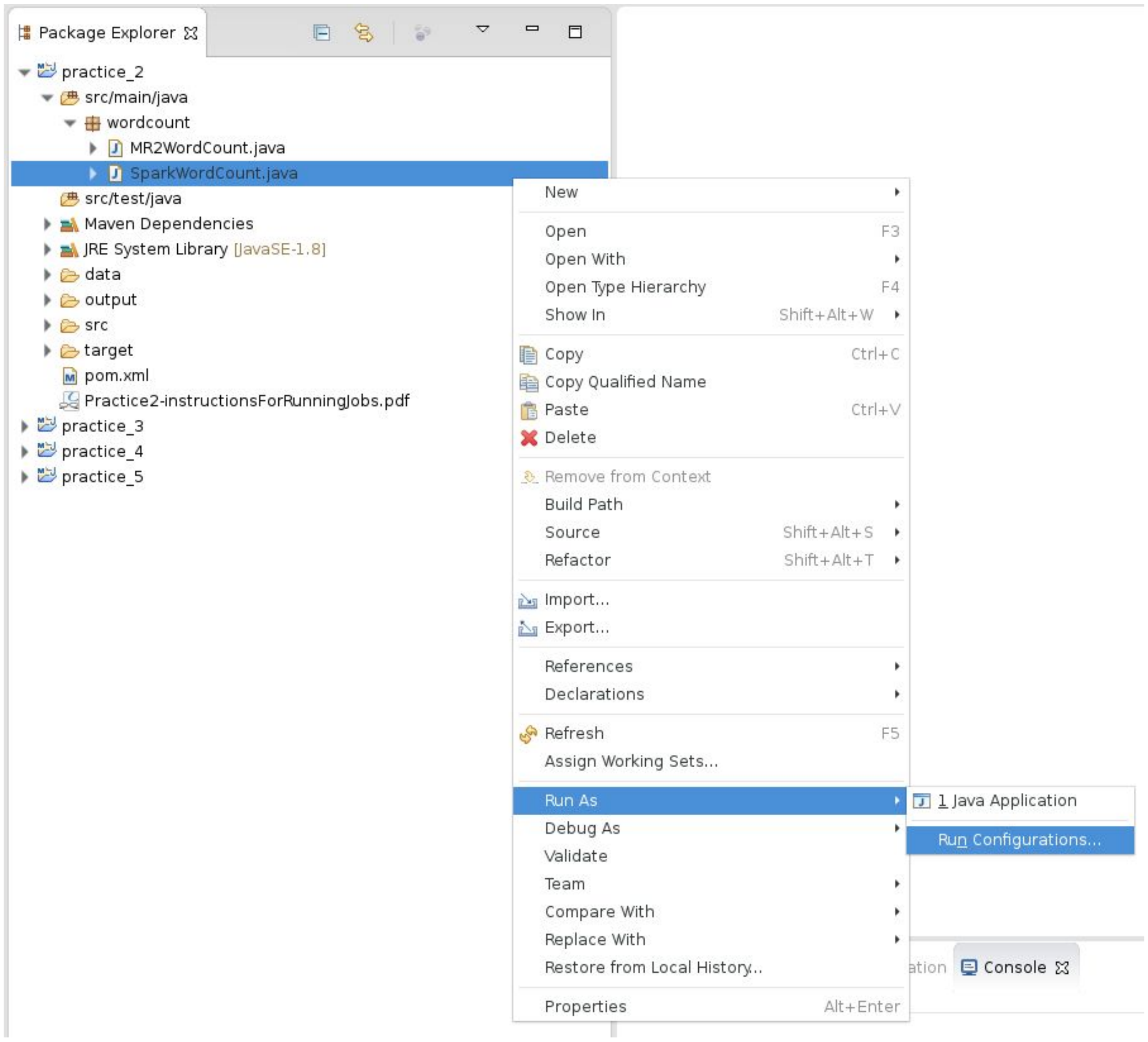
# Practice 2 - Learning to run a job in the VM

## Running SparkWordCount.java from Eclipse

### Create the run configuration

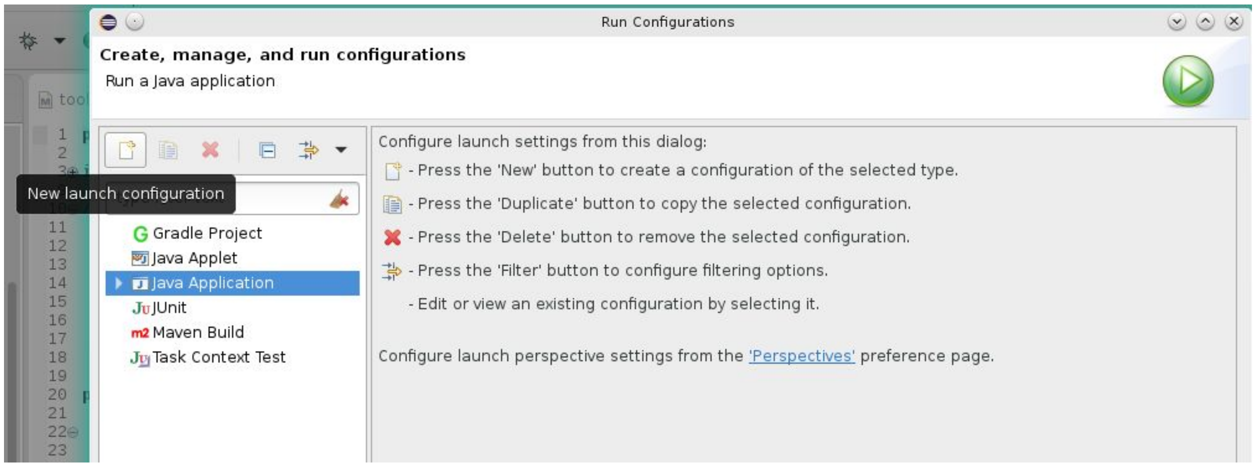
Start in Eclipse's package explorer:

- Select the project containing **`SparkWordCount.`**`java`
- Select the **`SparkWordCount.`**`java` file -> select "Run As" -> select "Run Configurations"
- An example is shown here:



## The Run Configuration dialog.

First, create a new launch configuration for a Java Application by selecting "Java Application" and then clicking on the 'New' button.



Configure launch settings from this dialog:

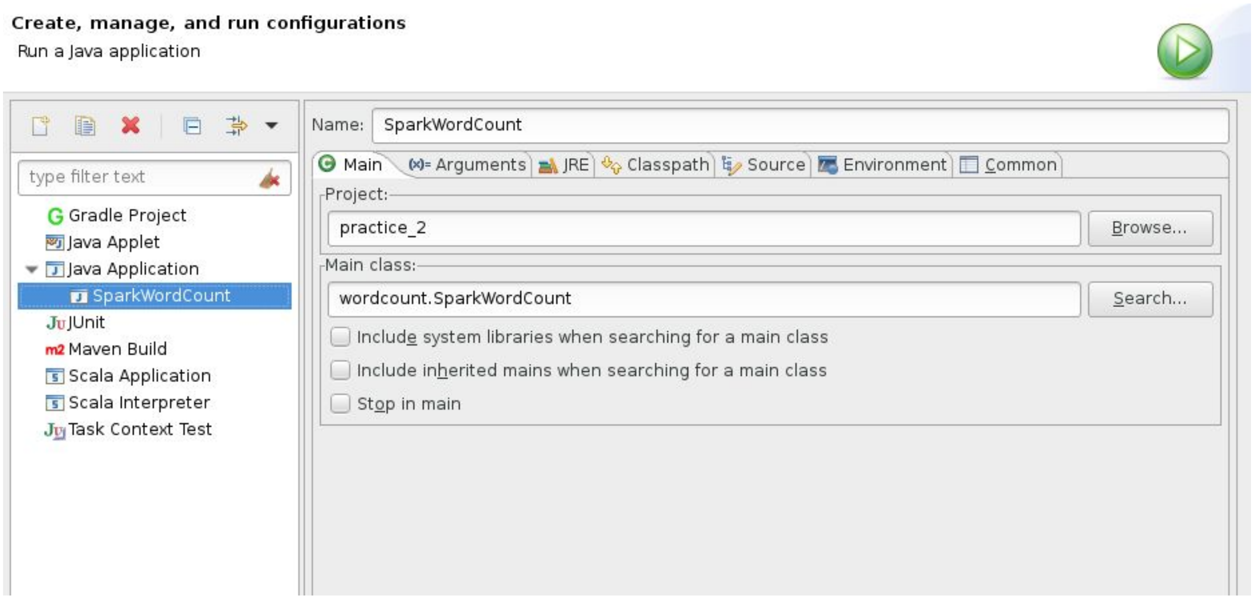
- Press the 'New' button to create a configuration of the selected type.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.

Configure launch perspective settings from the 'Perspectives' preference page.

Now, you see a Java Application called SparkWordCount in the dialog for configuring the Application.

It already has

- Name:  SparkWordCount (you can change this)
- Project: practice_2
- Main class:   wordcount.SparkWordCount



Name: SparkWordCount

Project: practice_2

Main class: wordcount.SparkWordCount

☐ Include system libraries when searching for a main class

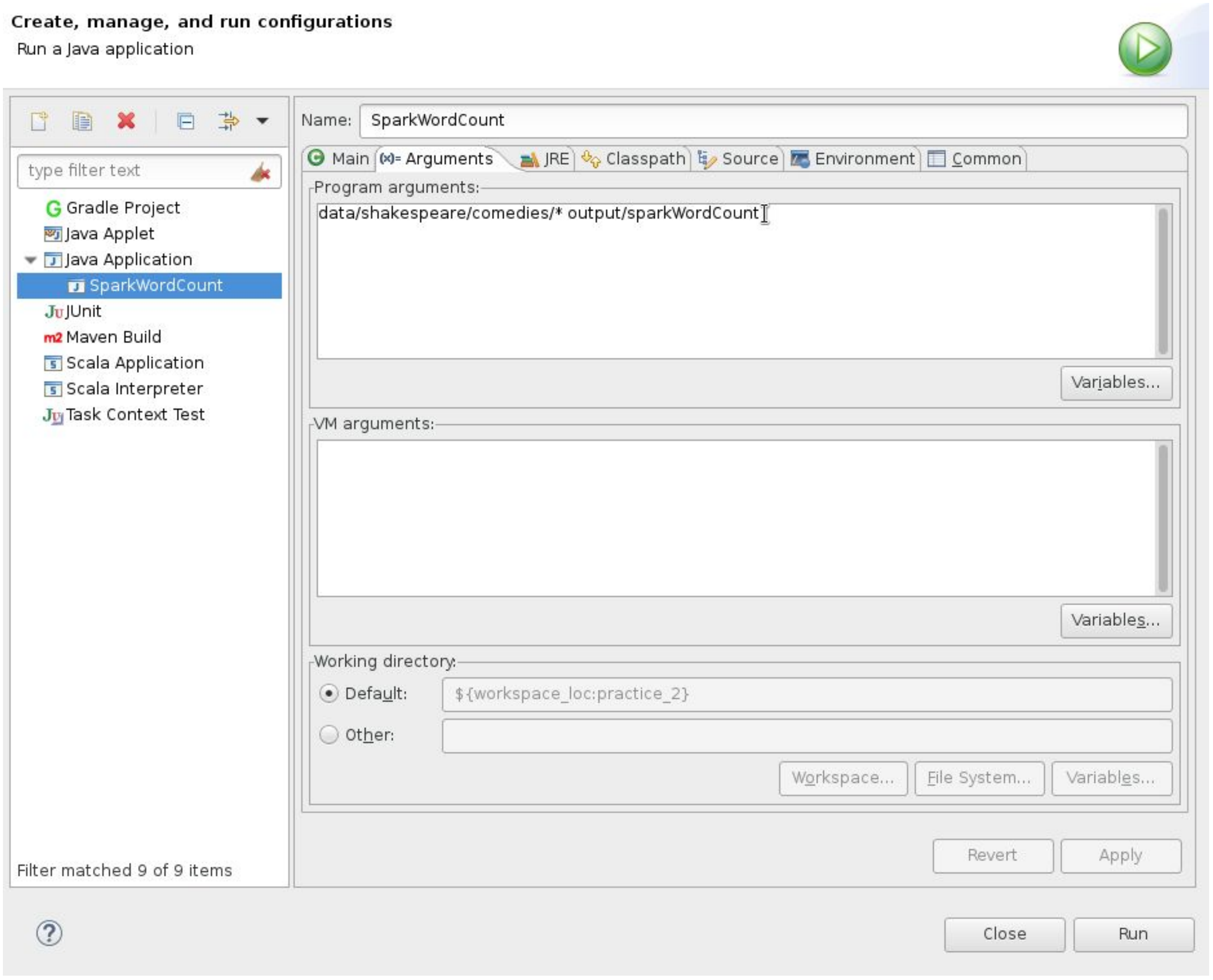
☐ Include inherited mains when searching for a main class

☐ Stop in main

**Click on the arguments tab and fill in the arguments.**

- The first argument is the location of the input data.
    - Note: example shows a *relative path* (does not start with a slash "/")
    - Because `shakespeare/comedies` is in a folder in our Eclipse project, we can use a relative path.
    - To refer to data NOT to your project, you need to use the full path.

- The second argument is the location of your output directory.
    - Again, this uses a relative path to the "output" folder in your project.



Create, manage, and run configurations

Run a Java application

Name: SparkWordCount

Program arguments:

```
data/shakespeare/comedies/* output/sparkWordCount
```

VM arguments:

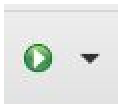
Working directory:

Default: ${workspace_loc:practice_2}

## Run

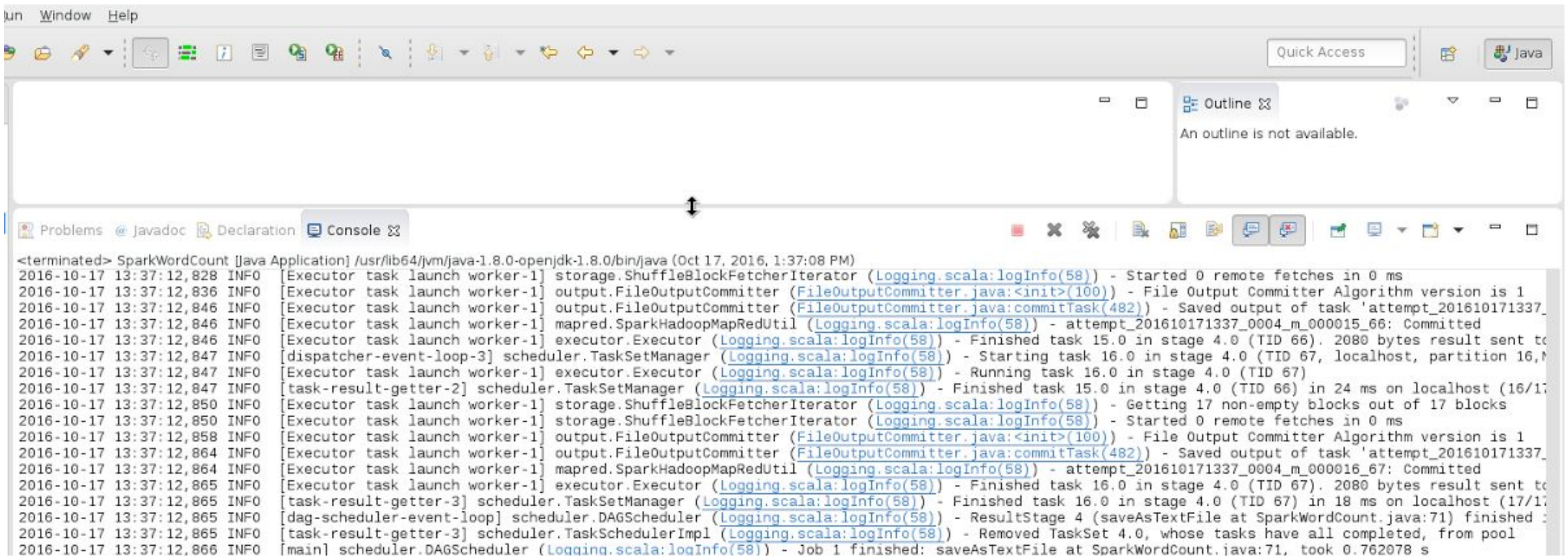
At this point you can run the program by clicking "Run".

You can always rerun this program by clicking the "Run" arrow.  It will show you which run configuration you are using when you mouse-over it.



## Processing is shown in the console

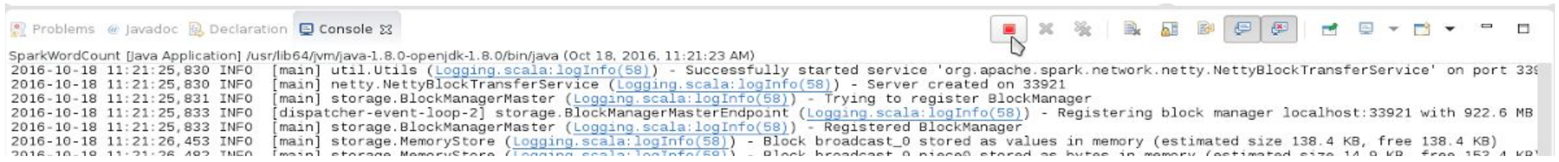
After you select run, the logged messages for the run will scroll onto your console.  You can resize the console to see the results more clearly.  Here are the results of running SparkWordCount:



```
<terminated> SparkWordCount [Java Application] /usr/lib64/jvm/java-1.8.0-openjdk-1.8.0/bin/java (Oct 17, 2016, 1:37:08 PM)
2016-10-17 13:37:12,828 INFO  [Executor task launch worker-1] storage.ShuffleBlockFetcherIterator (Logging.scala:logInfo(58)) - Started 0 remote fetches in 0 ms
2016-10-17 13:37:12,836 INFO  [Executor task launch worker-1] output.FileOutputCommitter (FileOutputCommitter.java:<init>(100)) - File Output Committer Algorithm version is 1
2016-10-17 13:37:12,846 INFO  [Executor task launch worker-1] output.FileOutputCommitter (FileOutputCommitter.java:commitTask(482)) - Saved output of task 'attempt_201610171337_
2016-10-17 13:37:12,846 INFO  [Executor task launch worker-1] mapred.SparkHadoopMapRedUtil (Logging.scala:logInfo(58)) - attempt_201610171337_0004_m_000015_66: Committed
2016-10-17 13:37:12,846 INFO  [Executor task launch worker-1] executor.Executor (Logging.scala:logInfo(58)) - Finished task 15.0 in stage 4.0 (TID 66). 2080 bytes result sent to
2016-10-17 13:37:12,847 INFO  [dispatcher-event-loop-3] scheduler.TaskSetManager (Logging.scala:logInfo(58)) - Starting task 16.0 in stage 4.0 (TID 67, localhost, partition 16,
2016-10-17 13:37:12,847 INFO  [Executor task launch worker-1] executor.Executor (Logging.scala:logInfo(58)) - Running task 16.0 in stage 4.0 (TID 67)
2016-10-17 13:37:12,847 INFO  [task-result-getter-2] scheduler.TaskSetManager (Logging.scala:logInfo(58)) - Finished task 15.0 in stage 4.0 (TID 66) in 24 ms on localhost (16/17
2016-10-17 13:37:12,850 INFO  [Executor task launch worker-1] storage.ShuffleBlockFetcherIterator (Logging.scala:logInfo(58)) - Getting 17 non-empty blocks out of 17 blocks
2016-10-17 13:37:12,850 INFO  [Executor task launch worker-1] storage.ShuffleBlockFetcherIterator (Logging.scala:logInfo(58)) - Started 0 remote fetches in 0 ms
2016-10-17 13:37:12,858 INFO  [Executor task launch worker-1] output.FileOutputCommitter (FileOutputCommitter.java:<init>(100)) - File Output Committer Algorithm version is 1
2016-10-17 13:37:12,864 INFO  [Executor task launch worker-1] output.FileOutputCommitter (FileOutputCommitter.java:commitTask(482)) - Saved output of task 'attempt_201610171337_
2016-10-17 13:37:12,864 INFO  [Executor task launch worker-1] mapred.SparkHadoopMapRedUtil (Logging.scala:logInfo(58)) - attempt_201610171337_0004_m_000016_67: Committed
2016-10-17 13:37:12,865 INFO  [Executor task launch worker-1] executor.Executor (Logging.scala:logInfo(58)) - Finished task 16.0 in stage 4.0 (TID 67). 2080 bytes result sent to
2016-10-17 13:37:12,865 INFO  [task-result-getter-3] scheduler.TaskSetManager (Logging.scala:logInfo(58)) - Finished task 16.0 in stage 4.0 (TID 67) in 18 ms on localhost (17/17
2016-10-17 13:37:12,865 INFO  [dag-scheduler-event-loop] scheduler.DAGScheduler (Logging.scala:logInfo(58)) - ResultStage 4 (saveAsTextFile at SparkWordCount.java:71) finished
2016-10-17 13:37:12,865 INFO  [task-result-getter-3] scheduler.TaskSchedulerImpl (Logging.scala:logInfo(58)) - Removed TaskSet 4.0, whose tasks have all completed, from pool
2016-10-17 13:37:12,866 INFO  [main] scheduler.DAGScheduler (Logging.scala:logInfo(58)) - Job 1 finished: saveAsTextFile at SparkWordCount.java:71, took 0.762078 s
```

## Stopping a job

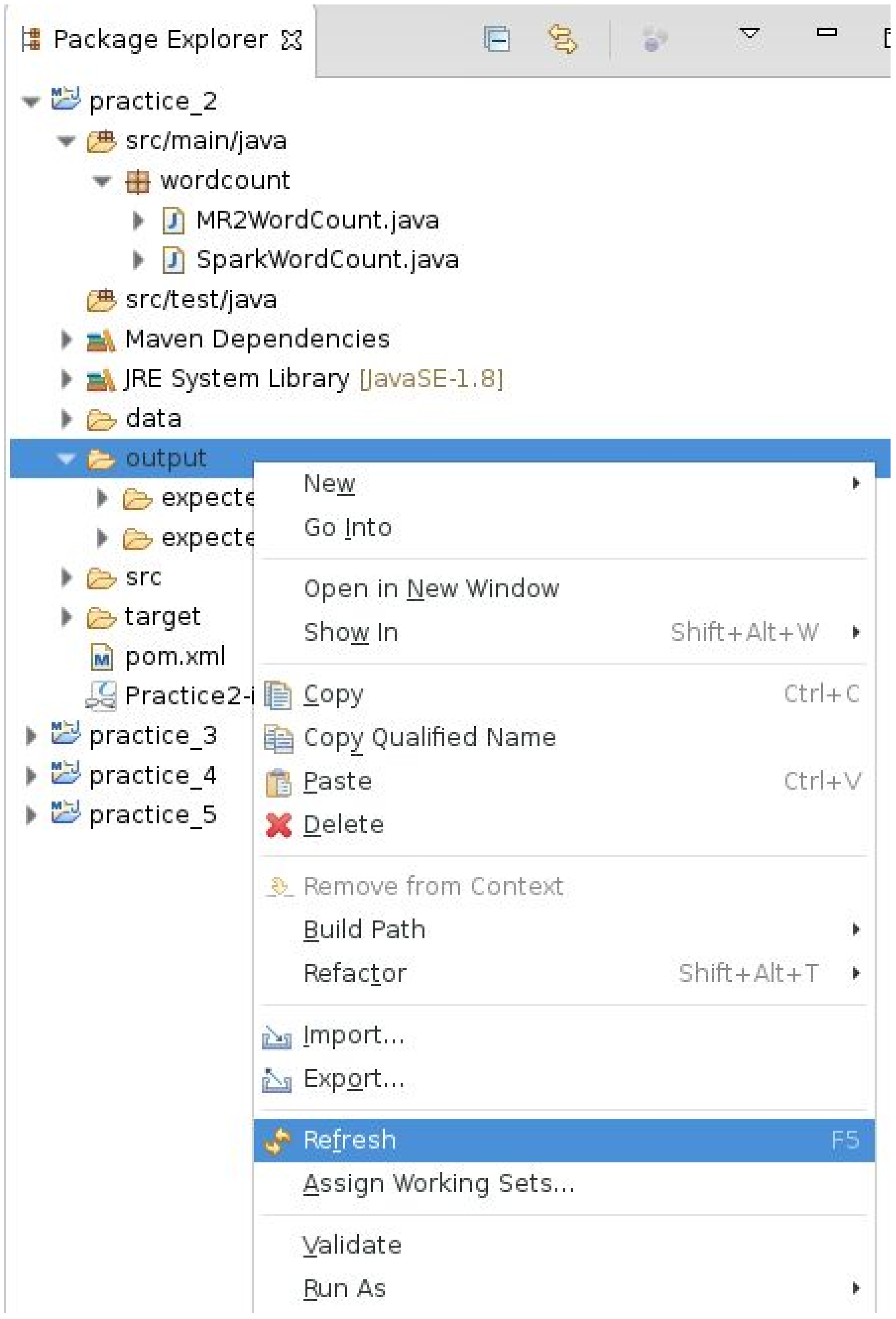
Simple - just click on the little red box shown on the console



```
SparkWordCount [Java Application] /usr/lib64/jvm/java-1.8.0-openjdk-1.8.0/bin/java (Oct 18, 2016, 11:21:23 AM)
2016-10-18 11:21:25,830 INFO  [main] util.Utils (Logging.scala:logInfo(58)) - Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 339
2016-10-18 11:21:25,830 INFO  [main] netty.NettyBlockTransferService (Logging.scala:logInfo(58)) - Server created on 33921
2016-10-18 11:21:25,831 INFO  [main] storage.BlockManagerMaster (Logging.scala:logInfo(58)) - Trying to register BlockManager
2016-10-18 11:21:25,833 INFO  [dispatcher-event-loop-2] storage.BlockManagerMasterEndpoint (Logging.scala:logInfo(58)) - Registering block manager localhost:33921 with 922.6 MB
2016-10-18 11:21:25,833 INFO  [main] storage.BlockManagerMaster (Logging.scala:logInfo(58)) - Registered BlockManager
2016-10-18 11:21:26,453 INFO  [main] storage.MemoryStore (Logging.scala:logInfo(58)) - Block broadcast_0 stored as values in memory (estimated size 138.4 KB, free 138.4 KB)
```

## Output

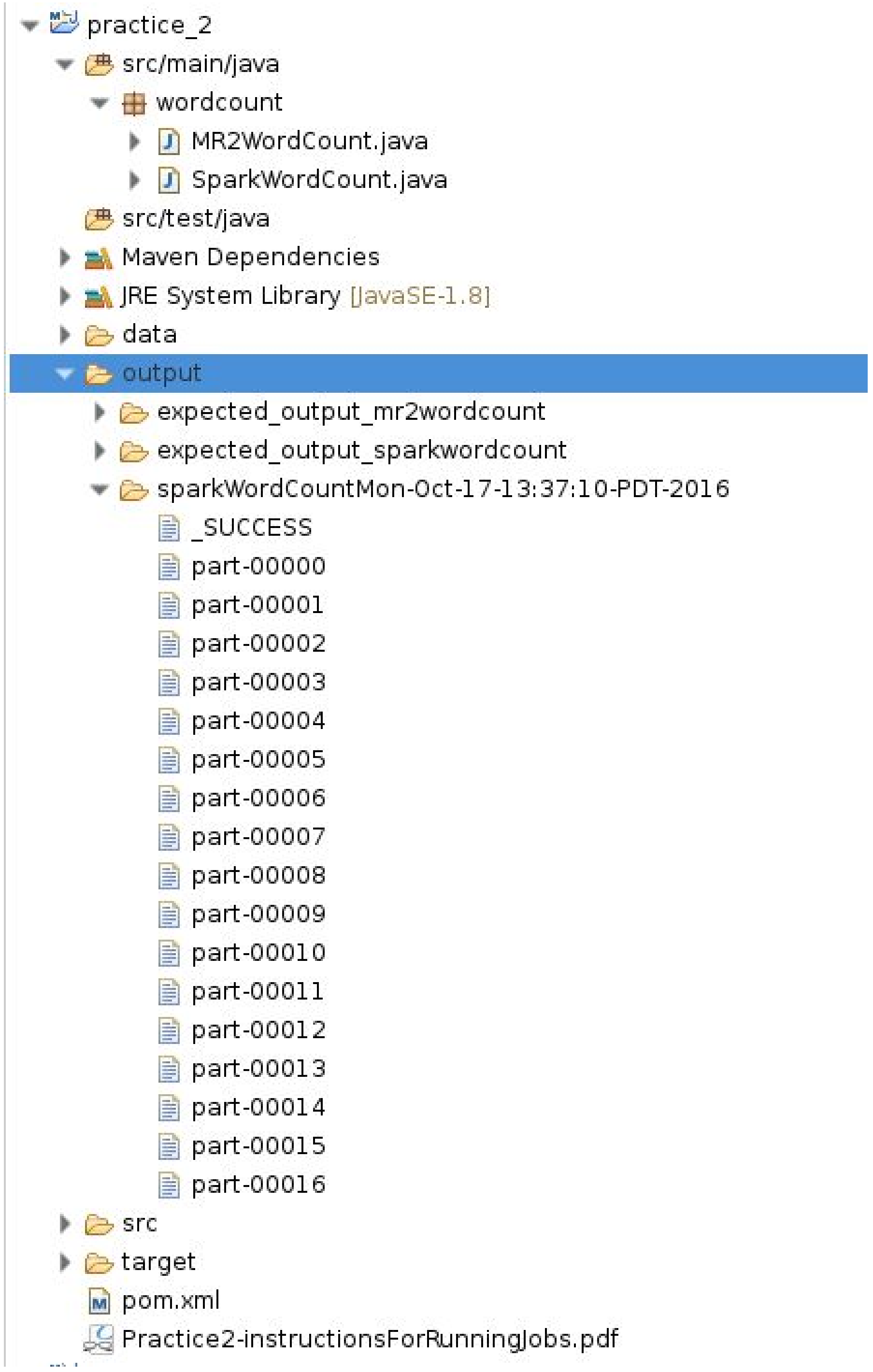
To see the output, right-click on the output folder and refresh:



**Finally, you will see the output.** Click on the SparkWordCount<date> folder to expand.

The _SUCCESS file indicates that the job completed successfully. You can ignore it.

The 'part-*' files contain the output. Each part file is the output from one task. In this case, there are 17 reduce tasks running, so there are 17 output files.



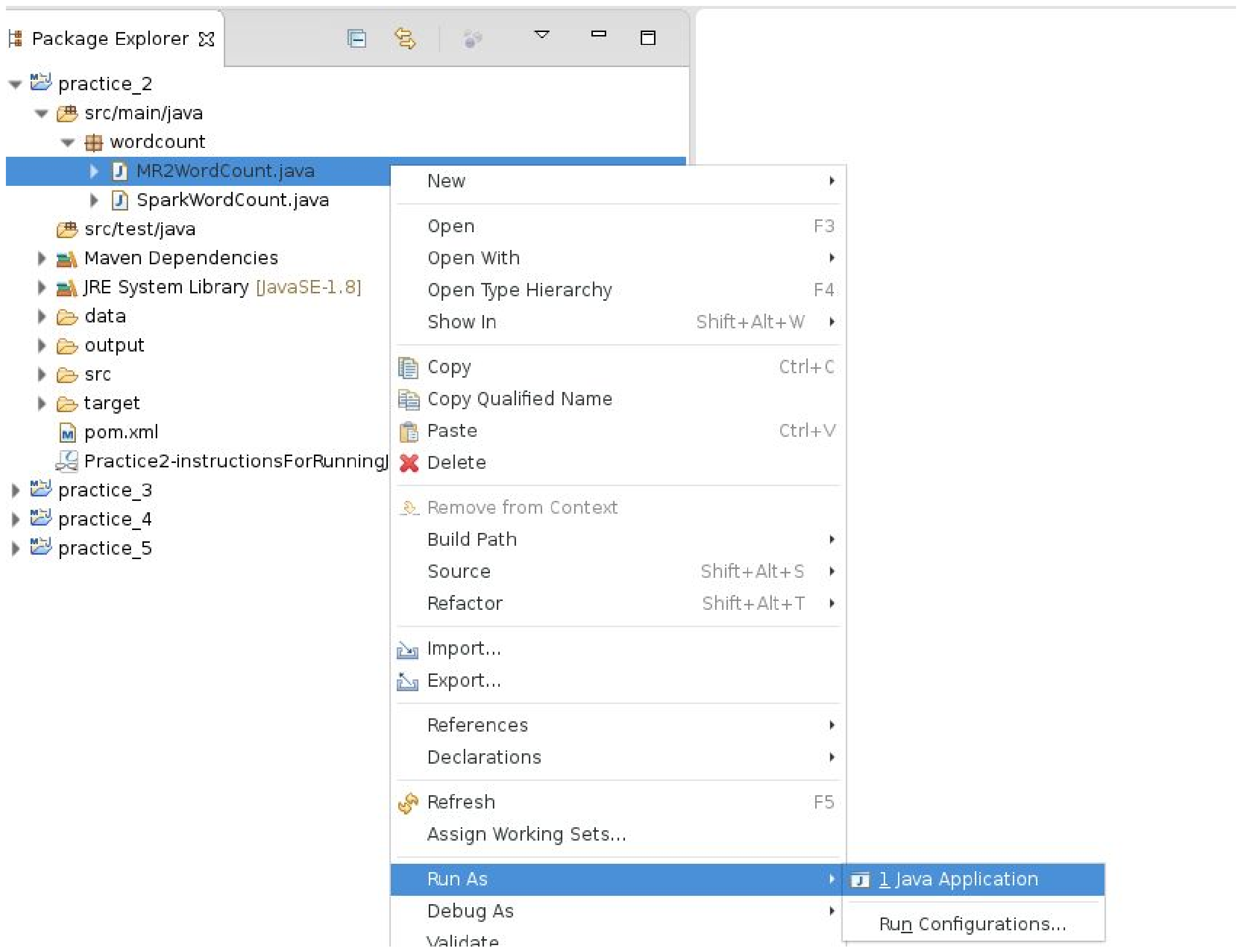
```
practice_2
  src/main/java
    wordcount
      MR2WordCount.java
      SparkWordCount.java
  src/test/java
  Maven Dependencies
  JRE System Library [JavaSE-1.8]
  data
  output
    expected_output_mr2wordcount
    expected_output_sparkwordcount
    sparkWordCountMon-Oct-17-13:37:10-PDT-2016
      _SUCCESS
      part-00000
      part-00001
      part-00002
      part-00003
      part-00004
      part-00005
      part-00006
      part-00007
      part-00008
      part-00009
      part-00010
      part-00011
      part-00012
      part-00013
      part-00014
      part-00015
      part-00016
  src
  target
  pom.xml
  Practice2-instructionsForRunningJobs.pdf
```

## Running MR2WordCount.java from Eclipse.

You may notice that there is an MR2WordCount.java program in Practice 2. The run configuration for MR2WordCount.java is already setup, so you can just run it.

Once a Run Configuration exists, you can use it (shown below):

- right-click on the app you want to run in the Package Explorer
- selecting “RunAs”
- select “Java Application”



## Running from the commandline

### 1) Create the jar file for the project:

- In the **Package Explorer, right-click on the project (practice_2)**

- Select **Run-As** and then select **Maven Install.**
  - This will create the jar file
  - You will see the process in the Eclipse console
  - Notice, you can cleanup old jar files by running **Maven clean** *before* you run Maven install.

- In the **Package Explorer,** open **target** folder
  - it should now contain a jar file called: **practice-2.1.0.0-SNAPSHOT.jar**
  - this is the jar file you will use to run the jobs below

### 2) Run a Spark job on the cluster

When you run locally, Spark's master does not get involved and there is no webUI for viewing the job and its execution. To get the master involved, you can run on the cluster. The syntax is:

```
spark-submit \
--master <REST URL> \
--deploy-mode cluster \
--class <MainClassName> \
<myJar>.jar <input_in_HDFS> <output_in_HDFS>
```

- make sure you are in the target directory containing your jar file.
  - Note, you can type pwd to check your present working directory.
  - if necessary, `cd /home/cloudera/workspace/practice_2/target`

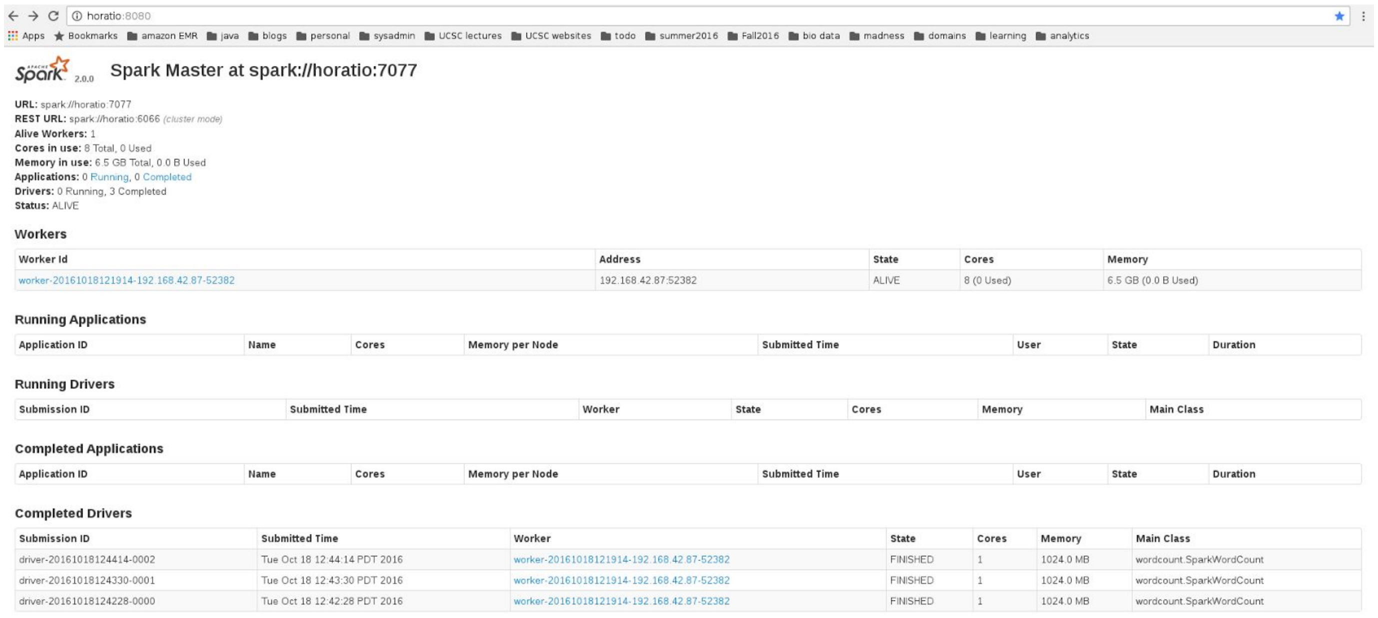
- make sure your input data is on `HDFS`

- run the SparkWordCount job *on the cluster* via the REST server on port 6066:

```
$ spark-submit \
--master spark://quickstart.cloudera:6066  \
--deploy-mode cluster \
--class wordcount.SparkWordCount \
practice_2-1.0.0-SNAPSHOT.jar \
/user/cloudera/shakespeare/* /user/cloudera/practice2_sparkout
```

## View the job - Use the Spark Master UI

- Open a browser and go to the master's UI at **quickstart.cloudera:8080.**
- On my machine, it looks like this.  On your VM, you will see a different master url.



**Spark Master at spark://horatio:7077**

**URL:** spark://horatio:7077
**REST URL:** spark://horatio:6066 *(cluster mode)*
**Alive Workers:** 1
**Cores in use:** 8 Total, 0 Used
**Memory in use:** 6.5 GB Total, 0.0 B Used
**Applications:** 0 Running, 0 Completed
**Drivers:** 0 Running, 3 Completed
**Status:** ALIVE

**Workers**

| Worker Id | Address | State | Cores | Memory |
|---|---|---|---|---|
| worker-20161018121914-192.168.42.87-52382 | 192.168.42.87:52382 | ALIVE | 8 (0 Used) | 6.5 GB (0.0 B Used) |

**Running Applications**

| Application ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|

**Running Drivers**

| Submission ID | Submitted Time | Worker | State | Cores | Memory | Main Class |
|---|---|---|---|---|---|---|

**Completed Applications**

| Application ID | Name | Cores | Memory per Node | Submitted Time | User | State | Duration |
|---|---|---|---|---|---|---|---|

**Completed Drivers**

| Submission ID | Submitted Time | Worker | State | Cores | Memory | Main Class |
|---|---|---|---|---|---|---|
| driver-20161018124414-0002 | Tue Oct 18 12:44:14 PDT 2016 | worker-20161018121914-192.168.42.87-52382 | FINISHED | 1 | 1024.0 MB | wordcount.SparkWordCount |
| driver-20161018124330-0001 | Tue Oct 18 12:43:30 PDT 2016 | worker-20161018121914-192.168.42.87-52382 | FINISHED | 1 | 1024.0 MB | wordcount.SparkWordCount |
| driver-20161018124228-0000 | Tue Oct 18 12:42:28 PDT 2016 | worker-20161018121914-192.168.42.87-52382 | FINISHED | 1 | 1024.0 MB | wordcount.SparkWordCount |

**Refresh the page.**  You should see at least one "Running Drivers" or  "Completed Drivers". You can keep refreshing until your job is listed under "Completed Drivers".  It should take less than a minute.

Under **Completed Drivers** click on the Worker for the job you just ran.

- The **Worker UI** will show your job and also logs for the job.
- Click on *stderr* to see the information logged for the job.

## Killing a Spark job[1]

When you do need to kill a Spark application,  you can do so by: using spark-class:

```
$ spark-class org.apache.spark.deploy.Client kill <master url> <submission ID>
```

You can find the `master url` and the `submission ID` on the master's web UI, **quickstart.cloudera:8080.**

[1] Don't try this now, you will drive yourself crazy trying to kill it before it completes.

## 3) Run an MR2 job on the cluster

The simplest way to submit job to Hadoop is to use the following syntax:

```
 hadoop jar <myJar>.jar <MainClassName> \
<input_in_HDFS> <output_in_HDFS>
```

For this practice:

- if you haven't already done so, open a terminal window and cd to the target dir:

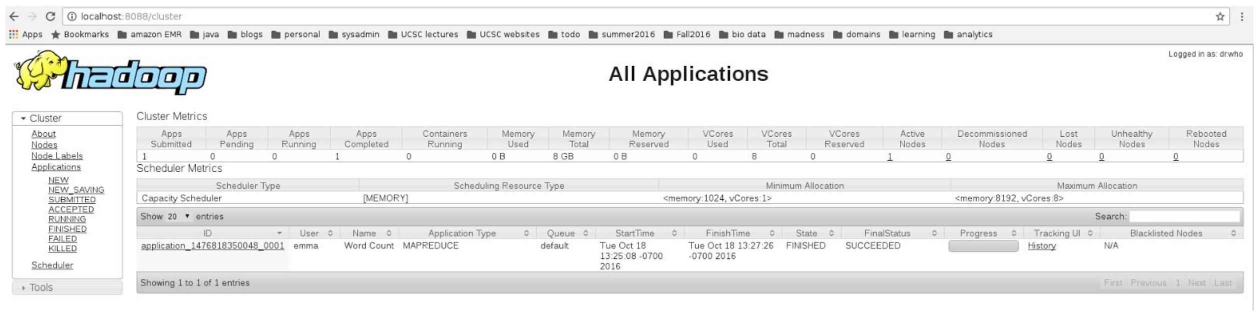
  ```
  $ cd /home/cloudera/workspace/practice_2/target
  ```

- make sure your input data is on `HDFS`

- run the MR2WordCount job by typing:

  ```
  $ hadoop jar practice_2-1.0.0-SNAPSHOT.jar \
  wordcount.MR2WordCount \
  /user/cloudera/shakespeare/* /user/cloudera/practice2_mr2out
  ```

- open the UI for Hadoop jobs at **quickstart.cloudera:8088**



localhost:8088/cluster

Logged in as: dr.who

All Applications

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 | Rebooted Nodes |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 1 | 0 | 0 B | 8 GB | 0 B | 0 | 8 | 0 | 1 | 0 | 0 | 0 | 0 |

Scheduler Metrics

| Scheduler Type | Scheduling Resource Type | Minimum Allocation | Maximum Allocation |
|---|---|---|---|
| Capacity Scheduler | [MEMORY] | <memory:1024, vCores:1> | <memory:8192, vCores:8> |

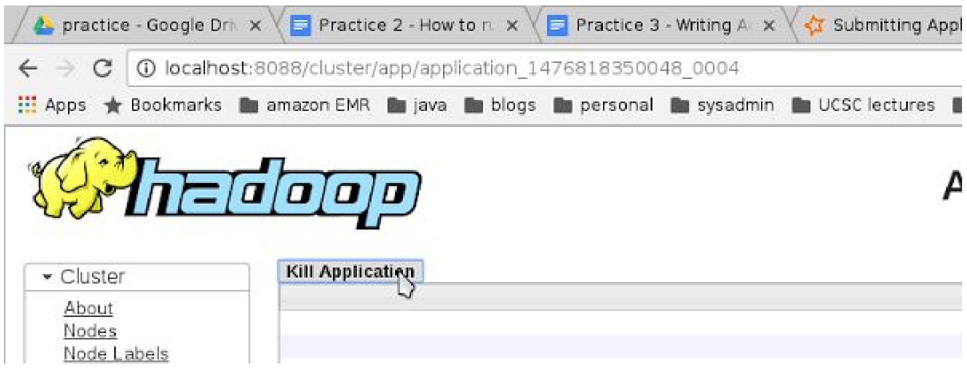
Show 20 entries

| ID | User | Name | Application Type | Queue | StartTime | FinishTime | State | FinalStatus | Progress | Tracking UI | Blacklisted Nodes |
|---|---|---|---|---|---|---|---|---|---|---|---|
| application_1476818350048_0001 | emma | Word Count | MAPREDUCE | default | Tue Oct 18 13:25:08 -0700 2016 | Tue Oct 18 13:27:26 -0700 2016 | FINISHED | SUCCEEDED | | History | N/A |

Showing 1 to 1 of 1 entries

## Killing an MR2 job

The simplest way to kill a Hadoop job is to use the UI.  From the same UI, drill-down on the link for the running application, and select kill.



# End of Practice 2