Docker Training

Draft Notes

swarm is equivalent to Kubernetes

Docker compose enables you to manage multiple containers at the same time

From Virtualization to Docker

Virtualization = group of techniques to run multiple OS in the same machine by spawning VMs. These VMs will have to share resources: bandwidth, ram, network card.

Why virtualize? Optimize the use of a server before 1 app = 1 server: This costs too much money and energy. Multiple VMs helps us make use of the same machine for multiples tasks, while creating a clean separation between the different environments (instead of deploying multiple apps in the same place).
It's also easier and more dynamic (way more direct and quicker to add ram on a VM than on a physical machine).

Different types of Virtualization

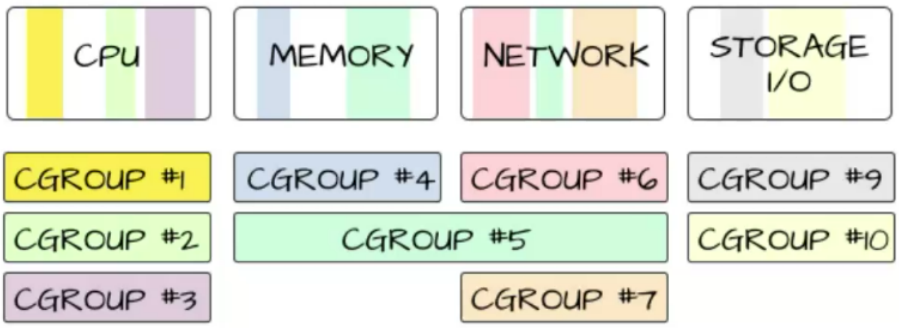
Material Virtualization: Hypervisors installed and directly manipulates the hardware

Type 2 hypervisors: using software

Type 3: Container notion Kind of the same concept as type 2 hypervisors in that they are software. The main difference between a container and a VM is that in a VM has a full-on OS installed in it. Not the case in a container.

Stuff that you find in a container:

- /rootfs: file system
- namespaces: they help isolate process
- Control groups: limit the use of resources like this:



you see that each control group has a certain share of the machine resources.

Introducing Docker

Docker Environment

Containers are managed by a docker daemon.

We are on a microservice architecture, one container = one service (we do not put DB and app in the same container)

We are on Ephemeral storage By default, a container is stateless, if you destroy it, it won't remember/store its store for when you spawn it again.

CLI helps us manage these things.

Architecture Docker

What you will find when you play with docker:

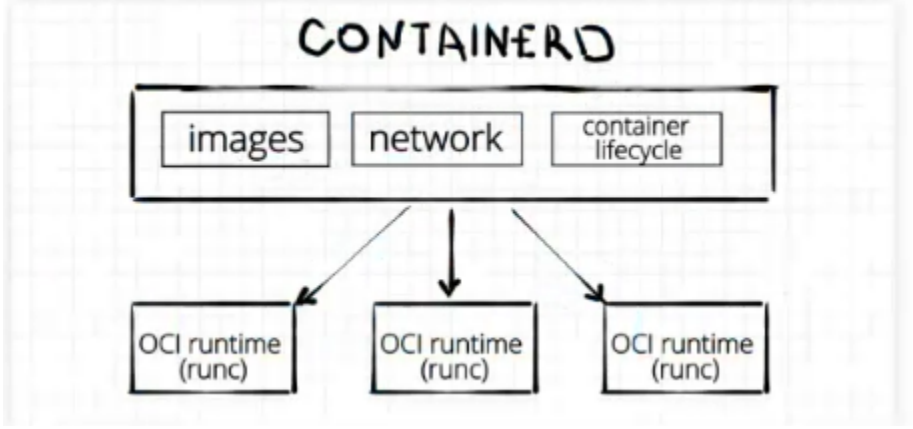
- the client = the interface that allows you to communicate with the daemon
- the docker daemon running in the background in your host machine
- docker registry: backup of the container image
- docker file: to build images

- Docker image includes instructions to build the container
- container = instance of an image running.

Components of Docker

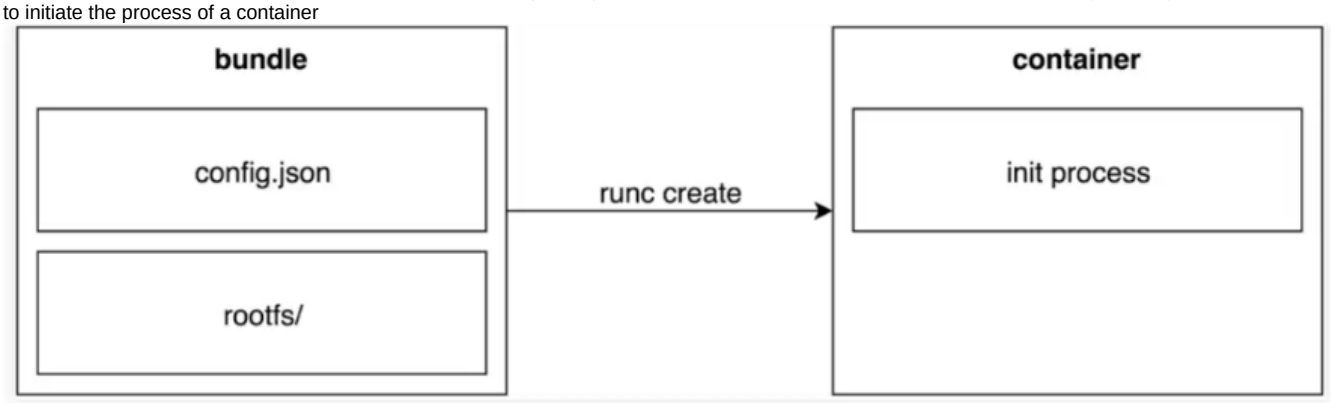
Docker Engine = allows you to interact with the container

Containerd = tool that executes the containers. Contains local images, manages networking and lifecycle of a container



. It runs in the Docker engine.

RunC = the lowest couche there is in Docker's architecture (kind of). runc runs a runc create function on a bundle containing a config file and a file system to initiate the process of a container



Installing Docker

you can install docker desktop that will have docker engine, docker compose, and a little UI.

you can also install the different elements: engine, docker build, dockerhub etc...

On client machines, docker desktop works fine. On server however, we install only the bare minimum elements.

Desktop: Dev Environnements are "the packages of compose"

Life Cycle of a Docker Container

A docker container can be in different states:

- created: the file system is created
- running
- paused
- killed
- stopped

Basic commands to handle a container:

Command	Description
docker create image	Create container
docker run image [command]	= create + start
docker rename container new_name	Rename container
docker update container	Update container config
docker start container	Start container
docker stop container	Stop container
docker kill container	Kill container
docker restart container	= stop + start
docker pause container	Paused container
docker unpause container	Unpaused container
docker rm [-f] container	Remove container

run, stop, restart, rm are the most used.

run container in background : -d = detached option.

get a pseudo-terminal on container: -t, -i for interactif

Use user: -u [username] inside the contaienr

Definer container hostname: -h

start at a certain point in file system tree: -w

define env variable: -e

docker diff <container id> difference between the container and the image from which it has been spawned.

docker inspect gives a whole bunch of info on a container.

docker cp to copy from host to container and vis-vers-ça

Root Privilege

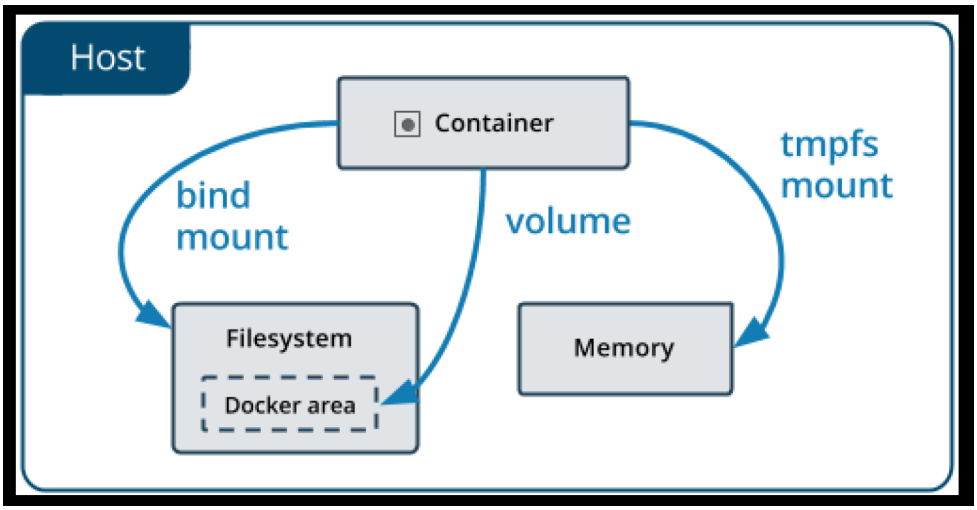
Docker is used by default with root privileges. (in linux and mac, needs to be run with sudo privileges, or create a group:

(root on linux), to avoid this it is possible to create a docker group (when launching the docker
daemon, this group will be automatically called) and add your user.
• Under Ubuntu for example:
– sudo groupadd docker
– sudo usermod -aG docker $USER
– It is necessary to log out and log in again for this to be taken into
account

Different types of mounts

3 types of mounts:

1. Volumes are stored in hosts's file system: Best choices for persisting
2. Bind mount data: Can be stored
   anywhere on the host's filesystem. Processes in the docker container and on
   the host system can modify the files (useful for sharing code for example)
3. Tmpfs mount: are stored only in memory and are never persisted on the file
   system



Docker Image

- snapshot of a filesystem + metadata

Produce the Image

Most of the time, we start from an existing image, and build on top of it, layers after layers.
Each image is identified by a sha1 id.

Image commands

Command	Description
docker images	List local images
docker history image	Show image's history
docker inspect image	Show image's low-level information (json)
docker tag image tag	Tag image
docker commit container image	Create image from running container
docker import url	- [tag]
docker rmi image	Remove image

Image Tag

Some images do not have the tag "latest". By default, when you pull or push an image from dockerhub without specifying the tag "latest", it won't work for those images. You'll have to explicitly put the tag of the version that you want.

Images have 3 namespaces: a registry, a username or organization name, and a the image name.

docker ssh to transfer images from a machine to another.

What is a Dockerfile

a Dockerfile contains instructions (capital letter words).

ENTRYPOINT cannot be overridden. CMD can.

Command	Description
FROM image	Starting image for build
COPY path dst	Copy path in context in directory dst in container
ADD src dst	As copy but able to untar archives and get URLs with HTTPS
RUN command	Launch command when building container
CMD command	Launch command when running container : can be overrident on docker run
ENTRYPOINT command	Launch command when running container : can't be overrident on docker run
USER name[:group]	User who launch command
WORKDIR path	Working directory
ENV name="value"	Environment variable
STOPSIGNAL signal	Signal to send when finishing container (instead of SIGTERM)
HEALTHCHECK CMD command	Test command to check if running inside container
EXPOSE port..	Expose TCP/UDP port

STOPSIGNAL to do some event-driven stuff. If the signal is not the one by default, I can use the detection of that signal to trigger another action.

Always prefer RUN to SHELL, because SHELL is specific to linux.