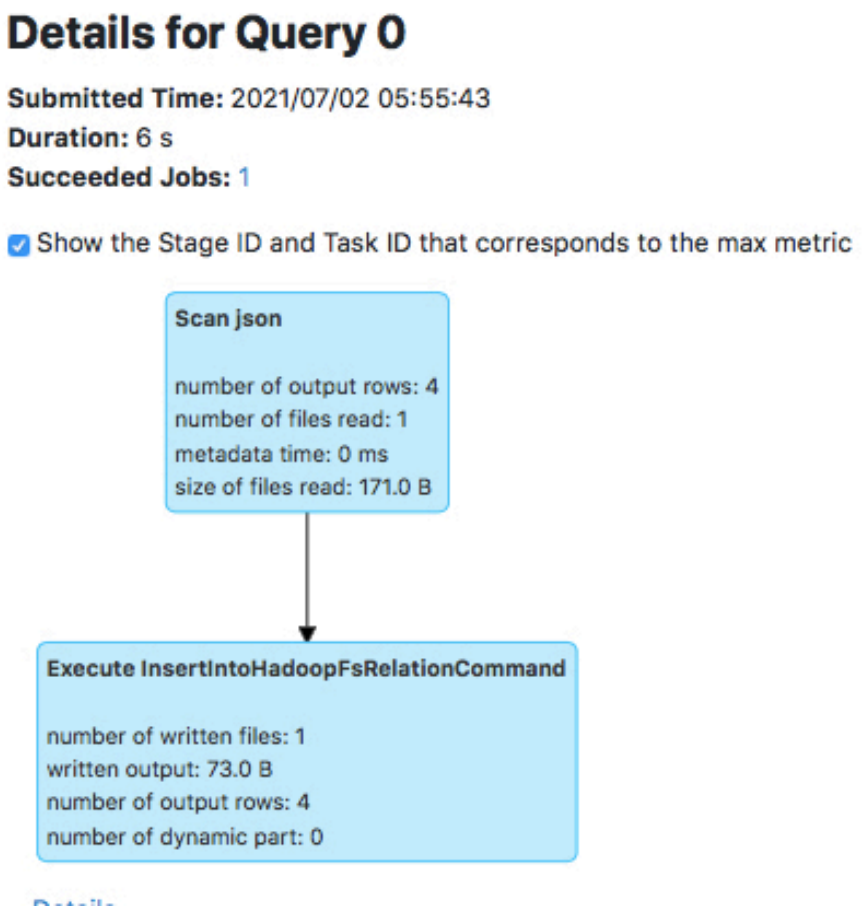
| | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Active(3)** | 1 | 23.4 KB / 1.3 GB | 0.0 B | 2 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B |
| **Dead(4)** | 2 | 46.8 KB / 1.7 GB | 0.0 B | 4 | 0 | 0 | 7 | 7 | 1.0 min (6 s) | 21 KB | 0.0 B | 0.0 B |
| **Total(7)** | 3 | 70.2 KB / 3 GB | 0.0 B | 6 | 0 | 0 | 7 | 7 | 1.0 min (6 s) | 21 KB | 0.0 B | 0.0 B |

**Executors**

Show 20 entries Search:

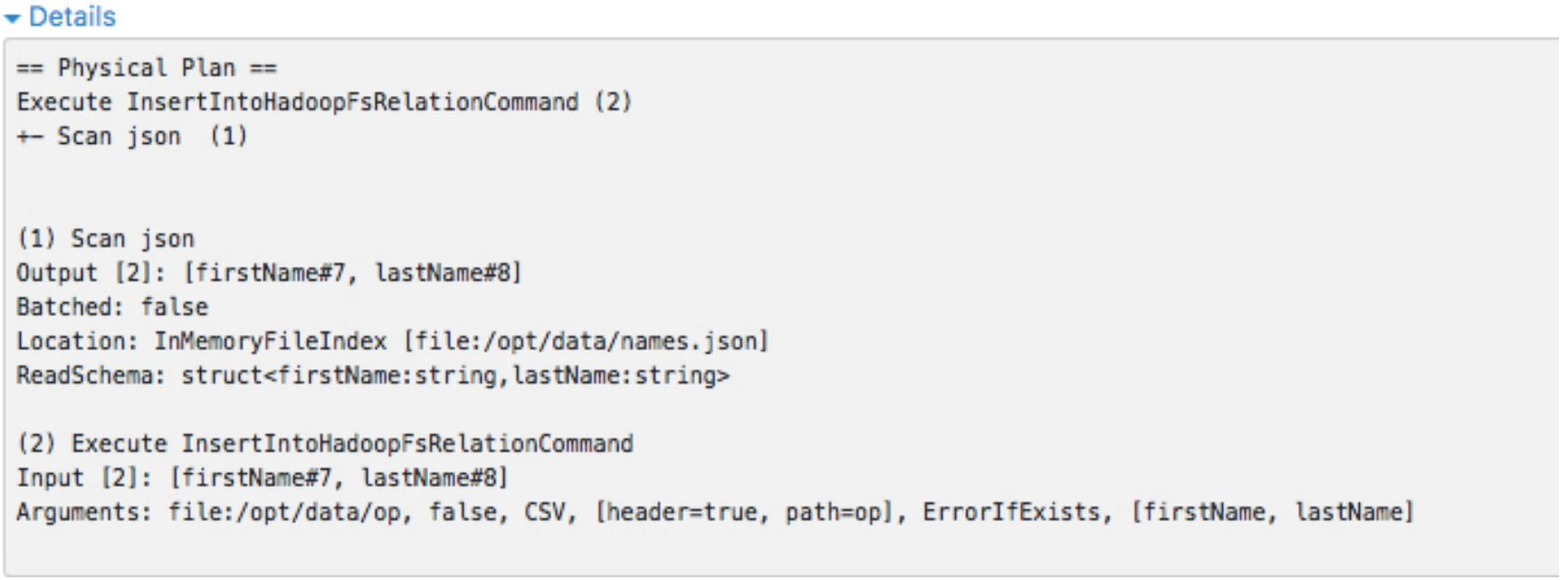
| Executor ID | Address | Status | RDD Blocks | Storage Memory | Disk Used | Cores | Active Tasks | Failed Tasks | Complete Tasks | Total Tasks | Task Time (GC Time) | Input | Shuffle Read | Shuffle Write | Logs | Thread Dump |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| driver | 192.168.188.173:54112 | Active | 1 | 23.4 KB / 434 MB | 0.0 B | 0 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B | | Thread Dump |
| 0 | 192.168.188.173:54931 | Dead | 1 | 23.4 KB / 434 MB | 0.0 B | 1 | 0 | 0 | 4 | 4 | 37 s (6 s) | 9.5 KB | 0.0 B | 0.0 B | stdout stderr | Thread Dump |
| 1 | 192.168.188.174:34320 | Dead | 1 | 23.4 KB / 434 MB | 0.0 B | 1 | 0 | 0 | 3 | 3 | 25 s (0.3 s) | 11.5 KB | 0.0 B | 0.0 B | stdout stderr | Thread Dump |
| 2 | 192.168.188.174:46874 | Dead | 0 | 0.0 B / 434 MB | 0.0 B | 1 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B | stdout stderr | Thread Dump |
| 3 | 192.168.188.173:34382 | Dead | 0 | 0.0 B / 434 MB | 0.0 B | 1 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B | stdout stderr | Thread Dump |
| 4 | 192.168.188.174:44548 | Active | 0 | 0.0 B / 434 MB | 0.0 B | 1 | 0 | 0 | 0 | 0 | 0 ms (0 ms) | 0.0 B | 0.0 B | 0.0 B | stdout stderr | Thread Dump |

Click on SQl tab




You can view the statistics of the SQL – How much Rows read and how much bytes etc.

Verify the plan (Scroll down the SQL tab.)




---------- Lab Ends Here ----------

## 18. Persisting Data – 40 Minutes

In this lab, you will learn how to cache RDD and uncache it, when it's done.

Data being used in this lab:

Create two files in /Software/data folder with the following content.

people.csv

```
pcode,lastName,firstName,age
02134,Hopper,Grace,52
94020,Turing,Alan,32
94020,Lovelace,Ada,28
87501,Babbage,Charles,49
02134,Wirth,Niklaus,48
```

pcodes.csv

```
pcode,city,state
02134,Boston,MA
94020,Palo Alto,CA
87501,Santa Fe,NM
60645,Chicago,IL
```

Load the data using dataframe.

Open a terminal and start the spark-shell using pyspark. Specify the data path of the file appropriately.

```
over20DF = spark.read.option("header","true").csv("people.csv").where("age > 20")
pcodesDF = spark.read.option("header","true").csv("pcodes.csv")
joinedDF = over20DF.join(pcodesDF, "pcode").persist()
```

Fetch the records which pincode is “94020”

```
joinedDF.where("pcode = 94020").show()
```

Fetch another pincode.

```
joinedDF.where("pcode = 87501").show()
```

```
scala> val over20DF = spark.read.option("header","true").csv("people.csv").where("age > 20")
over20DF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [pcode: string, lastName: string ... 2 more fields]

scala> val pcodesDF = spark.read.option("header","true").csv("pcodes.csv")
pcodesDF: org.apache.spark.sql.DataFrame = [pcode: string, city: string ... 1 more field]

scala> val joinedDF = over20DF.join(pcodesDF, "pcode").persist()
joinedDF: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [pcode: string, lastName: string ... 4 more fields]

scala> 20/11/11 06:19:19 WARN HeartbeatReceiver: Removing executor driver with no recent heartbeats: 137346 ms exceeds timeout 120000 ms
20/11/11 06:19:19 WARN SparkContext: Killing executors is not supported by current scheduler.


scala> joinedDF.where("pcode = 94020").show()
+-----+--------+---------+---+---------+-----+
|pcode|lastName|firstName|age|     city|state|
+-----+--------+---------+---+---------+-----+
|94020|  Turing|     Alan| 32|Palo Alto|   CA|
|94020|Lovelace|      Ada| 28|Palo Alto|   CA|
+-----+--------+---------+---+---------+-----+


scala> joinedDF.where("pcode = 87501").show()
+-----+--------+---------+---+--------+-----+
|pcode|lastName|firstName|age|    city|state|
+-----+--------+---------+---+--------+-----+
|87501| Babbage|  Charles| 49|Santa Fe|   NM|
+-----+--------+---------+---+--------+-----+
```

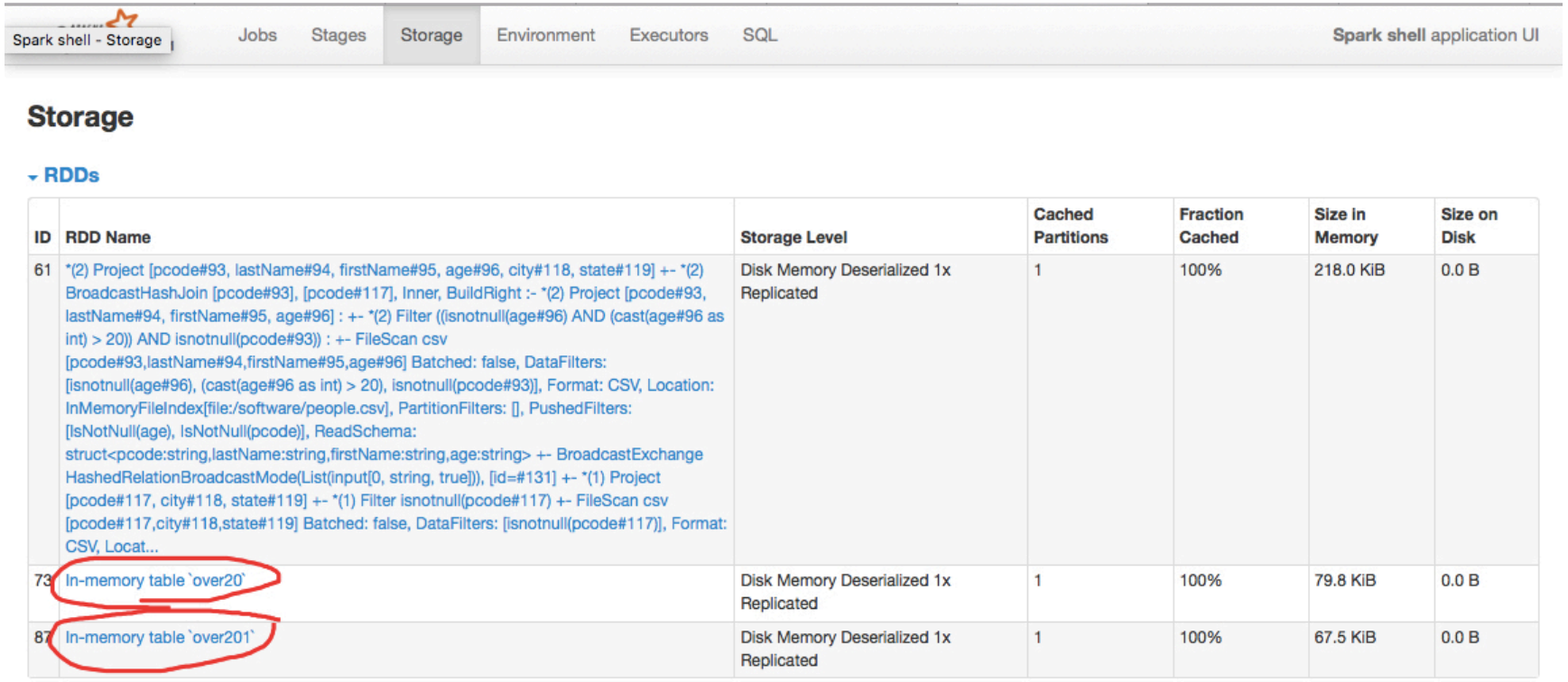
Register as a table and cache.

```
over20DF.createTempView("overAge")

spark.sql("CACHE TABLE overAge")
```

spark.sql("CACHE TABLE over_age AS SELECT *  FROM overAge WHERE age > 20")
spark.sql("select * from over_age").show

```
scala> over20DF.createTempView("overAge")

scala> spark.sql("CACHE TABLE overAge")
20/11/11 06:50:19 WARN CacheManager: Asked to cache already cached data.
res16: org.apache.spark.sql.DataFrame = []

scala> spark.sql("CACHE TABLE over_age AS SELECT *  FROM overAge WHERE age > 20")
20/11/11 06:50:43 WARN CacheManager: Asked to cache already cached data.
res17: org.apache.spark.sql.DataFrame = []

scala> spark.sql("select * from over_age").show
+-----+--------+---------+---+
|pcode|lastName|firstName|age|
+-----+--------+---------+---+
|02134|  Hopper|    Grace| 52|
|94020|  Turing|     Alan| 32|
|94020|Lovelace|      Ada| 28|
|87501| Babbage|  Charles| 49|
|02134|   Wirth|  Niklaus| 48|
+-----+--------+---------+---+
```

View the cache details using the URL - http://localhost:4040/jobs/




| ID | RDD Name | Storage Level | Cached Partitions | Fraction Cached | Size in Memory | Size on Disk |
|---|---|---|---|---|---|---|
| 61 | *(2) Project [pcode#93, lastName#94, firstName#95, age#96, city#118, state#119] +- *(2) BroadcastHashJoin [pcode#93], [pcode#117], Inner, BuildRight :- *(2) Project [pcode#93, lastName#94, firstName#95, age#96] : +- *(2) Filter ((isnotnull(age#96) AND (cast(age#96 as int) > 20)) AND isnotnull(pcode#93)) : +- FileScan csv [pcode#93,lastName#94,firstName#95,age#96] Batched: false, DataFilters: [isnotnull(age#96), (cast(age#96 as int) > 20), isnotnull(pcode#93)], Format: CSV, Location: InMemoryFileIndex[file:/software/people.csv], PartitionFilters: [], PushedFilters: [IsNotNull(age), IsNotNull(pcode)], ReadSchema: struct<pcode:string,lastName:string,firstName:string,age:string> +- BroadcastExchange HashedRelationBroadcastMode(List(input[0, string, true])), [id=#131] +- *(1) Project [pcode#117, city#118, state#119] +- *(1) Filter isnotnull(pcode#117) +- FileScan csv [pcode#117,city#118,state#119] Batched: false, DataFilters: [isnotnull(pcode#117)], Format: CSV, Locat... | Disk Memory Deserialized 1x Replicated | 1 | 100% | 218.0 KiB | 0.0 B |
| 73 | In-memory table `over20` | Disk Memory Deserialized 1x Replicated | 1 | 100% | 79.8 KiB | 0.0 B |
| 87 | In-memory table `over201` | Disk Memory Deserialized 1x Replicated | 1 | 100% | 67.5 KiB | 0.0 B |

Changing the storage tab. Verify the size in memory.

Let us change the storage type.

Specify the Storage location as Disk.

```
import org.apache.spark.storage.StorageLevel
joinedDF.persist(StorageLevel.DISK_ONLY)
```

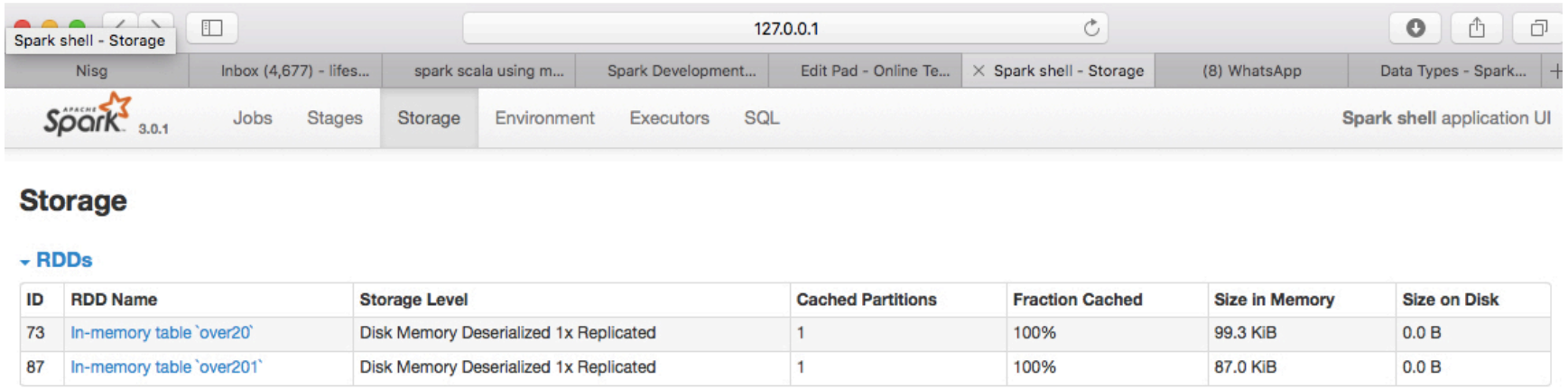
Remove the cache and cache again to change the persist level.

```
joinedDF.unpersist()
joinedDF.cache()
```

```
scala>

scala> import org.apache.spark.storage.StorageLevel
import org.apache.spark.storage.StorageLevel

scala> joinedDF.persist(StorageLevel.DISK_ONLY)
20/11/11 06:56:26 WARN CacheManager: Asked to cache already cached data.
res19: joinedDF.type = [pcode: string, lastName: string ... 4 more fields]

scala> joinedDF.unpersist()
res20: joinedDF.type = [pcode: string, lastName: string ... 4 more fields]

scala> joinedDF.persist(StorageLevel.DISK_ONLY)
res21: joinedDF.type = [pcode: string, lastName: string ... 4 more fields]

scala> joinedDF.cache()
20/11/11 06:57:39 WARN CacheManager: Asked to cache already cached data.
res22: joinedDF.type = [pcode: string, lastName: string ... 4 more fields]

scala>
```

To Remove the table/view – over20 from cache follow the steps given below.
From the web UI – storage section. You can verify the cache details. Currently there are two objects in cache.

Before removing the cache:

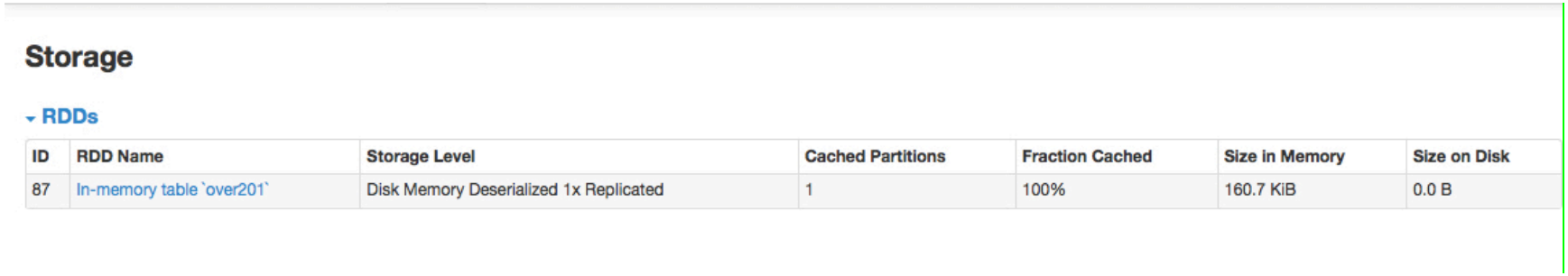



Display the list of tables:

```
spark.catalog.listTables.show
```

```
spark.catalog.uncacheTable("overage")
```

After executing the above command: the above mention cache should be removed as shown below:




----------Lab Ends Here ----------

## 19. Broadcast & Repartition – 45 Minutes

Open a spark shell terminal and perform the following steps.
In this we are broadcasting the states and countries details to all the executor.

**Understanding Broadcast.**

```
#pyspark
```

Perform the following steps on the spark shell.

Import the necessary packages.

```
from pyspark.sql.functions import broadcast
```

Define the States and Countries data set.

```
states = [("NY","New York"),("CA","California"),("FL","Florida"),("MH","Mahrasthra")]
countries = [("USA","United States of America"),("IN","India")]
```

Covert the state and countries information into Dataframe.

```
stDF = spark.createDataFrame(data=states, schema = ["cs","state"])
csDF = spark.createDataFrame(data=countries, schema = ["cc","country"])
```

Define a dataset of person details.

```
data = [("James","Smith","USA","CA"), \
   ("Michael","Rose","USA","NY"), \
   ("Robert","Williams","USA","CA"), \
   ("Maria","Jones","USA","FL"),("Ram","Kumar","IN","MH")]
```

Define the column structure of the data frame.

```
columns = ["firstname","lastname","country","state"]
```

Person Dataframe construction with schema.

```
perDF = spark.createDataFrame(data=data, schema = columns)
perDF.printSchema()
perDF.show(truncate=False)
```

```
In [59]: perDF = spark.createDataFrame(data=data, schema = columns)
         perDF.printSchema()
         perDF.show(truncate=False)
```

```
root
 |-- firstname: string (nullable = true)
 |-- lastname: string (nullable = true)
 |-- cc: string (nullable = true)
 |-- sc: string (nullable = true)

+---------+--------+---+---+
|firstname|lastname|cc |sc |
+---------+--------+---+---+
|James    |Smith   |USA|CA |
|Michael  |Rose    |USA|NY |
|Robert   |Williams|USA|CA |
|Maria    |Jones   |USA|FL |
|Ram      |Kumar   |IN |MH |
+---------+--------+---+---+
```

Get the Country name for each state using the country code Using broadcast.

```
result = perDF.join(broadcast(csDF), perDF["cc"] == csDF["cc"])
result.show()
```

Derive the country from the code using broadcast

```
In [60]: result = perDF.join(broadcast(csDF), perDF["cc"] == csDF["cc"])

In [67]: result.show()
```

```
+---------+--------+---+---+---+--------------------+---+----------+
|firstname|lastname| cc| sc| cc|             country| cs|     state|
+---------+--------+---+---+---+--------------------+---+----------+
|    James|   Smith|USA| CA|USA|United States of ...| CA|California|
|  Michael|    Rose|USA| NY|USA|United States of ...| NY|  New York|
|   Robert|Williams|USA| CA|USA|United States of ...| CA|California|
|    Maria|   Jones|USA| FL|USA|United States of ...| FL|   Florida|
|      Ram|   Kumar| IN| MH| IN|               India| MH|Mahrasthra|
+---------+--------+---+---+---+--------------------+---+----------+
```

Derive the State from the code using broadcast.

```
result = result.join(broadcast(stDF), result["sc"] == stDF["cs"])
result.show()
```

```
In [62]: result = result.join(broadcast(stDF), result["sc"] == stDF["cs"])

In [63]: result.show()
```

```
+---------+--------+---+---+---+--------------------+---+----------+
|firstname|lastname| cc| sc| cc|             country| cs|     state|
+---------+--------+---+---+---+--------------------+---+----------+
|    James|   Smith|USA| CA|USA|United States of ...| CA|California|
|  Michael|    Rose|USA| NY|USA|United States of ...| NY|  New York|
|   Robert|Williams|USA| CA|USA|United States of ...| CA|California|
|    Maria|   Jones|USA| FL|USA|United States of ...| FL|   Florida|
|      Ram|   Kumar| IN| MH| IN|               India| MH|Mahrasthra|
+---------+--------+---+---+---+--------------------+---+----------+
```

Drop the code - state and country from the Dataframe.

result.drop("cc","sc","cs").show()

```
In [64]: result.drop("cc","sc","cs").show()
```

```
+---------+--------+--------------------+----------+
|firstname|lastname|             country|     state|
+---------+--------+--------------------+----------+
|    James|   Smith|United States of ...|California|
|  Michael|    Rose|United States of ...|  New York|
|   Robert|Williams|United States of ...|California|
|    Maria|   Jones|United States of ...|   Florida|
|      Ram|   Kumar|               India|Mahrasthra|
+---------+--------+--------------------+----------+
```

## Understanding Repartition

Create a simple RDD using the `sc.parallelize` method to determine the number of partitions spark creates by default.

```
data = [1, 2, 3, 4, 5,6,7,8,9,10,11]

rdd = sc.parallelize(data)
print(rdd.getNumPartitions())
```

```
In [21]: data = [1, 2, 3, 4, 5,6,7,8,9,10,11]

In [22]: rdd = sc.parallelize(data)

In [23]: print(rdd.getNumPartitions())
6
```

As shown above, spark creates 6 partitions by default here (Number of cores of your machine). Now, you must be wondering where this default number has come from.

Spark has a default parallelism parameter which is determined by, `sc.defaultParallelism`. When we run this method, it returns 6 as shown below, which is having 6 cores. It may be different in your case, depending on your machine cores.

## sc.defaultParallelism

This is how the data is present inside each partition. Spark uses an internal Hash Partitioning Scheme to split the data into these smaller chunks.

We can use the `rdd.glom` method to display the partitions in a list.
`glom` - returns an RDD having the elements within each partition in a separate list

## # rdd.glom().collect()

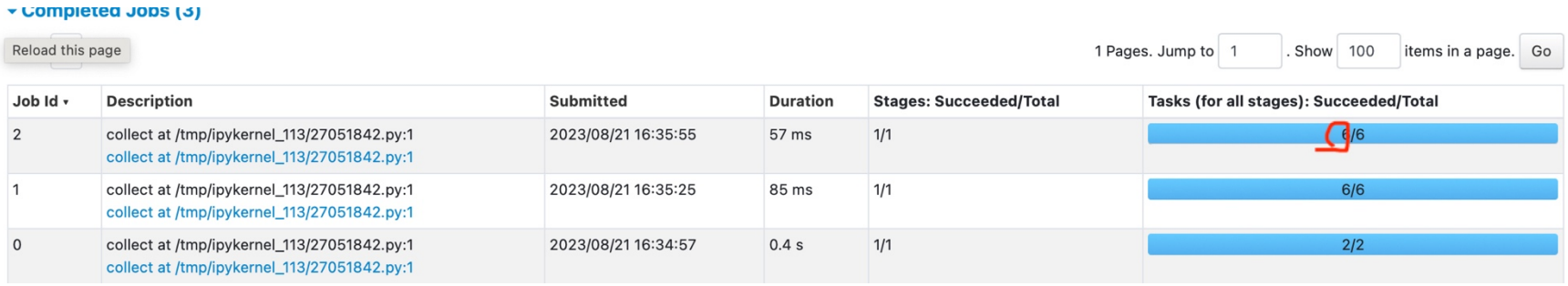
```
In [24]: sc.defaultParallelism
Out[24]: 6
```

Reload this page

```
In [25]: rdd.glom().collect()
Out[25]: [[1], [2, 3], [4, 5], [6, 7], [8, 9], [10, 11]]
```

In above, Inner Array represents the Partition. As shown above it represent 6 partitions.
Let's visualize the above collect operation in the Spark WebUI(if you are using spark locally, navigate to http://localhost:4040 to see the spark webui or if spark is being accessed via a cluster navigate to your cluster specific localhost webUI)
Note: Spark shell notifies a port number before creating a Sparksession.

Since 2 partitions are present, 2 tasks would be launched for this action (Depends if you have that cores or executor), as shown below,



How to see number of Partitions in a Dataframe?

The above method used for an RDD can also be applied to a dataframe. Let's convert the RDD to a Dataframe and apply the same method. We can see that the same number of partitions are present.

```
df = rdd.toDF("Integer")
```

Converting rdd to Dataframe

```
df.rdd.glom().collect()
```

```
In [28]: df = rdd.toDF("Integer")

In [30]: df.rdd.glom().collect()
Out[30]: [[Row(value=1)],
          [Row(value=2), Row(value=3)],
          [Row(value=4), Row(value=5)],
          [Row(value=6), Row(value=7)],
          [Row(value=8), Row(value=9)],
          [Row(value=10), Row(value=11)]]
```

By Default, partition is divided by size in case of file.
Now, you must've had the question in your mind as to how spark partitions the data when reading textfiles.

Let's read a simple textfile and see the number of partitions here.

```
peopleDF = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/Software/data/zcodes.csv")

peopleDF.rdd.getNumPartitions
```

```
[Row(value=10), Row(value=11)]]

In [32]: peopleDF = spark.read.format("csv").option("sep", ",") \
         .option("inferSchema", "true").option("header", "true").load("/Software/data/zcodes.csv")

In [34]: peopleDF.rdd.getNumPartitions()

Out[34]: 1
```

We can see that only 1 partition is created here. Alright let's break this down,
*Spark by default creates 1 partition for every 128 MB of the file.*
So if you are reading a file of size 1GB, it creates 10 partitions.

**So how to control the number of bytes a single partition can hold ?**
The file being read here is less than that of the minimum size, hence only 1 partition is created in this case.
The no. of partitions is determined by `spark.sql.files.maxPartitionBytes` parameter, which is set to 128 MB, by default.

```
spark.conf.get("spark.sql.files.maxPartitionBytes") # -> 128 MB by default
```

```
In [35]: spark.conf.get("spark.sql.files.maxPartitionBytes")
Out[35]: '134217728b'
```

**Note:** The files being read must be splitable by default for spark to create partitions when reading the file. So, in case of compressed files like snappy, gz or lzo etc, a single partition is created irrespective of the size of the file.
Let us upload a file of larger size.

```
sfDF = spark.read.format("csv").option("sep", ",").option("inferSchema",
"true").option("header", "true").load("/Software/data/departuredelays.csv")

sfDF.rdd.getNumPartitions()
```

```
In [37]: sfDF = spark.read.format("csv").option("sep", ",") \
         .option("inferSchema", "true").option("header", "true").load("/Software/data/departuredelays.csv")

In [38]: sfDF.rdd.getNumPartitions()
Out[38]: 6
```

Reload this page

Let's manually set the `spark.sql.files.maxPartitionBytes` to 4MB, so that we get 9 partitions upon reading the file.

```
spark.conf.set("spark.sql.files.maxPartitionBytes", 4000000)
spark.conf.get("spark.sql.files.maxPartitionBytes")
```

Upload the file again

```
sfDF1 = spark.read.format("csv").option("sep", ",").option("inferSchema",
"true").option("header", "true").load("/Software/data/departuredelays.csv")
sfDF1.rdd.getNumPartitions()
```

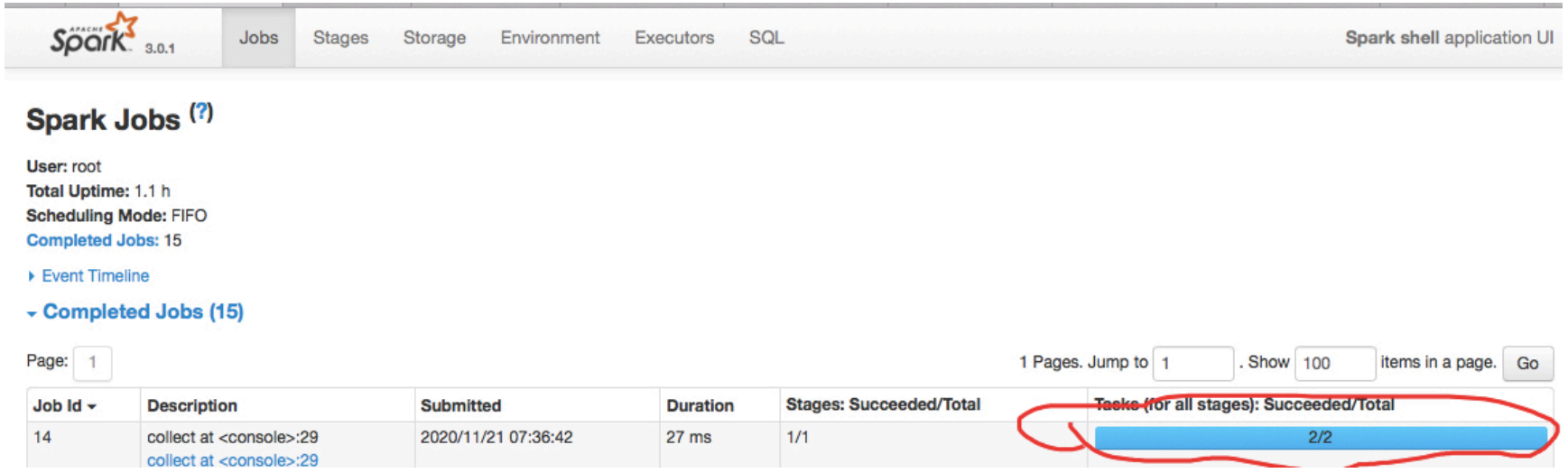
Now, there are 9 Partitions. It depends on the size of your file.

```
In [39]: spark.conf.set("spark.sql.files.maxPartitionBytes", 4000000)
         spark.conf.get("spark.sql.files.maxPartitionBytes")

Out[39]: '4000000'

In [40]: sfDF1 = spark.read.format("csv").option("sep", ",").option("inferSchema", "true").option("header", "true").load("/S
         sfDF1.rdd.getNumPartitions()

Out[40]: 9
```

Reload this page

Let's analyse with a simple example where we create an RDD with a single partition and perform an action on the same.

When the above action is seen on the Spark WebUI, only 2 tasks would be assigned to process this data. So, imagine a case where you are processing a huge volume of data without the concept of partitioning, then the entire data would be processed by a single executor taking a lot of time and memory.



Now what if you wish to control these partitions by yourself. This is where the methods of `repartition` and `coalesce` come to effect.

**Repartition and Coalesce**

Create a data collection.

```
mydata = [1, 2, 3, 4, 5,6,7,8,9,10,11,12]
```

Convert the collection into RDD with one Partition only.

```
rdd1 = sc.parallelize(mydata,1)
```

Determine the number of partitions. It should be only 1.

```
rdd1.getNumPartitions()
```

Repartition and Coalesce

```
In [2]: mydata = [1, 2, 3, 4, 5,6,7,8,9,10,11,12]

In [6]: rdd1 = sc.parallelize(mydata,1)

In [9]: rdd1.getNumPartitions()
Out[9]: 1
```

Let us convert RDD to data frame for further analysis.

```
df = rdd1.toDF("Integer")
```

Repartition the above dataframe to 10 partitions from 1.

```
partDF = df.repartition(10)
```

```
partDF.rdd.getNumPartitions()
```

Now the number of partitions should be 10.

```
In [11]: df = rdd1.toDF("Integer")

In [12]: partDF = df.repartition(10)

In [13]: partDF.rdd.getNumPartitions()
Out[13]: 10
```

After repartitioning, the partition size has increased from 1 to 10 as shown above.

You can observe the partitions information or data as shown below.

```
spark.conf.get("spark.sql.shuffle.partitions")
partDF.rdd.glom().collect()
```

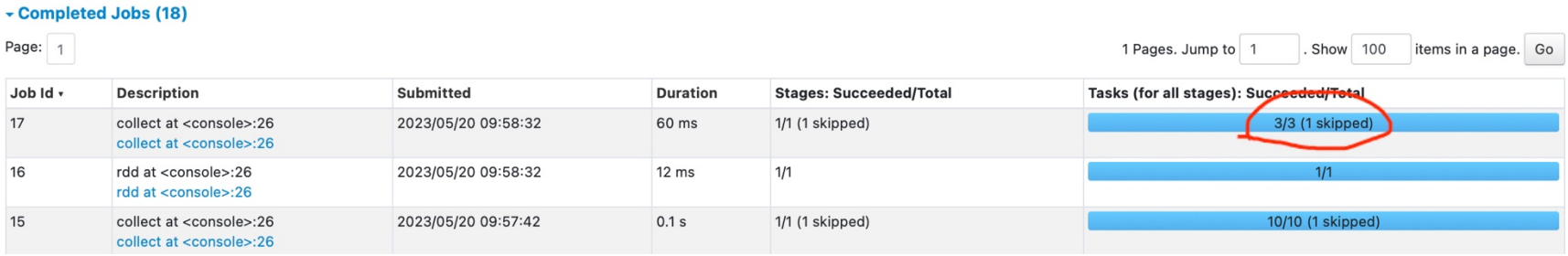
```
In [14]: spark.conf.get("spark.sql.shuffle.partitions")
Out[14]: '200'

In [16]: partDF.rdd.glom().collect()
Out[16]: [[Row(value=4)],
          [Row(value=11), Row(value=1)],
          [Row(value=3), Row(value=12)],
          [Row(value=7)],
          [Row(value=10)],
          [Row(value=5)],
          [Row(value=6)],
          [Row(value=8)],
          [Row(value=2)],
          [Row(value=9)]]
```

Let's visualise the same on the Spark WebUI

```
dfre = df.repartition(3)
dfre.rdd.glom().collect()
```

```
In [17]: dfre = df.repartition(3)

In [18]: dfre.rdd.glom().collect()

Out[18]: [[Row(value=7), Row(value=6), Row(value=9), Row(value=12)],
          [Row(value=11), Row(value=10), Row(value=8), Row(value=4)],
          [Row(value=3), Row(value=5), Row(value=2), Row(value=1)]]
```

The above execution when seen on the spark web console would show 3 tasks being issued on the collect operation(stage 2).

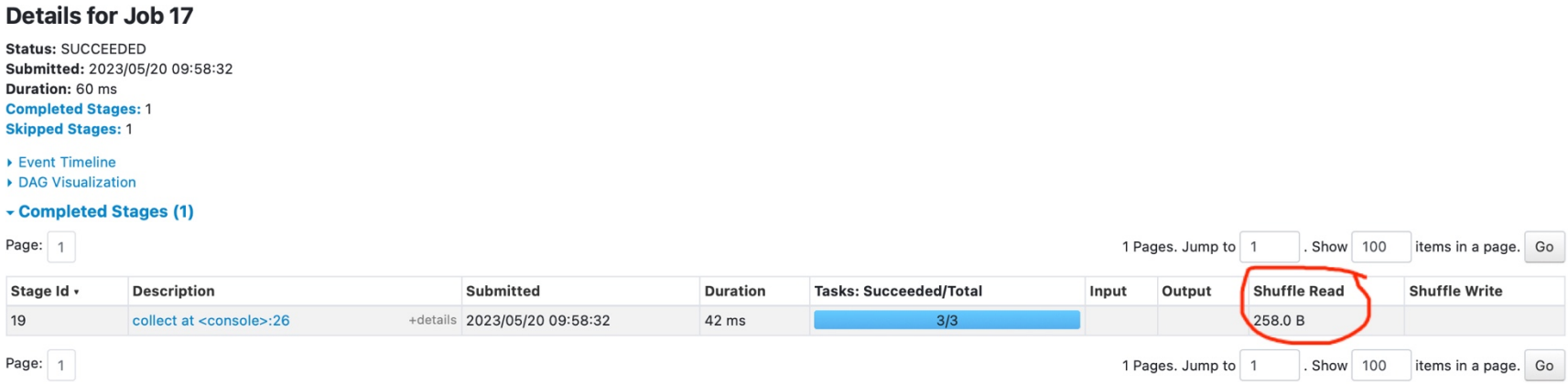


▾ Completed Jobs (18)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

| Job Id ▾ | Description | Submitted | Duration | Stages: Succeeded/Total | Tasks (for all stages): Succeeded/Total |
|---|---|---|---|---|---|
| 17 | collect at <console>:26<br>collect at <console>:26 | 2023/05/20 09:58:32 | 60 ms | 1/1 (1 skipped) | 3/3 (1 skipped) |
| 16 | rdd at <console>:26<br>rdd at <console>:26 | 2023/05/20 09:58:32 | 12 ms | 1/1 | 1/1 |
| 15 | collect at <console>:26<br>collect at <console>:26 | 2023/05/20 09:57:42 | 0.1 s | 1/1 (1 skipped) | 10/10 (1 skipped) |

Click on the Job description of Job ID 17 – It may be different for your case.



**Details for Job 17**

**Status:** SUCCEEDED
**Submitted:** 2023/05/20 09:58:32
**Duration:** 60 ms
**Completed Stages:** 1
**Skipped Stages:** 1

▸ Event Timeline
▸ DAG Visualization

▾ **Completed Stages (1)**

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

| Stage Id ▾ | Description | Submitted | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|
| 19 | collect at <console>:26 +details | 2023/05/20 09:58:32 | 42 ms | 3/3 | | | 258.0 B | |

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

As shown above, repartition causes shuffling of data.

**Coalesce**

Unlike repartition, **coalesce doesn't perform a shuffle** to create the partitions.

Let's analyse this using our dataset, let's reduce the number of partitions to 2 now,

```
partDF.rdd.glom().collect()
partDF.rdd.getNumPartitions()
```

Currently it has 10 partitions, let us reduce to 2.

```
part3 = partDF.coalesce(2)
part3.rdd.getNumPartitions()
```

```
In [21]: part3 = partDF.coalesce(2)
         part3.rdd.getNumPartitions()

Out[21]: 2
```

https://datanoon.com/blog/spark_repartition_coalesce/
https://medium.com/@mrpowers/managing-spark-partitions-with-coalesce-and-repartition-4050c57ad5c4

## 20. Spark – Hive Integration – 45 Minutes

When working with Hive, one must instantiate SparkSession with Hive support, including connectivity to a persistent Hive metastore, support for Hive serdes, and Hive user-defined functions. Users who do not have an existing Hive deployment can still enable Hive support.

use spark.sql.warehouse.dir to specify the default location of database in warehouse

Open a pyspark terminal (Using CLI only)

```
#pyspark --conf spark.sql.catalogImplementation=hive
```

Execute the following code.

```
from os.path import abspath

from pyspark.sql import SparkSession
from pyspark.sql import Row

# warehouse_location points to the default location for managed databases
and tables
warehouse_location = abspath('/opt/spark/spark-warehouse')
```

Only for CLI Mode.

```
spark = SparkSession \
    .builder \
    .appName("Python Spark SQL Hive integration example") \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .config("spark.sql.catalogImplementation", "hive") \
    .enableHiveSupport() \
    .getOrCreate()
```

Create a hive datawarehouse table.

```
spark.sql("CREATE TABLE IF NOT EXISTS src (key INT, value STRING) USING
hive")
```

Load the data in the hive table, src

```
spark.sql("LOAD DATA LOCAL INPATH
'/opt/spark/examples/src/main/resources/kv1.txt' INTO TABLE src")
```

Some Queries to retrieve data from the src table.

```
# Queries are expressed in HiveQL
spark.sql("SELECT * FROM src").show()
```

```
# +---+-------+
# |key|  value|
# +---+-------+
# |238|val_238|
# | 86| val_86|
# |311|val_311|
# ...
```

```
# Aggregation queries are also supported.
spark.sql("SELECT COUNT(*) FROM src").show()
```

```
# +--------+
# |count(1)|
# +--------+
# |    500 |
# +--------+
```

```
# The results of SQL queries are themselves DataFrames and support all
normal functions.
sqlDF = spark.sql("SELECT key, value FROM src WHERE key < 10 ORDER BY
key")

# The items in DataFrames are of type Row, which allows you to access each
column by ordinal.
stringsDS = sqlDF.rdd.map(lambda row: "Key: %d, Value: %s" % (row.key,
row.value))
for record in stringsDS.collect():
    print(record)

# Key: 0, Value: val_0
# Key: 0, Value: val_0
# Key: 0, Value: val_0
# ...

# You can also use DataFrames to create temporary views within a
SparkSession.
```

```
Record = Row("key", "value")
recordsDF = spark.createDataFrame([Record(i, "val_" + str(i)) for i in range(1,
101)])
recordsDF.createOrReplaceTempView("records")

# Queries can then join DataFrame data with data stored in Hive.
spark.sql("SELECT * FROM records r JOIN src s ON r.key = s.key").show()
```

Execution Output:

```
[root@spark0 data]# pyspark --conf spark.sql.catalogImplementation=hive
Python 3.6.8 (default, Nov 16 2020, 16:55:22)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-44)] on linux
Type "help", "copyright", "credits" or "license" for more information.
21/07/03 10:09:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using built
in-java classes where applicable
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
21/07/03 10:10:04 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.1.2
      /_/

Using Python version 3.6.8 (default, Nov 16 2020 16:55:22)
Spark context Web UI available at http://spark0:4041
Spark context available as 'sc' (master = local[*], app id = local-1625307004585).
SparkSession available as 'spark'.
>>> from os.path import abspath
>>>
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql import Row
```

Profile: (null)
Command: None

```
>>> spark.sql("LOAD DATA LOCAL INPATH '/opt/spark/examples/src/main/resources/kv1.txt' INTO TABLE src")
21/07/03 10:11:29 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
DataFrame[]
>>> spark.sql("SELECT * FROM src").show()
+---+-------+
|key|  value|
+---+-------+
|238|val_238|
| 86| val_86|
|311|val_311|
| 27| val_27|
|165|val_165|
|409|val_409|
|255|val_255|
|278|val_278|
| 98| val_98|
```

```
>>> spark.sql("SELECT COUNT(*) FROM src").show()
+--------+
|count(1)|
+--------+
|     500|
+--------+

>>> sqlDF = spark.sql("SELECT key, value FROM src WHERE key < 10 ORDER BY key")
>>> stringsDS = sqlDF.rdd.map(lambda row: "Key: %d, Value: %s" % (row.key, row.value))
>>> for record in stringsDS.collect():
...     print(record)
...
Key: 0, Value: val_0
Key: 0, Value: val_0
Key: 0, Value: val_0
Key: 2, Value: val_2
Key: 4, Value: val_4
Key: 5, Value: val_5
Key: 5, Value: val_5
Key: 5, Value: val_5
```

```
>>> Record = Row("key", "value")
>>> recordsDF = spark.createDataFrame([Record(i, "val_" + str(i)) for i in range(1, 101)])
>>> recordsDF.createOrReplaceTempView("records")
>>> spark.sql("SELECT * FROM records r JOIN src s ON r.key = s.key").show()
+---+------+---+------+
|key| value|key| value|
+---+------+---+------+
|  2| val_2|  2| val_2|
|  4| val_4|  4| val_4|
|  5| val_5|  5| val_5|
|  5| val_5|  5| val_5|
|  5| val_5|  5| val_5|
|  8| val_8|  8| val_8|
|  9| val_9|  9| val_9|
| 10|val_10| 10|val_10|
| 11|val_11| 11|val_11|
| 12|val_12| 12|val_12|
| 12|val_12| 12|val_12|
| 15|val_15| 15|val_15|
| 15|val_15| 15|val_15|
```

---------- Lab Ends Here ----------

## 21. Spark integration with Hive Cluster (Local Mode)

Pre requisites: Spark Installation and Hive Local Mode installation.

You can refer Hive Lab for Hive Local Installation.

In this lab, we will configure to use Apache Spark with Hive.

```
export SPARK_HOME=/opt/spark
export HIVE_HOME=/opt/hive_local
```

Set HIVE_HOME and SPARK_HOME accordingly.

## 22. Annexure & Errata:

Quit Scala CLI :- :q
Error: No suitable device found: no device found for connection 'System eth0'.
You could try removing/renaming /etc/udev/rules.d/70-persistent-net.rules and reboot. A new file will be created, sometimes that fixes this kind of problems.

You could also add a file "/etc/sysconfig/network-scripts/ifcfg-eth1", can you bring up eth1 then?

Changing hostname:

- /etc/hosts
- vi /etc/sysconfig/network
- sysctl kernel.hostname=slave
- bash

Unable to start spark shell with the following error

```
  at org.apache.hadoop.hive.ql.session.SessionState.createSessionDirs(SessionState.java:554)
  at org.apache.hadoop.hive.ql.session.SessionState.start(SessionState.java:508)
  ... 85 more
Caused by: java.net.ConnectException: Connection refused
  at sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
  at sun.nio.ch.SocketChannelImpl.finishConnect(SocketChannelImpl.java:717)
  at org.apache.hadoop.net.SocketIOWithTimeout.connect(SocketIOWithTimeout.java:206)
  at org.apache.hadoop.net.NetUtils.connect(NetUtils.java:531)
  at org.apache.hadoop.net.NetUtils.connect(NetUtils.java:495)
  at org.apache.hadoop.ipc.Client$Connection.setupConnection(Client.java:614)
  at org.apache.hadoop.ipc.Client$Connection.setupIOstreams(Client.java:712)
```

```
  at org.apache.hadoop.ipc.Client$Connection.access$2900(Client.java:375)
  at org.apache.hadoop.ipc.Client.getConnection(Client.java:1528)
  at org.apache.hadoop.ipc.Client.call(Client.java:1451)
```

Solution: unset HADOOP_CONF_DIR  and unset HADOOP_HOME

https://jaceklaskowski.gitbooks.io/mastering-apache-spark/

## issue: Cannot assign requested address

```
17/03/15 21:44:15 ERROR spark.SparkContext: Error initializing SparkContext.
java.net.BindException: Cannot assign requested address: Service 'sparkDriver' f
ailed after 16 retries (starting from 0)! Consider explicitly setting the approp
riate port for the service 'sparkDriver' (for example spark.ui.port for SparkUI)
to an available port or increasing spark.port.maxRetries.
      at sun.nio.ch.Net.bind0(Native Method)
      at sun.nio.ch.Net.bind(Net.java:433)
      at sun.nio.ch.Net.bind(Net.java:425)
      at sun.nio.ch.ServerSocketChannelImpl.bind(ServerSocketChannelImpl.java:
223)
```

Solution:

- include host to ip entry in /etc/hosts --> 192.168.188.178 master
- update in conf/spark-env.sh
    - SPARK_LOCAL_IP=127.0.0.1

- SPARK_MASTER_HOST=192.168.188.178

- comment the following in conf/spark-defaults.conf
  - # spark.master spark://master:7077

## Unable to start spark shell:

Caused by: java.io.IOException: Error accessing /opt/jdk/jre/lib/ext/._cld*.jar
Such kind of error resolve by removing all the hidden file. .* (You can determine the file by running #ls -alt). List all the hidden dot file and remove it.

## Yum repo config

# mkdir -p /redhatimg
# mount -o loop,ro /mnt/hgfs/MyExperiment/rhel-server-6.4-x86_64-dvd.iso /redhatimg

Create an entry in /etc/fstab so that the system always mounts the DVD image after a reboot.
/mnt/hgfs/MyExperiment/rhel-server-6.4-x86_64-dvd.iso /redhatimg iso9660 loop,ro 0 0

/etc/yum.repos.d/rheldiso.repo

[rhel6dvdiso]
name=RedHatOS DVD ISO
mediaid=1359576196.686790
baseurl=file:///redhatimg
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

## *Issue in Pyspark: TypeError: Too many parameters for typing.Iterable; actual 2, expected 1*

**Background:**
Python 3.6.8
3.3.0

File "/usr/local/lib/python3.6/site-packages/typing_extensions.py", line 113, in _check_generic
raise TypeError(f"Too {'many' if alen > elen else 'few'} parameters for {cls};"
TypeError: Too many parameters for typing.Iterable; actual 2, expected 1

**Solution:**

Check python version :

```
#python3 -V
```

```
#yum install python3.8
# python3.8 --version
```

Let's set default python3 to python3.8 on RHEL 8:
Perform the following steps on your machine

```
sudo alternatives --config python3
or
sudo update-alternatives --config python3
```

```
[root@spark0 /]# alternatives --config python3

There are 2 programs which provide 'python3'.

  Selection    Command
-----------------------------------------------
*+ 1           /usr/bin/python3.6
   2           /usr/bin/python3.8

Enter to keep the current selection[+], or type selection number: 2
[root@spark0 /]#
```

Accept 2 as atlernatives.

Issue: Traceback (most recent call last):

File "/usr/local/lib/python3.6/site-packages/notebook/notebookapp.py", line 2027, in init_server_extensions

python3.8 -m pip install jupyter_contrib_nbextensions

## Bash profile

```
export  JAVA_HOME=/opt/jdk
export PATH=$PATH:$JAVA_HOME/bin
export PATH=$PATH:/opt/spark/bin:/Software/maven/bin
export PYSPARK_PYTHON=python3.8
export SPARK_CONF_DIR=/opt/spark
export SPARK_HOME=/opt/spark
export PYSPARK_DRIVER_PYTHON=jupyter
#export PYSPARK_DRIVER_PYTHON=python3.8
#export PYSPARK_SUBMIT_ARGS='--packages org.apache.spark:spark-sql-kafka-0-
10_2.12:3.3.0,org.apache.spark:spark-streaming-ka
fka-0-10_2.12:3.3.0 pyspark'
export PYSPARK_DRIVER_PYTHON_OPTS='notebook'

export PACKAGES="io.delta:delta-core_2.12:2.1.0",org.apache.spark:spark-sql-kafka-0-10_2.12:3.3.0
#export PYSPARK_SUBMIT_ARGS="--packages ${PACKAGES} pyspark-shell"
export PYSPARK_PYTHON=python3.8
# This is for spylon jupyter integration required for pyspark in Delta.
alias snotebook='pyspark --packages io.delta:delta-core_2.12:2.1.0 --conf
spark.sql.extensions=io.delta.sql.DeltaSparkSessio
nExtension --conf "spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog" --
conf spark.sql.warehou
se.dir=/delta/spark-warehouse'

#Required for pyspark+kafka integration.
alias pnotebook='pyspark --packages org.apache.spark:spark-sql-kafka-0-
10_2.12:3.3.0,org.apache.spark:spark-streaming-kafka-
```

```
0-10_2.12:3.3.0 --conf spark.sql.catalogImplementation=hive'
```