RICE UNIVERSITY

COMP543 Lab 1: Getting started with Docker

COMP543 Instructors

Last edited: August 28, 2018

Throughout this class, we'll be making use of Docker to set up our local work environments. This will be a short lab to make sure that everyone has Docker installed on their computer and knows a little bit about how to use it in the class.

1 Installation

Installation **should** be straightforward on recent versions of Mac, Windows, and Linux operating systems. Follow the links below to the Docker website for detailed installation instructions for your OS. Please ask for help if anything is confusing and/or not working!

1.1 Windows

Note that virtualization must be enabled for Docker to work properly. To check if you have virtualization enabled, look at the Task Manager's performance tab. If you find that it is disabled, please find a TA to help you enable it.

- 64bit Windows 10 Pro, Education, or Enterprise: Docker for Windows
- Other Windows: Docker Toolbox

After installation, "Docker for Windows" users should be able to execute `docker` at a Powershell prompt, and "Docker Toolbox" users should be able to from a new "Docker Toolbox Command Prompt" that was installed. A help dialogue will be printed.

1.2 Mac

- OSX >= OSX 10.11 and hardware >= 2010: Docker for Mac

- Other Mac: Docker Toolbox

After installation, users should be able to execute `docker` from a terminal to see a help dialogue printed.

1.3 Linux

Follow one of these guides:

- Ubuntu
- Debian
- CentOS
- Fedora

Then follow these steps: Linux Post-Installation Steps. Just follow the first two steps (managing docker as non-root user and starting at boot).

2 Getting Started

2.1 Terminology

- **image** - a static snapshot of a container built from a Dockerfile or CLI
- **container** - a running instance of a container, "feels like" a lightweight VM
- **Dockerfile** - a file with the "definition" of an image
- **docker daemon** - a server process that executes commmands sent from the docker CLI
- **docker machine** - On Windows / Mac, docker runs a lightweight linux VM called the docker machine. `docker-machine` is also a command line tool that lets you interact with the docker daemon.
- **Docker Hub** - like GitHub, but stores public images

2.2 Hello, World!

Let's dive in and see if the installation worked!

```
$ docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
 3. The Docker daemon created a new container from that image which runs the
```

```
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://cloud.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/engine/userguide/
```

If you see roughly that output, then the installation worked and you should be good to go! If not, please find a TA to help you debug.

2.3 Mounting Volumes

Usually, you'll want to write code in your preferred text editor or IDE on your local machine, and then run it in one of the containers we have set up for you. Or you'll fill in cells in a Jupyter notebook and then want to save that notebook back to your local machine. Docker provides utilties to "sync" directories between the host filesystem and the container filesystem called **volumes**.

Consider the following directory structure:

```
/home/user/comp543
  /lab1
    /lab1.ipynb
  /lab2
    /lab2.ipynb
```

To share all the contents of the `/home/user/comp543` directory with a container, one might execute:

```
$ docker run -i -t -v /home/user/comp543:/data ubuntu bash
```

Let's go ahead and unpack that command:

1. `run` : indicates we're running a container
2. `-i` : runs the container interactively (keeps STDIN open)
3. `-t` : attaches your terminal STDIN/STDOUT to the containers STDIN/STDOUT
4. `-v /home/user/comp543:/data`
   creates a **volume** that allows containers to read / write data to `/home/user/comp543` on the host filesystem and allows users to read / write data to `/data` on the container filesystem.
5. `ubuntu` : the container image to run
6. `bash` : the command to run on the container

On the container, the previous directory structure would correspond to:

```
/data
  /lab1
    /lab1.ipynb
  /lab2
    /lab2.ipynb
```

2.4 Publishing Ports

Containers are rarely run in isolation, and it's often useful to interact with a container over a network rather than by executing commands on a running container. Docker provides extensive utilties for managing container networking, such as providing an IP address for each running container and a network with which to access them, however we'll only be scratching the surface.

Consider a container running a webserver at port `8888`, such as a Jupyter notebook server. In order to access that webserver without tracking down the generated IP address, one usually opts to `-p, --publish` a container's port to a host port:

```
$ docker run -it -p 8000:8888 jupyter/scipy-notebook
```

The newly added argument (`-p 8000:8888`) "publishes" the container's port `8888` to the host's[1] port `8000`. The syntax is: `-p [host port]:[container port]`. Jupyter's default port is `8888`, so this allows us to access Jupyter at `http://localhost:8000/` (plus the generated token from the Jupyter console output).

Some Mac and Windows users may not be able to access the notebook server at `localhost`, since the actual host for the container is a small linux VM. If this is the case, you'll need to go to the terminal where you've been running docker commands and run:

```
$ docker-machine ip
```

to get the IP address of your "docker machine".

3 Exercises

To receive credit for the lab, please get these exercises checked off by a TA.

3.1 Jupyter + Docker

Here you will run a Jupyter notebook server from a docker container. One of the reasons we'll be running our Jupyter notebooks in Docker is that we can assume everyone is working in the exact same environment, as well as provide access to applications with complex installations.

[1] Note that on Mac and Windows the host is actually a small linux VM.

The Jupyter team publishes and maintains a set of docker images on GitHub and DockerHub. They tend to run relatively large, but there is typically one that already has all the packages you need.

The image you'll want to run is `jupyter/scipy-notebook`. Jupyter will be accessible at the container port `8888`. Note that earlier we mapped the container port to a localhost port. Be sure to make the same or a similar mapping when you start the container.

There are some examples on how you might run the docker stacks notebooks on the README. Once you get a notebook server running, follow these steps:

1. Create a new Python 3 notebook
2. Add a markdown cell with your name and netID
3. Add a markdown cell with the command you used to kick off docker
4. Add a markdown cell with the URL you used to access Jupyter
5. Add a markdown cell with a cool mathematical function, e.g. $f(x) = sin(x)$
6. Add a Python cell that plots this function over an interesting domain, e.g.

```
import numpy as np
import matplotlib.pyplot as plt
# necessary for jupyter notebooks
%matplotlib inline
# 1000 points evenly spaced btwn 0 and 100
domain = np.linspace(0, 100, 1000)
f = lambda x: np.sin(x)
plt.plot(domain, f(domain))
```

7. Save the Jupyter notebook as a PDF file, by clicking File→Download As→PDF via Latex (.pdf).