Beginning Docker

Docker is a sort of virtual machine. This lab talks you through installing it and using it for the first time on your Linux machine. It does not cover Docker Compose and it does not cover docker configuration files.

One of the main things Docker will give you is that it will allow you immediate access to a "fresh" machine. So, if you need to check whether something will work from a "fresh install", or if you need an already defined install (i.e. a computer with specific packages already installed), then you can use a Docker image.

You need to know what is installed in your Linux box. Realistically, all you will ever need is Docker and possibly git. You can find out if it is already installed using:
which docker
If it isn't installed, go ahead and use:
sudo apt install docker.io
to install docker. Then check it has installed using:
which docker

Actual Docker

Now let's take a look at what we get with docker.
docker -h
If you get a "permission error", then be aware that you might have to use "sudo" in front of all your docker commands, like:
sudo docker -h
And docker should spit out all possible commands. We will be interested in:

- docker start – to start a container
- docker images – to see which images are stored on the computer
- docker run – to start a container from an image
- docker ps – to see which images are currently running (use the **-a** option)
- docker pull – to pull an image from docker hub
- docker search – to find an image on dockerhub (although you can also just search online for a suitable image name)

We want an image that has some sort of web server. We'd like node at minimum, but we can still modify the container once it is running and save the modifications as an image. Go ahead and use docker search to look for some node images (something with lots of stars is good). Or just look on dockerhub from a browser: https://hub.docker.com/_/node

Getting a Docker image
The command to get a docker image from dockerhub is:
docker pull <image name>
Where you replace <image name> with the name of the image you want to pull plus any tags. I've chosen node:latest as the image I want to pull (name: node, tag: latest). Go ahead and pull an image. Once it's downloaded, use:
docker images
to check that your image has downloaded. It should be there in the list.

Starting a Docker container

This is probably the most complicated task. There are a number of things you need to know about before you do this:

- Everything generated in the docker container is kept in the docker container, *unless* you map a suitable volume
- The docker container cannot "serve" to the internet, *unless* you specify some port mappings (think port 8080…and other ones if you're also serving a database).

We could make our docker container use a folder on the host machine by mapping a volume using the -v option. In the first instance, we don't need this because our web system can come from GitHub. We do, however want to map a port. The easiest thing to do is just map ports inside the container to the same port outside the container.

To start your container, use the command (remember to use your particular port number):
docker run -ti -p 8000:8000 node bash

Now you will be inside your docker container. And…it looks something like another linux box. But notice a few things:

- whoami returns what?
- The command prompt starts with what?

Now, press **ctrl+p,q** (that is, hold down ctrl and then press p, release, press q, release) to step out of your docker container and run the same command. You will now know how to tell if you are inside your docker container or outside of it on the host system.

On the host system, to see which docker containers are running, type:
docker ps
and you will see your container. It has a name. When you use docker run, you can specify a name with a --name option, but if you don't (which we didn't), then you are allocated a verb plus a scientist as the name for the container. I got "gifted_euclid".
To step back inside the container, type (replace <container_name> with your container's name):
docker attach <container name>
and now browse to a folder that you can use to hold your server code. If you used the node docker image to start your container, then you can try:
cd /home/node
Docker always logs the user in as root when it starts up a container. This is not the best idea because it violates the rule of least privileges. For this reason, the node docker image has a user called "node" set up and ready. If you want to use this rather than the root user (it's a good idea), then you can use:
su node
To switch to a lower privilege level. Type
exit
to get back to being root.

Installing yet another computer…?

To get your server running on the Docker container, you need to fetch it from GitHub (using **git clone**), move to the directory where package.json lives (using **cd**) and then start the server running (**node server.js** – make sure you get your path to the server file correct). You can then check to see if the web server is running (either on the machine that is hosting docker, or another machine on the same network). You don't actually have to install anything. As your web system gets more complicated, you might end up installing more and more things. At the very least, you might want to investigate how to use cron to schedule the web server to update automatically at a certain time every day.

I never want to install another computer….

If you do end up installing stuff on your docker container, then you can turn it back into a docker image which you can push to docker hub. This gives you a customised docker image that you can use time and time again. To do this, step out of the docker container (**ctrl+p,q**), or stop it by typing "exit". Stop the container from outside of the container by using:
docker stop <container name>
replacing <container name> with your container name (mine is still gifted_euclid).
And finally:
docker commit <container name> <image_name>
In my case, I used my_node_image as my image name. You can now see your image in with the other node image in the docker images repository on your linux host machine (**docker images**). This can be pushed to docker hub as in the optional section below.

DockerHub?

(you do not need to push to DockerHub, unless you want to!)
Go to DockerHub https://hub.docker.com/ and create an account. Use:
docker login
To log in to your docker account.

Create a repository, name it and add yourself as a collaborator. My repository was called "my_node".
Rename your docker image from the previous step so that you have your DockerHub username in it:
docker tag <original image name> <Docker_username>/<image name>:1
For example, for me, I originally called my image my_node_image, my username on DockerHub is pamelaajohnston and I want this docker image to be called node_image. So I would type:
docker tag my_node_image pamelaajohnston/node_image:1

Finally, push the whole image to dockerhub:
docker push docker.io/<Docker_username>/<image name>:1
then visit your DockerHub repository and check that the image is there. Note that DockerHub does not keep images forever. If the image is not used for more than 6 months, then it will get deleted.

Now what?

Now you have a docker image installed with everything you need to run your server (it might be the default image!). You can set it up on just about any machine simply by pulling the image from docker hub, grabbing your server code and starting it up. Your server code is currently stored in GitHub, but there is no reason it couldn't be stored within an image on DockerHub. This means that if you have to switch machines for any reason, then your set-up time is minimal.

Housekeeping!

Docker images are efficient, but the images and containers do take up some space on your computer. If you're wondering how much space, run:
docker ps -a

This will show you all the containers (running and stopped) on the system. If you've stored your stopped container as an image, then there's no need to keep the container. You can remove any containers that you don't need using:

docker rm <container_name>
For example, my container was called gifted_euclid, so I would type:
docker rm gifted_euclid
once I was ready to get rid of it. In practice, I tend to keep a single docker container hanging around for all of my experiments, and just keep my code up to date. Docker containers can be built using a docker config file, and this might prove to be a more space conserving way.

Some useful links:

https://github.com/nodejs/docker-node/blob/main/docs/BestPractices.md
https://hub.docker.com/_/node/tags
https://nodejs.org/en/docs/guides/nodejs-docker-webapp/