

A

Installing Apache Spark

Starting with Apache Spark can be intimidating. However, after you have gone through the process of installing it on your local machine, in hindsight, it will not look so scary.

In this chapter, we will guide you through the requirements of Spark 2.0, the installation process of the environment itself, and through setting up the Jupyter notebook so that it is convenient and easy to write your code.

The topics covered are:

- Requirements
- Installing Spark
- Jupyter on PySpark
- Installing in the cloud

Requirements

Before we begin, let's make sure your computer is ready for Spark installation. What you need is Java 7+ and Python 2.6+/3.4+. Spark also requires R 3.1+ if you want to run R code. For the Scala API, Spark 2.0.0 Preview uses Scala 2.11. You will need to use a compatible Scala version (2.11.x).

Spark installs Scala during the installation process, so we just need to make sure that Java and Python are present on your machine.



Throughout this book we will be using Mac OS X El Capitan, Ubuntu as our Linux flavor, and Windows 10; all the examples presented should run on either of these machines.

Checking for presence of Java and Python

On a Unix-like machine (Mac or Linux) you need to open **Terminal** (or **Console**), and on Windows you need to open Command Line (navigate to **Start | Run | cmd** and press the *Enter* key).



Throughout this book we will refer to Terminal, Console, or Command Line as **CLI**, which stands for a **Command Line Interface**.



Once the window opens, type the following:

```
java -version
```

If the command prints out something like this:

```
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
```

It means you have Java present on your machine. In the preceding case, we are running Java 8, so we meet the first criterion. If, however, executing the preceding command returns an error on Mac or Linux, it might look similar to the following:

```
-bash: java: command not found
```

Or, on Windows, it might resemble the following error:

```
'java' is not recognized as an internal or external command, operable
program or batch file
```

This means that either Java is not installed on your machine, or it is not present in the PATH.

PATH is an environment variable that a CLI checks for binaries. For example, if you type the `cd` (change directory) command and try to execute in the CLI, your system will scan the folders listed in the PATH, searching for the `cd` executable and, if found, will execute it; if the binary cannot be found, the system will produce an error.



To learn more about what the PATH variable does, go to http://www.tldp.org/paths_env_var.html for more information.



If you are sure you have Java installed (or simply do not know) you can try locating Java binaries. On Linux you can try executing the following command:

```
locate java
```

Check the `/usr/lib/jvm` location for a `jvm` folder.



Refer to your flavor of Linux documentation to find an equivalent method or an exact location of the `jvm` folder.



On Mac, check the `/Library/Java/JavaVirtualMachines/` location for a `jdk` or `jre` folder; on Windows you can navigate to `C:\Program Files (x86)\` and check for the `Java` folder. If the preceding efforts fail, you will have to install Java (see the following section, *Installing Java*).

In a similar fashion to how we checked for Java, let's now check whether Python is present on your machine. In your CLI type, use the following command:

```
python --version
```

If you have Python installed, the Terminal should print out its version. In our case, this is:

```
Python 3.5.1 :: Anaconda 2.4.1 (x86_64)
```

If, however, you do not have Python, you will have to install a compatible version on your machine (see the following section, *Installing Python*).

Installing Java

It goes beyond the scope of this book to provide detailed instructions on how you should install Java. However, it is a fairly straightforward process and the high-level steps you need to undertake are:

1. Go to https://www.java.com/en/download/mac_download.jsp and download the version appropriate for your system.
2. Once downloaded, follow the instructions to install on your machine.

That is effectively all you have to do.

If you run into trouble, check https://www.java.com/en/download/help/mac_install.xml for help on how to install Java on Mac.

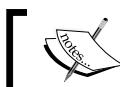
Check https://www.java.com/en/download/help/ie_online_install.xml for steps outlining the installation process on Windows.

Finally, check https://www.java.com/en/download/help/linux_install.xml for Linux installation instructions.

Installing Python

Our preferred flavor of Python is Anaconda (provided by Continuum) and we strongly recommend this distribution. The package comes with all the necessary and most commonly used modules included (such as pandas, NumPy, SciPy, or Scikit, among many others). If a module you want to use is not present, you can quickly install it using the conda package management system.

The Anaconda environment can be downloaded from <https://www.continuum.io/downloads>. Check the correct version for your operating system and follow the instructions presented to install the distribution.



Note that, for Linux, we assume you install Anaconda in your HOME directory.



Once downloaded, follow the instructions to install the environment appropriate for your operating system:

- For Windows, see <https://docs.continuum.io/anaconda/install#anaconda-for-windows-install>
- For Linux, see <https://docs.continuum.io/anaconda/install#linux-install>
- For Mac, see <https://docs.continuum.io/anaconda/install#anaconda-for-os-x-graphical-install>

Once both of the environments are installed, repeat the steps from the preceding section, *Checking for presence of Java and Python*. Everything should work now.

Checking and updating PATH

If, however, your CLI still produces errors, you will need to update the PATH. This is necessary for CLI to find the right binaries to run Spark.

Setting the PATH environment variable differs between Unix-like operating systems and Windows. In this section, we will walk you through how to set these properly in either of these systems.

Changing the PATH on Linux and Mac

First, open your `.bash_profile` file; this file allows you to configure your bash environment every time you open the CLI.



To learn what bash is, check out the following link https://www.gnu.org/software/bash/manual/html_node/What-is-Bash_003f.html

We will use `vi` text editor in CLI to do this, but you are free to choose a text editor of your liking:

```
vi ~/.bash_profile
```



If you do not have the `.bash_profile` file present on your system, issuing the preceding command will create one for you. If, however, the file already exists, it will open it.

We need to add a couple of lines, preferably at the end of the file. If you are using `vi`, press the `I` key on your keyboard (that will initiate the edit mode in `vi`), navigate to the end of the file, and insert a new line by hitting the `Enter` key. Starting on the new line, add the following two lines to your file (for Mac):

```
export PATH=/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home/bin:$PATH  
export PATH=/Library/Frameworks/Python.framework/Versions/3.5/bin:$PATH
```

On Linux:

```
export PATH=/usr/lib/jvm/java-8-sun-1.8.0.40/jre/bin/java:$PATH  
export PATH=$HOME/anaconda/bin:$PATH
```



Note that we are referring to Java 1.8 update 40 above. Also, note that, on Mac, there are only two lines; due to space constraints they appear as four lines, as the words `Contents` and `bin` were wrapped at the end of the line.

Once done typing hit the `Esc` key and type the following command:

```
:wq
```

Once you hit the `Enter` key, `vi` will write and quit.

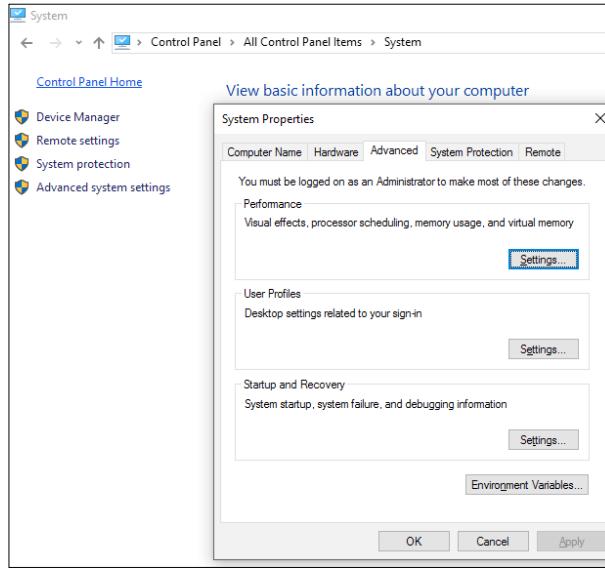


Do not forget the colon at the beginning of the `:wq` as it is necessary.

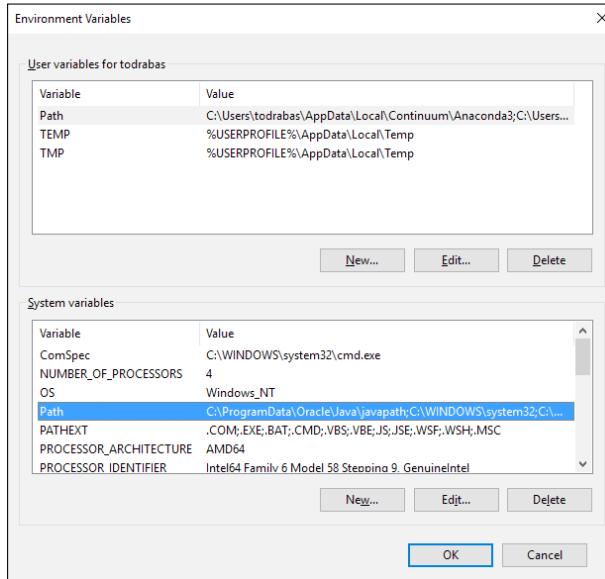
You will need to restart CLI for the changes to be reflected.

Changing PATH on Windows

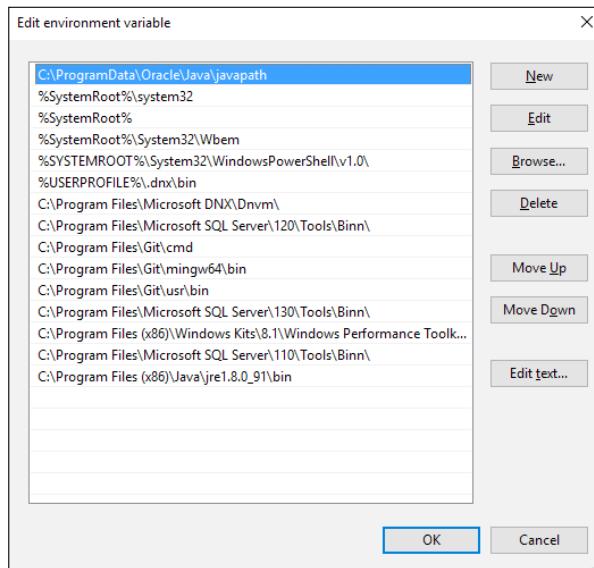
First, navigate to **Control Panel** and click on **System**. You should see a screen similar to the one shown in the following screenshot:



Click on **Environment Variables** and in the **System variables** search for **Path**. Once found, click on **Edit**:



In the new window that opens click **New** and then **Browse**. Navigate to C:\Program Files (x86)\Java\jre1.8.0_91\bin and click **OK**:



Once done, close the window by clicking on **OK**.

Next, check if under **User variables** for <your login> (where <your login> is the name of your account, such as **todrabas** in the preceding example) there exists a variable **Path** and if it lists any reference to Anaconda (it does in our preceding example).

However, if not then select the **PATH** and click **Edit**. Next, similarly to how we added the **Java** folder earlier, click **New** and then **Browse**. Now, navigate to C:\Users\\AppData\Local\Continuum\Anaconda3\Library\bin and click **OK**.

Once this is done, continue clicking the **OK** button until the **System** window closes.

Finally, open CLI and type the following command:

```
echo %PATH%
```

Your newly added folders should be listed there.

As a final step loop back to the *Checking for presence of Java and Python* section and check if both Java and Python can now be accessed.

Installing Spark

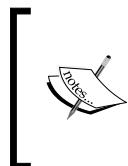
Your machine is now ready to install Spark. You can do this in two ways:

1. Download source codes and compile the environment yourself; this gives you the most flexibility.
2. Download pre-built binaries.
3. Install PySpark libraries through PIP (see here: <http://bit.ly/2ivVhbH>)

The following instructions for Mac and Linux guide you through the first way. We will show you how to configure your Windows machine while showcasing the second option of installing Spark.

Mac and Linux

We describe these two systems together as they both are Unix-like systems: Mac OS X's kernel (called Darwin) is based on BSD, while the Linux kernel borrows heavily from the Unix-world functionality and security.



Check <https://developer.apple.com/library/mac/documentation/Darwin/Conceptual/KernelProgramming/Architecture/Architecture.html> or <http://www.ee.surrey.ac.uk/Teaching/Unix/unixintro.html> for more information if you feel so inclined.



Downloading and unpacking the source codes

First, go to <http://spark.apache.org/downloads.html> and go through the following steps:

1. **Choose a Spark release:** 2.1.0 (Dec 28, 2016). Note that at the time you read this, the version might be different; simply select the latest one for Spark 2.0.
2. **Choose a package type:** Source code.
3. **Choose a download type:** Direct download.
4. Click on the link next to **Download Spark:** It should state something similar to `spark-2.1.0.tgz`.

Once the download finishes, go to your CLI and navigate to the folder you have downloaded the file to; in our case it is `~/Downloads/`:

```
cd ~/Downloads
```



The tilde sign ~ denotes your home folder on both Mac and Linux.



To confirm the authenticity and completeness of the file, in your CLI, type the following (on Mac):

```
md5 spark-2.1.0.tgz
```

Or use this code:

```
md5sum spark-2.1.0.tgz
```

On Linux, it should produce a long string of numbers and letters; on our machine it looks like this:

```
MD5 (spark-2.1.0.tgz) = 80029fb4e3ff8Cf500ff7baf2728daf9
```

You can then compare this with the corresponding md5 checksum provided by Spark:
<http://www.apache.org/dist/spark/spark-2.1.0/spark-2.1.0.tgz.md5>

Next, we need to unpack the archive. This can be achieved with the following command:

```
tar -xvf spark-2.1.0.tgz
```

The -xvf options of the tar command makes it easy to extract the archive (the x part) and produce a verbose output (the v option) from a file (the f) that we specified.

Installing the package

Now, let's install the package. Go to the directory to the unpacked Spark codes:

```
cd spark-2.1.0
```

We will be building Spark with Maven and sbt, which we will later use to package up our applications deployed in the cloud.



Maven is a build automation system that is used to install Spark. You can read more at <https://maven.apache.org>. sbt stands for scala build tool and it is an incremental compiler for Scala. Scala is a scalable programming language (that is where its name comes from: Scalable Language). The code written in Scala compiles to a Java-bytecode, so it can run in **Java Virtual Environment (JVM)**. For more information, check out the following link <http://www.scala-lang.org/what-is-scala.html>.

Installing with Maven

You do not need to install it explicitly as Spark source codes ship with mvn located in the build folder; that will get us started.

First, we need to change default memory settings for Maven so it allocates more memory for the installer to use. You attain this with executing the following command in your CLI (everything in one line):

```
export MAVEN_OPTS="-Xmx2g -XX:MaxPermSize=512M -XX:ReservedCodeCacheSize=512m"
```

Also, you need to have the JAVA_HOME system variable specified properly and pointing to where your Java JDK distribution is installed. This can be done with the help of the following command on Mac:

```
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_40.jdk/Contents/Home
```

Or the following command on Linux:

```
export JAVA_HOME=/usr/lib/jvm/open-jdk
```



Note that your distribution locations might be different, so you will have to adapt the preceding commands to your system.



Having the Maven options and the JAVA_HOME environment variable set we can proceed to build Spark.

We will build Spark with Hadoop 2.7 and Hive. Execute the following command in your CLI (again, everything in one line):

```
./build/mvn -Pyarn -Phadoop-2.7 -Dhadoop.version=2.7.0 -Phive -Phive-thriftserver -DskipTests clean package
```

Once you issue the preceding command the installer will download Zinc, Scala, and Maven.



Zinc is a standalone version of sbt's incremental compiler.



If everything goes well, you should see a screen similar to the following screenshot:

```
[INFO] -----
[INFO] Reactor Summary:
[INFO]
[INFO] Spark Project Parent POM ..... SUCCESS [ 2.612 s]
[INFO] Spark Project Tags ..... SUCCESS [ 5.155 s]
[INFO] Spark Project Sketch ..... SUCCESS [ 6.345 s]
[INFO] Spark Project Networking ..... SUCCESS [ 8.141 s]
[INFO] Spark Project Shuffle Streaming Service ..... SUCCESS [ 4.775 s]
[INFO] Spark Project Unsafe ..... SUCCESS [ 6.784 s]
[INFO] Spark Project Launcher ..... SUCCESS [ 7.271 s]
[INFO] Spark Project Core ..... SUCCESS [01:50 min]
[INFO] Spark Project ML Local Library ..... SUCCESS [ 6.066 s]
[INFO] Spark Project GraphX ..... SUCCESS [ 11.841 s]
[INFO] Spark Project Streaming ..... SUCCESS [ 24.800 s]
[INFO] Spark Project Catalyst ..... SUCCESS [ 59.887 s]
[INFO] Spark Project SQL ..... SUCCESS [01:21 min]
[INFO] Spark Project ML Library ..... SUCCESS [01:02 min]
[INFO] Spark Project Tools ..... SUCCESS [ 0.886 s]
[INFO] Spark Project Hive ..... SUCCESS [ 38.901 s]
[INFO] Spark Project REPL ..... SUCCESS [ 3.463 s]
[INFO] Spark Project YARN Shuffle Service ..... SUCCESS [ 5.193 s]
[INFO] Spark Project YARN ..... SUCCESS [ 8.081 s]
[INFO] Spark Project Hive Thrift Server ..... SUCCESS [ 16.256 s]
[INFO] Spark Project Assembly ..... SUCCESS [ 2.667 s]
[INFO] Spark Project External Flume Sink ..... SUCCESS [ 4.421 s]
[INFO] Spark Project External Flume ..... SUCCESS [ 9.387 s]
[INFO] Spark Project External Flume Assembly ..... SUCCESS [ 2.294 s]
[INFO] Spark Integration for Kafka 0.8 ..... SUCCESS [ 8.363 s]
[INFO] Spark Project Examples ..... SUCCESS [ 14.318 s]
[INFO] Spark Project External Kafka Assembly ..... SUCCESS [ 3.098 s]
[INFO] Spark Integration for Kafka 0.10 ..... SUCCESS [ 6.825 s]
[INFO] Spark Integration for Kafka 0.10 Assembly ..... SUCCESS [ 2.987 s]
[INFO] Kafka 0.10 Source for Structured Streaming ..... SUCCESS [ 7.260 s]
[INFO] Spark Project Java 8 Tests ..... SUCCESS [ 3.987 s]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 08:57 min
[INFO] Finished at: 2017-01-15T16:29:36-08:00
[INFO] Final Memory: 92M/952M
[INFO] -----
```

Installing with sbt

Installing Spark with sbt drills down to executing the following command:

```
./build/sbt -Pyarn -Phadoop-2.7 package
```

If all goes well, you will see a final screen similar to this:

```
[info] Packaging /Users/drabast/Downloads/spark-2.1.0/examples/target/scala-2.11/jars/spark-examples_2.11-2.1.0.jar ...
[info] Done packaging.
[success] Total time: 238 s, completed Jan 16, 2017 8:40:00 PM
```

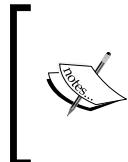
Testing the environment

Now that we have successfully built the environment, let's test it. Run the following commands in your CLI:

```
./build/mvn -DskipTests clean package -Phive
./python/run-tests --python-executables=python
```

The preceding commands will clean up the installation and run tests of all modules of PySpark (since we execute the `run-tests` inside the `python` folder). If you want to test only a specific module of PySpark, you can use the following command:

```
./python/run-tests --python-executables=python --modules=pyspark-sql
```



If you run the `run-tests` command without the `--python-executables` switch, the environment will use the default Python installation (for us it was Python 2.6). This might generate some errors, so we would recommend you explicitly specify the `--python-executables` option.

Once the tests finish, assuming all went well, you should see a screen similar to the following screenshot:

```
Running PySpark tests. Output is in /Users/drbast/Downloads/spark-2.1.0/python/unit-tests.log
Will test against the following Python executables: ['python']
Will test the following Python modules: ['pyspark-core', 'pyspark-ml', 'pyspark-mllib', 'pyspark-sql', 'pyspark-streaming']
Finished test(python): pyspark.sql.tests (63s)
Finished test(python): pyspark.accumulators (8s)
Finished test(python): pyspark.broadcast (5s)
Finished test(python): pyspark.conf (4s)
Finished test(python): pyspark.context (19s)
Finished test(python): pyspark.ml.classification (30s)
Finished test(python): pyspark.tests (140s)
Finished test(python): pyspark.ml.clustering (23s)
Finished test(python): pyspark.ml.evaluation (14s)
Finished test(python): pyspark.ml.linalg._init__ (0s)
Finished test(python): pyspark.ml.recommendation (18s)
Finished test(python): pyspark.ml.feature (31s)
Finished test(python): pyspark.streaming.tests (187s)
Finished test(python): pyspark.ml.regression (25s)
Finished test(python): pyspark.ml.tuning (23s)
Finished test(python): pyspark.mllib.tests (214s)
Finished test(python): pyspark.mllib.classification (26s)
Finished test(python): pyspark.mllib.evaluation (20s)
Finished test(python): pyspark.mllib.feature (26s)
Finished test(python): pyspark.mllib.clustering (42s)
Finished test(python): pyspark.mllib.linalg._init__ (0s)
Finished test(python): pyspark.mllib.fpm (21s)
Finished test(python): pyspark.mllib.random (10s)
Finished test(python): pyspark.ml.tests (89s)
Finished test(python): pyspark.mllib.stat.KernelDensity (0s)
Finished test(python): pyspark.mllib.recommendation (27s)
Finished test(python): pyspark.mllib.linalg.distributed (31s)
Finished test(python): pyspark.mllib.regression (27s)
Finished test(python): pyspark.mllib.stat._statistics (14s)
Finished test(python): pyspark.mllib.util (11s)
Finished test(python): pyspark.profiler (9s)
Finished test(python): pyspark.mllib.tree (17s)
Finished test(python): pyspark.shuffle (1s)
Finished test(python): pyspark.serializers (15s)
Finished test(python): pyspark.rdd (21s)
Finished test(python): pyspark.sql.conf (5s)
Finished test(python): pyspark.sql.catalog (18s)
Finished test(python): pyspark.sql.column (19s)
Finished test(python): pyspark.sql.context (21s)
Finished test(python): pyspark.sql.group (34s)
Finished test(python): pyspark.sql.dataframe (39s)
Finished test(python): pyspark.sql.functions (41s)
Finished test(python): pyspark.sql.types (9s)
Finished test(python): pyspark.sql.window (5s)
Finished test(python): pyspark.streaming.util (0s)
Finished test(python): pyspark.sql.readwriter (33s)
Finished test(python): pyspark.sql.session (16s)
Tests passed in 372 seconds
```

Moving the environment

Once all the tests have passed, let's move our Spark environment away from its temporary location in the ~/Downloads folder to our home folder and rename it Spark:

```
cd ~/Downloads  
mv -r spark-2.1.0 ~/Spark
```

Once this is done, you will be able to run Spark from your directory.

First run

Now you can try running `pyspark`. Let's navigate to `~/Spark/bin` and start the interactive shell by using the following command:

```
cd ~/Spark/bin  
./pyspark
```

You should see something similar to this:

You can test the environment by typing the following:

```
print(sc.version)
```

The output should read:

2.1.0

The `sc` is `SparkContext` that is automatically created for you when PySpark starts. Initializing a PySpark session creates `sqlContext` as well. We will get to describing what these are later in this book.

To exit the pyspark session, type `quit()`.

Windows

Installing Spark on Windows is also fairly straightforward. However, as mentioned earlier, instead of building the whole environment from scratch, we will download a precompiled version of Spark.

Downloading and unpacking the archive

Go to <http://spark.apache.org/downloads.html> and select the following:

1. **Choose a Spark release:** 2.1.0 (Dec 28, 2016). Note that, when you read this, the version might be different; simply select the latest one for Spark 2.0.
2. **Choose a package type:** Pre-built for Hadoop 2.7 and later.
3. **Choose a download type:** Direct download.
4. Click on the link next to **Download Spark:** it should state something similar to `spark-2.1.0-bin-hadoop2.7.tgz`.

In order to check the integrity and completeness of your download, go to <https://www.microsoft.com/en-us/download/details.aspx?id=11533>, download the setup file, and install it. Once installed, go to CLI, navigate to your Downloads folder, and use the following tool:

```
cd C:\Users\<your login>\Downloads  
fciv -md5 spark-2.1.0-bin-hadoop2.7.tgz
```

This should produce a string of random looking letters and numbers similar to the following one:

```
50E73F255F9BDE50789AD5BD657C7A71 spark-2.1.0-bin-hadoop2.7.tgz
```

You can compare the cited number with the corresponding `md5` checksum found here <http://www.apache.org/dist/spark/spark-2.1.0/spark-2.1.0-bin-hadoop2.7.tgz.md5>.



Check this video for step-by-step instructions on how to install and use the `fciv` tool <https://www.youtube.com/watch?v=G08xum0AuFg>

Let's unpack the archive now. If you have 7-Zip or another unarchiver that can handle `.tgz` archives, you are ready to go. However, if you do not have an unarchiver we suggest you go to <http://www.7-zip.org>, download the version of 7-Zip that is compatible with your system, and install it.

Once installed, go to (most likely) c:\Users\<your login>\Downloads and right-click on the archive; a menu option **7-Zip** should now have been added. Go to the **7-Zip- | Extract here** option. Note that this will be a two-step process: first, you extract the .tar archive from the .tgz file, and then follow the same process to extract the Spark folder from the freshly extracted .tar archive.

Once done, open the CLI and navigate to the folder you have just extracted the Spark binaries to (we assume it is C:\Users\<your login>\Downloads\spark-2.1.0-bin-hadoop2.7). Inside the folder, go to the bin folder and type pyspark:

```
cd C:\Users\<your login>\Downloads\spark-2.1.0-bin-hadoop2.7\bin  
pyspark
```

Once you hit the *Enter* key, you should see a screen similar to the following screenshot:

Note that, while running Spark locally, Hadoop installation is not required. However, even though Spark might print some error messages when starting up, complaining about not finding Hadoop binaries, it will still execute your PySpark code. Spark, according to the FAQ from <http://spark.apache.org>, requires Hadoop (or any other distributed file system) only when you deploy Spark on a cluster; running locally, it is not necessary and the error can be ignored.

Jupyter on PySpark

Jupyter is a convenient and powerful shell for Python where you can create notebooks with embedded code. Jupyter's notebooks allow you to include regular text, code, and images, and you can also create tables or use the LaTeX typesetting system. All in one place, running above Python, it is a really convenient way of writing your applications, where you essentially keep your thoughts, documentation, and code in one place.

If you have never used Jupyter, to bring you up to speed, check out the Jupyter documentation: <http://jupyter-notebook.readthedocs.io/en/latest/examples/Notebook/Notebook%20Basics.html> to learn how to navigate Jupyter notebooks.

Also, a word of warning is warranted here: JVM log messages from Spark are not currently passed to Jupyter, so for debugging, you will have to go back to CLI to read detailed debug messages.

Installing Jupyter

If you run the Anaconda distribution of Python, you can easily install Jupyter by running the following command:

```
conda install jupyter
```

The command will install all the necessary modules Jupyter depends on, as well as Jupyter itself. If, however, you do not run Anaconda, follow the instructions on <http://jupyter.readthedocs.io/en/latest/install.html> to install Jupyter manually.

Throughout this book, we will be using Jupyter exclusively to develop and run our PySpark applications, so it is vital that you install Jupyter.

Setting the environment

Once you have Jupyter installed on your machine, let's get it to work with PySpark. Getting it done, though, is a bit tricky.

We need to add the Spark environment's `bin` folder to the `PATH` environment variable and set a couple of new variables.

Mac and Linux

First, let's again open your `.bash_profile` file:

```
vi ~/.bash_profile
```

Then, add the following lines:

```
export PATH=$HOME/Spark/bin:$PATH
export PYSPARK_PYTHON=$HOME/anaconda/bin/python3
export PYSPARK_DRIVER_PYTHON=jupyter
export PYSPARK_DRIVER_PYTHON_OPTS='notebook' pyspark
```

In the first line, we allow the bash environment to find the newly compiled binaries (`pyspark` lives in the `~/Spark/bin` folder). So, now, you can simply type the following:

```
pyspark
```

Instead of the following:

```
~/Spark/bin/pyspark
```

Every time you want to start `pyspark`.

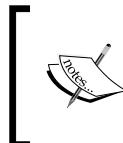
Adding the `PYSPARK_PYTHON` variable ensures that executors also run Python 3.5 (instead of the default Python 2.7).

Setting the `PYSPARK_DRIVER_PYTHON` environment variable to `jupyter` instructs the system to, instead of running a PySpark interactive shell in the command line, to start a Jupyter instance.

The `PYSPARK_DRIVER_PYTHON_OPTS` variable instructs Jupyter to pass the `notebook` parameter to Jupyter and link it with a new instance of `pyspark`.

Windows

Following the same way we added and changed the environment variables earlier (see the *Changing PATH on Windows* section), change the `PATH` variable to point to your Spark distribution's `bin` folder. Next, add `PYSPARK_DRIVER_PYTHON` and set its value to `jupyter`.



There is a small difference on Windows with regards to the last variable: the `PYSPARK_DRIVER_PYTHON_OPTS` variable should only be set to '`notebook`', that is, you need to omit the `pyspark` at the end.

Starting Jupyter

Now, every time you type `pyspark` in CLI, a new instance of Jupyter and PySpark will be created, and your default browser will launch with the starting screen from Jupyter:



HelloWorld from PySpark

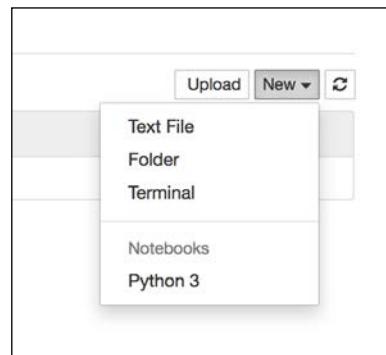
Jupyter, in the background, starts a local web server that allows you to visually navigate your notebooks. Let's create our first example that will be our own version of the required 'helloWorld' example.



The code for this book is located on GitHub: <https://github.com/drabastomek/learningPySpark>. Go to Chapter02 for the Jupyter notebook for this example.



First, click on **New** and select the **Python 3** notebook:



It will open a notebook in another tab in your browser, as shown in the following screenshot:



Now we can start coding. Type `sc` in the first cell and hit the *Alt + Enter* keys (or the *[Alt-Option + Enter]* keys on Mac). This will execute the command and create a new cell for our following code. The output of the command should look similar to the following screenshot:

```
Out[1]: <pyspark.context.SparkContext at 0x1050456a0>
```

Remember that starting `pyspark` automatically creates the `SparkContext` object (and aliases it as `sc`) as well as `SQLContext` (aliased as `sqlContext`). Let's see if `sqlContext` has properly started: in the new cell type `sqlContext` and execute. The notebook should return something similar to the following screenshot:

```
Out[2]: <pyspark.sql.context.SQLContext at 0x10b832a58>
```

Now we know all is running well.

Similarly to our example from the CLI PySpark interactive shell, let's print the version of PySpark we're running: type `print (sc.version)` and execute. This should result in the following:

```
2.1.0
```

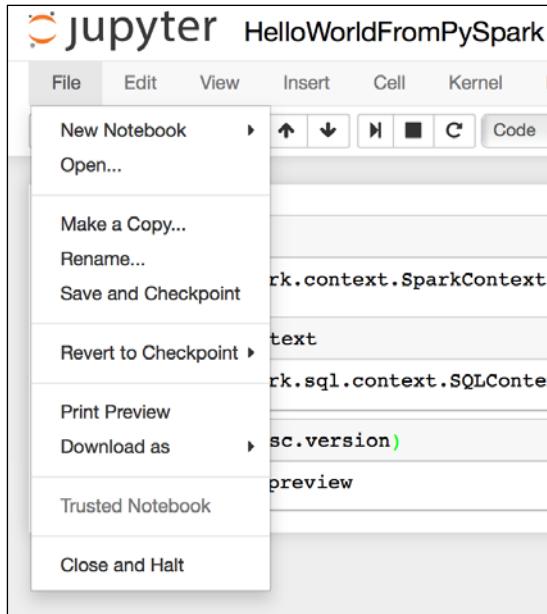
We are in business!

As a last step, let's rename our notebook as it normally starts with the 'Untitled' title. Navigate to **File | Rename...** and change the name to your liking; we changed it to `HelloWorldFromPySpark`; this automatically changes the filename for you as well.

To stop the notebook (what you should do every time you want to finish working with a notebook) you go to **File** and click on the **Close and Halt** option. What this does is it closes the notebook, but it also stops the Python kernel from releasing the memory.

Installing in the cloud

If you do not want to install Spark locally on your computer, you can jump to *Bonus Chapter B, Free Spark Cloud Offering*, where we present how to sign up for the Community Edition of Spark on Databricks' cloud, or how to set up your own clusters on Microsoft's Azure or Amazon AWS/EMR.



Summary

In this chapter, we walked you through the (sometimes painful) process of setting up your local Spark environment. We showed you how to check whether two required environments (Java and Python) were present on your machine. We also provided some guidance on how to install them on your system if these two packages were missing.

Even though the process of installing Spark itself might be intimidating at times, we hope, with our help, you were able to successfully install the engine and execute the minimal code presented in this chapter. At this point, you should be able to run code in Jupyter notebook.