Docker

Chapter-content:

2020-2021

- Virtualization and docker
- Les conteneurs
- Docker

Table of content

2020-2021

Virtualization

O.S. Virtualization: technique that consists in running on a computer (simultaneously) operating systems as if they were running on distinct computers.
Those simulated O.S. are called Virtual Machines (VM).

hypervisor: process that creates and executes virtual machines, allocating the host's computer's hardware ressources (memory, CPU) to guest machines.

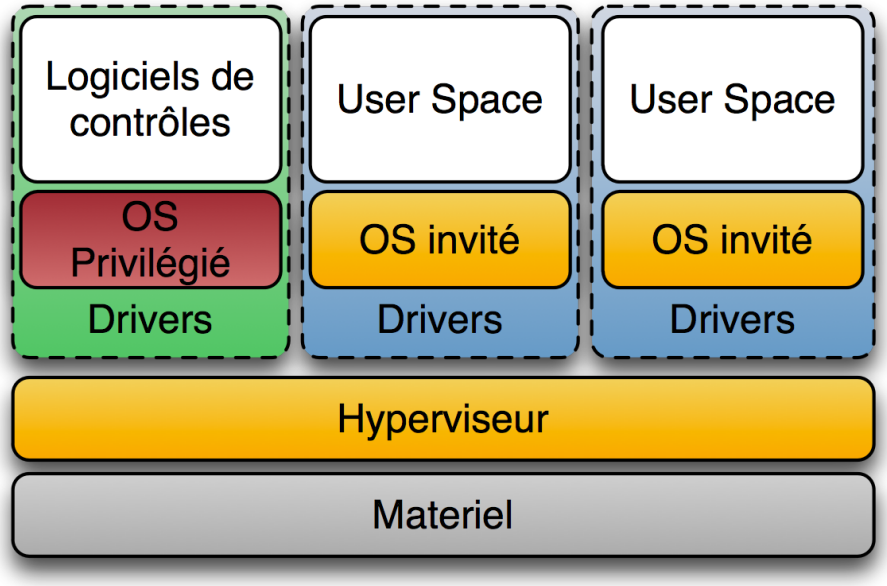
[https://doc.ubuntu-fr.org/virtualisation]

Why virtualize?

- Minimize hardware cost by reducing downtime
  - execute simultaneously several OS/applications on a same machine
- Facilitates management, faster provisioning. Improves scalability, versatility, resiliency. Allows to
  - run software/peripheral that cannot run on host hardware
  - test software in controlled environment
  - transfer application from a computer to any other computer featuring a compatible hypervisor
  - enhance security (isolation of guests)
  - facilitate recovery

Inconvénient: augmente la complexité, et la couche d'abstraction nuit aux performances/consomme des ressources.

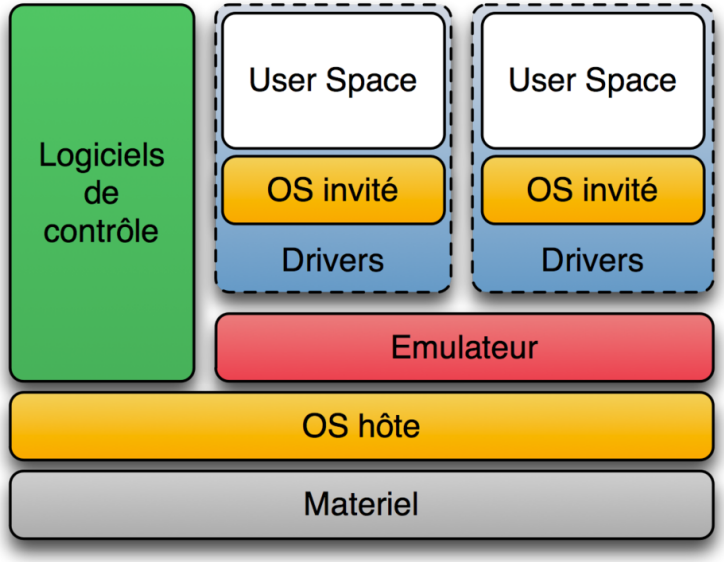
Paravirtualization = type1



paravirtualisation (hypervisor type 1 = bare metal): hypervisor is simple, runs directly on host hardware, performance close to real hardware. But guest OS must be adapted.

ex: VMware vSphere, MS Hyper-V Server

Full Virtualization= type2

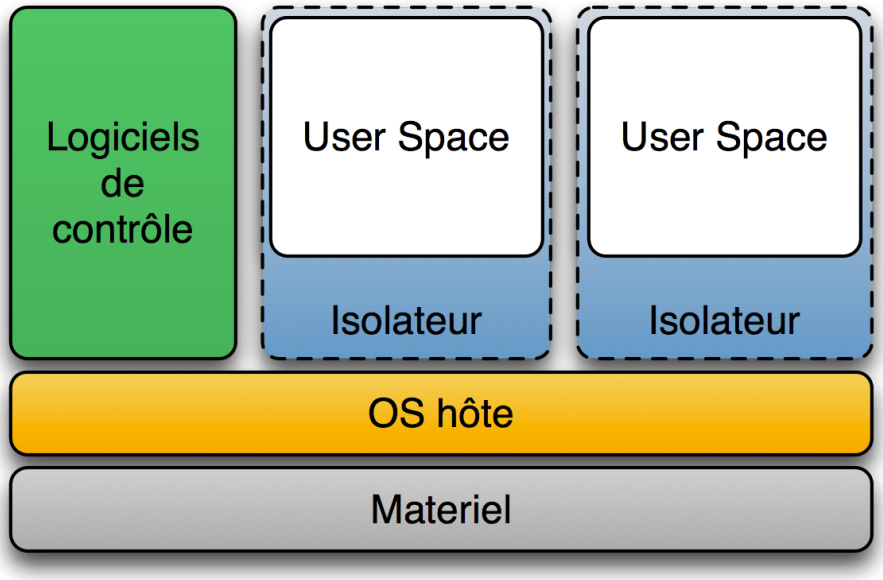


full virtualization (hypervisor type 2): runs on guest OS by simulating the whole computer (harware included). However, guest maintain access to host's CPU, RAM, storage on file.
Most common solution on personal computers. Can only virtualize a guest OS with the same hardware architecture as host (ex: x86).

ex: MS VirtualPC, Oracle VM VirtualBox, VMWare Player, VMWare Workstation

Très similaire à:
emulation: like full virtualization, but also emulates CPU etc. Poor performance.

Containerization



environnement virtuel (isolateur): Each environment has dedicated/isolated memory, but they share the host kernel, drivers. Containers are "light" but the environment must support the host OS.

Linux containers, Docker, LXD

Main virtualization techniques

- *paravirtualization (hypervisor type 1 = bare metal):* hypervisor simple, s'exécute directement sur le matériel de l'hôte, performance proche du matériel réel. Mais l'OS invité doit être adapté. ex: VMware vSphere, MS Hyper-V Server
- *virtualisation complète (hypervisor type 2):* s'exécute sur l'OS hôte en simulant complètement l'ordinateur (matériel). Par contre l'unité centrale (processeur, RAM et stockage dans fichier) sur l'hôte restent accessibles aux invités.
  La solution la plus courante pour les particuliers. On ne peut virtualiser qu'un OS invité utilisant une archi matérielle similaire à l'hôte (ex: x86).
  ex: MS VirtualPC, Oracle VM VirtualBox, VMWare Player, VMWare Workstation
- *émulation:* comme virtualisation complète mais en plus on simule l'unité centrale. L'architecture matérielle invitée et hôte peut alors différer. Performance médiocre.
- *environnement virtuel (isolateur):* Chaque environnement utilise son propre espace mémoire isolé, mais ressources systèmes (pilote, noyau) partagées sur l'hôte. Conteneurs "légers", mais les environnements doivent pouvoir s'exécuter sur l'OS hôte. Conteneurs linux, Docker, LXD

Table of content

2020-2021

Containers

Un conteneur est un silo léger et isolé qui permet d'exécuter une application sur l'OS hôte.

- isolation : containers cannot access others without explicit authorization.
- container contains own files and its data.
- when the container is destroyed, data is not kept.
- every container is created from an image
- container can be stopped, restarted, transfered to another machine.

Process vs Container vs VM

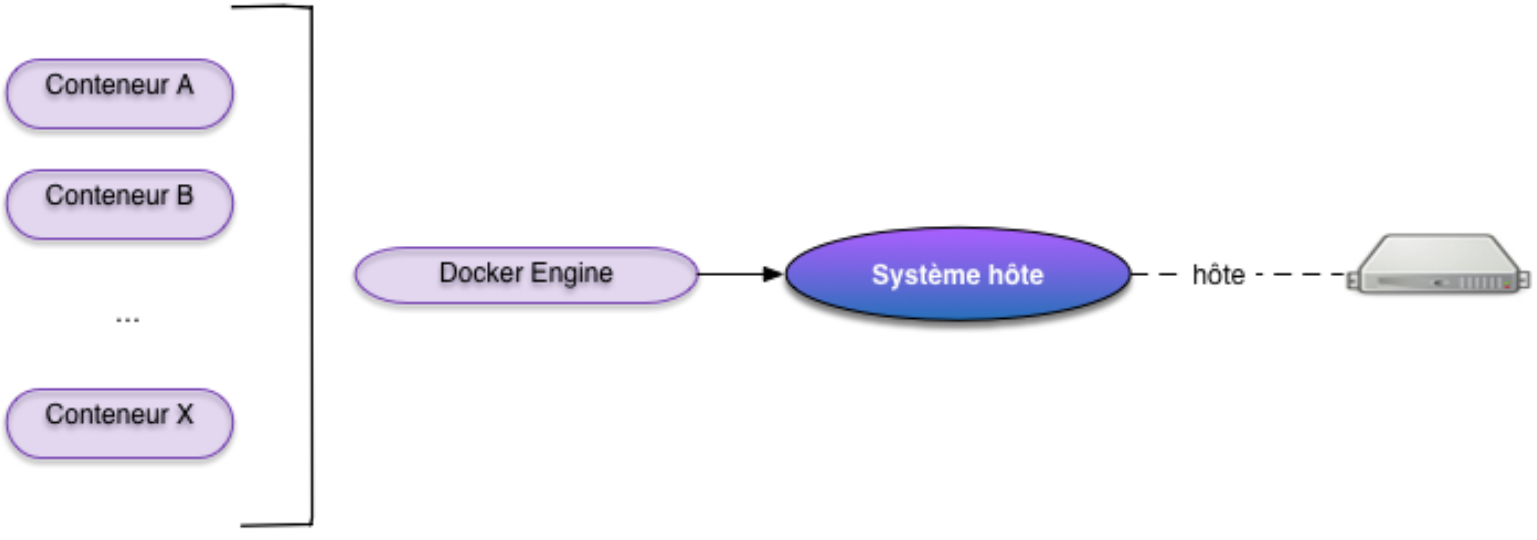
	Process	**Container**	**VM**
definition	running instance of a program	isolated group of processes managed by a shared kernel	an OS that shares host hardware through a supervisor
OS for virtualized apps	same OS	same kernel	multiple indepdt OS
Isolation	memory space and user privileges	namespaces and cgroups	full OS isolation
Security	-	improving	higher
Size	user application	image = some MBs + user app	image = some GBs + user app
Lifecycle	created by forking, generally short-lived	runs on kernel with hardly any boot (1s), generally short lived	needs boot process (>10s), generally long lived.
Computer overhead	-	<5%	>10%
Disk I/O overhead	-	generally 0	>50%
Communication	inter-process (IPC)	IPC	network devices
Support	all OS	mostly linux	all OS

Table of content

2020-2021

Docker

Slides are from :
`http://b3d.bdpedia.fr/docker.html`
Docker (stems from *Dock worker*, working qui travailler avec les conteneurs (maritimes). Docker is written in Go.
Docker engine is a client-server application, whose CLI client `docker` communicates with the server (`dockerd` for *docker daemon* through a REST API.
The docker deamon manages the containers (and images, networks, volumes. . .).
Docker allows to emulate a *distributed system* of servers, and orchestrate multiple multiple daemons with their respective containers as a service.
A *server* is a machine that is permanently connected to the network through ports; it is identified through its IP adresse.



Docker: images

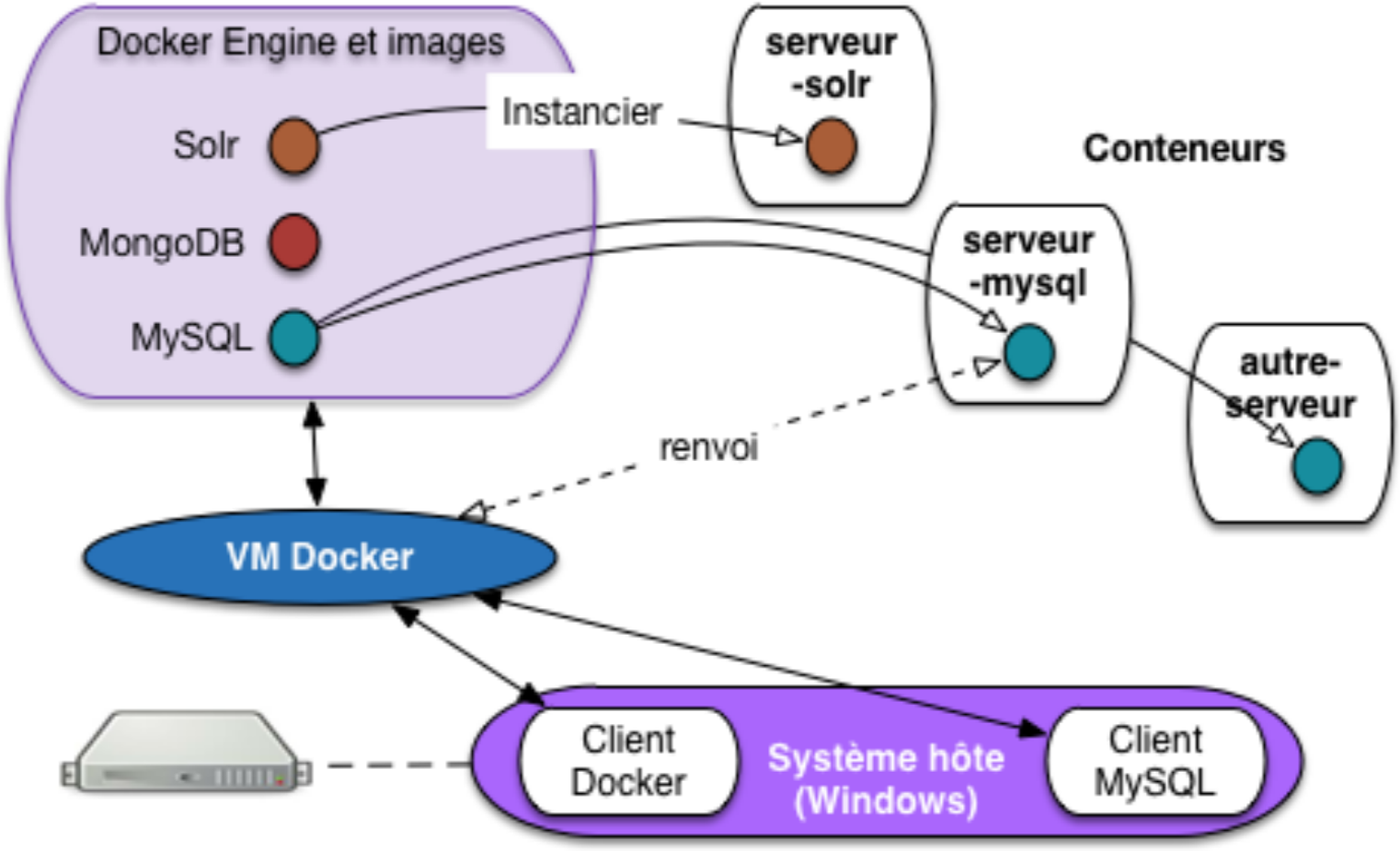


Fig. 2.3 Les images docker, constituant un pseudo système distribué

Images are downloaded into the docker engine, from an online repository. Each image can then be instantiated in one or more container.

Docker: instructions to manage images

`docker pull <image>` downloads a docker image (but run downloads automatically if required).

`docker images` lists downloaded images.

`docker rmi <image>` removes a downloaded image.

Docker: instructions to manage containers

`docker run <image> [programme sur l'image]` to create and start a container.

Options for docker run:

`--help`

`-d` container in detached mode (i.e., runs in background).

`--name <conteneur>` to choose container name

`--rm <conteneur>` to remove automatically container when we exit.

`-it <conteneur>` to launch an interactive program such as shell in container

`-v <source>:<chemin>` to mount a volume into directory "chemin" inside container

`docker exec -it <conteneur> <application>` executes the application within a running container.

Docker: instructions to manage containers (2)

`docker cp <fichier> <conteneur>:<fichier>` copies file into container. Behavious similar to Unix's cp -a sous Unix; when cp directory, recursively copies content.

`docker ps -a` list containers, whether running or not.

`docker rm -f <conteneur>` deletes a container (`-f` (force) to delete a running container).

`docker restart <conteneur>` restarts a container that had been stopped.

Examples:

```
docker run -it --name mon-linux-shell ubuntu bash
```

container name — image — program inside image

```
docker exec -it mon-linux-shell sh
```

(2^{nd} shell sur même conteneur)

Docker: instructions for cleaning up

Once you are done:

```
#!/bin/bash

# remove  containers:
docker ps -aq | xargs -r docker rm -f

# remove unused images (those without running containers):
docker images --no-trunc | awk '{ print $3 }' | xargs -r docker rmi

# remove unused volumes:
docker volume ls -qf dangling=true | xargs -r docker volume rm
```

References

https://www.cse.wustl.edu/~jain/cse570-18/ftp/m_21cdk4.pdf

https://medium.com/@jessgreb01/
what-is-the-difference-between-a-process-a-container-and-a-vm-f36ba0f8a8f7

https://fr.wikipedia.org/wiki/Virtualisation

http://b3d.bdpedia.fr/docker.html