# Advanced Apache Spark, DeepLearni.ng and TensorFlow Lab

## 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:

* Docker ([Docker - Download Docker](https://www.docker.com/community-edition#/download))
* meetup-zeppelin.img ([Google Drive - meetup-zeppelin.img](https://drive.google.com/file/d/0Bw5bW0et-TNZcVFFWjVIVnBjVDA/view?usp=sharing))
* practical-learnings GitLab project ([GitLab - Practical Learning](https://gitlab.com/deeplearni.ng/practical-learnings))

## 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).

### 

## Setup Docker Container:

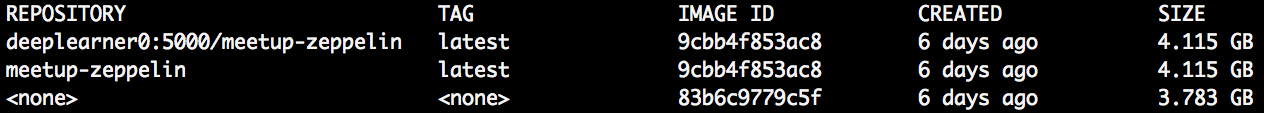
To get our container up and running we first need to load our image into Docker. Open your terminal of choice with Docker running and repeat after me!

1. $ cd /directory/with/downloaded/meetup-zeppelin.img
2. $ docker load < meetup-zeppelin.img

This step will take some time as the image is 4.34GB.

1. $ docker tag deeplearner0:5000/meetup-zeppelin meetup-zeppelin
2. $ docker images

We should see something like the following:



The “<none>” is simply a leftover from the build processes that Docker uses. The next few steps are clean up.

1. $ docker rmi deeplearner0:5000/meetup-zeppelin

“rmi” means “remove image”. You can specify either the repository, repository:tag, or id.

1. $ docker rmi **[IMAGE ID of <none>]**

For me **[IMAGE ID of <none>]** is **83b6c9779c5f**.

1. $ docker run -it -p 8080:8080 -p 4040:4040 -v **/path/to/notebooks**:/zeppelin/notebook -v **/path/to/datasets**:/data-sets --rm meetup-zeppelin

Hold up! Let’s break this line down a little bit and figure out what in the name of Bill Gates is going on here.

**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:**

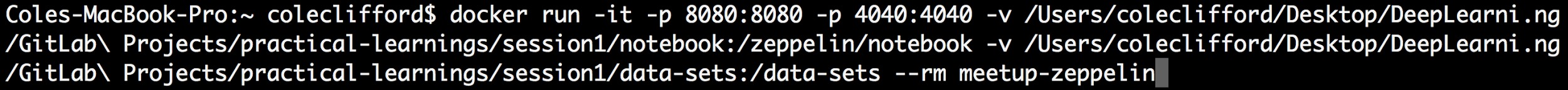
By default (without this argument), Docker will keep around containers in a shut down state. This can take up memory if you don’t handle it the right way.

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

**meetup-zeppelin:**

This is the name of the image that we want to create a container out of. You could use the image id or even the image name and the tag (meetup-zeppelin:latest).

My full command would look like this:



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

1. We can now go to our favourite browser and see if Zeppelin started. Go to one of the following addresses to check.

* localhost:8080
* 192.168.99.100:8080

## 

## 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.