**Docker**

Docker fundamentals :

* Docker is linux based so we always need the Linux kernel to be installed on the host machine to be able to use containers.
* Docker handles containers which contains applications (Dependencies…)
* It is lightweight compared to a VM.
* It does not work well for every app out there for example, there may be cases where an app runs better if it is not containerised.

Install Docker on Ubuntu :

sudo apt install docker.io

systemctl enable docker

systemctl start docker

**Run a container :**

sudo docker run hello-world

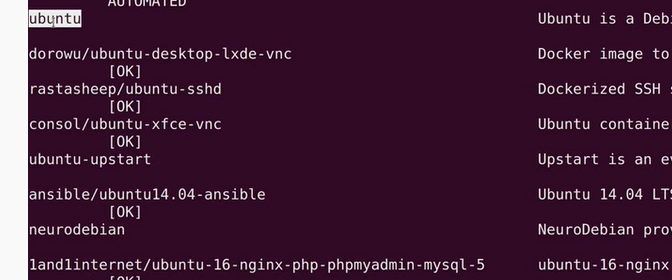
**Remove the sudo for using docker :** It would be nicer if we remove the sudo demand for our docker commands, to do that :

sudo usermod -aG docker elie

So here, we added the user elie to the docker group. A logout and login must be done.

Docker images :

**See container images :** The command below will show us images downloaded and stored locally on our machines. docker images

**Search an image :** To seach for a specific container image we can search by using a keyword docker search ubuntu

Here we can see that there are some repositories like ‘dorowu’ where inside of it is the actual contrainer image. We can also see in the first line, the container ‘ubuntu’ which is directly the image that we can use so we will install it.

**Download images :** The command below will let us download a container image

docker pull ubuntu

docker images

Now, we will see the the image ‘ubuntu’ that is stored locally on our machine.

We could have skip this step by using the following :

docker run ubuntu

This will automatically download the container and then run it but we learned the above step so that if we wanted to download a container image and run it later.

N.B :

In fact, docker containers quit if they have nothing to do so that there will not be a waste of ressources etc… So for example, when we ran the ‘ubuntu’ container, we will stay in the bash shell and the container will quit after running because there is no application that is running inside of this container.

**Running images :** To display the running images on our system we can use the command :

docker ps

We can can also have more details and see the history of containers, so we can see which container images where running and when it stopped :

docker ps -a

**Access images shell :** We can access the shell of a specific container so that we could launch commands on that container…

docker run -it ubuntu /bin/bash

or

docker run -it ubuntu

This will return a bash shell for that specific container so we can launch commands. The ‘-it’ creates an interactive shell inside a container.

N.B :

Containers are not stateful, so when the container dies or quits, everything added to this container will be erased, so when we relaunch the container (The container will be built from the image), only the apps or things that comes by default with the container image will be present. So the image is the blueprint.

**Make containers persist :** In this section, we will learn how we can keep containers running in the background so that it never quits.

docker run -it -d ubuntu

The ‘-d’ means deamon mode and deamon means a service that runs in the background. However, here we will not have the acces to the shell of the container because we used the

‘-d’ option that will send the container to the background directly.

So now, we need to find a way to interact with the shell and to do that we first launch the following command to see the container ID for our container :

docker ps

Then we will do the following :

docker attach 454dz4z484

So here we attached the container with the specific ID to the shell so we gain access to the shell of the container. We could also only put the first two numbers of the container ID because docker is smart enough to know that we would want to reference this container.

However, here when we quit from the shell of the container, the container will still quit so to fix that we access to the shell of the container with the attach command and then we hold ctrl and pres p and while you’re still holding ctrl press q and now when we quit from the shell and run ‘docker ps’ we will see that our container image is always running in the background.

So evertime we attach to the container shell and would want that the container stays running we should use the combination of keys described above.

Access containerized apps :

docker run nginx

Here, we launched the nginx container image and we are stuck to the command line of nginx because by default nginx stays running when we first run it.

docker run -it -d -p 8080:80 nginx

So here, we created the interactive mode (shell) for our container and we sent it to the background. Moreover, the nginx listens on the port 80 by default but it is not accessible from outside of the container so we will link the port 80 of nginx to the port 8080 of our machine. So we are opening the port 8080 of our physical machine and now we can access nginx server by going to the browser of our machine and accessing ‘localhost:8080’

N.B :

For the case of nginx container, we do not need to use the combination of key clicks to say to this container to stay running even if we quit from it because nginx by default stays running by default.

**Stop a container :** To stop a container we can do the following :

docker stop 45ce56de

**Restart a container :** To

docker run -it -d -–restart unless-stopped -p 8080:80 nginx

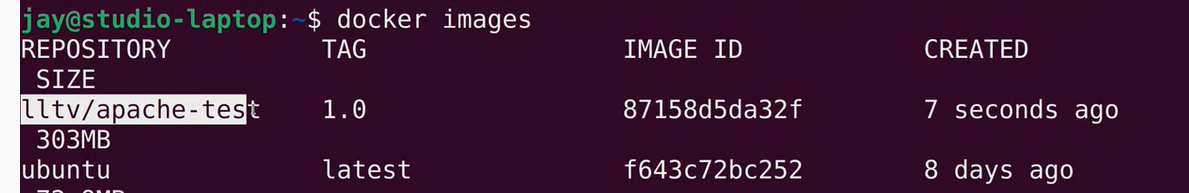
The following example starts a nginx container and configures it to always restart unless it is explicitly stopped or Docker is restarted. When the container is stopped (manually or otherwise), it is not restarted even after Docker daemon restarts.

Create images :

Now let’s say that we have a container image that is running and we have done some changes to that image, now we want to create a new image that clones our running image so that our changes will be saved in a new image that we can reuse later.

docker commit 15dzz5dc1 lltv/apache-test:1.0

So here, we created a new image under the lltv repo and we give it a tag which is the version of our apache so we can see the image that is stored locally on our machine.



docker run -it -d -p 8080:80 lltv/apache-test:1.0

Here after we run this new image and stopped the other image because we want that port 8080 be free and used by this container, we will notice that when we try to go ‘localhost:8080’ it will not work because this container image is an image that we created and not the one that we downloaded so we need to manually add an entrypoint for this container image.

docker commit –-change=’ENTRYPOINT [“apachectl”, “-DFOREGROUND”]’ 15dzz5dc1 lltv/apache-test:1.1

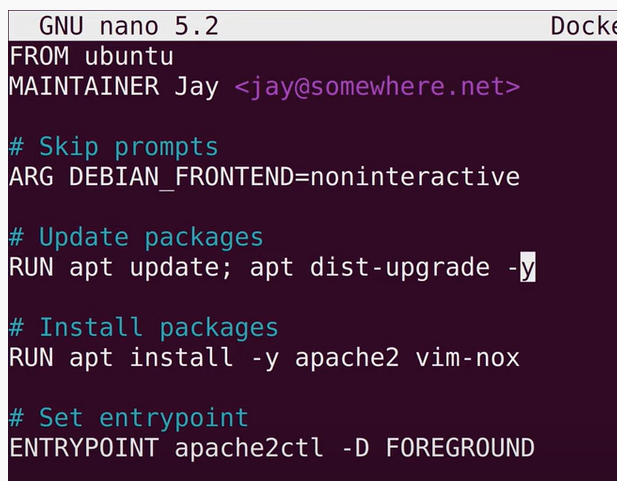
docker run -it -d -p 8080:80 lltv/apache-test:1.1

So now, when we launch the default page of the apache server we will see the page.

**Automise the process :** We will create a docker file

nano DockerFile

We will write inside of the following text file the following :



Now we will build the image by using this DockerFile :

docker build -t lltv/apache-test:1.2 .

**Delete an image :**

docker rmi 15dzz5dc1

It will not work because we had a container that uses this image. Even if the container is stopped, we can not delete the image before removing the container in the first place.

docker rm 6ddgz45j

docker rmi 15dzz5dc1