1) Nvidia Drivers [https://github.com/saiprashanths/dl-setup#nvidia-drivers]

- Find your graphics card model

```
lspci | grep -i nvidia
```

- Go to the Nvidia website and find the latest drivers for your graphics card and system setup. You can download the driver from the website and install it, but doing so makes updating to newer drivers and uninstalling it a little messy. Also, doing this will require you having to quit your X server session and install from a Terminal session, which is a hassle.
- We will install the drivers using apt-get. Check if your latest driver exists in the "Proprietary GPU Drivers" PPA. Note that the latest drivers are necessarily the most stable. It is advisable to install the driver version recommended on that page. Add the "Proprietary GPU Drivers" PPA repository. At the time of this writing, the latest version is 364: [https://launchpad.net/%7Egraphics-drivers/+archive/ubuntu/ppa]

```
sudo add-apt-repository ppa:graphics-drivers/ppa
sudo apt-get update
sudo apt-get install nvidia-364
```

- Restart your system

```
sudo shutdown -r now
```

- Check to ensure that the correct version of NVIDIA drivers are installed

```
cat /proc/driver/nvidia/version
```

- Install nvidia-modprobe (prerequisite for nvidia-docker)

```
sudo apt-get install nvidia-modprobe
```

2) Docker [https://docs.docker.com/engine/installation/linux/ubuntulinux/]

Update your apt sources

Docker's `APT` repository contains Docker 1.7.1 and higher. To set `APT` to use packages from the Docker repository:

Log into your machine as a user with `sudo` or `root` privileges.

Open a terminal window.

Update package information, ensure that APT works with the `https` method, and that CA certificates are installed.

```
$ sudo apt-get update
```

```
$ sudo apt-get install apt-transport-https ca-certificates
```

Add the new `GPG` key.

```
$ sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.net:80 --recv-keys 58118E89F3A912897C070ADBF76221572C52609D
```

Find the entry for your Ubuntu operating system.

The entry determines where APT will search for packages. The possible entries are:

Ubuntu version	Repository
Precise 12.04 (LTS)	deb https://apt.dockerproject.org/repo ubuntu-precise main
Trusty 14.04 (LTS)	deb https://apt.dockerproject.org/repo ubuntu-trusty main
Xenial 16.04 (LTS)	deb https://apt.dockerproject.org/repo ubuntu-xenial main

> Note: Docker does not provide packages for all architectures. Binary artifacts are built nightly, and you can download them from https://master.dockerproject.org. To install docker on a multi-architecture system, add an `[arch=...]` clause to the entry. Refer to Debian Multiarch wiki for details.

Run the following command, substituting the entry for your operating system for the placeholder `<REPO>`.

```
$ sudo su (need to become root, otherwise the following command will return a permission error)
```

```
$ sudo echo "<REPO>" > /etc/apt/sources.list.d/docker.list
```

Update the `APT` package index.

```
$ sudo apt-get update
```

Verify that `APT` is pulling from the right repository.

When you run the following command, an entry is returned for each version of Docker that is available for you to install. Each entry should have the URL `https://apt.dockerproject.org/repo/`. The version currently installed is marked with `***`. The output below is truncated.

```
$ apt-cache policy docker-engine

 docker-engine:
   Installed: 1.13.2~rc4~trusty
   Candidate: 1.12.2~rc4~trusty
   Version table:
  *** 1.13.2~rc4~trusty 0
         500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
         100 /var/lib/dpkg/status
      1.13.1~rc1~trusty 0
         500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
      1.13.0~rc3~trusty 0
         500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
      1.11.2~rc3~trusty 0
         500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
      1.11.1~rc1~trusty 0
         500 https://apt.dockerproject.org/repo/ ubuntu-trusty/main amd64 Packages
```

From now on when you run `apt-get upgrade`, `APT` pulls from the new repository.

Prerequisites by Ubuntu Version [Ubuntu Xenial 16.04 (LTS) and Ubuntu Trusty 14.04 (LTS)]

For Ubuntu Trusty, and Xenial, it's recommended to install the `linux-image-extra-*` kernel packages. The `linux-image-extra-*` packages allows you use the `aufs` storage driver.

To install the `linux-image-extra-*` packages:

Open a terminal on your Ubuntu host.

Update your package manager.

```
$ sudo apt-get update
```

Install the recommended packages.

```
$ sudo apt-get install linux-image-extra-$(uname -r) linux-image-extra-virtual
```

Go ahead and install Docker.

Install

Make sure you have installed the prerequisites for your Ubuntu version.

Then, install Docker using the following:

Log into your Ubuntu installation as a user with `sudo` privileges.

Update your `APT` package index.

```
$ sudo apt-get update
```

Install Docker.

```
$ sudo apt-get install docker-engine
```

Start the `docker` daemon.

```
$ sudo service docker start
```

Verify `docker` is installed correctly.

```
$ sudo docker run hello-world
```

This command downloads a test image and runs it in a container. When the container runs, it prints an informational message. Then, it exits.

3) Nvidia-docker

Assuming the NVIDIA drivers and Docker are properly installed (see installation)

Ubuntu distributions

```
# Install nvidia-docker and nvidia-docker-plugin
wget -P /tmp https://github.com/NVIDIA/nvidia-docker/releases/download/v1.0.0-rc.3/nvidia-docker_1.0.0.rc.3-1_amd64.deb
sudo dpkg -i /tmp/nvidia-docker*.deb && rm /tmp/nvidia-docker*.deb

# Test nvidia-smi
nvidia-docker run --rm nvidia/cuda nvidia-smi
```

4) Building Docker Image [https://github.com/saiprashanths/dl-docker]

Option 2: Build the Docker image locally

Alternatively, you can build the images locally. Also, since the GPU version is not available in Docker Hub at the moment, you'll have to follow this if you want to GPU version. Note that this will take an hour or two depending on your machine since it compiles a few libraries from scratch.

```
git clone https://github.com/saiprashanths/dl-docker.git
cd dl-docker
```

GPU Version

```
docker build -t floydhub/dl-docker:gpu -f Dockerfile.gpu .
```

This will build a Docker image named `dl-docker` and tagged either `cpu` or `gpu` depending on the tag your specify. Also note that the appropriate `Dockerfile.<architecture>` has to be used.

4) Running the Docker Image as a container [https://github.com/saiprashanths/dl-docker]

Once we've built the image, we have all the frameworks we need installed in it. We can now spin up one or more containers using this image, and you should be ready to go deeper

CPU Version

```
docker run -it -p 8888:8888 -p 6006:6006 -v /sharedfolder:/root/sharedfolder floydhub/dl-docker:cpu bash
```

GPU Version

```
nvidia-docker run -it -p 8888:8888 -p 6006:6006 -v /sharedfolder:/root/sharedfolder floydhub/dl-docker:gpu bash
```

Note the use of `nvidia-docker` rather than just `docker`

Parameter	Explanation
`-it`	This creates an interactive terminal you can use to iteract with your container
`-p 8888:8888 -p 6006:6006`	This exposes the ports inside the container so they can be accessed from the host. The format is `-p <host-port>:<container-port>`. The default iPython Notebook runs on port 8888 and Tensorboard on 6006
`-v /sharedfolder:/root/sharedfolder/`	This shares the folder `/sharedfolder` on your host machine to `/root/sharedfolder/` inside your container. Any data written to this folder by the container will be persistent. You can modify this to anything of the format `-v /local/shared/folder:/shared/folder/in/container/`. See Docker container persistence
`floydhub/dl-docker:cpu`	This is the image that you want to run. The format is `image:tag`. In our case, we use the image `dl-docker` and tag `gpu` or `cpu` to spin up the appropriate image
`bash`	This provides the default command when the container is started. Even if this was not provided, bash is the default command and just starts a Bash session. You can modify this to be whatever you'd like to be executed when your container starts. For example, you can execute `docker run -it -p 8888:8888 -p 6006:6006 floydhub/dl-docker:cpu jupyter notebook`. This will execute the command `jupyter notebook` and starts your Jupyter Notebook for you when the container starts

Some common scenarios

Jupyter Notebooks

The container comes pre-installed with iPython and iTorch Notebooks, and you can use these to work with the deep learning frameworks. If you spin up the docker container with `docker-run -p <host-port>:<container-port>` (as shown above in the instructions), you will have access to these ports on your host and can access them at `http://127.0.0.1:<host-port>`. The default iPython notebook uses port 8888 and Tensorboard uses port 6006. Since we expose both these ports when we run the container, we can access both from the localhost.

However, you still need to start the Notebook inside the container to be able to access it from the host. You can either do this from the container terminal by executing `jupyter notebook` or you can pass this command in directly while spinning up your container using the `docker run -it -p 8888:8888 -p 6006:6006 floydhub/dl-docker:cpu jupyter notebook` CLI. The Jupyter Notebook has both Python (for TensorFlow, Caffe, Theano, Keras, Lasagne) and lTorch (for Torch) kernels.

Data Sharing

See Docker container persistence. Consider this: You have a script that you've written on your host machine. You want to run this in the container and get the output data (say, a trained model) back into your host. The way to do this is using a Shared Volumne. By passing in the `-v /sharedfolder/:/root/sharedfolder` to the CLI, we are sharing the folder between the host and the container, with persistence. You could copy your script into `/sharedfolder` folder on the host, execute your script from inside the container (located at `/root/sharedfolder`) and write the results data back to the same folder. This data will be accessible even after you kill the container.