Installing and Configuring PySpark with Docker

[(https://github.com/learn-co-curriculum/dsc-spark-docker-installation)](https://github.com/learn-co-curriculum/dsc-spark-docker-installation) [(https://github.com/learn-co-curriculum/dsc-spark-docker-installation/issues/new)](https://github.com/learn-co-curriculum/dsc-spark-docker-installation/issues/new)

Introduction

In a large-scale enterprise environment, you would typically run (Py)Spark on a distributed cloud system or possibly a dedicated hardware in a datacenter. However it is also possible to run it on your local computer as a standalone cluster. In this lesson we'll walk through how to do just that!

Objectives

You will be able to:

- Understand how to install PySpark on your local computer
- Explain the utility of Docker when dealing with package management
- Install a Docker container that comes packaged with Spark

Installing PySpark without Docker

It is possible to install PySpark without Docker, but it will require more-advanced systems administration skills. Follow these instructions if you want to get the best possible Spark performance from your personal computer, but be aware that **you can feel free to skip this** because there are a lot of little places that things can go wrong, and it can be difficult to troubleshoot.

Creating a New `conda` Environment

If you want to work on a project using PySpark, we recommend that you make a new `conda` environment. Execute the following commands in the terminal:

```
conda activate base
```

```
conda create --name spark-env python=3.8
```

```
conda activate spark-env
```

Checking for Successful Environment Creation

Run this command in the terminal:

```
which python
```

Make sure that the path displayed includes `spark-env` before proceeding to the next step. If it doesn't, try `conda deactivate` repeatedly until no environment is shown, then `conda activate spark-env` again.

Installing Java

With `spark-env` activated, execute this line in the terminal:

```
conda install -c conda-forge openjdk=11
```

This installs OpenJDK, an open-source version of Java, within your `conda` environment.

Checking for Successful Java Installation

Run this command in the terminal:

```
which java
```

Make sure the path displayed includes `spark-env` .

Launch an interactive **Java shell** by running:

```
jshell
```

This should launch a CLI application that displays a `jshell>` prompt.

If you want to write some Java code here, you can! Or you can just quit the Java shell by typing `/exit` and hitting Enter.

Installing PySpark

With `spark-env` activated, execute this line in the terminal:

```
pip install pyspark==3
```

This installs a setup for a standalone Spark cluster as well as the PySpark library to interact with that cluster.

Checking for Successful PySpark Installation: Spark

First, check that Spark installed successfully by launching an interactive **Spark shell**:

```
spark-shell
```

This should launch a CLI application that displays a `scala>` prompt. It's normal for this to take several seconds, and also to print out several lines of warnings, such as `WARNING: Illegal reflective access` , as well as some fun ASCII art:

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\
      /_/
```

Try typing `sc` and hitting Enter to see the string representation of the SparkContext in Scala. Then quit the Scala shell by typing `:quit` and hitting Enter.

Checking for Successful PySpark Installation: PySpark

Then, check that PySpark installed successfully by launching an interactive **PySpark shell**. Run this command in the terminal:

```
pyspark
```

This should launch a CLI application that displays a `>>>` prompt, as well as the same warnings and ASCII art as the `spark-shell` did. Try typing `sc` and hitting Enter to see the string representation of the SparkContext in Python (and you can test out any other random Python commands). Then quit the PySpark shell by typing `quit()` and hitting Enter.

Installing Jupyter Notebook and Other Useful Libraries

In theory `spark-env` already has everything you need to start developing PySpark code! But there are a couple more tools that we typically use for data science that you probably want to install as well.

With `spark-env` activated, run these commands in the terminal to install Jupyter Notebook and Matplotlib. Afterwards you can install any other libraries you like to use as well.

```
conda install -c conda-forge notebook
```

```
python -m ipykernel install --user --name spark-env --display-name "Python (spark-env)"
```

```
conda install matplotlib
```

Running a PySpark Notebook Locally

Now you can clone this notebook and run `jupyter notebook` to start it in your `spark-env`. The three cells below should run with no errors (although there may be warnings):

```
import pyspark
```

```
sc = pyspark.SparkContext('local[*]')
```

```
22/03/07 20:38:37 WARN NativeCodeLoader: Unable to load native-hadoop library for your pla
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(newLeve
```

```
rdd = sc.parallelize(range(1000))
rdd.takeSample(False, 5)
```

```
[597, 803, 304, 458, 603]
```

Installing PySpark with Docker

If you're lucky, all of the commands above ran smoothly and you are now able to use PySpark directly on your machine using `conda`. However we recognize that this process frequently goes wrong, and troubleshooting often involves some tricky systems administration tasks, configuring environment variables such as `$PATH`, `$JAVA_PATH`, etc.

In the rest of this lesson, we'll describe an alternative way to install PySpark using **Docker**.

Why Docker?

Docker is a container technology that allows **packaging** and **distribution** of software so that it takes away the headache of things like setting up an environment, configuring logging, configuring options, etc. Docker basically removes the excuse of *It doesn't work on my machine*.

[Visit this link learn more about docker and containers (https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/)](https://www.zdnet.com/article/what-is-docker-and-why-is-it-so-darn-popular/)

How Does Docker Work?

Without getting too much into the details of virtualization and underlying operating systems, a Docker **image** is deployed in a **container** where it essentially operates as a self-contained computer wherever it is run. It can be running on a Windows desktop, a Mac laptop, or an AWS cloud server, and the dependencies and environment should work exactly the same.

Kind of like the `.yml` file we used to create `learn-env`, Docker containers have a static specification that tells the software what to install. Only instead of just Python packages, the **Dockerfile** (as it's called) specifies the operating system, shell language, permissions, environment variables, etc.

The Dockerfile we'll be using is for an image maintained by Jupyter called `pyspark-notebook`. You can view the full Dockerfile **here (https://github.com/jupyter/docker-stacks/blob/master/pyspark-notebook/Dockerfile)**.

In order to turn that file into a functioning container, we need to install Docker.

Installing Docker

Go to the **Get Docker (https://docs.docker.com/get-docker/)** page, click on your operating system, and follow the instructions. Note that there is a graphical user interface called "Docker Desktop" available for Mac and Windows users, whereas at the time of this writing there is not an equivalent tool for Linux users. Linux users can install the "server" version.

Pulling the PySpark Stack from DockerHub

If you were developing your own Dockerfiles, you could just work locally, similarly to how you could write Python code locally without connecting to any remote repositories. But we want to run an image created by someone else, so we want to use the `docker pull` command from **DockerHub (https://hub.docker.com/r/jupyter/pyspark-notebook)**. This is roughly equivalent to running `git pull` from GitHub, except you're downloading a pre-built computer image.

Specifically, run this command in the terminal:

```
docker pull jupyter/pyspark-notebook
```

This will initiate a download that will likely take a while, then finally you should see a message like this:

```
Status: Downloaded newer image for jupyter/pyspark-notebook:latest
```

You have now pulled down the PySpark stack!

Running Jupyter Notebook with Docker

Now that you have pulled down `pyspark-notebook`, run this command in the terminal:

```
docker run -p 8888:8888 jupyter/pyspark-notebook
```

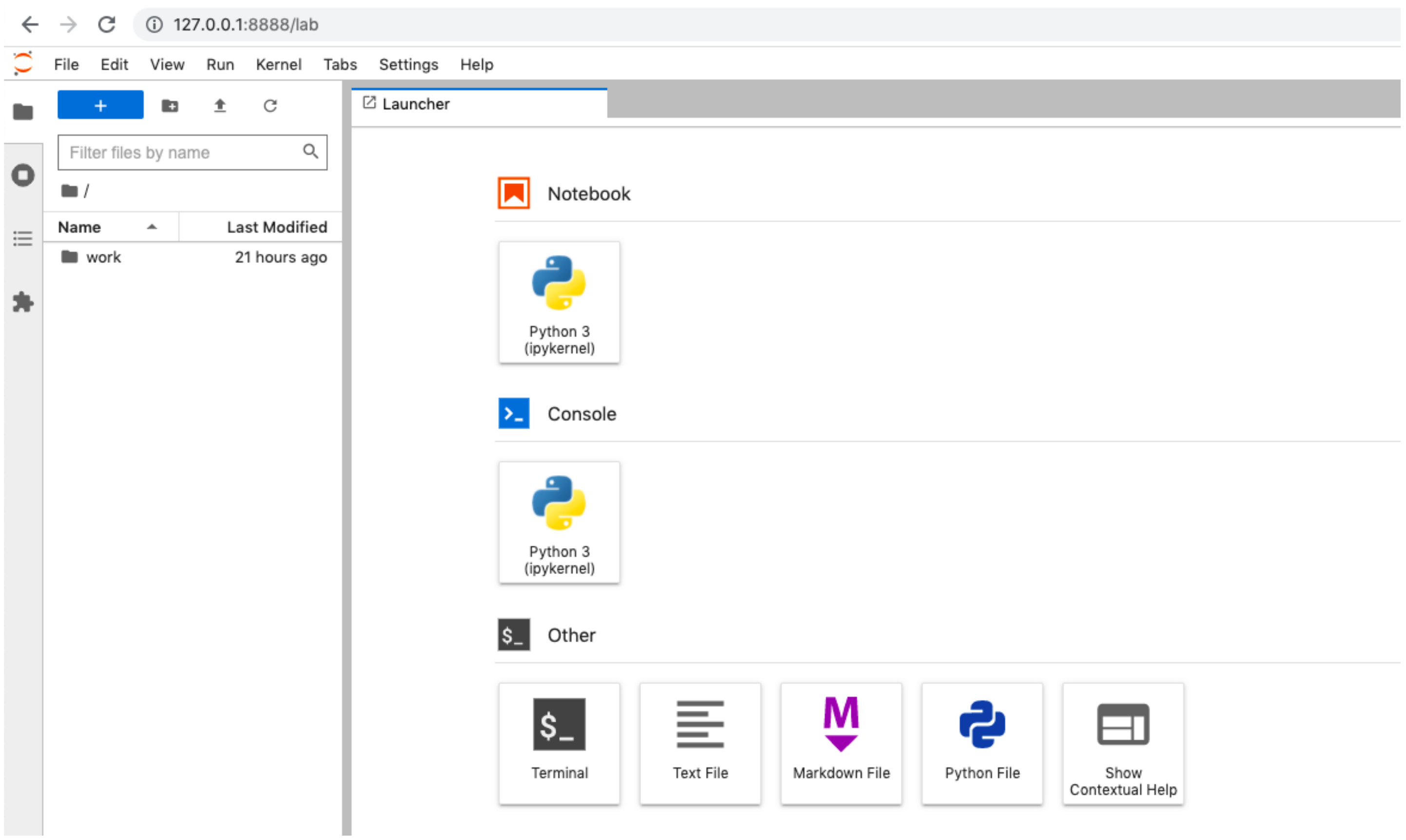
(The `-p` flag is setting the ports on your computer as well as the container to be connected.)

This will launch a notebook server that should look fairly similar to when you run a regular `jupyter notebook` command!

However you will most likely need to copy the URL displayed in the terminal and paste it into a browser window, rather than having it automatically open like `jupyter notebook` usually does. The URL will look something like this:

```
http://127.0.0.1:8888/lab?token=<token>
```

Then once you paste it into the browser address bar, you'll be redirected to just `http://127.0.0.1:8888/lab`, which is a Jupyter Lab interface:



If you want to navigate back to the classic `jupyter notebook` file window, simply enter `http://127.0.0.1:8888/tree` in the address bar (replacing `lab` with `tree`).

Checking for Successful PySpark Image Installation

From here, you can create a new notebook and run these lines of code, which should not produce an error:

```
import pyspark
sc = pyspark.SparkContext('local[*]')
rdd = sc.parallelize(range(1000))
rdd.takeSample(False, 5)
```

Connecting Docker Container to Your File System

You might have noticed something strange when you were creating that notebook: there was only a directory called `work` , nothing related to the directory on your computer where you launched `docker run` !

This is because even though the notebook server looked very similar to one being run directly on your computer, this one only had access to the container's file system.

If you want to be able to use notebooks from the curriculum or files on disk (e.g. CSV files), it's useful to be able to connect your computer's file system to the container.

Shutting Down Previous Docker Container

Shut down the currently-running container by typing `control-C` in the terminal window where it is currently running. If you accidentally closed that terminal window, you can:

1. Use a command-line approach:
   - Run `docker ps` to see a list of all currently-running docker containers
   - Run `docker stop <container id>` where `<container id>` is from the `docker ps` print-out. For example, `docker stop efb990e0e054`
2. Or, use Docker Desktop:
   - Open Docker Desktop and locate the currently-running container in the "Containers / Apps" list
   - Click the square stop button

Starting Docker Again, Connected to Your File System

The formal language of this is called "mounting a volume", so it uses the `-v` command-line option.

The general structure looks like this:

```
docker run -p 8888:8888 -v {absolute file path of current directory}:/home/jovyan/work ju
```

We are mapping `{absolute file path of current directory}` on your computer onto `/home/jovyan/work` in the container.

(Fun fact: the username `jovyan` is a **[play on the name Jupyter (https://docs.jupyter.org/en/latest/community/content-community.html#what-is-a-jovyan)](https://docs.jupyter.org/en/latest/community/content-community.html#what-is-a-jovyan)** . *Jovyan* is to

***Jovian* (https://en.wiktionary.org/wiki/Jovian)** as *Jupyter* is to *Jupiter*.)

For **Mac** or **Linux**, the actual command looks like this:

```
docker run -p 8888:8888 -v $(pwd):/home/jovyan/work jupyter/pyspark-notebook
```

For **Windows**, the actual command looks like this (executed in Command Prompt, not Git Bash):

```
docker run -p 8888:8888 -v %cd%:/home/jovyan/work jupyter/pyspark-notebook
```

Now you should be able to navigate to the `work` directory and find this notebook there!

A Couple More Command-Line Options

```
-it
```

This starts the container in "interactive mode" and allows you to access the Bash shell inside the container.

```
--rm
```

This removes the container from your list of images as soon as you shut it down. Since you are storing your data on your computer's file system, this is a good option to avoid creating a lot of extra unnecessary files.

Therefore we recommend that you run this complete command:

On **Mac/Linux**:

```
docker run -p 8888:8888 -v $(pwd):/home/jovyan/work -it --rm jupyter/pyspark-notebook
```

On **Windows**:

```
docker run -p 8888:8888 -v %cd%:/home/jovyan/work -it --rm jupyter/pyspark-notebook
```

Summary

In this lesson, we looked at installing Spark with and without a Docker container.

To recap the steps:

Without Docker

Run all of these commands, following the instructions above to ensure that each step worked as expected:

```
conda activate base
conda create --name spark-env python=3.8
conda activate spark-env
conda install -c conda-forge openjdk=11
pip install pyspark==3
conda install -c conda-forge notebook
python -m ipykernel install --user --name spark-env --display-name "Python (spark-env)"
conda install matplotlib
jupyter notebook
```

With Docker

Go to the **Get Docker (https://docs.docker.com/get-docker/)** page, click on your operating system, and follow the instructions.

For Mac/Linux:

```
docker pull jupyter/pyspark-notebook
docker run -p 8888:8888 -v $(pwd):/home/jovyan/work jupyter/pyspark-notebook
```

For Windows:

```
docker pull jupyter/pyspark-notebook
docker run -p 8888:8888 -v %cd%:/home/jovyan/work -it --rm jupyter/pyspark-notebook
```