#mkdir -p src/main/scala/

---------- ----------

Actual code section:

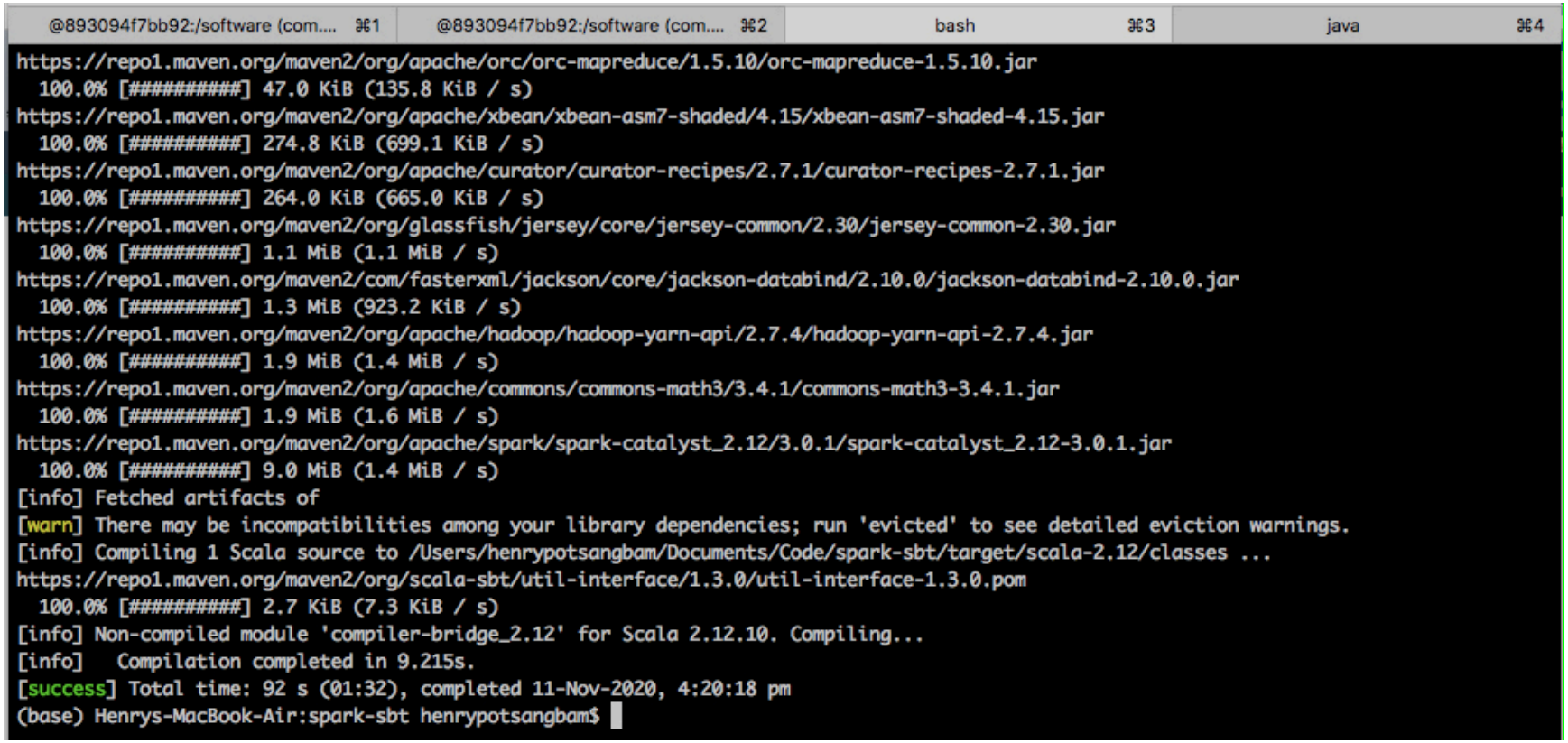
#cd src/main/scala/

Write the code as shown below in a file, NameList.scala in the same folder.

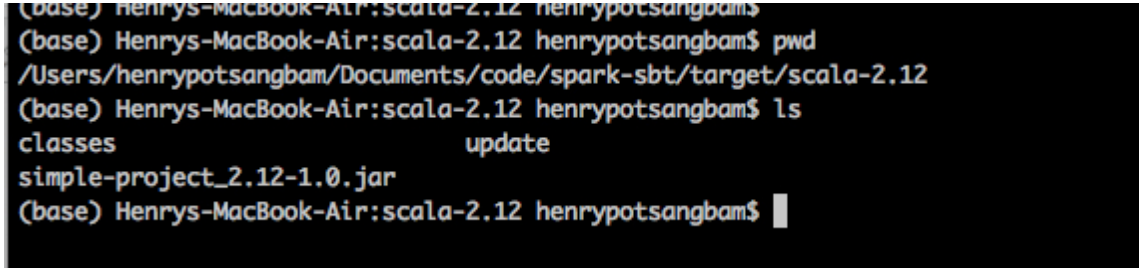
```
import org.apache.spark.sql.SparkSession
object NameList {
        def main(args: Array[String]) {
        if (args.length < 2) {
        System.err.println(
                "Usage: NameList <input-file> <output-file>")
        System.exit(1)
        }
        val spark = SparkSession.builder.getOrCreate()
        spark.sparkContext.setLogLevel("WARN")
        val peopleDF = spark.read.json(args(0))
        val namesDF = peopleDF.select("firstName","lastName")
        namesDF.write.option("header","true").csv(args(1))
        spark.stop
    }
}
```

Change to the directory where build.sbt is located.
cd /opt/spark-sbt

```
# Package a jar containing your application

$ sbt package

Usually take sometimes to download all the packages depending on the speed, usually 5 to 10 minutes.
```



It created an application jar as shown below inside the target folder.



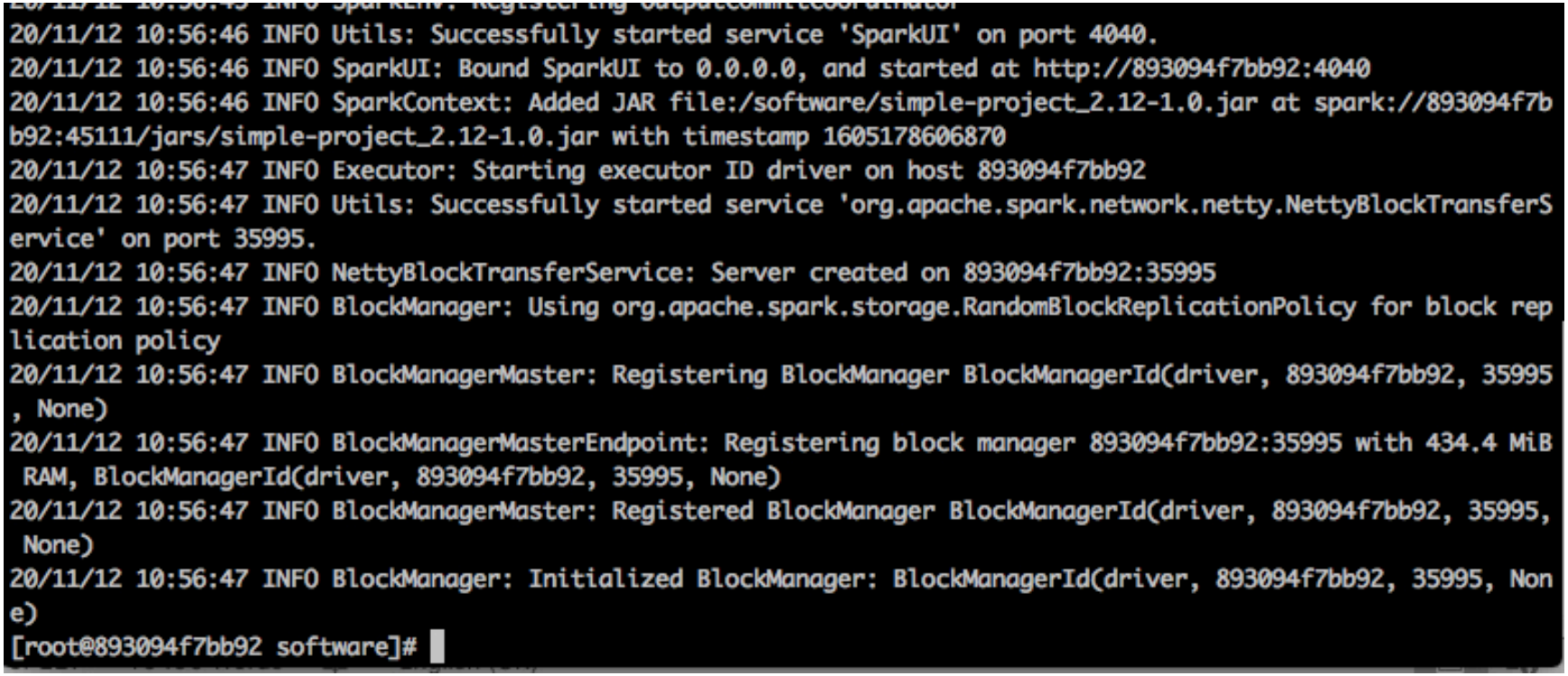
Copy the above jar to your cluster and deploy it.

Use spark-submit to run your application. Specify the files location appropriately. (e.x /opt/ or /software)

export PATH=$PATH:/opt/spark/bin
cd /opt

#spark-submit --class "NameList" --master local[2] simple-project_2.12-1.0.jar names.json myname.txt

```
[root@893094f7bb92 software]#
[root@893094f7bb92 software]# ls
J_AddCat.csv    derby.log                  namesAndAges.parquet  people.txt                 users.json
J_AddDist.csv   influxdb-1.8.2.x86_64.rpm  pcodes.csv            person.txt                 zcodes.csv
README.md       metastore_db               people-no-pcode.csv   postalcode.txt
age.json        mydata                     people.csv            sfpd.csv
age1.json       names.json                 people.json           simple-project_2.12-1.0.jar
[root@893094f7bb92 software]# spark-submit  --class "NameList" --master local[2] simple-project_2.12-1.0.jar n
ames.json myname.txt
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
20/11/12 10:56:43 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
20/11/12 10:56:44 INFO SparkContext: Running Spark version 3.0.1
20/11/12 10:56:44 INFO ResourceUtils: ==============================================================
```



You can verify the output : change the directory to myname.txt folder and view the output.

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

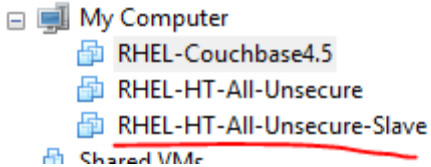
16. Spark Standalone Cluster – 60 Minutes.

You will learn to configure two nodes Spark Standalone Cluster.

Create another instance of the VM by copying the VM folder or making clone. You need to shutdown the VM for that.
If you use clone, then specify as shown below. Hints: Select the parent virtual machine and select VM > Manage > Clone.

It will take few minutes depending on your VM size.

At the end, you will get an additional VM as shown below: (ex)



You need to power on both VM at this step.

Add another node i.e slave node to the Cluster. Change the hostname of the slave node, slave.

Logon to the slave VM. Your slave node should be as shown below.

```
[root@slave sbin]#
[root@slave sbin]# hostname
slave
[root@slave sbin]#
```

Ensure to provide entries in /etc/hosts of both the VMs, so that it can communicate each other's using hostname.

192.168.188.178 master
192.168.188.174 slave

Logon to the first VM i.e master to configure password less connection between the nodes.
SSH access
The root user on the master must be able to connect
a) to its own user account on the master – i.e. ssh master in this context and not necessarily ssh localhost – and
b) to the root user account on the slave via a password-less SSH login.

You have to add the root@master‘s public SSH key (which should be in $HOME/.ssh/id_rsa.pub) to the authorized_keys file of root@slave (in this user’s$HOME/.ssh/authorized_keys).
The following steps will ensure that root user can ssh to its own account without password in all nodes.
Let us generate the keys for user root on the master node, to make sure that root user on master can ssh to slave nodes without password.
$ su - root
$ ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa

$cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
$ ssh master
You need to accept the key for the first time.

You can do this manually or use the following SSH command: to copy public key to all the slave nodes.
$ ssh-copy-id -i $HOME/.ssh/id_rsa.pub root@slave

Accept it, yes and supply the root password for the last time.

This command will prompt you for the login password for user root on slave, then copy the public SSH key for you, creating the correct directory and fixing the permissions as necessary.

The final step is to test the SSH setup by connecting with user root from the master to the user account root on the slave. This step is also needed to save slave‘s host key fingerprint to the root@master‘s known_hosts file.
So, connecting from master to master...
$ ssh master
And from master to slave.
$ ssh slave

```
[root@master sbin]# ssh-keygen -t rsa -P ""
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
/root/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
dd:87:61:51:6e:f9:1e:88:70:a0:ff:f6:92:e5:7f:fb root@master
The key's randomart image is:
+--[ RSA 2048]----+
|          .  ...  |
|         . .  o . |
|        . . .o +  |
|         ..oo.+.. |
|         S...o....|
|           .  o ..|
|            o+   .|
|           .o..  .|
|             ...oE|
+-----------------+
[root@master sbin]# scp
usage: scp [-1246BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]
           [-l limit] [-o ssh_option] [-P port] [-S program]
           [[user@]host1:]file1 ... [[user@]host2:]file2
[root@master sbin]# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
[root@master sbin]# ssh master
Last login: Thu Sep  1 23:45:46 2016 from localhost
[root@master ~]# ssh-copy-id -i $HOME/.ssh/id_rsa.pub root@slave
root@slave's password:
Now try logging into the machine, with "ssh 'root@slave'", and check in:

  .ssh/authorized_keys

to make sure we haven't added extra keys that you weren't expecting.

[root@master ~]# ssh slave
Last login: Thu Sep  1 23:45:46 2016 from localhost
[root@slave ~]#
```

Go to SPARK_HOME/conf/ and create new file with name spark-env.sh on the master node
*There will be **spark-env.sh.template** in same folder and this file gives you detail on how to declare various environment variables.*

Now we will give the IP address of Master.
SPARK_LOCAL_IP=192.168.188.173
SPARK_MASTER_HOST={IP Address}

```
# Options read by executors and drivers running inside the cluster
# - SPARK_LOCAL_IP, to set the IP address Spark binds to on this node
SPARK_LOCAL_IP=192.168.188.173
# - SPARK_PUBLIC_DNS, to set the public DNS name of the driver program
# - SPARK_CLASSPATH, default classpath entries to append
# - SPARK_LOCAL_DIRS, storage directories to use on this node for shuffle and RD
```

```
# Options for the daemons used in the standalone deploy mode
# - SPARK_MASTER_HOST, to bind the master to a different IP address or hostname
SPARK_MASTER_HOST=192.168.188.173
# - SPARK_MASTER_PORT / SPARK_MASTER_WEBUI_PORT, to use non-default ports for th
e master
```

// Note: You need to specify IP only.
On master node; Create SPARK_HOME/conf/slaves and enter the following entries.
Worker process will be started in both the nodes.

```
[root@master conf]# more slaves
slave
master
[root@master conf]#
```

Then go to Spark HOME_DIRECTORY/sbin and run following command from terminal

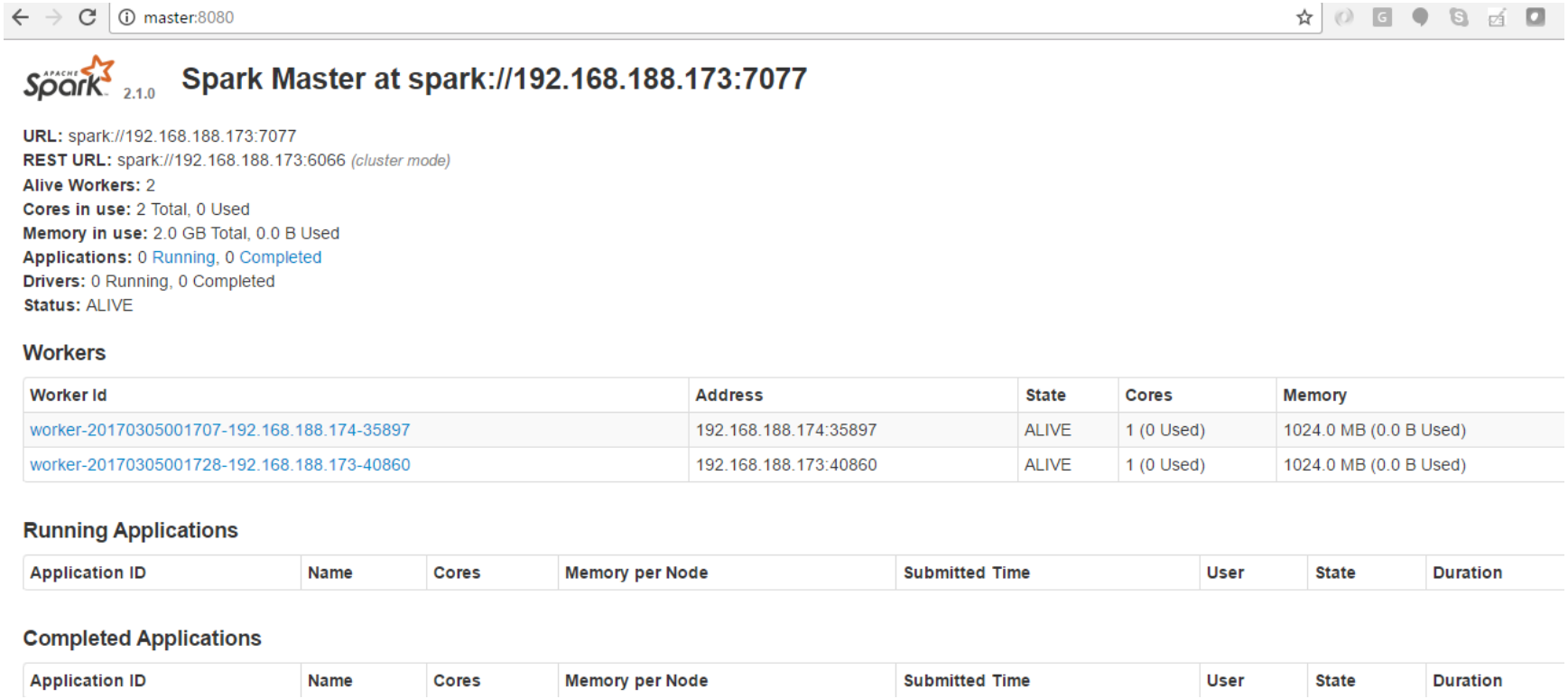
sh sbin/start-all.sh

```
[root@master sbin]# sh start-all.sh
starting org.apache.spark.deploy.master.Master, logging to /spark/spark-2.1/logs
/spark-root-org.apache.spark.deploy.master.Master-1-master.out
slave: starting org.apache.spark.deploy.worker.Worker, logging to /spark/spark-2
.1/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-slave.out
master: starting org.apache.spark.deploy.worker.Worker, logging to /spark/spark-
2.1/logs/spark-root-org.apache.spark.deploy.worker.Worker-1-master.out
master: failed to launch: nice -n 0 /spark/spark-2.1/bin/spark-class org.apache.
spark.deploy.worker.Worker --webui-port 8081 spark://192.168.188.173:7077
master: full log in /spark/spark-2.1/logs/spark-root-org.apache.spark.deploy.wor
ker.Worker-1-master.out
```

This will start a standalone master server by executing it

http://master:8080/

You can access the Spark Master with the above URL.



Once started, the master will print out a spark://HOST:PORT URL for itself, which you can use to connect workers to it, or pass as the “master” argument to SparkContext. You can also find this URL on the master’s web UI, which is http://localhost:8080 by default. Once you have started a worker, look at the master’s web UI (http://localhost:8080 by default). You should see the new node listed there, along with its number of CPUs and memory (minus one gigabyte left for the OS). In our case its 2 node.

Then you can verify the Node using the master UI:

Workers

Worker Id	Address	State	Cores	Memory
worker-20170305001707-192.168.188.174-35897	192.168.188.174:35897	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20170305001728-192.168.188.173-40860	192.168.188.173:40860	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

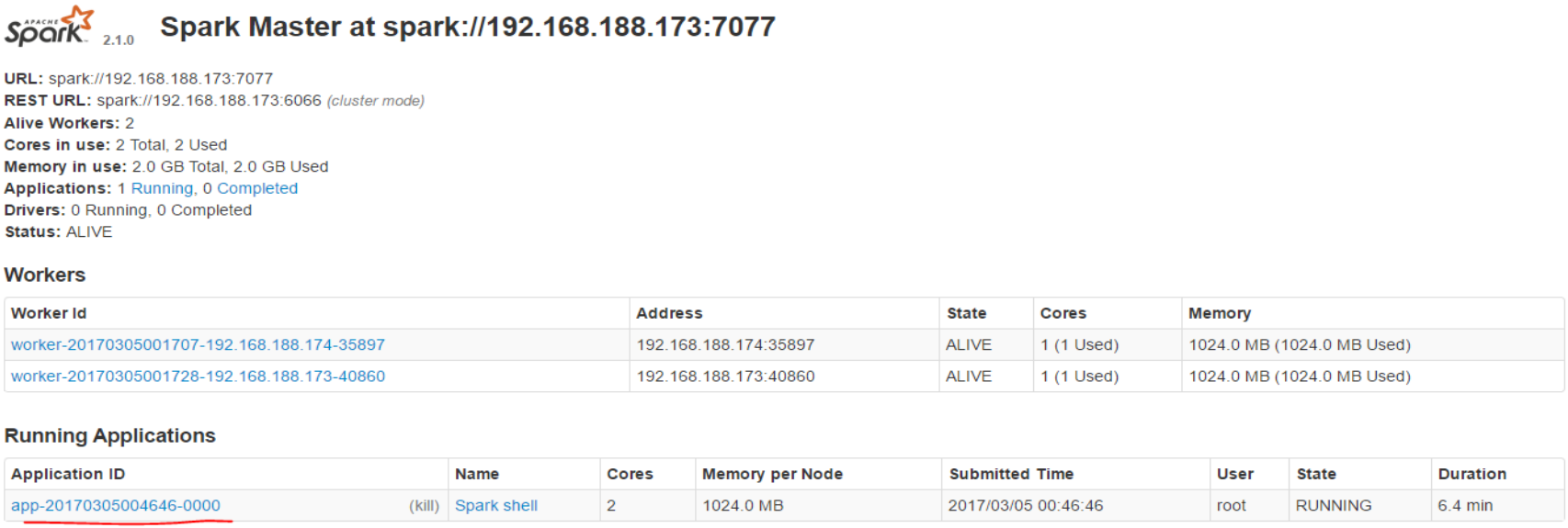
Let us execute Spark shell using Cluster mode:

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, run the following command from the bin folder:
We are executing from the slave node.

#bin/spark-shell --master spark://192.168.188.173:8090

You can verify the application execution from the web UI



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:

```
scala> val textFile = sc.textFile("README.md")
```

```
scala> val textFile  = sc.textFile("/MyTrainingWork/Spark/README.md")
15/06/03 23:32:06 INFO MemoryStore: ensureFreeSpace(73391) called with curMem=18
1810, maxMem=278019440
15/06/03 23:32:06 INFO MemoryStore: Block broadcast_1 stored as values in memory
 (estimated size 71.7 KB, free 264.9 MB)
15/06/03 23:32:06 INFO MemoryStore: ensureFreeSpace(31262) called with curMem=25
5201, maxMem=278019440
15/06/03 23:32:06 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in
memory (estimated size 30.5 KB, free 264.9 MB)
15/06/03 23:32:06 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on l
ocalhost:1599 (size: 30.5 KB, free: 265.1 MB)
15/06/03 23:32:06 INFO BlockManagerMaster: Updated info of block broadcast_1_pie
ce0
15/06/03 23:32:06 INFO SparkContext: Created broadcast 1 from textFile at <conso
le>:21
textFile: org.apache.spark.rdd.RDD[String] = /MyTrainingWork/Spark/README.md Map
PartitionsRDD[3] at textFile at <console>:21
```

Let's start with a few actions:

scala> textFile.count() *// Number of items in this RDD*

```
scala> textFile.count()
15/06/03 23:32:16 INFO FileInputFormat: Total input paths to process : 1
15/06/03 23:32:16 INFO SparkContext: Starting job: count at <console>:24
15/06/03 23:32:16 INFO DAGScheduler: Got job 0 (count at <console>:24) with 2 ou
tput partitions (allowLocal=false)
15/06/03 23:32:16 INFO DAGScheduler: Final stage: Stage 0(count at <console>:24)

15/06/03 23:32:16 INFO DAGScheduler: Parents of final stage: List()
15/06/03 23:32:16 INFO DAGScheduler: Missing parents: List()
15/06/03 23:32:16 INFO DAGScheduler: Submitting Stage 0 (/MyTrainingWork/Spark/R
EADME.md MapPartitionsRDD[3] at textFile at <console>:21), which has no missing
parents
```

```
tes result sent to driver
15/06/03 23:32:17 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in
 352 ms on localhost (1/2)
15/06/03 23:32:17 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in
 338 ms on localhost (2/2)
15/06/03 23:32:17 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have
all completed, from pool
15/06/03 23:32:17 INFO DAGScheduler: Stage 0 (count at <console>:24) finished in
 0.393 s
15/06/03 23:32:17 INFO DAGScheduler: Job 0 finished: count at <console>:24, took
 0.678076 s
res2: Long = 98

scala>
```

scala> textFile.first() *// First item in this RDD*

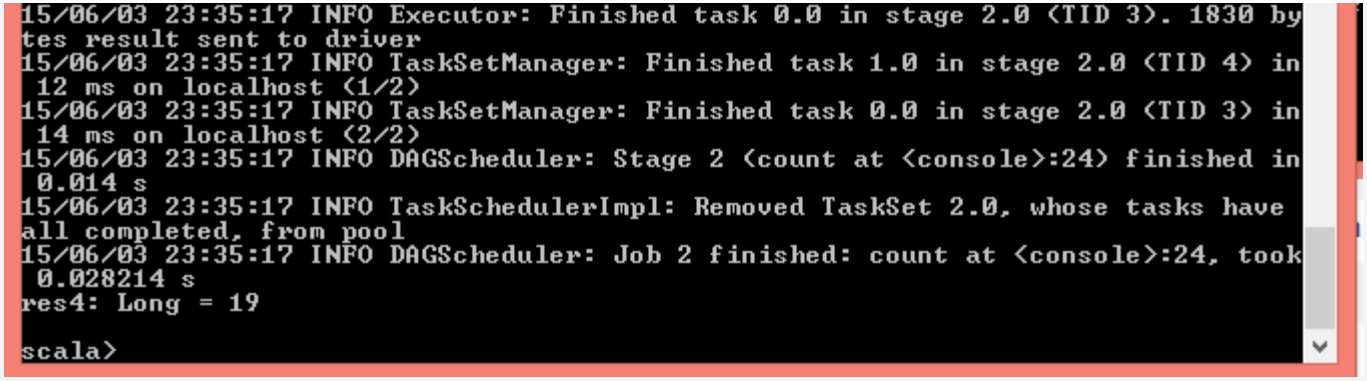
```
.md:0+1814
15/06/03 23:34:06 INFO Executor: Finished task 0.0 in stage 1.0 (TID 2). 1809 by
tes result sent to driver
15/06/03 23:34:06 INFO TaskSetManager: Finished task 0.0 in stage 1.0 (TID 2) in
 10 ms on localhost (1/1)
15/06/03 23:34:06 INFO DAGScheduler: Stage 1 (first at <console>:24) finished in
 0.011 s
15/06/03 23:34:06 INFO TaskSchedulerImpl: Removed TaskSet 1.0, whose tasks have
all completed, from pool
15/06/03 23:34:06 INFO DAGScheduler: Job 1 finished: first at <console>:24, took
 0.019875 s
res3: String = # Apache Spark

scala>
```

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.

```
scala> val linesWithSpark = textFile.filter(line => line.contains("Spark"))
```

We can chain together transformations and actions:

```
scala> textFile.filter(line => line.contains("Spark")).count() // How many lines contain "Spark"?
```



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

```
scala> textFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
```

Stop the spark process. Hints [sbin/stop-all.sh]

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

17. Spark Two Nodes Cluster Using Docker – 120 Minutes

Prerequisites:

- Install Docker
- Pull Centos Image (Preferable spark kafka Image)

Steps for deploying two node spark cluster in docker:

- Create two containers using the above image
  - Spark0
  - Spark1
- Create a network to connect between these two nodes.

  You should pull the following images from docker hub.

```
(base) Henrys-MacBook-Air:~ henrypotsangbam$ docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
spark-kafka         latest              a0c16a0aebc1        2 days ago          3.84GB
centos              7                   7e6257c9f8d8        4 weeks ago         203MB
(base) Henrys-MacBook-Air:~ henrypotsangbam$
```

Let us create a common network for our two nodes cluster.

#docker network create --driver bridge spark-net

Create first node, spark0 container

#docker run -it --name spark0 --privileged -p 8080:8080 -p 7077:7077 -p 4040:4040 -p 8081:8081 -p 8090:8090 centos:7 /usr/sbin/init

Docker command with Host folder mounted. (Skip)
docker run -it --name spark0 --hostname spark0 --privileged --network spark-net -v /Users/henrypotsangbam/Documents/Docker:/opt -p 8080:8080 -p 7077:7077 -p 4040:4040 -p 8081:8081 -p 8090:8090 centos:7 /usr/sbin/init

[Alternative Optional: Skip this step.

#docker create -it --name spark0 --privileged -p 8080:8080 -p 7077:7077 -p 4040:4040 -p 8081:8081 -p 8090:8090 --network spark-net --entrypoint /usr/sbin/init centos:7

or use tag - spark-kafka

]

To connect a **running** container to an existing user-defined bridge, use the docker network connect command. If the spark0 is already started without the network being attached execute the following else skip the following step.

```
$ docker network connect spark-net spark0
```

Start the first container:

#docker start spark0

Connect to you first node:

docker exec -it spark0 /usr/bin/bash

Start the second node and perform the installation as specify in the first lab.

#docker run -dit --name spark1 -p 8082:8081 -p 4041:4040 --network spark-net --entrypoint /bin/bash centos:7

or

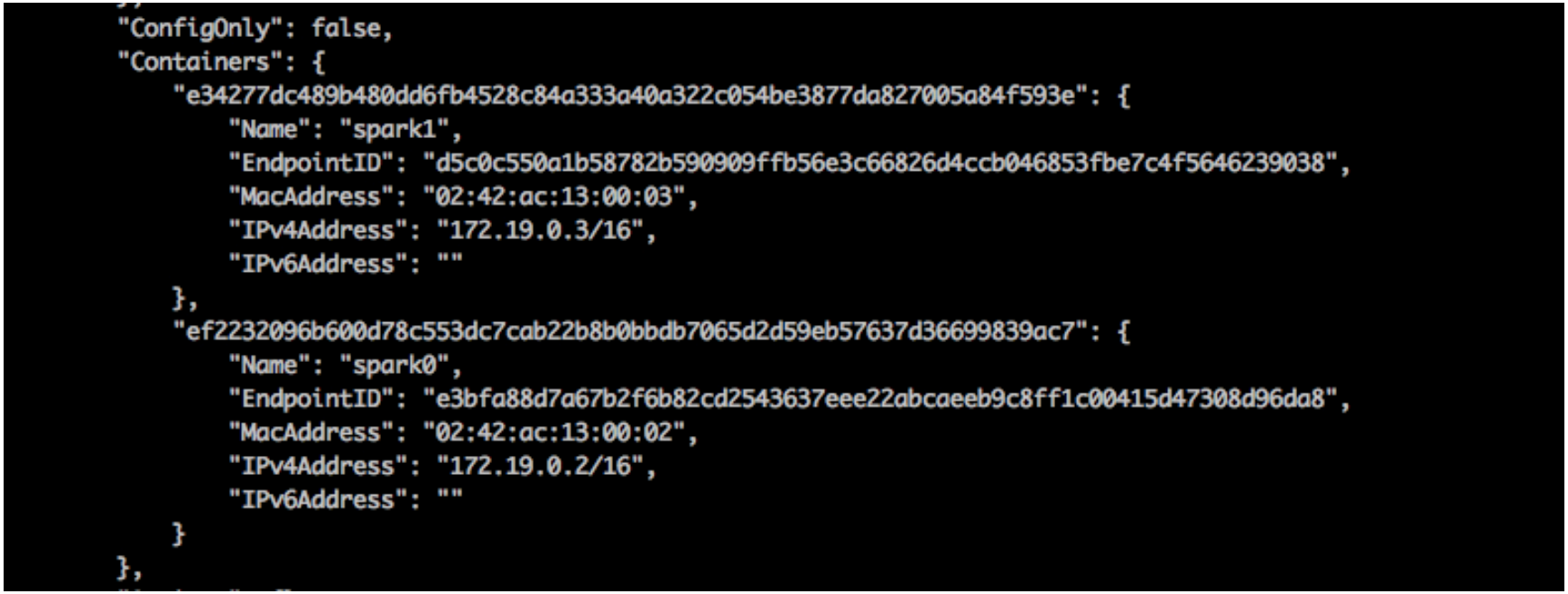
Docker command with Host folder mounted. (Skip)

docker run -it --name spark1 --hostname spark1 --privileged --network spark-net -v /Users/henrypotsangbam/Documents/Docker:/opt -p 4041:4040 -p 8082:8081 centos:7 /usr/sbin/init

Confirm the Network status:

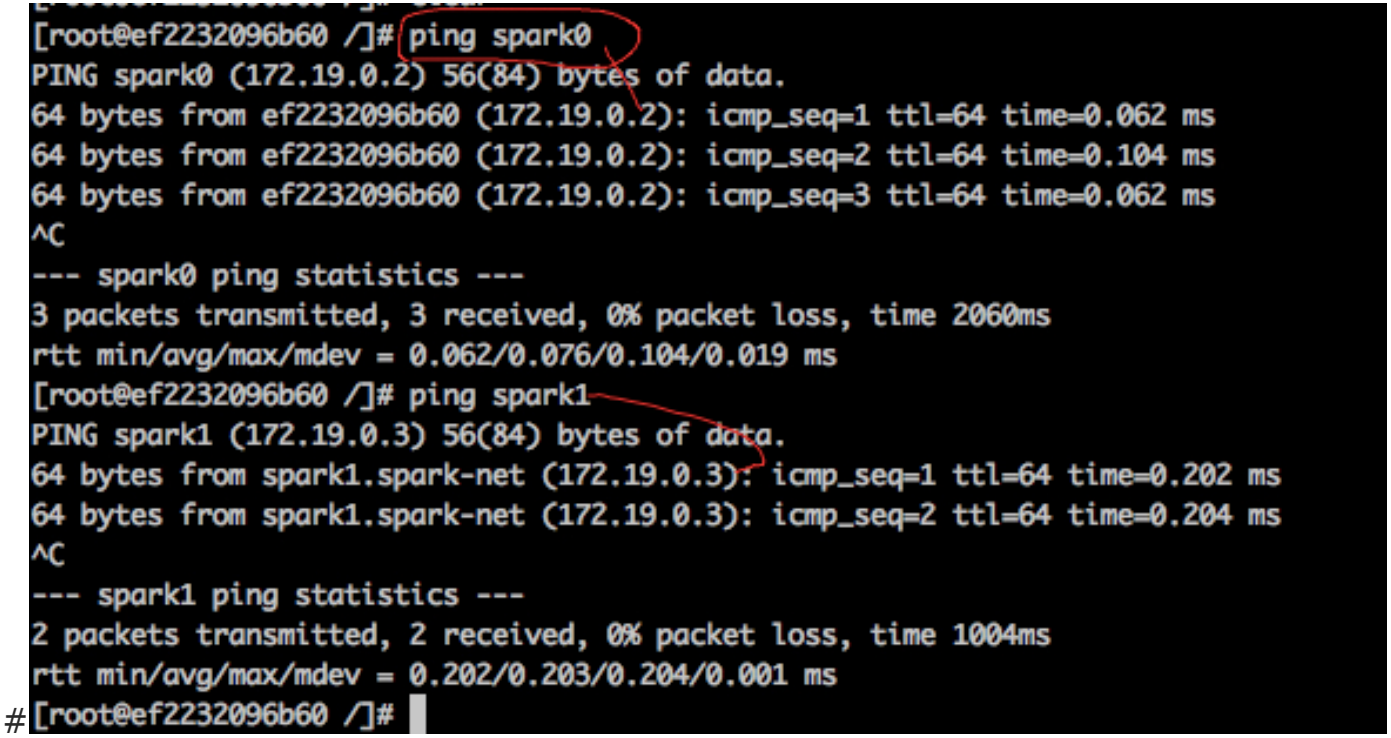
Inspect the network and find the IP addresses of the two containers

```
#docker network inspect spark-net
```

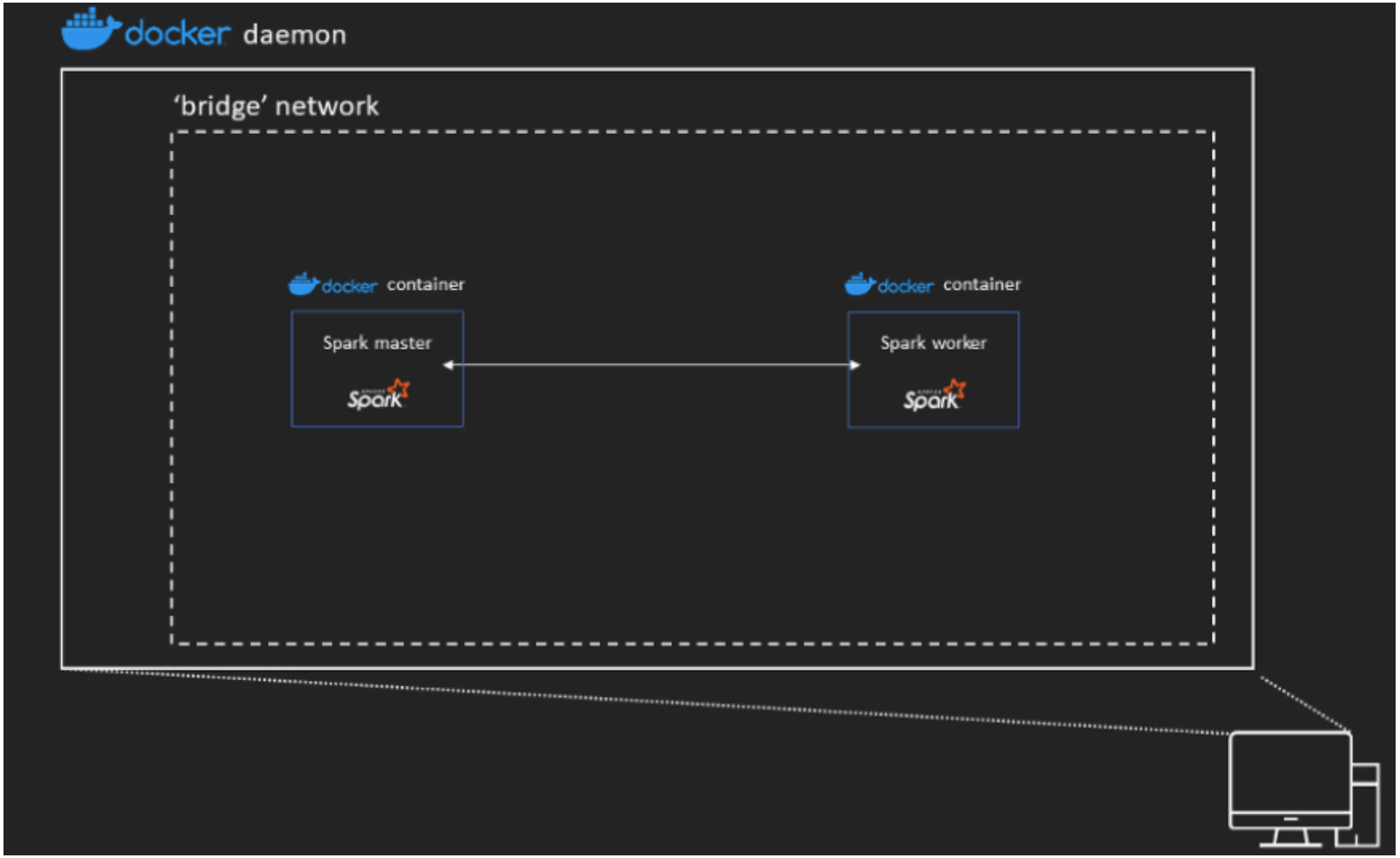


Ensure to substitute IPs accordingly.

Attach to the spark-master container and test its communication to the spark-worker container using both it's IP address and then using its container name.
ping spark0


[root@ef2232096b60 /]# ping spark0
PING spark0 (172.19.0.2) 56(84) bytes of data.
64 bytes from ef2232096b60 (172.19.0.2): icmp_seq=1 ttl=64 time=0.062 ms
64 bytes from ef2232096b60 (172.19.0.2): icmp_seq=2 ttl=64 time=0.104 ms
64 bytes from ef2232096b60 (172.19.0.2): icmp_seq=3 ttl=64 time=0.062 ms
^C
--- spark0 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2060ms
rtt min/avg/max/mdev = 0.062/0.076/0.104/0.019 ms
[root@ef2232096b60 /]# ping spark1
PING spark1 (172.19.0.3) 56(84) bytes of data.
64 bytes from spark1.spark-net (172.19.0.3): icmp_seq=1 ttl=64 time=0.202 ms
64 bytes from spark1.spark-net (172.19.0.3): icmp_seq=2 ttl=64 time=0.204 ms
^C
--- spark1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1004ms
rtt min/avg/max/mdev = 0.202/0.203/0.204/0.001 ms
#[root@ef2232096b60 /]#

Architecture



master – sparko and Slave – spark1

Let us start the Spark master.

Install the spark. You can refer the first lab.

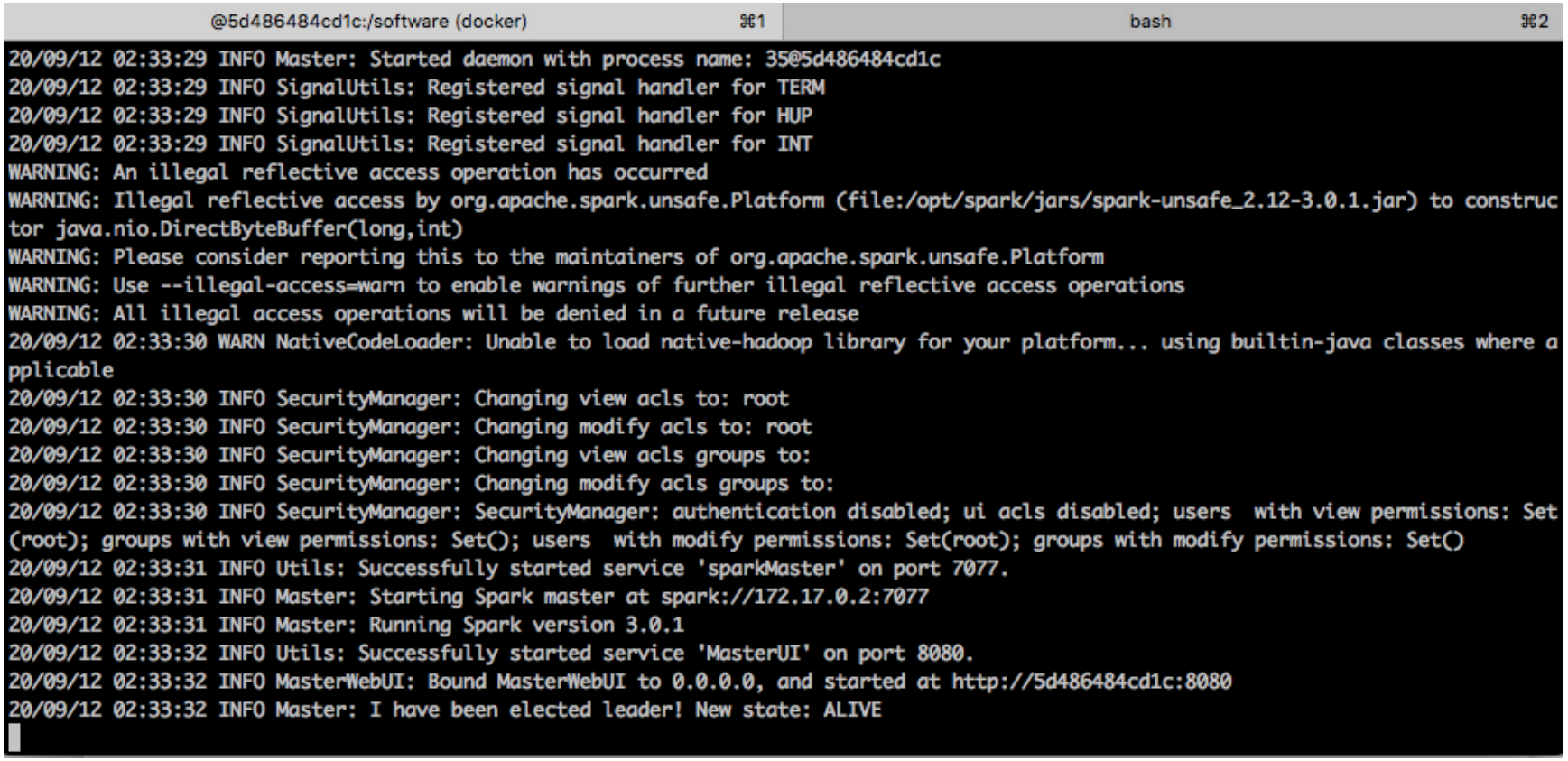
Execute the following in Sparko

Setup a Spark master node

Change the Master Port to 8090, there is issue when it is executed in 7077 in 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
```



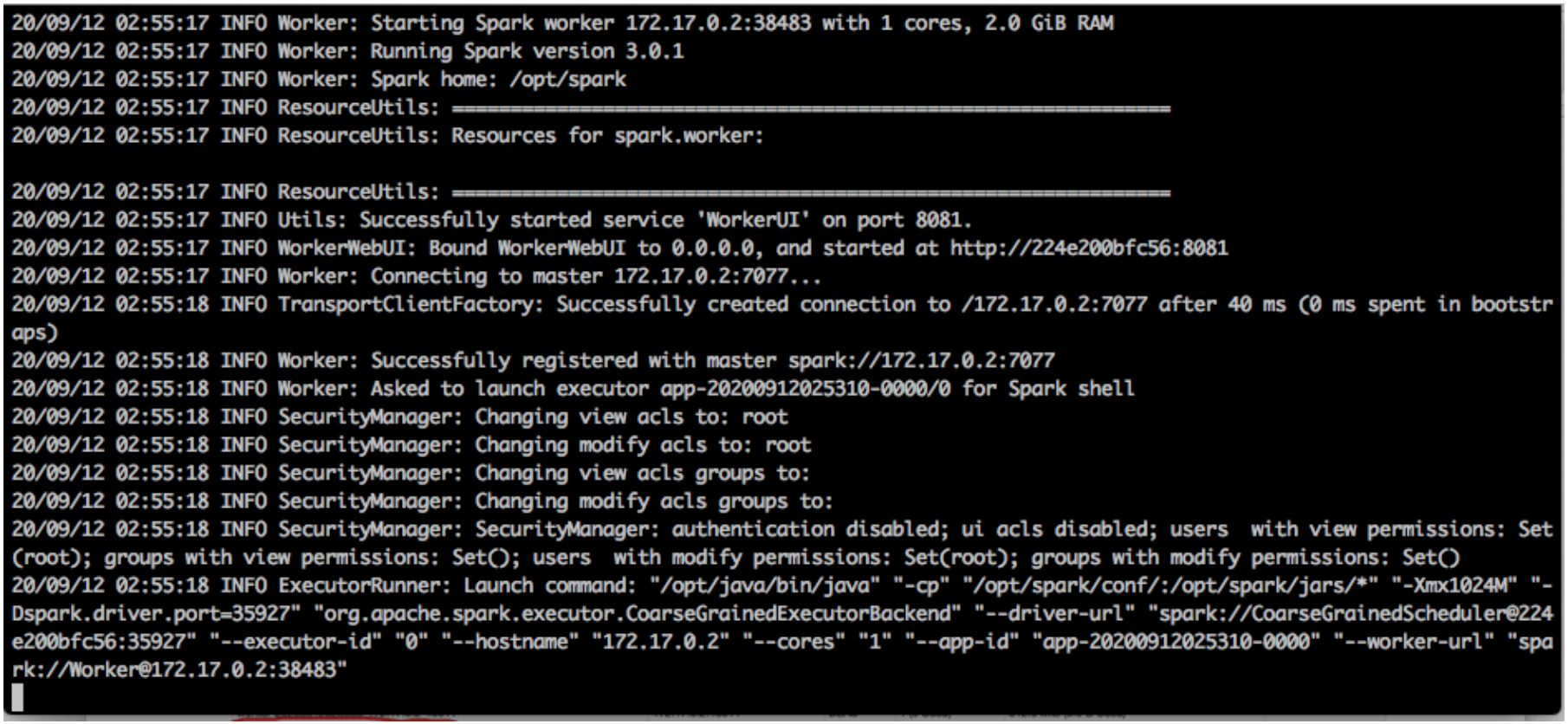
Start a worker process On Node/spark o: In a separate terminal.

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://172.17.0.2:8090
```

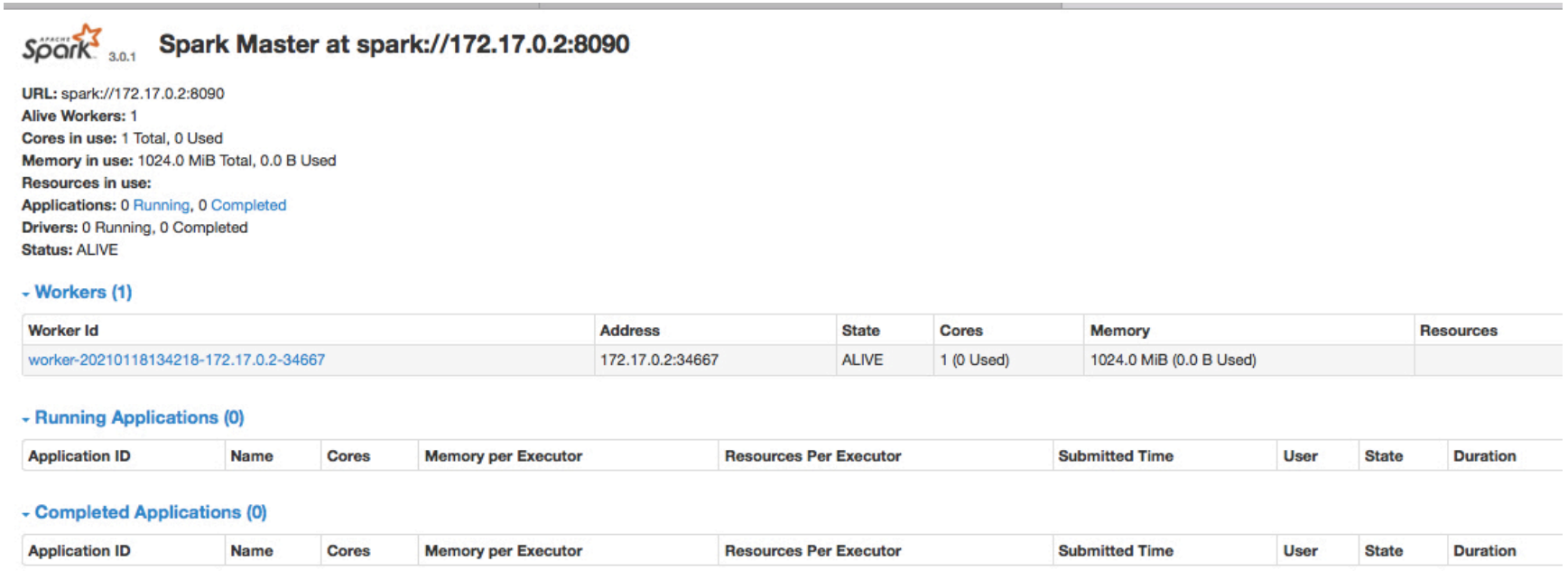
You need to replace with the IP of the sparko node. It can be retrieve using ifconfig command.



If unable to connect to localhost replace it with the container IP or the container alias i.e sparko.

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

http://127.0.0.1:8080

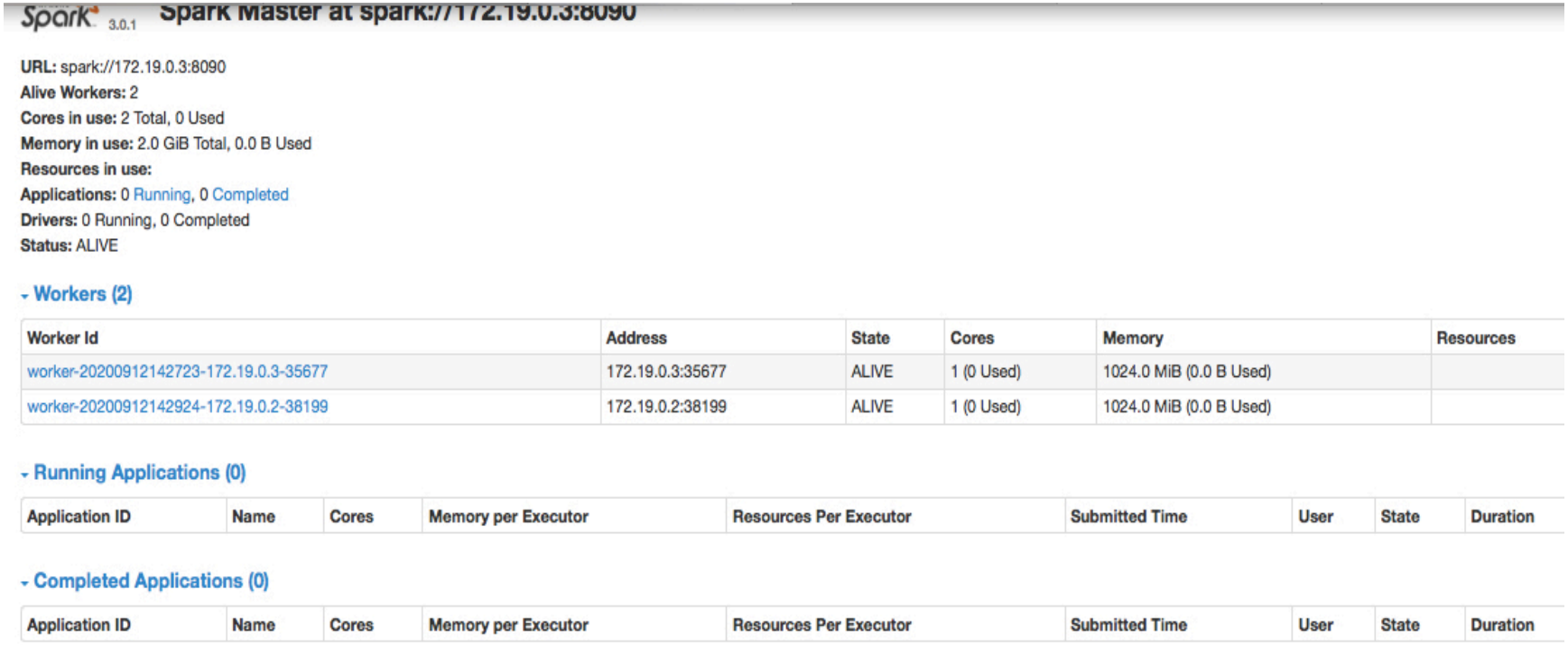


Start the Node on spark1
Ensure that you have install spark in the Spark1 too.

#docker exec -it spark1 bash

#export PATH=$PATH:/opt/spark/bin
spark-class org.apache.spark.deploy.worker.Worker -c 1 -m 1G spark://172.17.0.2:8090

At the end of this step, you should have 2 worker nodes as shown below:



Open a scala shell and connect to the Spark cluster

#SPARK_HOME/bin/spark-shell --conf spark.executor.memory=512mb --conf spark.executor.cores=1 --master spark://spark0:8090

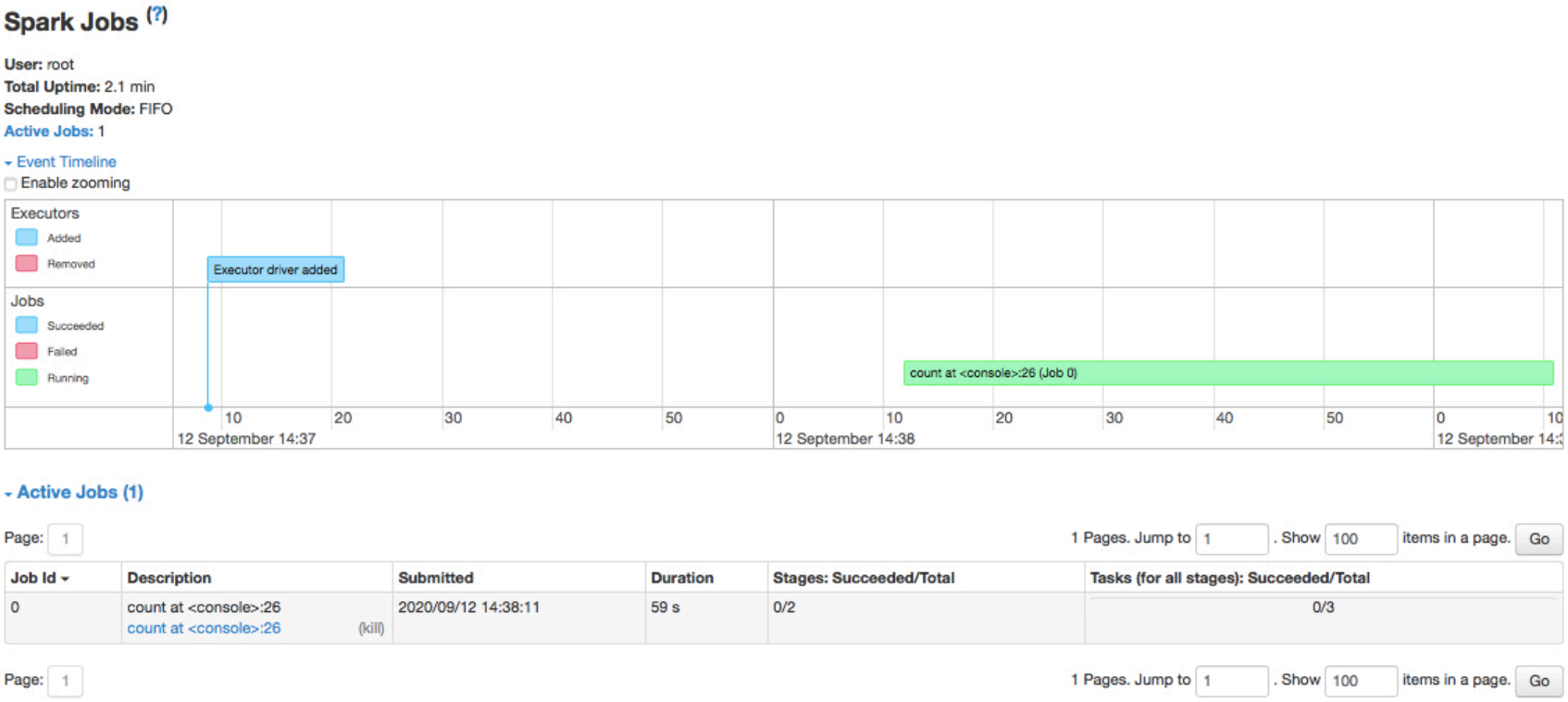
Run an example job in the interactive scala shell

val myRange = spark.range(100).toDF("number")

val divisBy2 = myRange.where("number % 2 = 0")

divisBy2.count()

Check the application UI by navigating to http://localhost:4040. You should see the following



The console should display the result as shown below:

```
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.0.1
      /_/

Using Scala version 2.12.10 (Java HotSpot(TM) 64-Bit Server VM, Java 14.0.2)
Type in expressions to have them evaluated.
Type :help for more information.

scala> val myRange = spark.range(100).toDF("number")
myRange: org.apache.spark.sql.DataFrame = [number: bigint]

scala> val divisBy2 = myRange.where("number % 2 = 0")
divisBy2: org.apache.spark.sql.Dataset[org.apache.spark.sql.Row] = [number: bigint]

scala> divisBy2.count()
res0: Long = 50

scala>
```

Hints: (For submitting job in cluster)

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

Copy the names.json in /software folder

export PATH=$PATH:/opt/spark/bin

#spark-submit --class "NameList" --master spark://spark0:8090 simple-project_2.12-1.0.jar names.json mynames --conf spark.executor.memory=256mb

Verify the output.

#ls /software/mynames

_temporary part-00000-ba4c474e-4f80-4824-83e7-4dc8d47cc79b-c000.csv

#more part-00000-ba4c474e-4f80-4824-83e7-4dc8d47cc79b-c000.csv

firstName,lastName
Grace,Hopper
Alan,Turing
Ada,Lovelace
Charles,Babbage
Optional only for Java.

#spark-submit   --class "com.ostech.spark.SimpleApp"   --master spark://spark0:8090 LearningSpark-0.0.1-SNAPSHOT.jar --conf spark.executor.memory=256mb

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

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

18. Launching on a Cluster: Hadoop YARN – 120 Minutes

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

Configure and install YARN.
One Node will be exclusively running YARN,
Options:

- Copy a VM and rename to HadoopVM-YARN.
- Create another container

Install YARN.

By now, you should have two VM on your workstation or Two container:

For Me, the first VM hostname is hp.com and the second one is ht.com. Ensure to follow the same nomencleature to avoid confusion. Its very important.

Host/VM	Container	Remarks
hp.com	Hadoop0	YARN Services
ht.com	Spark0	Spark Client or Spark.

Using Docker:

```
#docker run -it --name hadoop0 --privileged -p 8088:8088 -p 9870:9870 -p 9864:9864 -p 8032:8032 -p 8188:8188 -p 8020:8020 --hostname hadoop0  centos:7  /usr/sbin/init
```

Verify the hostname and the /etc/hosts. You need to enter each IP and host name as shown above. Update details accordingly in your local machine.

```
[root@hp ~]# hostname
hp.com
[root@hp ~]# more /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
#127.0.0.1               hp.com localhost
#::1                     hp.com localhost6

192.168.188.134 ht.com
192.168.188.136 hp.com localhost
[root@hp ~]#
```

You should see in your window as follows for ht.com. Changes the IP as your system.

```
[root@ht ~]# bash
[root@ht ~]# hostname
ht.com
[root@ht ~]# more /etc/hosts
# Do not remove the following line, or various programs
# that require network functionality will fail.
#127.0.0.1               localhost.localdomain localhost
#::1             localhost6.localdomain6 localhost6
192.168.188.134 ht.com localhost
192.168.188.136 hp.com
[root@ht ~]#
```

All the below commands should be executed on hp.com unless specify. We are configuring YARN cluster now.

Change directory to the location where you have the software.

Download Location: (hadoop-3.2.2.tar.gz)
https://hadoop.apache.org/releases.html

Untar as follows:
tar -xvf hadoop-X.tar.gz -C /opt
tar -xvf jdk-8u40-linux-i586.tar.gz -C /opt

Rename the folder:

#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

Unpack the downloaded Hadoop distribution. In the distribution, edit the file /opt/hadoop/`etc/hadoop/hadoop-env.sh` to define some parameters as follows:

```
# set to the root of your Java installation

export JAVA_HOME=/opt/jdk
```

cd /opt/hadoop

Try the following command:

```
$ bin/hadoop
```

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

Use the following:

etc/hadoop/core-site.xml:

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

etc/hadoop/hdfs-site.xml:

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

Setup passphraseless ssh

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

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

Change the passwd using passwd command.

```
$ ssh localhost
```

If you cannot ssh to localhost 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
```

copy the key to the slaves node:

```
scp ~/.ssh/id_rsa.pub master1:~/.ssh/master.pub
```

```
[root@master0 hadoop]# scp ~/.ssh/id_rsa.pub master1:~/.ssh/master.pub
Warning: the ECDSA host key for 'master1' differs from the key for the IP address '172.18.0.3'
Offending key for IP in /root/.ssh/known_hosts:4
Matching host key in /root/.ssh/known_hosts:7
Are you sure you want to continue connecting (yes/no)? yes
id_rsa.pub                                                    100%  401   324.3KB/s   00:00
[root@master0 hadoop]#
```

Log on the slave node. Adding the master public key as an authroized key in the slave node.

```
[root@master1 ~]#
[root@master1 ~]#
[root@master1 ~]# cat ~/.ssh/master.pub >> ~/.ssh/authorized_keys
[root@master1 ~]#
```

Add the environment variable.

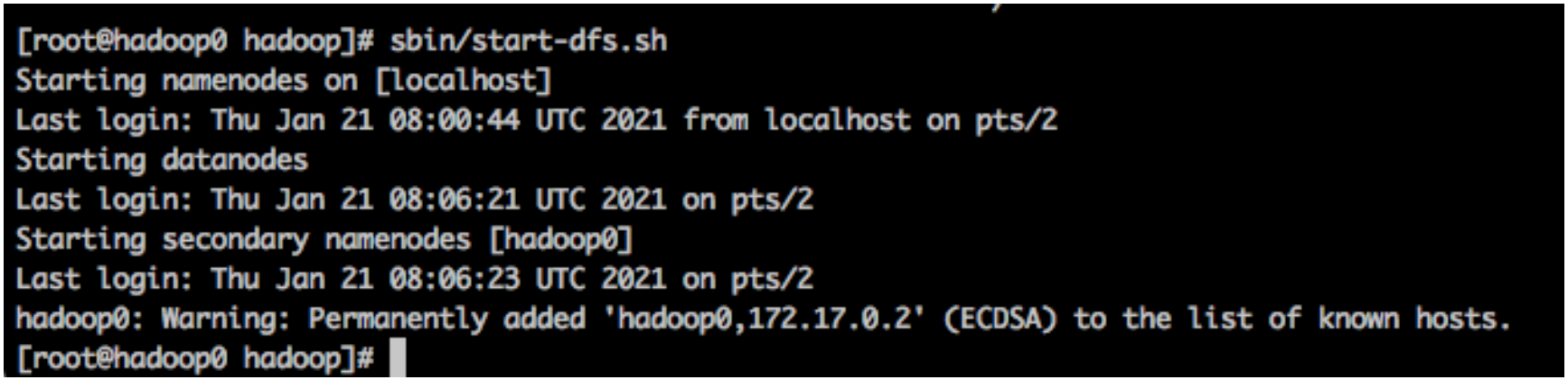
```
export HDFS_NAMENODE_USER="root"
export HDFS_DATANODE_USER="root"
export HDFS_SECONDARYNAMENODE_USER="root"
export YARN_RESOURCEMANAGER_USER="root"
export YARN_NODEMANAGER_USER="root"
```

Format the filesystem:

```
$ bin/hdfs namenode -format
```

Start NameNode daemon and DataNode daemon:

```
$ sbin/start-dfs.sh
```



1. Browse the web interface for the NameNode; by default it is available at:
   - NameNode - `http://localhost:9870/`
2. Make the HDFS directories required to execute MapReduce jobs:

```
3.   $ bin/hdfs dfs -mkdir /user
```

```
$ bin/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: hp.com

1. Configure parameters as follows:

   `etc/hadoop/mapred-site.xml`:

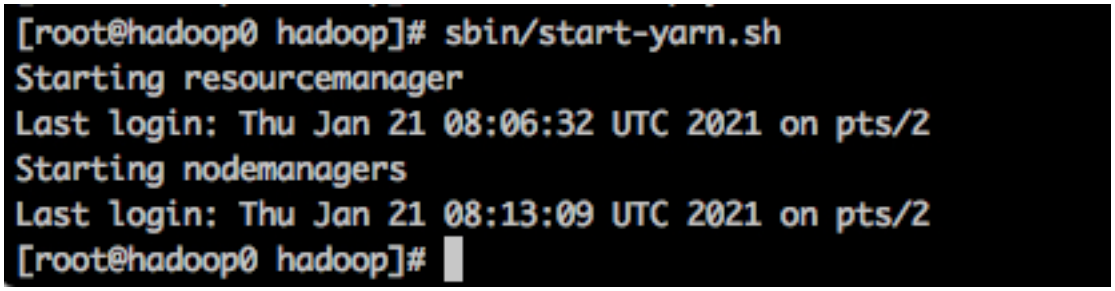
```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.application.classpath</name>

<value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*
</value>
    </property>
</configuration>
```

etc/hadoop/yarn-site.xml:

```
<configuration>
<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,HAD
OOP_YARN_HOME,HADOOP_MAPRED_HOME</value>
    </property>
</configuration>
```

2. Start ResourceManager daemon and NodeManager daemon:

```
$ sbin/start-yarn.sh
```



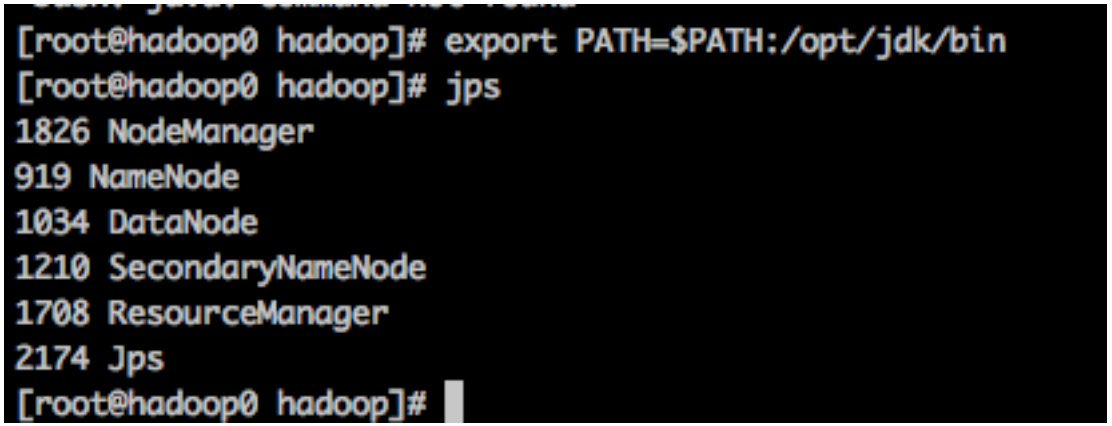
3. Browse the web interface for the ResourceManager; by default it is available at:

   - ResourceManager - `http://localhost:8088/`

4. When you're done, you can stop the daemons with: Optional.

```
$ sbin/stop-yarn.sh
```

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



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.

```
hadoop fs -mkdir /tmp
hadoop fs -chmod -R 1777 /tmp
hadoop fs -mkdir /tmp/in
hadoop fs -ls /tmp
```

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

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

```
#hadoop fs -mkdir  /tmp/spark-events
```

Congrats! You have successfully configure Yarn Cluster.

If you are using docker, join the hadoop0 and spark0 containers in a single network:

```
#docker network connect spark-net hadoop0
```

Verify it:
#docker network inspect spark-net


```
        "Containers": {
            "303b681436449f211ee3f828ec09af683947652bd1db52c68288e6ac75d71e04": {
                "Name": "spark0",
                "EndpointID": "56cfeed9c11107e578a688ddab77af96e72c1063ebd2631bd801629f328b17e2",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "c88c1c8cb01a6aedc1832d1bf3d3cb22dec209f45a5749ace85a66d1d1d0cf12": {
                "Name": "hadoop0",
                "EndpointID": "3c030821125613a7bf5c1cea99f510144a01220b9cc38391bbeb71274076c824",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        },
        "Options": {},
        "Labels": {}
    }
]
```

Start the Spark VM/spark container, if it's not yet started, i.e ht.com and issue the following command. Logon to the machine using telnet. You can use the VM, share folder options to copy the file to the VM from your workstation or else copy using the mouse from your workstation to the folder specify.

Let us copy the program, simple-project_2.12-1.0.jar to the spark nodes. It was developed earlier.

Let us configure hadoop client setting on the Spark VM --> ht.com or spark0 node.

Compress the Hadoop folder on the Hadoop0 instances or Hadoop node.
#cd /opt
#tar -cvzf hadoop.tar hadoop/

Copy the compressed hadoop folder to the spark0 or spark node and uncompressed in /opt folder.
[Docker:
copy from hadoop to host : docker cp hadoop0:/opt/hadoop.tar .
copy from host to spark : docker cp hadoop.tar spark0:/opt/]

#tar -xvf hadoop.tar

Update yarn-site.xml on the Spark Node. – Replace the RM hostname with that of your server.

```
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>master0</value>
      <description>The hostname of the RM.</description>
   </property>
<property>
      <name>yarn.resourcemanager.address</name>
      <value>master0:8032</value>
      <description>The hostname of the RM.</description>
   </property>
```

Create the following file in /opt on hadoop node and copy the file in to hdfs /user/root/input folder.
File names.json contains:

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

```
#hdfs dfs -mkdir /user/root/input
# hdfs dfs -copyFromLocal /opt/names.json  /user/root/input/
```

You can verify the file using the following command:

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

Open a command console to execute the Spark jobs to submit on YARN.
Export Hadoop and Yarn environment variable pointing to the Spark Node – Hadoop conf file.

```
#cd /opt/spark/bin
#export HADOOP_CONF_DIR=/opt/hadoop-spark/etc/hadoop
#export YARN_CONF_DIR=/opt/hadoop-spark/etc/hadoop
#./spark-submit --class "NameList"  --master yarn --deploy-mode cluster
/opt/data/simple-project_2.12-1.0.jar \
  hdfs://hadoop0:8020/user/root/input1 hdfs://hadoop0:8020/user/root/output4
```

or

```
./spark-submit --class "NameList"  --master yarn --driver-memory 1G --executor-memory 1G --num-executors 2 --deploy-mode cluster  /opt/data/simple-project_2.12-1.0.jar \
  hdfs://hadoop0:8020/user/root/input1 hdfs://hadoop0:8020/user/root/output7

./spark-submit --class "NameList"  --master yarn --driver-memory 512m --executor-memory 512m --num-executors 2 --deploy-mode cluster  /opt/data/simple-project_2.12-1.0.jar \
  hdfs://master0:8020/user/root/input hdfs://master0:8020/user/root/output7

./spark-submit --class "NameList"  --master yarn --driver-memory 512m --executor-memory 512m --num-executors 2 --deploy-mode client  /opt/data/simple-project_2.12-1.0.jar \
  hdfs://hadoop0:8020/user/root/input1 hdfs://hadoop0:8020/user/root/output7
```

or using spark-shell

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

./bin/spark-shell --master yarn --deploy-mode client

Note --> Application 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://hp.com:8088/cluster

Click on Cluster-- >Application --> accepted

Cluster Metrics

Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Re N
3	0	1	2	1	1 GB	8 GB	0 B	1	8	0	1	0	0	0	0

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Trackin
application_1434297324587_0003	root	com.ht.sparks.WordCount	SPARK	default	Sun, 14 Jun 2015 17:55:58 GMT	N/A	ACCEPTED	UNDEFINED		UNASSI
application_1434297324587_0002	root	com.ht.sparks.WordCount	SPARK	default	Sun, 14 Jun 2015 16:27:58	Sun, 14 Jun 2015 16:30:02	FINISHED	SUCCEEDED		History

Wait for few moment and verify on Running tab.

Cluster Metrics

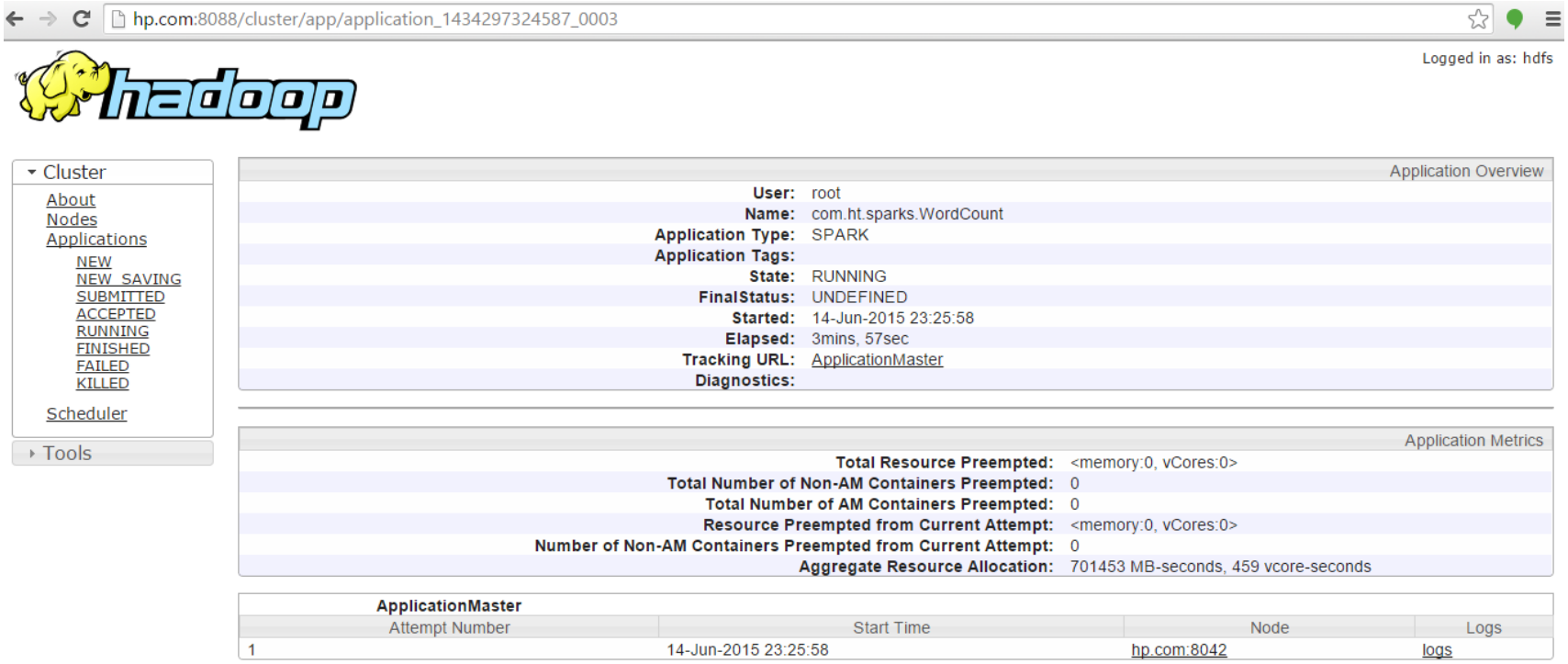
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Memory Used	Memory Total	Memory Reserved	VCores Used	VCores Total	VCores Reserved	Active Nodes	Decommissioned Nodes	Lost Nodes	Unhealthy Nodes	Rebo Noc
3	0	1	2	4	7 GB	8 GB	0 B	4	8	0	1	0	0	0	0

Show 20 entries Search:

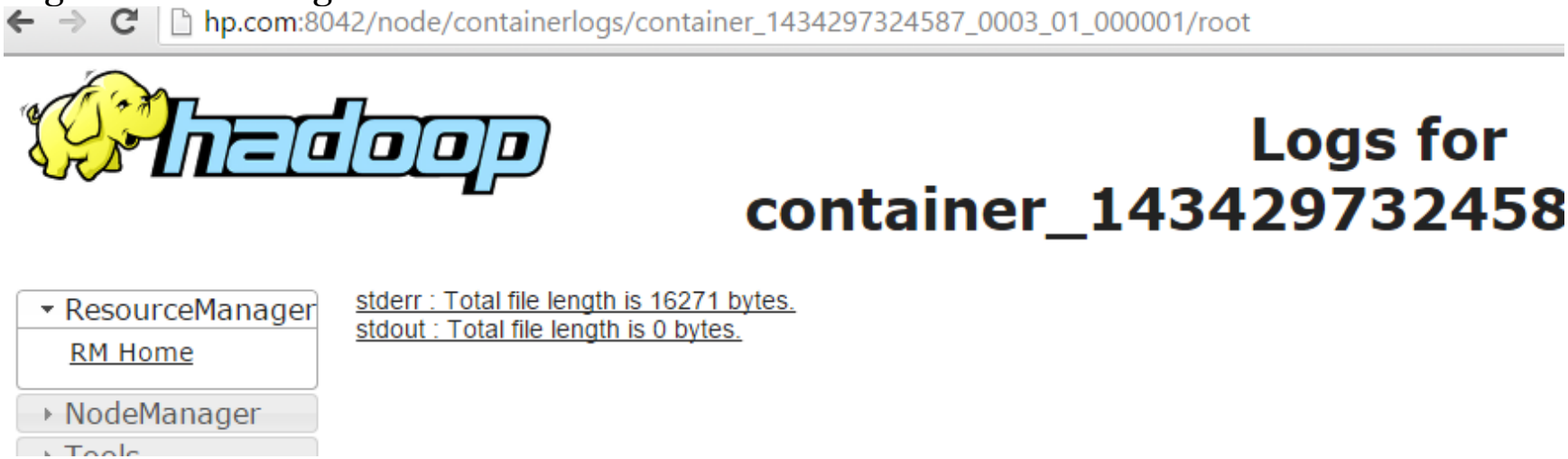
ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1434297324587_0003	root	com.ht.sparks.WordCount	SPARK	default	Sun, 14 Jun 2015 17:55:58 GMT	N/A	RUNNING	UNDEFINED		ApplicationMa

Showing 1 to 1 of 1 entries First Previous 1 Next L

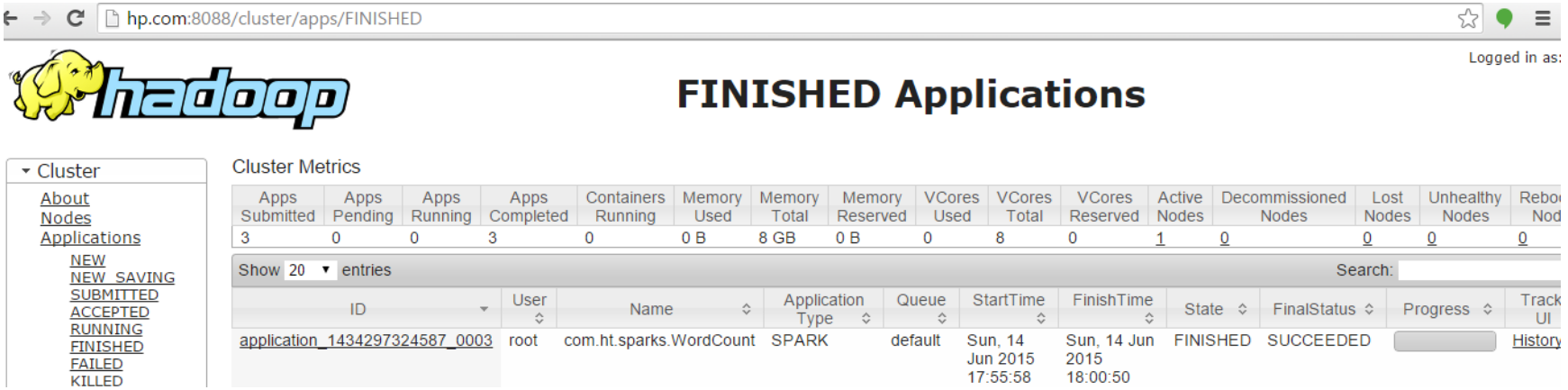
Click on the Application ID --> Logs



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]#
```

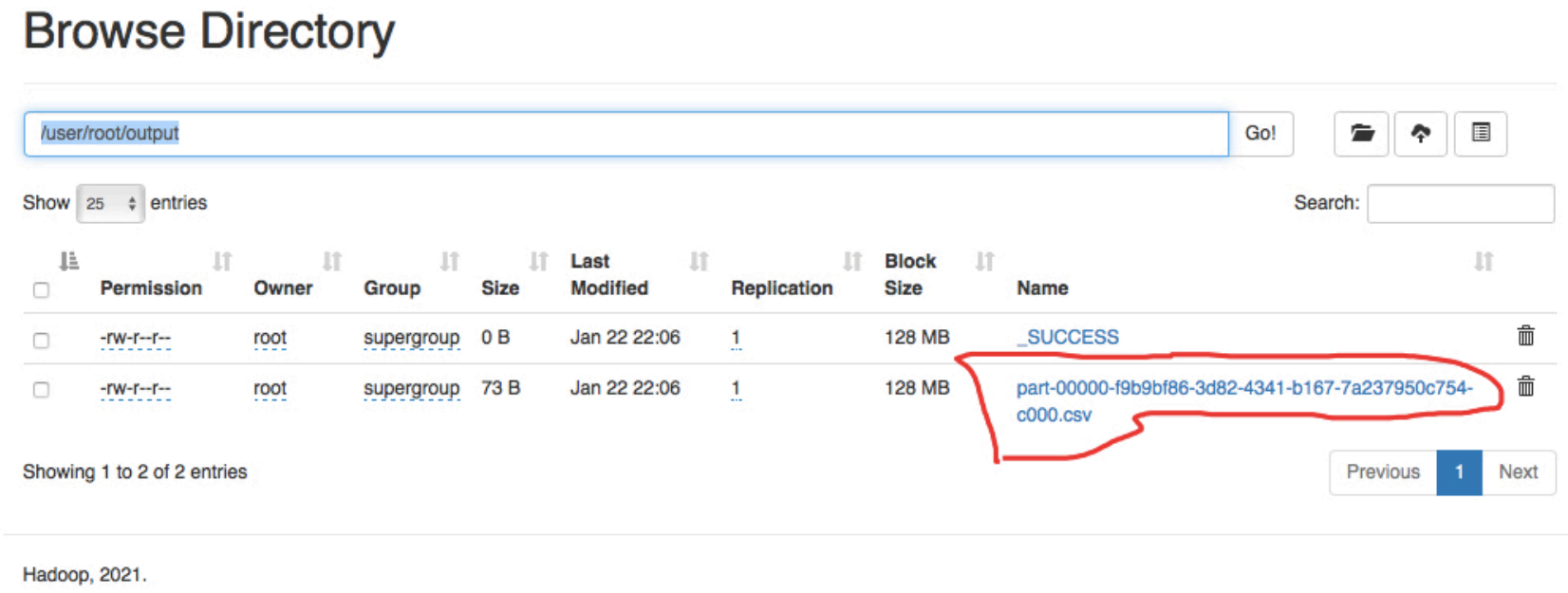
You can verify the output file in the hdfs as shown below:
#hdfs dfs -ls -R /user/root/output
#hdfs dfs -cat /user/root/output/part-00000-f9b9bf86-3d82-4341-b167-7a237950c754-c000.csv

```
[root@hadoop0 opt]# hdfs dfs -ls -R /user/root/output
-rw-r--r--   1 root supergroup          0 2021-01-22 16:36 /user/root/output/_SUCCESS
-rw-r--r--   1 root supergroup         73 2021-01-22 16:36 /user/root/output/part-00000-f9b9bf86-3d82-4341-b167-7a2
37950c754-c000.csv
[root@hadoop0 opt]# hdfs dfs -cat /user/root/output/part-00000-f9b9bf86-3d82-4341-b167-7a237950c754-c000.csv
firstName,lastName
Grace,Hopper
Alan,Turing
Ada,Lovelace
Charles,Babbage
[root@hadoop0 opt]#
```

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.

Browse Directory

/tmp/out Go!

Permission	Owner	Group	Size	Replication	Block Size	Name
-rw-r--r--	root	supergroup	0 B	3	128 MB	_SUCCESS
-rw-r--r--	root	supergroup	60.71 KB	3	128 MB	part-00000
-rw-r--r--	root	supergroup	61.17 KB	3	128 MB	part-00001
-rw-r--r--	root	supergroup	60.76 KB	3	128 MB	part-00002
-rw-r--r--	root	supergroup	57.2 KB	3	128 MB	part-00003

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

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9---
1  (Kakrafoon,,1)
2  (pitifully,1)
3  (mattered,2)
4  (propped,1)
5  (House,2)
6  (bone,8)
7  ("Conceited,1)
8  (afternoon's,1)
9  (screen.",7)
10 (five.,1)
11 (gaping,2)
12 (tents,1)
13 (2~Imports:,1)
14 (envelope,1)
15 (tone.,4)
16 (consideration,,1)
17 (conclusively,3)
18 (activation.,1)
```

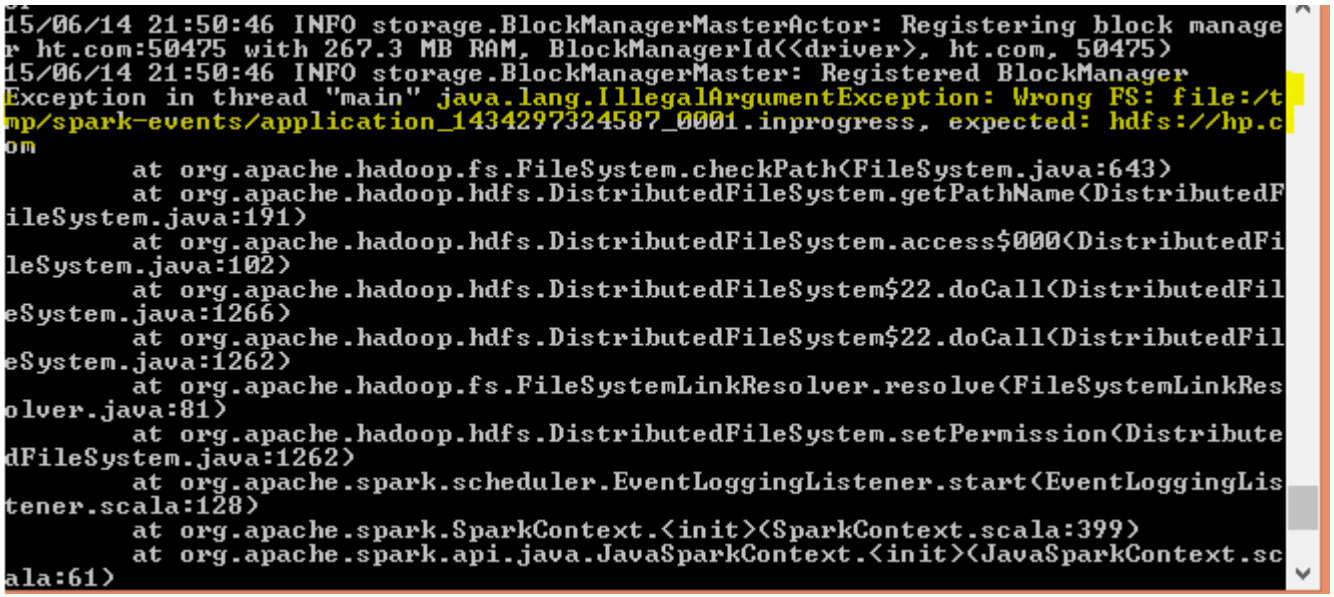
Congrats!.

---------- 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)

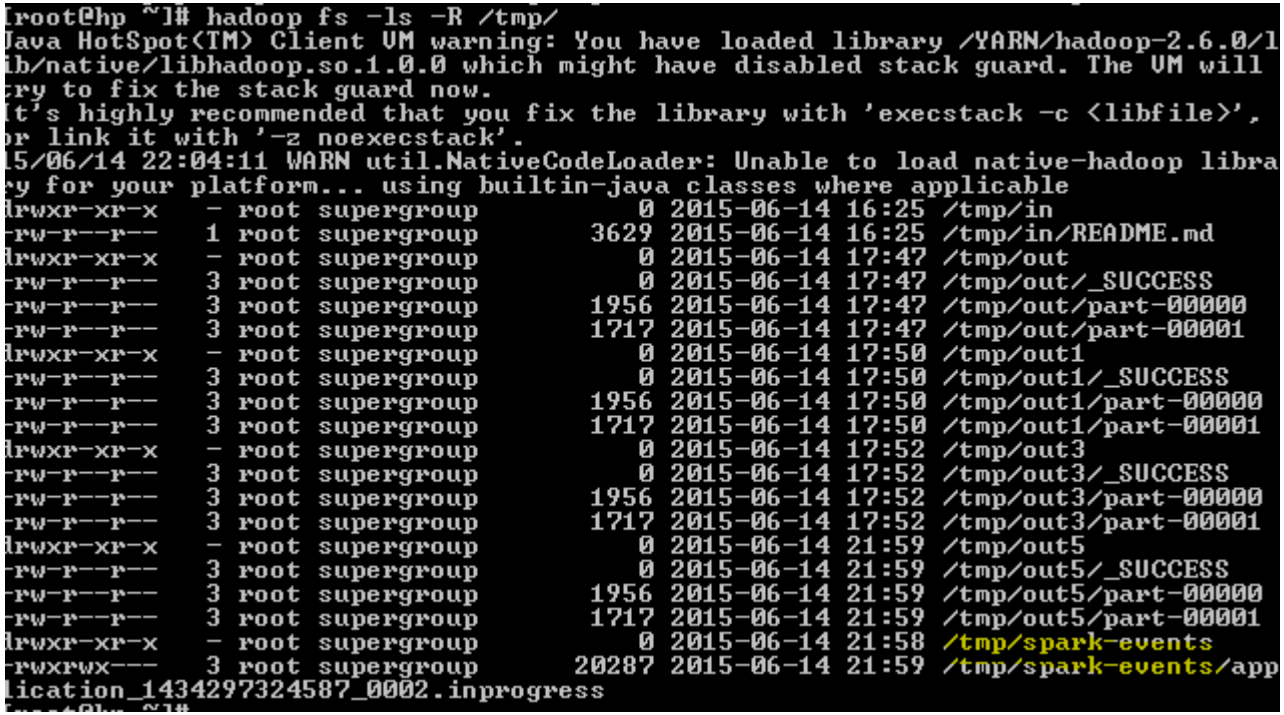
at org.apache.spark.scheduler.EventLoggingListener.start(EventLoggingListener.scala:128)



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/



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>
</property>
<property>
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>512</value>
</property>
<property>
<name>yarn.scheduler.increment-allocation-mb</name>
<value>256</value>
</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 is 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.

```
[root@hp ~]# jps
5082 DataNode
9633 CoarseGrainedExecutorBackend
9670 Jps
4980 NameNode
5345 NodeManager
9625 CoarseGrainedExecutorBackend
5029 SecondaryNameNode
9420 ApplicationMaster
5294 ResourceManager
9629 CoarseGrainedExecutorBackend
[root@hp ~]#
```

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>
```

Error:
Failing this attempt.Diagnostics: [2022-06-18 03:01:08.210]Container [pid=1686,containerID=container_1655521013018_0001_02_000001] is running 32197120B beyond the 'VIRTUAL' memory limit. Current usage: 198.8 MB of 1 GB physical memory used; 2.1 GB of 2.1 GB virtual memory used. Killing container.

Add following property in yarn-site.xml

```
 <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 for containers</description>
  </property>
```

Complete yarn-site.xml

```
<configuration>

<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
_M
APRED_HOME</value>
</property>
<property>
      <name>yarn.resourcemanager.hostname</name>
      <value>master0</value>
      <description>The hostname of the RM.</description>
   </property>
<property>
      <name>yarn.resourcemanager.address</name>
      <value>master0:8032</value>
      <description>The hostname of the RM.</description>
```

```
   </property>
<property>
<name>yarn.nodemanager.resource.memory-mb</name>
<value>4096</value>
</property>
<property>
<name>yarn.scheduler.minimum-allocation-mb</name>
<value>512</value>
</property>
<property>
<name>yarn.scheduler.increment-allocation-mb</name>
<value>256</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 for containers</description>
 </property>
```

```
</configuration>
```

---

19. Jobs Monitoring : Using Web UI. – 45 Minutes

Execute the Job as specified in the Spark standalone cluster lab/ Using Docker before proceeding ahead.

Copy the input file in both the nodes, if you are executing on the Standalone cluster.

Or start the Cluster and execute the following jobs:

The file should be in the Executing node.
File names.json contains:

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

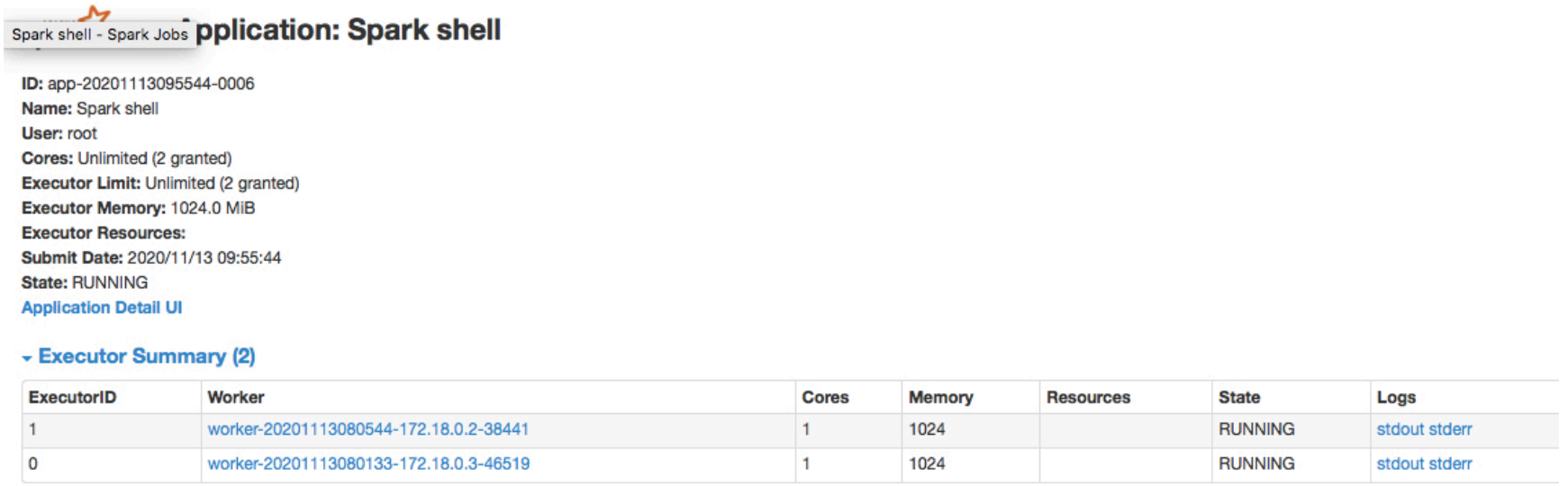
Enter the following commands one at a time.

```
#spark-shell --master spark://spark0:8090
val ip="/software/names.json"
val op="/software/nameop"
spark.sparkContext.setLogLevel("WARN")
val peopleDF = spark.read.json(ip)
val namesDF = peopleDF.select("firstName","lastName")
```

namesDF.write.option("header","true").csv(op)

http://127.0.0.1:8080

Click on Application ID → Application Detail UI



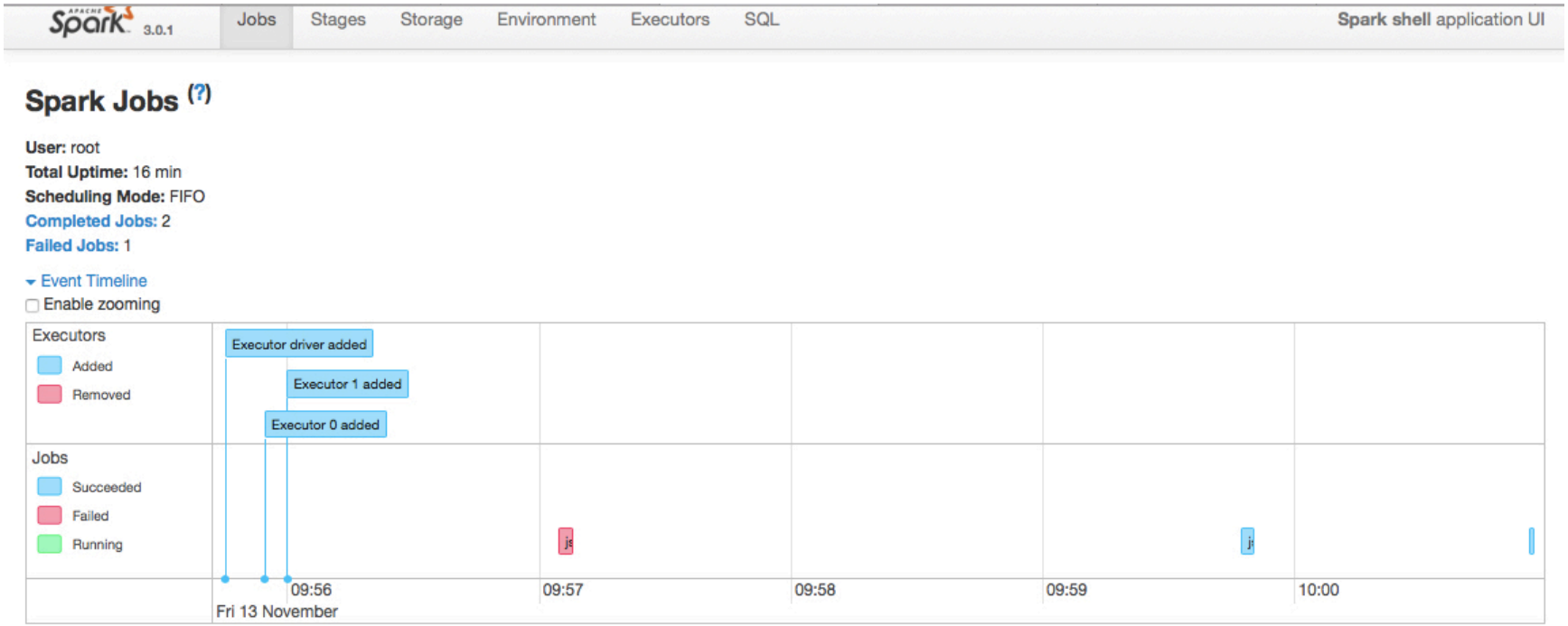
Or You can verify the Jobs details Using: http://master:4040/jobs/

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

The **Jobs Tab** shows status of all Spark jobs in a Spark application

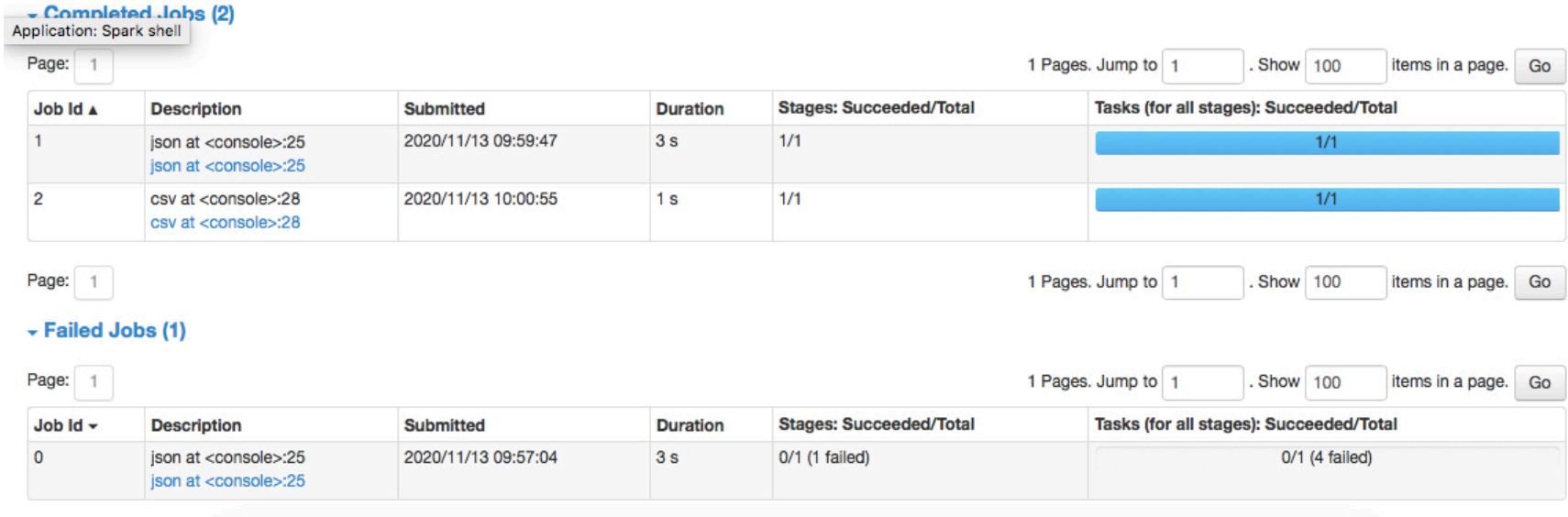


Display the timeline of Executor being added in the Job. When the Job Get completed etc. In above, you can verify that 2 executors have been added.

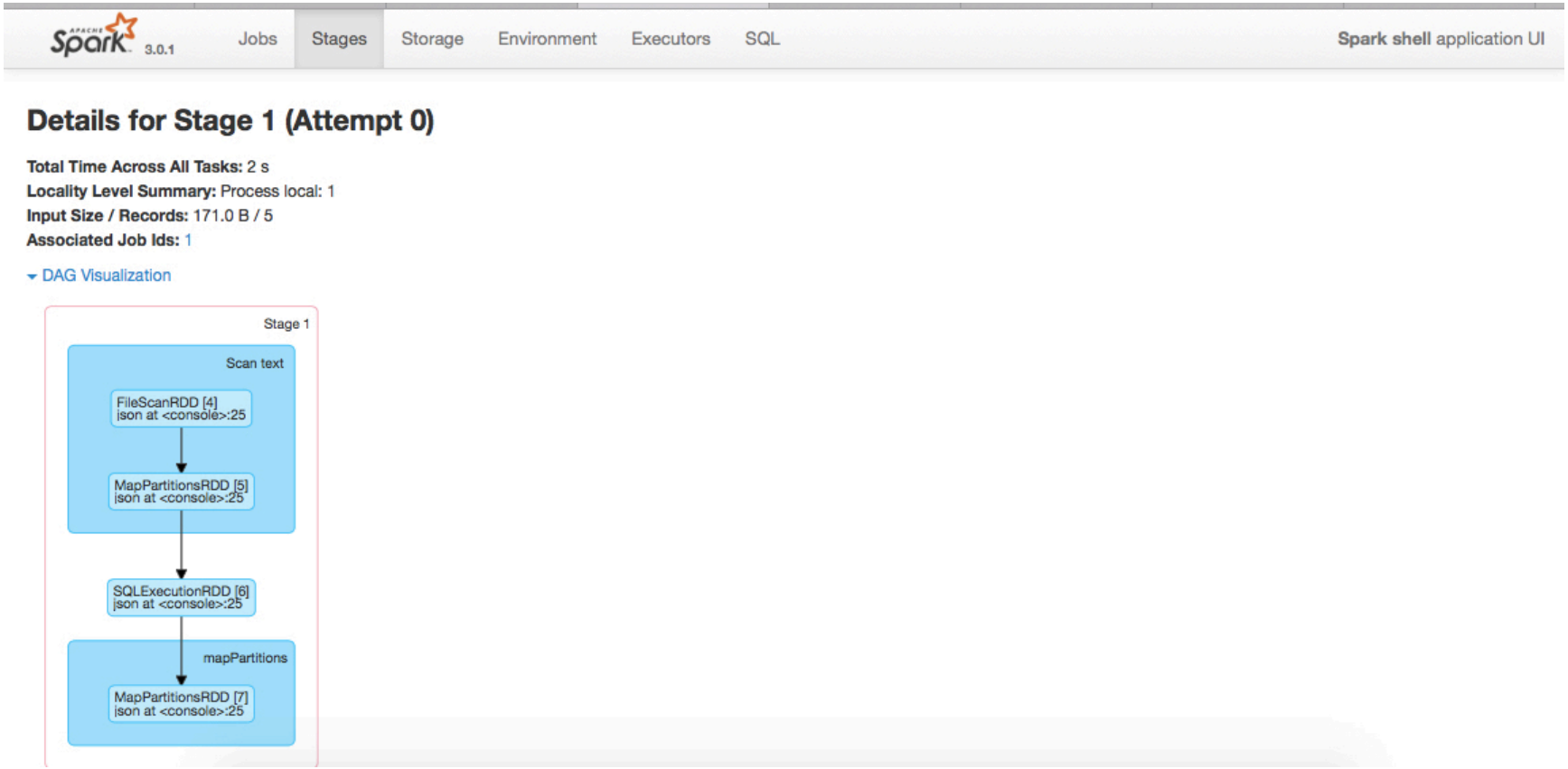
When you hover over a job in Event Timeline not only you see the job legend but also the job is highlighted in the Summary section.

The Event Timeline section shows not only jobs but also executors.

You can verify the completed and failed jobs too:

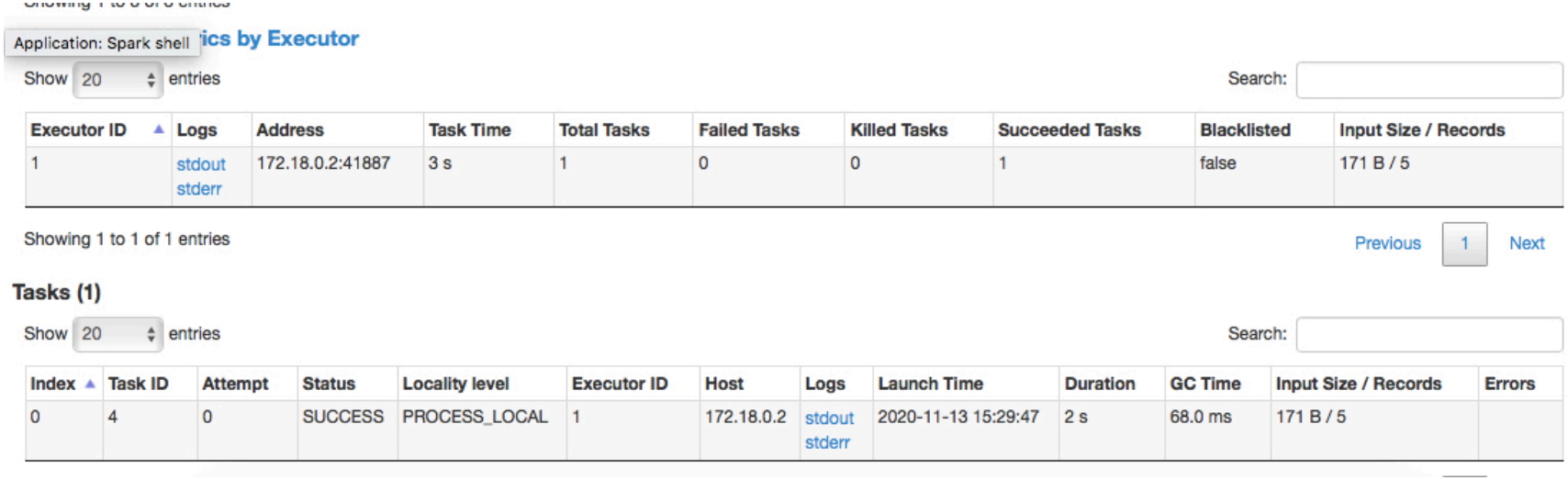


Click on On job -> for details on Description → stages to view DAG - When you click a job in All Jobs Page page, you see the **Details for Job** page.



It shows the DAG graph and the stage details.

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.



Executors Tab

Stats of each executor, how much time it spents on GC etc. How much data get shuffle?

Executors tab in web UI shows

Summary

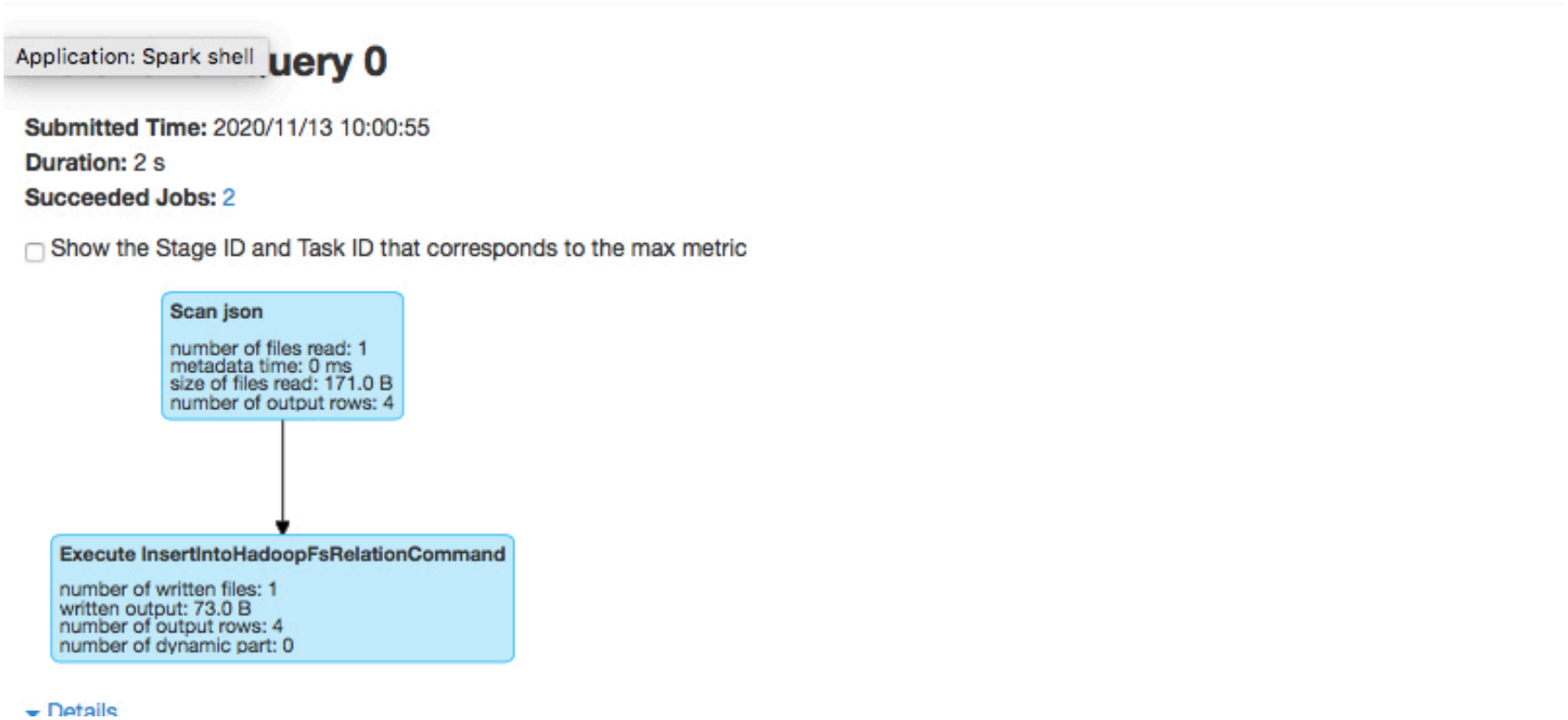
	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks	Task Time (GC Time)	Input	Shuffle Read	Shuffle Write
Active(3)	1	23.4 KB / 1.3 GB	0.0 B	2	0	0	0	0	0 ms (0 ms)	0.0 B	0.0 B	0.0 B
Dead(4)	2	46.8 KB / 1.7 GB	0.0 B	4	0	0	7	7	1.0 min (6 s)	21 KB	0.0 B	0.0 B
Total(7)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

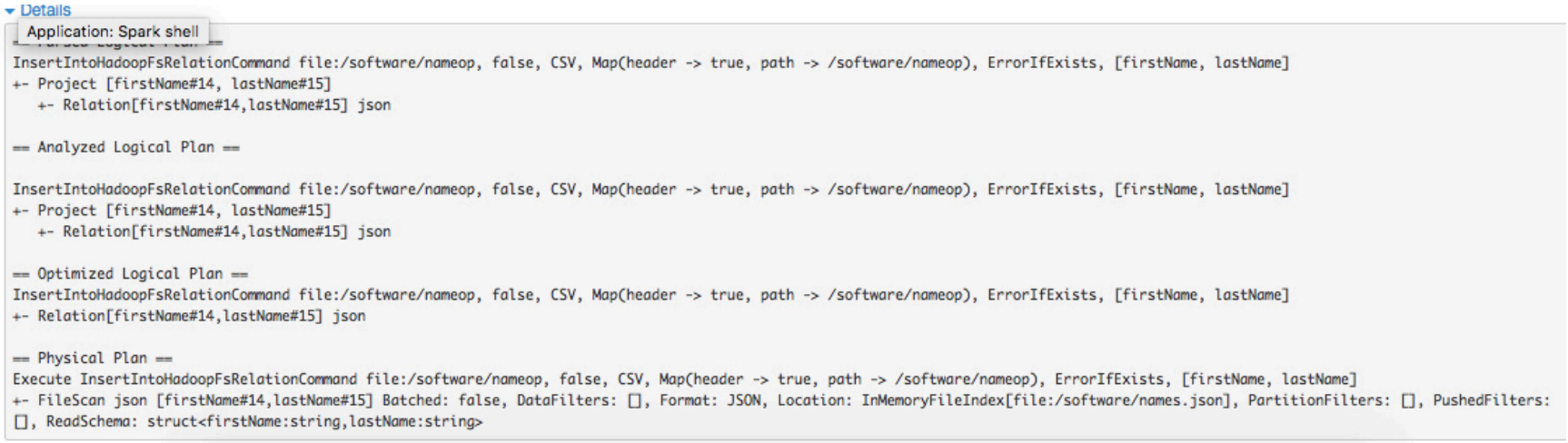
Show 20 entries Search:

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



Verify the plan



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

20. Persisting Data – 40 Minutes

In this lab, you will learn to cache the RDD and uncache it, when its done.

Data being used in this lab:

people.csv

```
pcode,lastName,firstNam
e,age
02134,Hopper,Grace,52
94020,Turing,Alan,33
```

pcodes.csv

```
pcode,city,state
02134,Boston,MA
94020,Palo
Alto,CA
87501,Santa
```

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

Fetch the records which pincode is “94020”

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

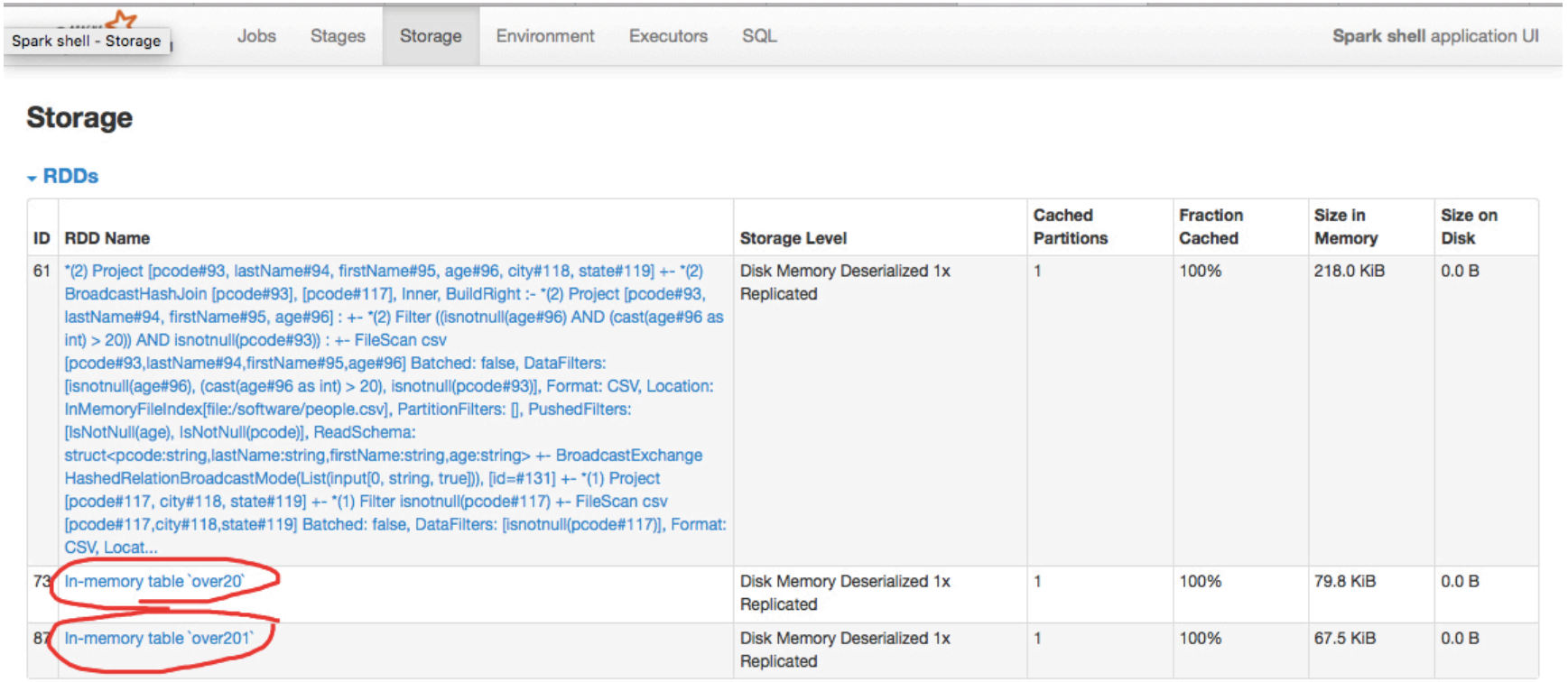
over20DF.createTempView("overAge")

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://127.0.0.1:4040/jobs/



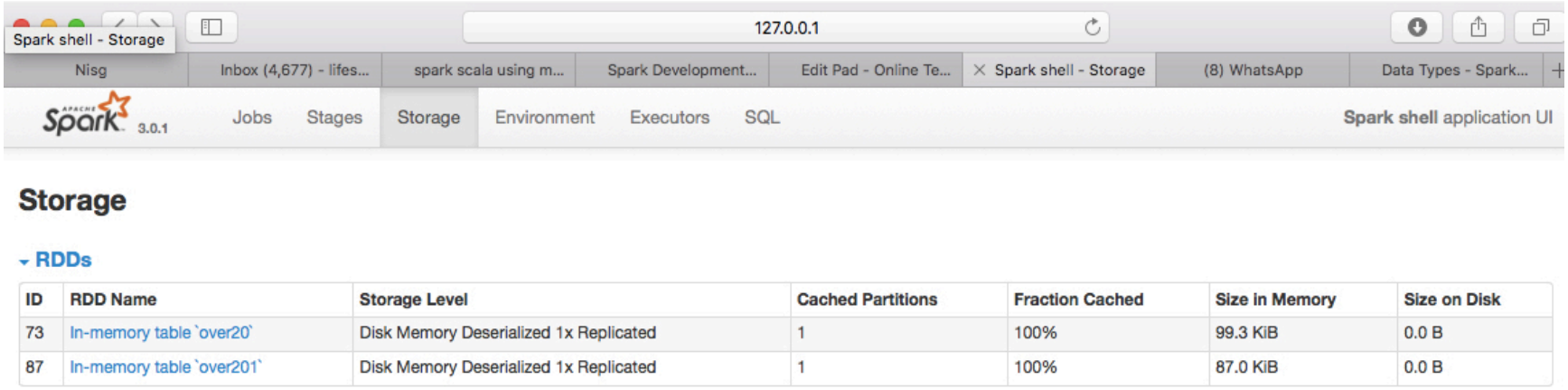
Changing the Persistence storage.

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

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>
```

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

Before:

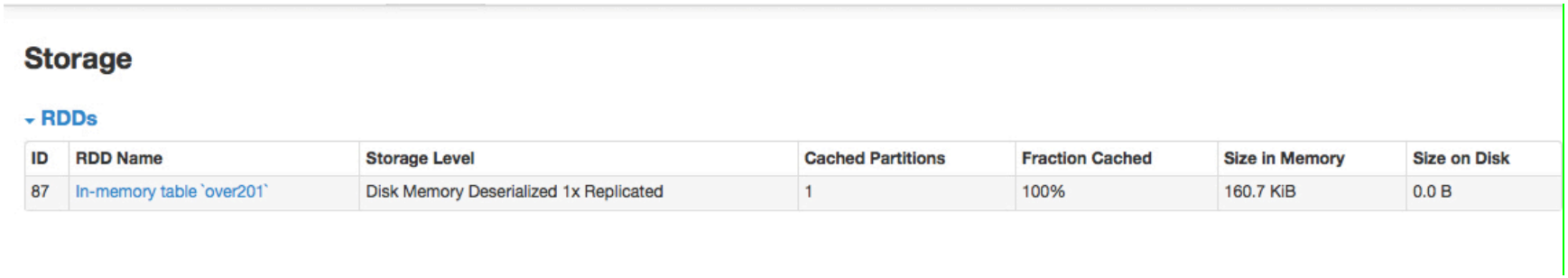


Display the list of tables:

spark.catalog.listTables.show

spark.catalog.uncacheTable("overage")

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



Storage

RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
87	In-memory table `over201`	Disk Memory Deserialized 1x Replicated	1	100%	160.7 KiB	0.0 B

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

21. Spark Streaming – Using Spark-Shell – 45 Minutes

We will count the number of words in text data received from a data server listening on a TCP socket.

Start the spark console.

```
[root@hp spark-2.1.0]# bin/spark-shell --master local[2]
18/04/05 22:46:31 WARN Utils: Service 'SparkUI' could not bind on port 4040. Attempting port 4041.
18/04/05 22:46:50 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.10.20.20:4041
Spark context available as 'sc' (master = local[2], app id = local-1522948591687).
Spark session available as 'spark'.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 2.1.0-mapr-1707
      /_/

Using Scala version 2.11.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_45)
Type in expressions to have them evaluated.
Type :help for more information.

scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@173b24c4

scala>
```

First, we import the names of the Spark Streaming classes and some implicit conversions from StreamingContext into our environment in order to add useful methods to other

classes we need (like DStream). StreamingContext is the main entry point for all streaming functionality. We create a local StreamingContext with two execution threads, and a batch interval of 1 second.

```
import org.apache.spark._
import org.apache.spark.streaming._

// Create a local StreamingContext with two working thread and batch interval of 10 second.
// The master requires 2 cores to prevent from a starvation scenario.

val ssc = new StreamingContext(sc, Seconds(10))
```

Using this context, we can create a DStream that represents streaming data from a TCP source, specified as hostname (e.g. localhost) and port (e.g. 9999).

```
// Create a DStream that will connect to hostname:port, like localhost:9999
val lines = ssc.socketTextStream("localhost", 9999)
```

This lines DStream represents the stream of data that will be received from the data server. Each record in this DStream is a line of text. Next, we want to split the lines by space characters into words.

```
// Split each line into words
val words = lines.flatMap(_.split(" "))
```

flatMap is a one-to-many DStream operation that creates a new DStream by generating multiple new records from each record in the source DStream. In this case, each line will be split into multiple words and the stream of words is represented as the words DStream. Next, we want to count these words.

```
// Count each word in each batch
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

// Print the first ten elements of each RDD generated in this DStream to the console
wordCounts.print()
```

The words DStream is further mapped (one-to-one transformation) to a DStream of (word, 1) pairs, which is then reduced to get the frequency of words in each batch of data. Finally, wordCounts.print() will print a few of the counts generated every second.
Note that when these lines are executed, Spark Streaming only sets up the computation it will perform when it is started, and no real processing has started yet. To start the processing after all the transformations have been setup, we finally call

```
ssc.start()             // Start the computation
ssc.awaitTermination()  // Wait for the computation to terminate
```

```
scala> val words = lines.flatMap(_.split(" "))
words: org.apache.spark.streaming.dstream.DStream[String] = org.apache.spark.streaming.dstream.FlatMappedDS
tream@650d5a3d

scala> val pairs = words.map(word => (word, 1))
pairs: org.apache.spark.streaming.dstream.DStream[(String, Int)] = org.apache.spark.streaming.dstream.Mappe
dDStream@53d94a6b

scala> val wordCounts = pairs.reduceByKey(_ + _)
wordCounts: org.apache.spark.streaming.dstream.DStream[(String, Int)] = org.apache.spark.streaming.dstream.
ShuffledDStream@490dc900

scala> wordCounts.print()

scala> ssc.start()             // Start the computation

scala> ssc.awaitTermination()  // Wait for the computation to terminate
-------------------------------------------
Time: 1522949060000 ms
-------------------------------------------
```

The above window will be changed every 10 seconds.

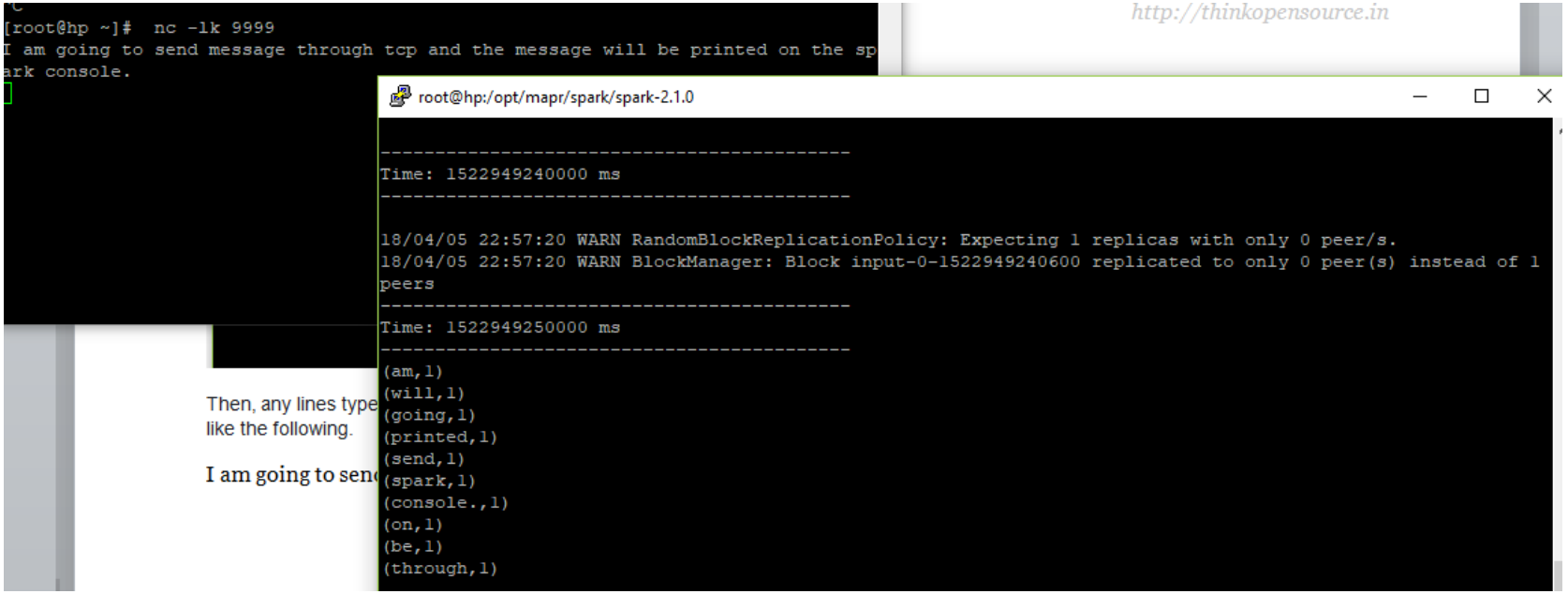
Open a terminal and perform the following.
You will need to run Netcat (a small utility found in most Unix-like systems) as a data server by using

```
$ nc -lk 9999
```

```
[root@hp ~]#  nc -lk 9999
```

Then, any lines typed in the terminal running the netcat server will be counted and printed on screen every second. It will look something like the following.

Type the following in the console:
I am going to send message through tcp and message will be printed on the spark console.



You should be able to determine the word count of each on the second window as shown above.

Type a few words after few seconds:

```
-------------------------------------------
Time: 1600101600000 ms
-------------------------------------------

20/09/14 16:40:07 WARN RandomBlockReplicationPolicy: Expecting 1 replicas with only 0 peer/s.
20/09/14 16:40:07 WARN BlockManager: Block input-0-1600101607400 replicated to only 0 peer(s) instead of 1 peers
-------------------------------------------
Time: 1600101610000 ms
-------------------------------------------
(Another,1)
(Message,1)

-------------------------------------------
Time: 1600101620000 ms
-------------------------------------------
```

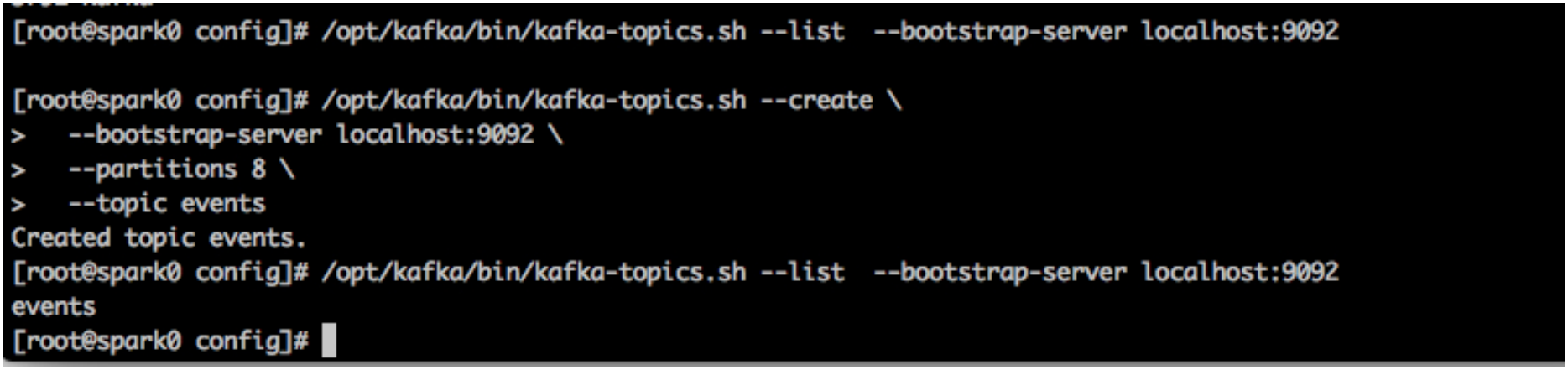
---------- Lab Ends Here ----------

22. Streaming Apache Kafka Messages – 90 Minutes

Install and start kafka before proceeding ahead. Refer Annexure for this.

Create topic events and verify it with the following kafka commands:

```
#/opt/kafka/bin/kafka-topics.sh --list  --bootstrap-server localhost:9092
#/opt/kafka/bin/kafka-topics.sh --create \
 --bootstrap-server localhost:9092 \
 --partitions 8 \
 --topic events
```



Using spark 3.0.1 in shell only

Use the Spark-Shell Only. (Use the following version Only.)

#spark-shell --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.0.0 org.apache.spark:spark-streaming-kafka-0-10_2.12:3.0.0

Execute the following commands to fetch records from Kafka.

#import spark.implicits._

#val inputDF = spark.readStream.format("kafka").option("kafka.bootstrap.servers","localhost:9092").option("subscribe", "events").option("startingOffsets", "earliest").option("checkpointLocation", "\tmp").load()

#inputDF.printSchema()

#inputDF.writeStream.format("console").option("truncate","false").start().awaitTermination()