This is a cheatsheet, where you can look up commands simply

# General Docker

**Listing docker**

* docker ps -a

List all containers

* docker ps -a --filter

List containers, but also filter then accordingly.

* docker image ls [OPTIONS] [REPOSITORY[:TAG]]

List images

* docker image COMMAND

Display options to help with the Manage images

**Images vs Containers**

It is important to note the difference between images and containers. This terminology is going to become important moving forward.

Docker Images:

An image is an inert, immutable, file that's essentially a snapshot of a container. Images are created with the build command, and they'll produce a container when started with run. Images are stored in a Docker registry such as registry.hub.docker.com. Because they can become quite large, images are designed to be composed of layers of other images, allowing a miminal amount of data to be sent when transferring images over the network.

It is a kind of ready-to-use software read-only template crafted with source codes, libraries, external dependencies, tools, and other miscellaneous files that are needed for any software application to run successfully on any platform or OS.

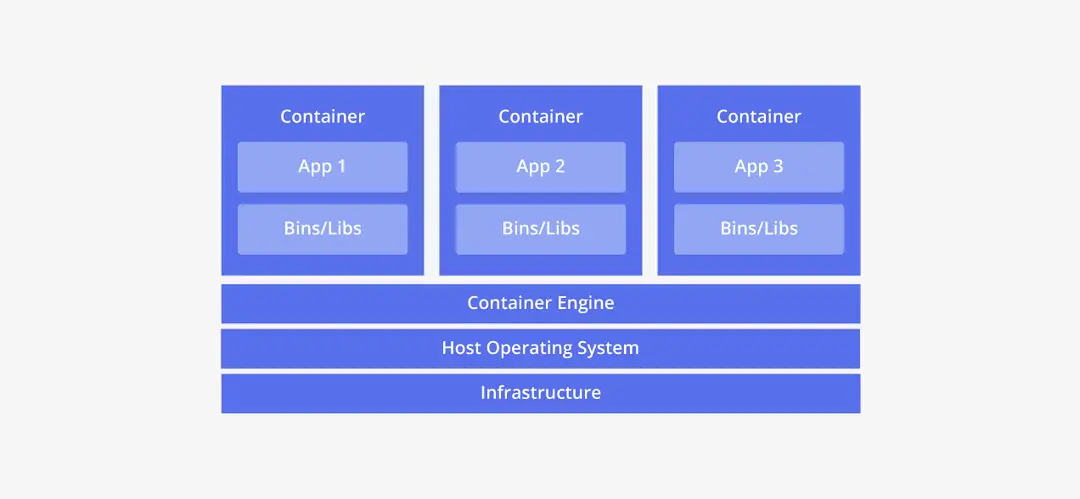
The developer community also likes to call it Snapshots, representing the app and its virtual environment at a specific point in time.

Docker Container:

To use a programming metaphor, if an image is a class, then a container is an instance of a class—a runtime object. Containers are hopefully why you're using Docker; they're lightweight and portable encapsulations of an environment in which to run applications.

A container is nothing but a box that has the ability to run the docker image templates. The moment you create a container using those immutable images you essentially end up creating a read-write copy of that filesystem (docker image) inside the given container. This adds a container layer which helps you to modify the entire copy of the given Docker image.

A container can also be considered as a cohesive software unit that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another.



**TLDR /Summary of key points** :

* Images are the things that hold all the software, containers are the things actually runs the images and by extension the software.
* You can have many images, each with different software,libaries and dependency
* You can have many containers of the same image, deployed in different places or deployed side by side. Flexibility here is key
* You can have containers without images, but nothing will happen without the software
* You can have images without containers, but nothing will happen as you need containers to run the software

Sources :

https://www.knowledgehut.com/blog/devops/docker-vs-container

https://www.geeksforgeeks.org/difference-between-docker-image-and-container/

<https://web.archive.org/web/20190519063425/http://paislee.io/how-to-automate-docker-deployments/>

**Creating an image(python based example)**

It is assumed that the python project has already been created, in this example app.py

Step 1: requirements.txt

* pip3 freeze > requirements.txt

A txt file is created in directory/current location. Here you can edit and add additional libraries/systems are required

Step 2 : Docker file

* type nul > Dockerfile

A file without an extension is created . The Dockerfile is “is a text document that contains all the commands a user could call on the command line to assemble an image”

* Step 3: Editing the Docker file

Open the docker file using your program of choice. Text editors can work

#decide what type of software we are dealing with. In this case, python

FROM python:3.10

#what file are we dockerising, in the case app.py

ADD app.py .

# this code copy the requirement, this is to install any libraries and dependencies.

COPY requirements.txt requirements.txt

RUN pip3 install -r requirements.txt

CMD [ "python","./app.py" ,"--host" ,"0.0.0.0","--port","80"]

An example of what a docker file can look like.

Step 4: Build the image

* docker build --tag [name-of-tage][name-of-docker-image]

Example

* docker build --tag python-docker

While the [--tag] option is optional it is highly recommended as the default will be “lastest”, and can get very messy

Step 4.5: Retag the image

* docker tag [SOURCE\_IMAGE][:TAG] [TARGET\_IMAGE][:TAG]

This if and when you want to chang the tag of an image.

Example 1 : using name

* docker tag python-docker:latest python-docker:v1.0.0

Example 2 : using ID

* docker tag 8cae92a8fbd6 python-docker:v1.0.0

Source for this section : https://docs.docker.com/language/python/build-images/

**Deleting/remove images**

docker image rm [OPTIONS] IMAGE [IMAGE...]

* To remove a docker image

Example 1: removing a single container

* docker rm 8cae92a8fbd6

Example 2: removing multiple containers

docker rm 277f2dfafaed 44f0562eb0fb 3017637ec75f

**Login to docker**

* docker login -u <username>

Example:

* docker login -u blueswordperson

**Start a docker container**

* docker start name of container

Example:

* docker start mongo-1

OR

Start it from the docker desktop app

**Stop a docker container**

* docker stop my\_container

Example:

* docker stop mongo-1

OR

* Stop it from the docker desktop app

**Remove docker container**

* docker rm -f <docker container ID\*>

Example:

* docker rm -f 58fa161e5c960e7aa1e1f6daa26639256d24db79523fd9f8771183f6fb0a1e07

\*This ID can be found in the docker desktop

**Change docker container name**

**View activity logged by Docker**

* docker logs -f <docker container ID\*>

Example:

* docker logs -f bc6feac09a0df444f16e0460d3a2a46d72fb1f630168e73f46011c1ecefa3f6b

\*This ID can be found in the docker desktop

# —----------------------------------------------------------------

Beyond this point are older, information that I dont fully know where to place. Probably wont have in the final submitted version, but might be useful until then

**Creation of key files when building a container**

* type nul > <file you want to create, including extension>

Example 1 : requirements.txt ( mostly python)

type nul > requirements.txt

A txt file is created in directory/current location. Here you can list if additional libraries/systems are required

Example 2 :

type nul > Dockerfile (optional)

Example 3 :

type nul > docker-compose.yml (required for image)

A dockercompose file is create. This is important as it is required to use to run the docker container.

# MongoDB-Docker

Start mongodb

Start sql