# **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:

Welcome to Lesson 2! Throughout this workshop we will try to give you a strong grasp on the basic concepts that make up TensorFlow.

We will be using both Docker and Zeppelin to get things set up. We won’t go through these technologies again since they were talked about during the first lesson, but you can read up on both Docker and Zeppelin in our previous document.

This lesson will go from the basic concepts of TensorFlow to a real use-case of a neural network. We try to show that a neural network follows the same concepts as a simple graph, but avoid going in-depth into the theory behind “Deep Neural Networks”.

## Requirements:

* Docker ([Docker - Download Docker](https://www.docker.com/community-edition#/download))
* meetup-zeppelin-tensorflow.img ([Google Drive - meetup-zeppelin-tensorflow.img](https://drive.google.com/open?id=0Bw5bW0et-TNZd3FlaE9VWl96aGM))
* practical-learnings GitLab project ([GitLab - Practical Learning](https://gitlab.com/deeplearni.ng/practical-learnings))

## Docker:

The docker command is very similar to Lesson 1 but we will lay out the differences below (highlighted in blue).

$ docker run -it

-p 8080:8080

-p 4040:4040

-p 6006:6006

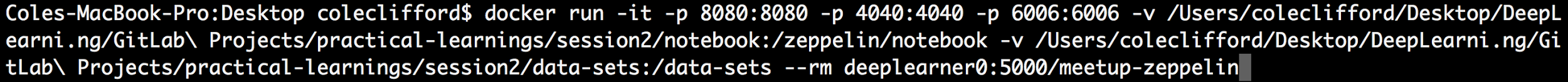
-v **/path/to/session2/notebooks**:/zeppelin/notebook

-v **/path/to/session2/datasets**:/data-sets

--rm meetup-zeppelin

**Note**: The “*meetup-zeppelin*” part could be “*deeplearner0:5000/meetup-zeppelin*” if you haven’t tagged the image after loading it.

My full command looks like this:



Remember that there will be a few warnings after hitting enter. We can ignore these.

Head to your browser of choice and make sure Zeppelin is running. Address will either be:

* localhost:8080
* 192.168.99.100:8080

## TensorFlow

For this lesson, the technology that everyone is talking about, TensorFlow!

We will cover some of the basic concepts before we go into the Zeppelin notebooks. The goal is to make sure you have at least heard some of the terms before attempting to use them.

In its most simplistic form, TensorFlow follows a data-flow paradigm. This is the concept of creating a graph, where each node is a mathematical computation, and letting data flow through it.

Data is held in a “Tensor”, which can be thought of as an *n*-dimensional array (similar to a Numpy array if you have used them before).

So there we have it! Data (in *Tensors*) *flowing* through a predefined graph… *TensorFlow!*

A Tensor, mainly, has a name, a shape, and a type associated with it. If you don’t specify any of these, they will be inferred/generated for you. **There are outliers that we will talk about as they come up.**

Notice that I called it a *predefined* graph previously. This is a very important concept to how TensorFlow works. When you define a graph you are not actually running it! In fact, no data is available unless you are actually running the graph through a “*session*” (this includes constants!).

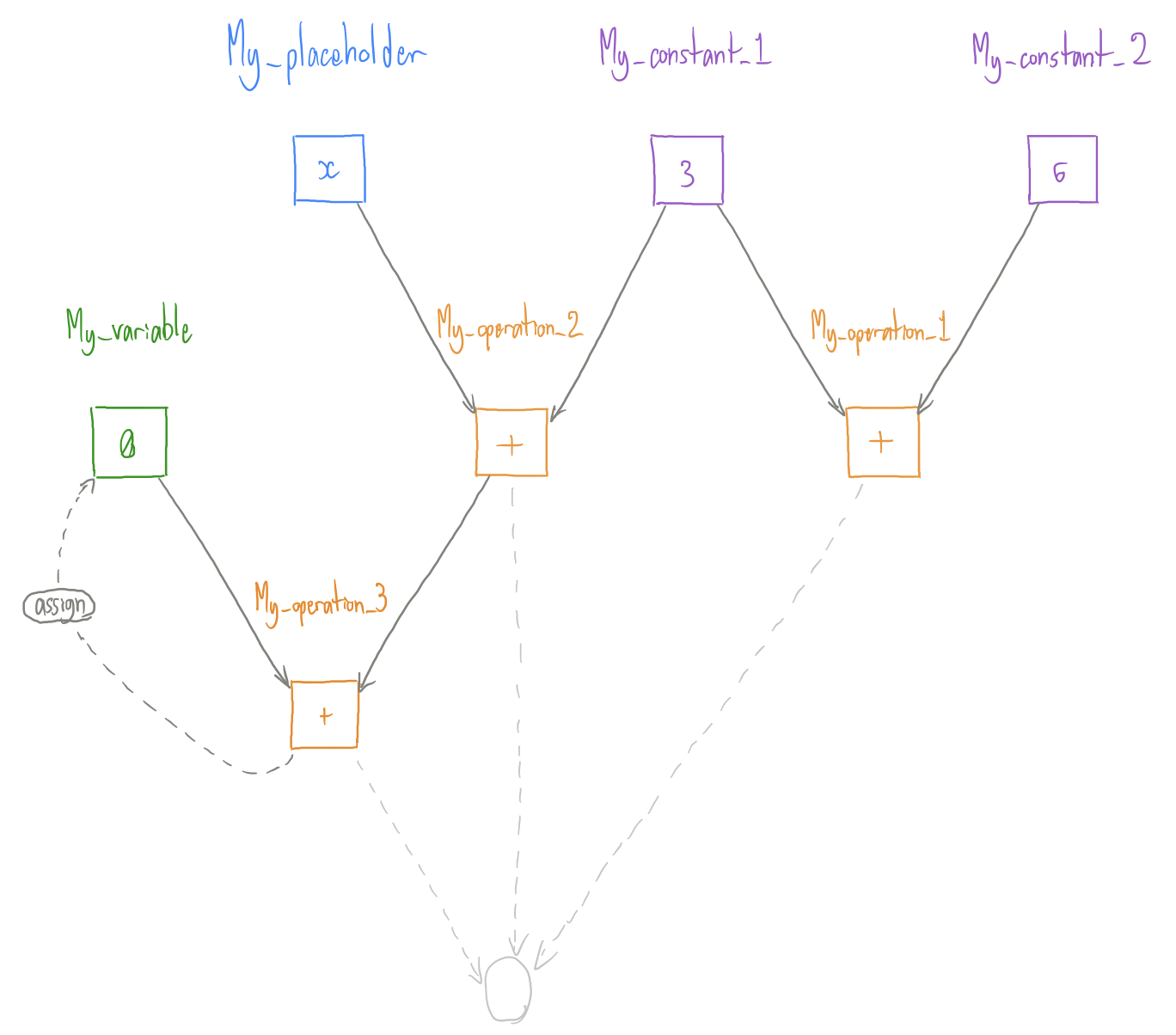
My two examples for understanding this concept are as follows:

1. A class in programming. You define the structure and logic of the class *and then* create an instance of the class to interact with it.
2. An RDD in Spark. You define the operations that will happen to the data *and then* run the operations through something like a .collect().

For TensorFlow, you create the structure and logic of the graph *and then* create a session to run the graph.

This design time vs. run time is very important and you should understand it before we go forward.

Finally, we will walk through the basics of a TensorFlow graph using the following diagram. This explanation will be high level, not actual code just yet.



This visual shows a basic graph that we will make in the notebook titled “**2: Graphs and How to Use Them**”(paragraph 9). It is designed to show a use case of four main concepts that make up a TensorFlow graph.

1. **Constant**

Fairly straightforward. This is a constant value on the graph. You can define a constant that will be the same every time you run the graph.

In the diagram, we define two constants *My\_constant\_1* and *My\_constant\_2*. We give them the values of *3* and *6* respectively.

1. **Placeholder**

Think of the graph as a function in regular programming. A placeholder would be an argument that the function requires. When defining the graph, you state that there will be a Tensor with a value but will not have access to it until run-time. You should definitely specify things like type and shape during design time!

We call our placeholder *My\_placeholder*.

1. **Variable**

This is not a *variable* in the classic programming sense, but a value that will *vary* across runs of the graph. You must designate the initial value of the variable. You then have access to the value during every run of the graph and can update it as a sort of “memory” from run to run.

We call our variable *My\_variable*.

1. **Operation**

This is a mathematical operation that makes up the nodes of the graph. There are multiple ways to specify operations which we give an example of in the notebook titled **1: TensorFlow Basics** (paragraphs 5 and 6).

We define three operations:

* *My\_operation\_1* which adds *My\_constant\_1* and *My\_constant\_2*
* *My\_operation\_2* which adds *My\_constant\_1* and *My\_placeholder*
* *My\_operation\_3*which adds the outcome of *My\_operation\_2* and *My\_variable*. It then assigns the outcome back to *My\_variable*.

With this new information, it is important to mention graph dependencies. If we run *My\_operation\_1* we will simply get back the value of our two constants added together, which is 9. However, if we want to run *My\_operation\_2* or *My\_operation\_3* we can see that we require a value from *My\_placeholder*. This means that we can run the first operation with no problems, but will have to give a value for our placeholder while running the other operations.

TensorFlow handles dependencies without you having to do any extra work!

The last thing to talk about is a TensorFlow *session*. We have discussed the fact that building the graph is different that running the graph. The session is how we actually run things. We will see how we use it in the notebooks but we will mention the *feed\_dict*. This is a dictionary that we pass to the Session().run() function that allows us to feed in values for our placeholders.

**Note**: TensorFlow seems scary, especially when you get to the later notebooks that touch on actual neural networks. Just try to remember that no matter what you do, TensorFlow works through a **data flow graph**. The later notebooks may look overwhelming but they are still graphs, they just do very interesting math at their nodes to allow for “learning”.