**Exercise 5**

*Get started with Apache Spark and Python*

**Prior Knowledge**

Unix Command Line Shell

Simple Python

**Learning Objectives**

Understand the Spark system

Understand the Jupyter Notebook model

Submit Spark jobs locally and using YARN

Write SparkSQL code in Python  
WordCount!

**Software Requirements**

(see separate document for installation of these)

* Apache Spark 2.0.0
* Python 2.7.12
* Jupyter notebooks

**Part A. Spark Python Shell (pySpark)**

1. We are going to do a wordcount against a set of books downloaded from Project Gutenberg. Wordcount is the definitive Big Data program (sort of Hello World for Big Data) and it is frankly embarrassing that we haven’t done one yet.
2. Apache Spark has a useful Python shell, which we can use to interactively test and run code. Since we have our data in HDFS, *we need to ensure HDFS is running.* (Follow the instructions from the Hadoop lab).
3. Let’s load some books into HDFS. In a terminal window (Ctrl-Alt-T)

hadoop fs -mkdir -p /user/oxclo/books

hadoop fs -put ~/datafiles/books/\* /user/oxclo/books/

1. Let’s make a directory for our code:  
     
   First make sure it is cleared out from any previous runs:  
     
   rm -rf ~/pse

Now create it again:  
  
mkdir ~/pse  
cd ~/pse

1. Now start the Spark Python command line tool –  
     
   ~/spark/bin/pyspark  
   1. You should see a lot of log come up, ending in something like:

Python 2.7.12 (default, Nov 19 2016, 06:48:10)

[GCC 5.4.0 20160609] on linux2

Type "help", "copyright", "credits" or "license" for more information.

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).

17/07/01 13:51:32 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable

17/07/01 13:51:32 WARN Utils: Your hostname, oxclo resolves to a loopback address: 127.0.0.1; using 172.16.64.199 instead (on interface ens33)

17/07/01 13:51:32 WARN Utils: Set SPARK\_LOCAL\_IP if you need to bind to another address

17/07/01 13:51:38 WARN ObjectStore: Failed to get database global\_temp, returning NoSuchObjectException

Welcome to

\_\_\_\_ \_\_

/ \_\_/\_\_ \_\_\_ \_\_\_\_\_/ /\_\_

\_\ \/ \_ \/ \_ `/ \_\_/ '\_/

/\_\_ / .\_\_/\\_,\_/\_/ /\_/\\_\ version 2.1.1

/\_/

Using Python version 2.7.12 (default, Nov 19 2016 06:48:10)

SparkSession available as 'spark'.

>>>

* 1. This is the “traditional” Spark Python command line tool. We aren’t going to use this just now.
  2. Type quit() to leave.

1. The VM has a “notebook” system called Jupyter configured by default. The result is that instead of starting a command line repl[[1]](#footnote-1), there is a web based editor / evaluator launched instead.
2. To start this, type  
   ~/pyspark-jupiter.sh
3. In the command-line you will see

[I 13:53:23.865 NotebookApp] Serving notebooks from local directory: /home/oxclo/pse

[I 13:53:23.866 NotebookApp] 0 active kernels

[I 13:53:23.866 NotebookApp] The Jupyter Notebook is running at: http://localhost:8888/?token=fd655aab32ed4840ceb47b8b7392b1243a27f56350888a91

[I 13:53:23.866 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).

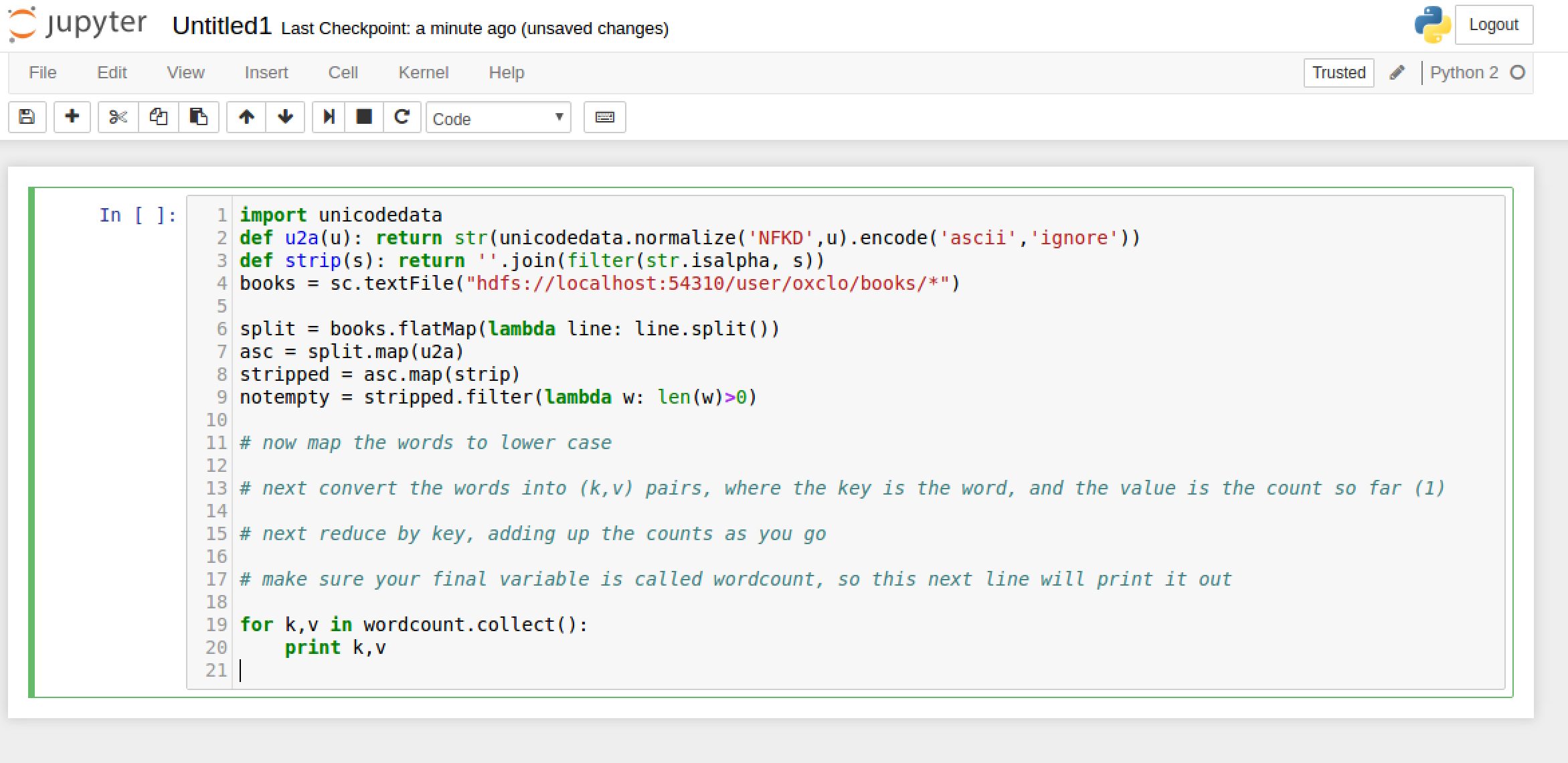
[C 13:53:23.868 NotebookApp]

Copy/paste this URL into your browser when you connect for the first time,

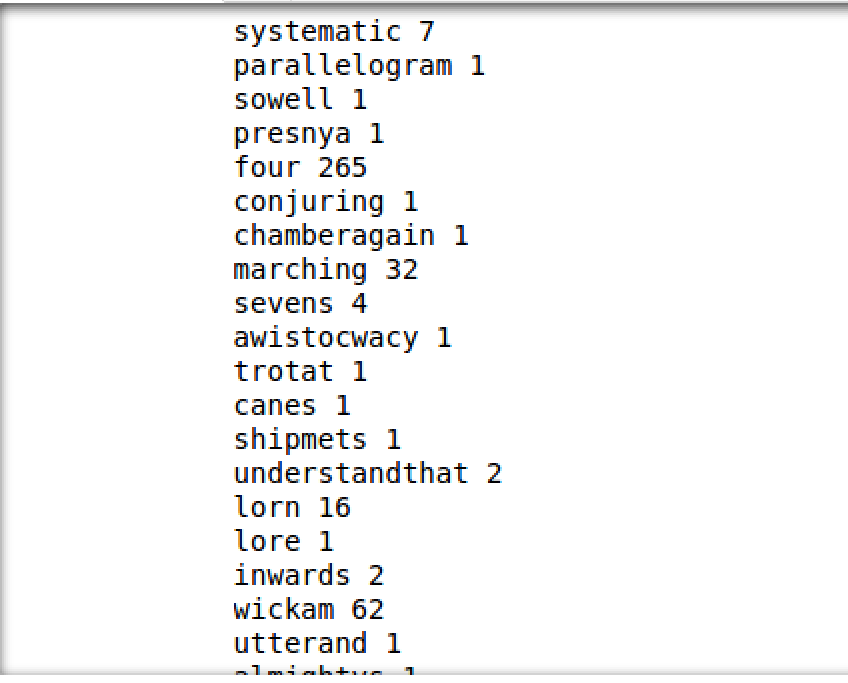
to login with a token:

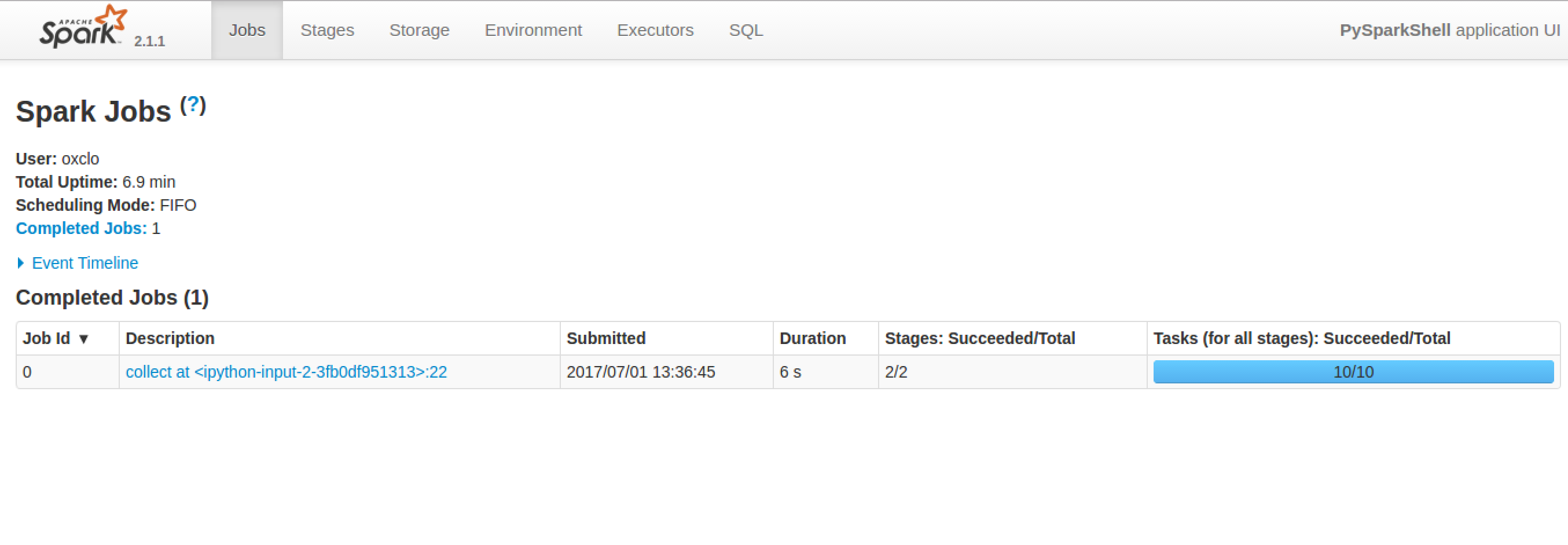
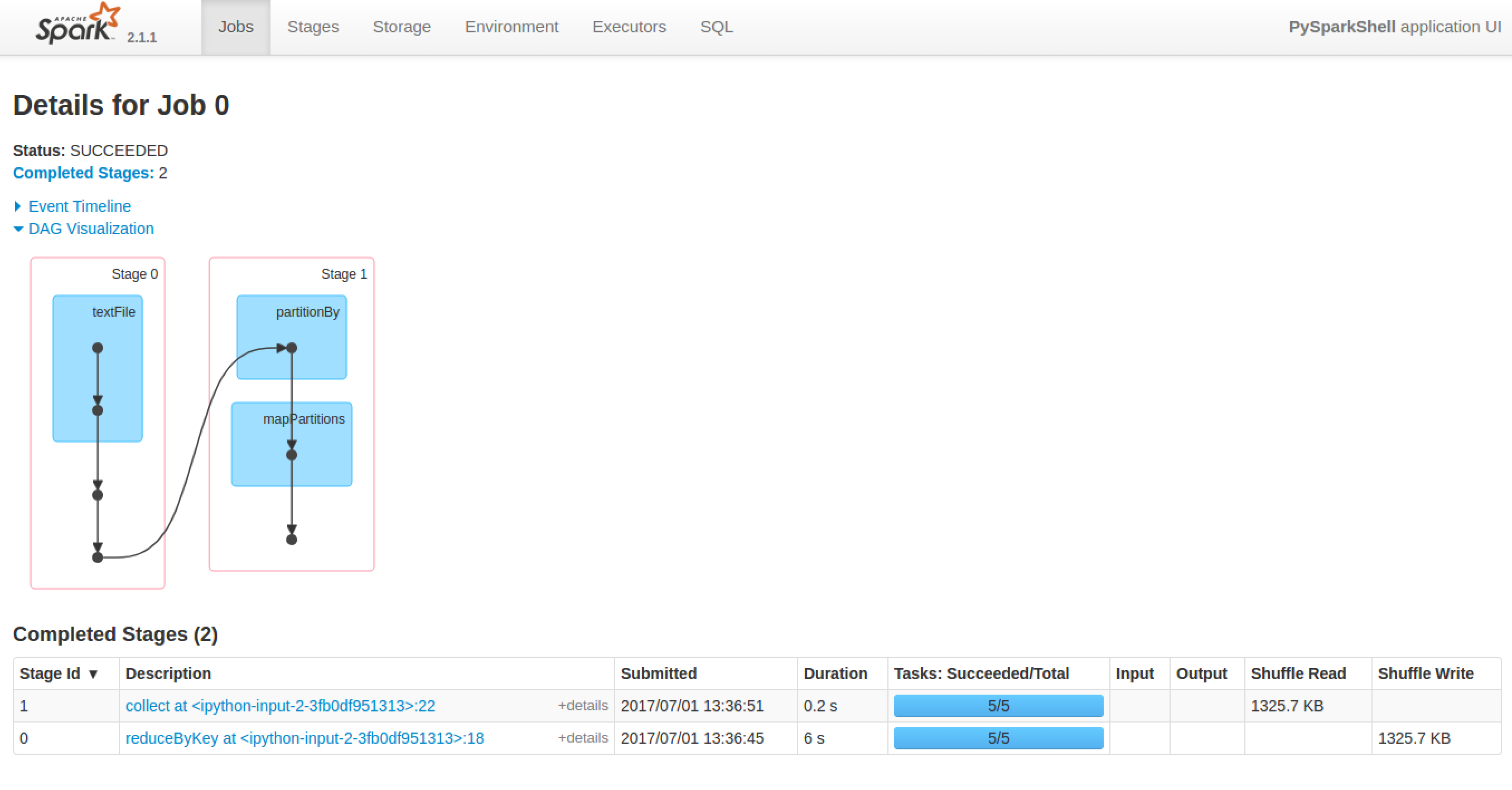
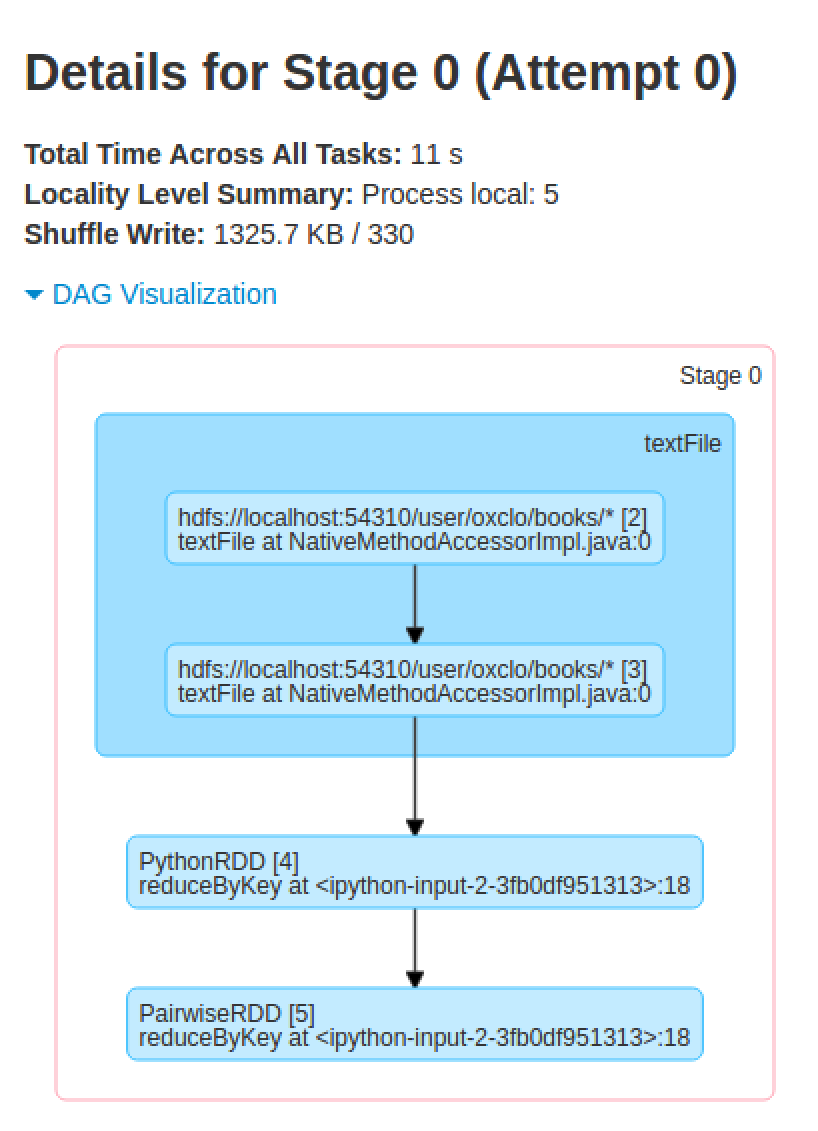
http://localhost:8888/?token=fd655aab32ed4840ceb47b8b7392b1243a27f56350888a91

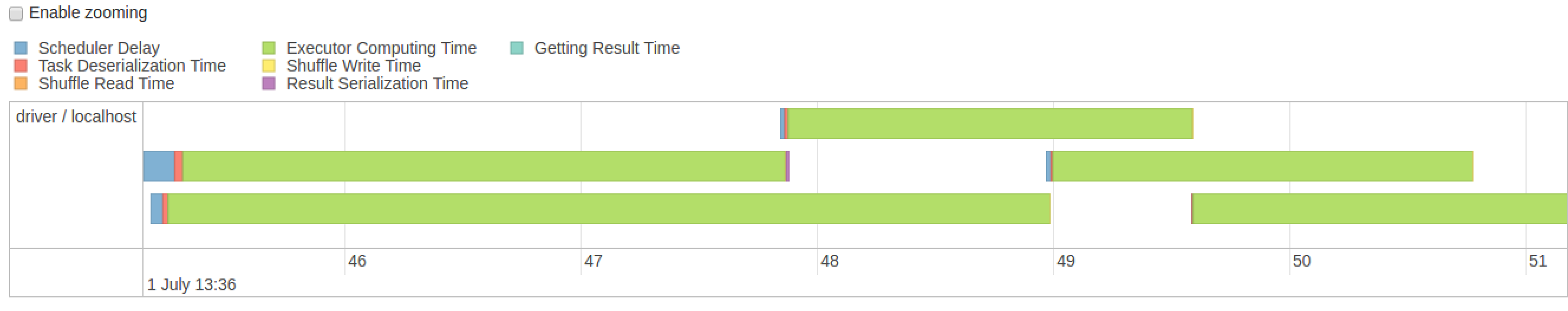
1. And then a browser window will pop up.  
   
2. Use the **New** button to create a new Python2 notebook:  
   
3. There is a starter of the code you need in the following URL:   
   <https://freo.me/first-notebook>   
     
   Paste that into the cell [1] so it looks like this:



1. There are some aspects that are not filled in that you need to write.   
   Basically this is a data-processing pipeline (also a directed acyclic graph)
2. *Let’s look at the parts that are there already.*
3. We already have a SparkContext object defined in the notebook (in a program you need to define one, which we will see later)
4. Unfortunately some of the input is handled as Unicode by Python and we want to get rid of that.  
   import unicodedata  
   def u2a(u): return str(unicodedata.normalize('NFKD',u).  
   encode('ascii','ignore'))
5. We also want to remove any non-alphanumeric characters:  
     
   def strip(s): return ''.join(filter(str.isalpha, s))
6. With the preliminaries over, the next line loads the data in:  
   books = sc.textFile("hdfs://localhost:54310/user/oxclo/books/\*")
7. Then splits the lines into separate words  
     
   split = books.flatMap(lambda line: line.split())
8. Deals with the Unicode problem  
   asc = split.map(u2a)  
     
   And removes non-alpha characters  
   stripped = asc.map(strip)  
     
   and removes empty items:  
   notempty = stripped.filter(lambda w: len(w)>0)
9. Now it is time for you to do something!  
     
   Convert all the words to lower case, using a map operation. In python, if *str* is a string, then str.lower() is the same string in lower case.
10. Now you need to get ready for a reduce. In order to do a reduce, we need some form of *key, value* pairs. I recommend using *tuples*  which are simply (k,v) in Python (the brackets group the items into a tuple).
11. Remembering how reduce works, we need each word to have a count. Before reducing, that count is 1. So we need a lambda that takes a word w and returns (w,1)
12. Now we can do a reduce that adds all those counts together.
13. Finally, we need to collect the results and print them. In Spark, they may be distributed across different RDD partitions on different machines, so the collect() method brings them together.  
      
    for k,v in wordcount.collect(): print k,v
14. Try running the cell, by clicking 
15. Be patient. I suggest you look at the command window and wait until you see spark start working.
16. You should see a word count appear below cell 1:



1. Congratulations!
2. While the pyspark is still running browse to <http://localhost:4040>
3. You will see the Spark web console:  
   
4. Click on the blue link “collect at ipython-input”  
   This shows you how Spark converted your code into stages:  
   
5. Click on Stage 0  
   
6. And expand the Event Timeline:



1. Make sure your code is saved from the notebook.
2. Quit the notebook shell by typing Ctrl-C on the command line, and then Y  
   Also close the notebook windows in the browser.
3. Now let’s run the same code as a “job” instead of interactively.
4. From <http://freo.me/oxclo-wc-py> copy the code into a file wc-job.py
5. You will notice that there is a bunch of “setup” code that we didn’t need in the pyspark command line tool. That is because pyspark assumes you want all this and does it for you.
6. We run jobs locally on a single node directly on Spark:  
   The local[\*] indicates to use as many threads as you have cores on your system:  
   ~/spark/bin/spark-submit --master local[\*] wc-job.py "hdfs://localhost:54310/user/oxclo/books/\*"
7. Congratulations, the lab is complete!  
     
   **Extension**
8. Re load the code into the Jupyter notebook and now improve it to show the wordcount in descending order, starting with the most common words.

1. Read Eval Print Loop [↑](#footnote-ref-1)