

# Advanced Apache Spark, DeepLearni.ng and TensorFlow Lab

## Session 1: Getting started with Docker and Spark

## Why This Meetup?

•We want to show members how to apply Spark and TensorFlow in a pragmatic manner to address industry use cases

•This first series of 4 lessons will combine the building blocks necessary to tackle these use cases

## Introduction:

The end goal of Lesson 1 is to give an overview of Apache Spark. Unfortunately, like most technologies today, there is a bit of effort required to get things set up. Fortunately, there are some other technologies that make this environment creation very easy.

This does mean that we will need to talk about two other technologies that will help us out. Enter stage left Docker and Zeppelin.

## Requirements:

* Virtualbox (available from <https://www.virtualbox.org/wiki/Downloads>)
* Ubuntu 16.04 desktop iso (available at <http://releases.ubuntu.com/16.04/ubuntu-16.04.3-desktop-amd64.iso> or on our local FTP server)
* 20 gigabytes of free space (or more) for the virtual machine
* practical-learnings GitLab project ([GitLab - Practical Learning](https://gitlab.com/deeplearni.ng/practical-learnings))
* (Optionally) the meetup image (available on our FTP server). This can otherwise be obtained through a build script described later in this document

## Docker:

Docker is, in a very basic sense, a “virtual machine” (VM) manager. It allows you to “spin up” a virtual machine in a quick and straightforward way. We will touch on a high-level view of what is going on and how it works.

To illustrate, I have drawn up a simplistic diagram of what’s happening in the background. We start with the concept of a “client” and a “host”.

The **Docker Client** is what you will be using to interact with Docker, either through a terminal or some other user interface.

The **Docker Host** is where all the magic happens. Within the host, there are two main concepts that we must know about. Images and Containers.

An **image** is NOT the machine but more of a description of what the machine will look like. In our example (meetup-zeppelin.img) the image already knows that it will have Zeppelin and Spark, as well as all the requirements needed for both applications (i.e. Java SDK, NodeJS).

A **container** is the actual virtual machine. You can do everything that you could to a regular virtual machine (i.e. SSH into it, stop it, start it up again).

### 

## The Setup

### 1. Setting up our Virtual Machine

Before we can do anything else, we will need to setup a virtual machine (or VM) to run all of our applications inside. The reason we do this is to ensure a consistent environment so that it is easier to debug any problems that may arise.

1. Download <http://releases.ubuntu.com/16.04/ubuntu-16.04.3-desktop-amd64.iso> – this will contain the ubuntu image that we will be using
2. Open up VirtualBox
3. Click ‘New’ at the top left of the screen
4. Type ‘ubuntu’ as the name, select ‘Linux’ as the type and ‘Ubuntu (64-bit)’ as the version
5. Type ‘2048’ for memory and click ‘Continue’
6. When asked to create a virtual hard drive, click ‘Continue’
7. Click ‘Continue’ for the current dialog and then select ‘Fixed Size’ for the next one to proceed.
8. When asked for the disk size, enter 20gb (or more if you want) and continue
9. Right click on the VM you just created and click ‘Settings’
10. Click on the ‘Storage’ tab and in the left bar click the little CD icon with a plus sign that’s on the same line as ‘Controller: IDE’ then select the .iso file you downloaded in step 1A
11. Now click on the ‘Network’ tab. In the first tab there will be settings for the network adapter this VM will use.   
     - The first select box should have ‘Bridged Adapter’ selected  
     - The second select box should have the adapter selected that is used to connect to the internet – this will depend on your machine but in most cases should be your WiFi connection
12. Double click on your VM to start it up.
13. Select ‘Install Ubuntu’
14. Select ‘Continue’ (no updates will be installed)
15. Select ‘Erase Disk’, ‘Install Ubuntu’ and then click ‘Continue’
16. Select Toronto when prompted for a region – this is used for your time zone
17. When prompted for a keyboard layout – please select English (US) and English (US). This will help avoid unexpected characters appearing when typing in commands
18. You will then be asked for the name, hostname, username and password settings. Please be creative with these as it will help with future meetups!
19. Finally press ‘Enter’ when asked by the screen and your machine will reboot

### 2. Preparing our VM

Now that we have our virtual machine we want to get the files from gitlab and run some shell scripts we have built to make set-up easier.

1. Select the ‘Devices’ menu from the virtual machine menu – in it, click ‘Insert Guest Additions CD Image’
2. In the VM, open up your terminal by clicking the Ubuntu button (start-menu like button) followed by typing ‘terminal’ and pressing enter
3. CD into the folder containing the guest additions. It should look like ‘/media/<your\_username\_here>/VBOXADDITIONS\_5.1.20\_114628’
4. Run the command `sudo sh VboxLinuxAdditions.run`
5. Reboot you VM by selecting the gear in the top right corner, clicking shutdown, then ‘Restart’
6. Select the ‘Devices’ menu from the virtual machine menu – in it, click ‘Shared Clipboard’ followed by ‘Host to Guest’. This will allow you to copy things from your real computer into the VM
7. Once the VM has rebooted, re-open your terminal as in step 2B
8. Type `sudo apt-get update`
9. Type `sudo apt-get install git`
10. Type `git clone <https://gitlab.com/deeplearni.ng/practical-learnings.git>`

* You will be asked to generate a personal access token. Create one at <https://gitlab.com/profile/personal_access_tokens>. Once you get the code retype step C and enter it as your password.

1. Type `cd practical-learnings/session1`

### 3. Preparing Docker

Now that the VM is set up completely, we can install Docker and get ready to open up Zeppelin.

1. Type `sh ./install-docker.sh`. You will be asked for your password and then the installation should go through.
2. Once this is done, you will be required to log out. This can be achieved by clicking the gear as in step 2E, and clicking ‘Log Out’.  
    - This step is required so that we can use Docker with our current user
3. Log back in as usual with your password
4. Open your terminal once again (step 2B) and enter the session1 directory as in step 2K
5. (Optionally – this is the preferred method) Copy the meetup image from our public ftp server. This can be done with the command `scp -i ./id\_ftp public-ftp@192.168.1.2:~/meetup-zeppelin.img.gz ./`  
    - You may need to change the permissions of the ‘id\_ftp’ file to 500 by running `chmod 0500 ./id\_ftp`
6. Once this is done, you can run `gzip -d -c meetup-zeppelin.img.gz | docker load`. This can take a while
7. (Only if you didn’t run steps 3E and 3F) Build the docker image by running `sh ./build.sh`.   
    - This can possibly take much longer than the 2E method  
    - However, this can be used if our local resources are too heavily consumed

### 3. Running Zeppelin

Once the image has been prepared, we can now start using Zeppelin.

1. Finally we can now run our docker container using the command `docker run -it --rm -p 8080:8080 -v `pwd`/notebook:/zeppelin/notebook -v `pwd`/data-sets:/data-sets meetup-zeppelin:session1`  
    - Note the backquotes around `pwd`. This will evaluate to the current working directory so Docker knows where to get our notebooks and data-sets
2. Once this is running, we can open Firefox on the left bar in the VM, and then navigate to [http://localhost:8080](http://localhost:8080/). If everything is successful, you should see the Zeppelin application running.

## Some Info About The Docker Container:

Let’s dissect the ‘docker run -it --rm -p 8080:8080 -p 4040:4040 -v `pwd`/notebook:/zeppelin/notebook -v `pwd`/data-sets:/data-sets meetup-zeppelin:session1’ command for a bit:

**run:**

This is Docker’s command to “spin up” a VM from an image.

**-it:**

These are two separate arguments. “i” is for “interactive” and “t” is for “tty”. When combined, this will open up an interactive console into the running container so that we can see what is going on.

**-p:**

This allows us to open up a port to the outside world. Both applications that will be running in our container, Zeppelin and Spark, offer web-based applications. Zeppelin is available on port 8080 and Spark is available on port 4040.

Syntax is as follows: **-p [local port]:[container port]**

Notice that we open a port for both applications!

**-v:**

This one is a bit weird to wrap your head around and you might want to refer back to the diagram.

We have a directory on our computer that we want the VM to have access to. This is where we use something called “mounting”. Mounting is simply stating that a directory on one computer is THE EXACT SAME as a directory on another computer. They become inter-connected, if I change something in one directory, it changes in the other as well.

Syntax is as follows: **-v [local path]:[container path]**

For example:

**-v /Users/coleclifford/Desktop/DeepLearni.ng/GitLab\ Projects/practical-learnings/session1/notebook:/zeppelin/notebook**

Grey is the path to the notebooks on my computer and green is the path on the container we want to link to. “/zeppelin/notebook” is where the Zeppelin application will look for notebooks on startup.

Note that the notebooks and datasets are both in the GitLab project (practical-learnings/session1/).

**--rm:**

By default (without this argument), Docker will keep around containers in a shut down state. This can take up memory if you don’t handle it the right way.

This is a “remove” command and will take delete the container once you stop using it.

**meetup-zeppelin:session1:**

This is the name of the image that we want to create a container out of.

After hitting enter, there will be a few things logged and you may even see some warnings, but things should be up and working!

## 

## Zeppelin

With our Docker container running, we no have access to both Zeppelin and Spark! First we should actually tell you what Zeppelin is though.

Zeppelin is a web based IDE (similar to iPython if you have used it before). It allows us to have many different language interpreters that we can use very quickly and in an interactive way.

In our Zeppelin page, we can open the “Simple Parallelization” notebook and see a bunch of blue blocks. These are the available interpreters. Hit save at the bottom and refresh the page for syntax highlighting of our code.

These interpreters can even be paragraph specific which means we can try stuff in different languages very quickly (sometimes even across languages).

Most importantly to us, Zeppelin will allow us to run Spark jobs very quickly and interactively!

## Spark

Now for the headlining technology of the night!

Most of Spark’s concepts will be covered in the Zeppelin notebooks, but there are some main concepts that make this technology so exciting.

It’s very compatible with other Apache projects which means that it can be very extensible. This is a list of Apache projects to see what’s out there: [Apache Projects](https://projects.apache.org/projects.html).

Spark is very fast. It uses both the map-reduce paradigm and an internal “Tungsten engine” to speed things up. The Tungsten engine is out of scope here but is talked about in this blog post: [Databricks - Tungsten Engine](https://databricks.com/blog/2015/04/28/project-tungsten-bringing-spark-closer-to-bare-metal.html).

It parallelizes tasks extremely well and without you having to know what that actually means. Spark handles all of the dirty details involved with splitting up a task and having it run across however many computers you have access to.

We will now switch over to Zeppelin and start walking through the code.

**Notebook Order:**

1. Simple Parallelization
2. RDD Management
3. DataFrames

## Important Notes on Spark:

An RDD is not *really* the data! It is a representation of the transformation that have been done on the data. This means that you have to actually run the transformations at some point. Users can use something like the “collect()” function for this.

A DataFrame is similar in concept to an RDD but also stores information about the schema and structure of the data.