CONTAINERS

Jean-marc Pouchoulon EMA
Avril 2019

Programme

- Présentation et point sur vos compétences.
- Hyperviseur versus Container
- Container systèmes et applicatifs
- Briques de bases des containers
- Docker
- Point sur l’actualité

Principes Généraux

Deux types majeurs de « virtualisation »

- Par containerisation (LXC/LXD, Docker…). Ce **n’est pas** de la virtualisation.
- Par utilisation d’un hyperviseur avec des machines virtuelles indépendantes (Vmware ,Kvm ,Xen, Hyper V …).
  - Un hyperviseur repose soit sur le Kernel Linux (KVM/QEMU) , soit sur un Kernel durci dit bare-metal (VMWare , Xen).
  -Un Hyperviseur permet la containérisation.

L’hyperviseur est encore essentiel mais Docker a changé la donne...

Hyperviseur versus Container: Principes

Hyperviseur

- Une machine hôte supporte l'hyperviseur qui gère les vm (virtual machines) et les devices physique.
- Chaque VM a son propre OS, linux ou windows et donc son propre kernel indépendant de la machine hôte. Elle voit des devices virtuels ou a parfois accès aux devices physiques.

Container

- Une machine hôte supporte une distribution Linux qui va gérer les containers. On a l'illusion d'une machine virtuelle indépendante.
- Chaque container:
  1. Partage le kernel Linux de la machine hôte.
  2. Fonctionne sur le principe d'un chroot
  3. Est isolé grâce à des briques de bases du Kernel Linux (capabilities, namespace, cgroup) ou par modification du Kernel

Hyperviseur versus Container: avantages et inconvénients

Hyperviseur

- \+ Une Vm est une véritable machine indépendante isolée de l'hyperviseur
- \+ Le Multi OS est permis
- \+ L'Offre pro est très mature avec des fonctionnalités très avancées
- \+ En standard la migration à chaud d'une vm d'une machine physique à l'autre.
- Peut servir à virtualiser le poste client.

\- L'hyperviseur

- \- Ralentie les performances
- \- Consomme des ressources
- \- Utilise l'artillerie lourde pour fonctionner.
- \- Peut être cher (très cher)

Container

- \+ Les performances sont identiques dans le container et l'hôte.
- \+ Les containers ne nécessitent pas de ressources supplémentaires
- \+ Techniquement astucieux le container évolue (docker)
- \+ Le container est une techno légère.

Le container:

- \- Peut être complexe à générer
- \- N'est pas aussi isolé qu'une Vm et la sécurité est dépendante de l'hôte physique.
- \- Le multios n'est pas permis.
- La migration à chaud des vm n'est pas toujours possible.
- +- Pas de poids lourds du web sur ce segment .. Jusqu'à l'arrivée de Docker

CONTAINERS

Une virtualisation en trompe l’œil mais efficace.

Les containers à l’ordre du jour

- La manipulation de VM est lourde. (Taille importante, liens avec les clones)
- Monter un VM pour une application est souvent superfétatoire et cher.
- On cherche à augmenter la densité de systèmes par machines physiques. Du fait du point précédent c’est rarement le cas.
- L’architecture doit être scalable mais la vitesse de création des VM ne permet pas de répondre facilement à ce besoin ou l’unité de réaction est la minute.

Le container est une vieille solution mais remise au goût du jour par Docker. La vision container peut être complémentaire de la vision machine virtuelle. (Des containers sur une VM sont courants).

Les solutions containers

Deux solutions intégrées aux noyaux des OS:

- LXC/LXD (Containers systèmes Linux)
- Docker (Containers applicatifs Windows/Linux)

Les briques de la containérisation

Les namespace Linux

Ils permettent à un processus Linux une vision customisée et différente des autres processus. C'est un vieux concept dont l'implémentation continue dans le Kernel au fil des années.

- Mnt (Points de montages, systèmes de fichiers c'est ce qui permet à des processus d'être chrooté et de ne pas accéder à la même arborescence).
- PID (Deux process peuvent avoir le même PID dans deux namespaces différents).
- Net (pile réseaux , on peut faire voir des cartes réseaux et tables de routages différentes par process).
- User (Deux noms d'utilisateurs peuvent avoir le même PID dans deux namespaces différents).
- Hostname (Chaque process peut voir un nom de machine dédié)
- IPC (communication inter process).

Les appels systèmes utilisés par les namespaces

- Clone(): créé un nouveau process et un nouveau namespace.
- Unshare(): Créé un nouveau namespace et y attache le processus en cours.
- Setns() : Se rattache à un namespace existant.

Les outils userland liés au namespace

unshare

root@debian71:~# unshare -u /bin/bash
root@linux:~# hostname
Linux
root@linux:~# hostname newhostname
root@linux:~# hostname
newhostname

Les outils userland liés au namespace

- **Ip route2**

root@debian71:~# ip netns add myns1
root@debian71:~# ip netns add myns2
root@debian71:~# ip netns list
myns2
myns1

Les devices réseaux (cartes physiques ou virtuelles) peuvent être migrés d’un net namespace à l’autre sauf lO (127.0.0.1)

Les sous systèmes des Cgroups

- cpuset : affectation des cpu et de la mémoire à un groupe de process (NUMA)
- blkio : permet de répartir la bande passante par device.
- perf_event : permet de monitorer un cgroup avec des outils d'analyse de la performance (sans ca on ne monitore q'un seul process).
- cpu: permet de répartir les cpu par cgroups.
- cpuacct: permet de générer des rapports automatiquement sur le CPU consommé par les process d'un cgroup, y compris les tâches filles.
- memory: permet de générer des rapports automatiquement sur la mémoire consommé par les process d'un cgroup, de la limiter à une valeur et même de killer le process par "oom killer" si il dépasse cette limite.
- devices: permet ou interdit l'accès des devices au process des cgroups.
- freeze: permet de suspendre ou de réveiller des process des cgroups.
- net_prio : permet d'affecter une priorité au traffic réseau par cgroups The Network Priority (net_prio) subsystem provides a way to dynamically set the priority of network traffic per each network interface for applications within various cgroups
- net_cls : permet de tagger des paquets réseaux par cgroup afin de leur donner une priorité (par l'outil tc)

Cgroups

Cgroups (control groups) sont une fonctionnalité du Kernel Linux pour limiter , compter et isoler l'utilisation de processus).

Source wikipedia

On distingue des sous systèmes qui permettent de contrôler l'appétence d'un groupe de process (cgroup) dans des domaines telles que le cpu , la mémoire , les IO, la bande passante... Ou d'isoler ce groupe de devices.

Les cgroups sont dans le Kernel. Ils sont accessibles en userland via un virtual file system. Ils sont manipulables via la création de directories ou de fichier dans ce VFS.

Cgroups cas d’usage

- Source https://www.kernel.org/doc/Documentation/cgroups/cgroups.txt

```
CPU :           "Top cpuset"
                  /       \
         CPUSet1          CPUSet2
            |                |
         (Professors)     (Students)

         In addition (system tasks) are attached to topcpuset (so
         that they can run anywhere) with a limit of 20%

Memory : Professors (50%), Students (30%), system (20%)

Disk : Professors (50%), Students (30%), system (20%)

Network : WWW browsing (20%), Network File System (60%), others (20%)
                           / \
         Professors (15%)  students (5%)
```

Manipuler les Cgroups

Deux façons de faire:

1. Via un montage d’un filesystem de type cgroup
2. Via le package cgroup-bin

Les cgroup sont positionnés sous /sys/fs/cgroup par les distributions Redhat et Debian mais on peut affecter un autre point de montage (voir TP)

Les Capabilities

- En natif sous linux, chaque process pour effectuer une action privilégiée doit avoir un jeton (posix capabilities) prouvant qu'il est autorisé à le faire.

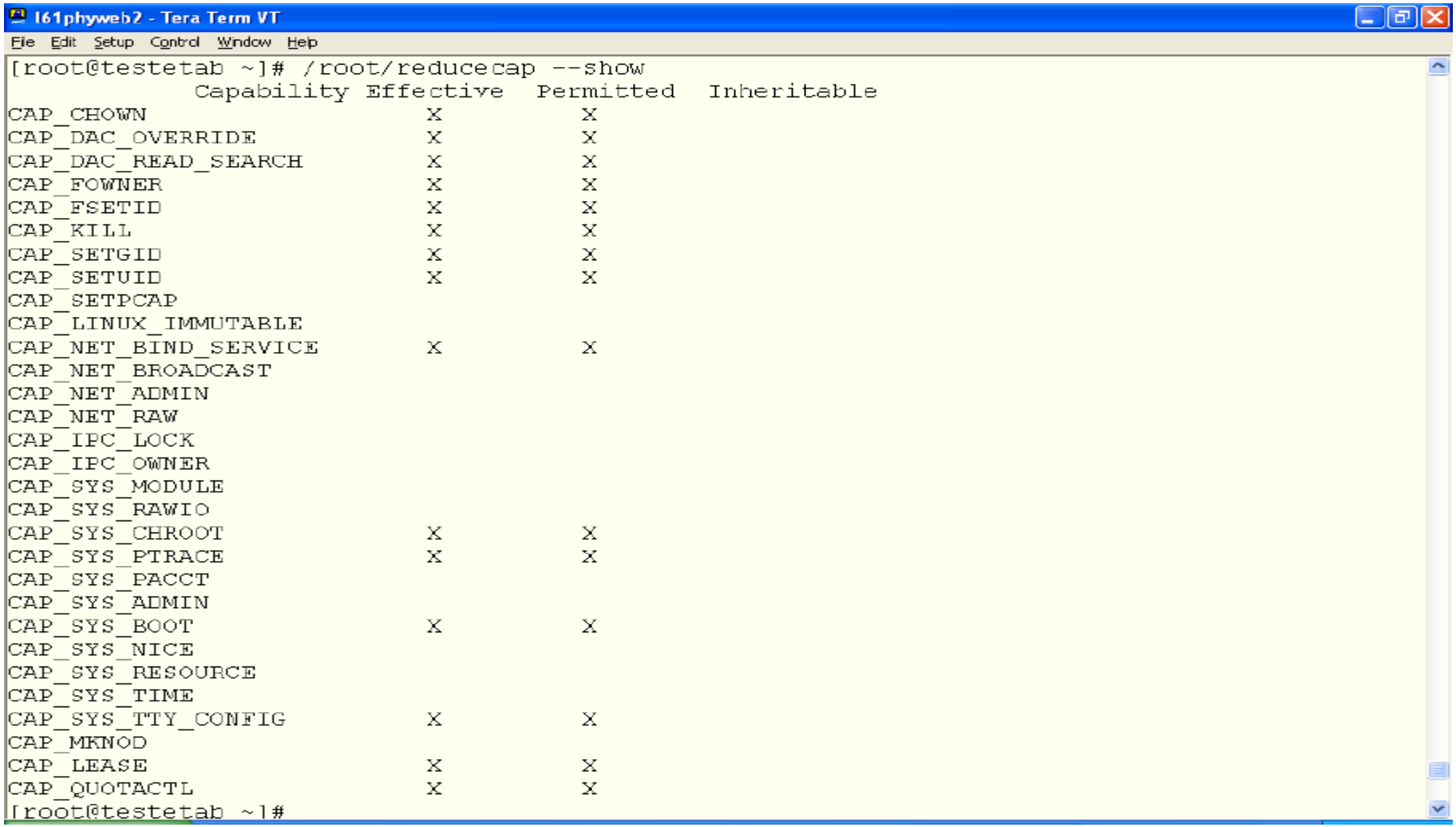
Ex: On peut interdire à root de sniffer en supprimant CAP_SYS_NET

[root@testetab /]# tcpdump -i any
tcpdump: socket: Operation not permitted

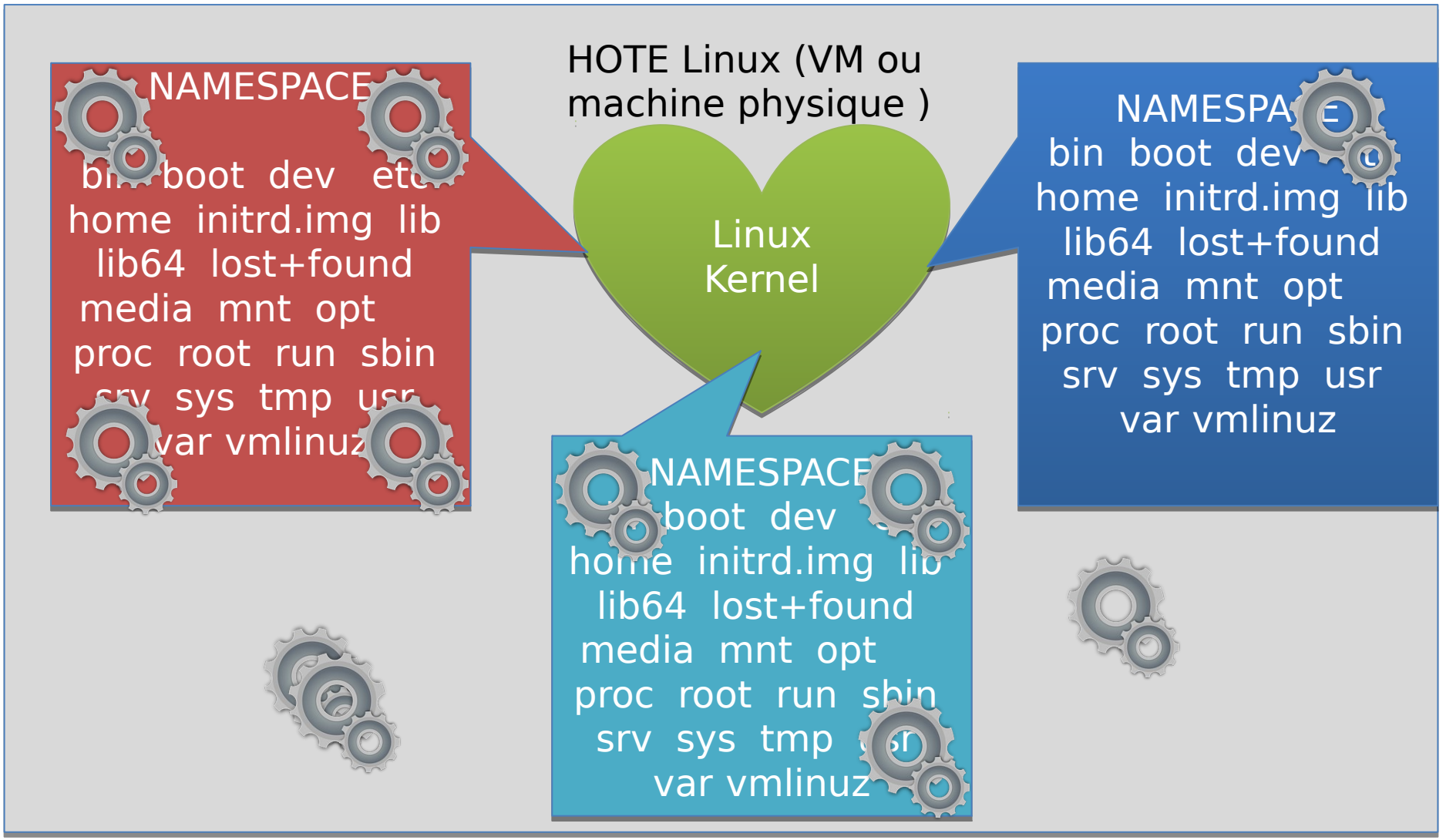
Capabilities sur une Debian standard

```
root@debian71:~# reducecap --show
          Capability Effective  Permitted  Inheritable
CAP_CHOWN                    X          X
CAP_DAC_OVERRIDE             X          X
CAP_DAC_READ_SEARCH          X          X
CAP_FOWNER                   X          X
CAP_FSETID                   X          X
CAP_KILL                     X          X
CAP_SETGID                   X          X
CAP_SETUID                   X          X
CAP_SETPCAP                  X          X
CAP_LINUX_IMMUTABLE          X          X
CAP_NET_BIND_SERVICE         X          X
CAP_NET_BROADCAST            X          X
CAP_NET_ADMIN                X          X
CAP_NET_RAW                  X          X
CAP_IPC_LOCK                 X          X
CAP_IPC_OWNER                X          X
CAP_SYS_MODULE               X          X
CAP_SYS_RAWIO                X          X
CAP_SYS_CHROOT               X          X
CAP_SYS_PTRACE               X          X
CAP_SYS_PACCT                X          X
CAP_SYS_ADMIN                X          X
CAP_SYS_BOOT                 X          X
CAP_SYS_NICE                 X          X
CAP_SYS_RESOURCE             X          X
CAP_SYS_TIME                 X          X
CAP_SYS_TTY_CONFIG           X          X
CAP_MKNOD                    X          X
CAP_LEASE                    X          X
CAP_QUOTACTL                 X          X
root@debian71:~#
```

Capabilities sur un guest vserver



Containers: Schéma général



Docker

Le container applicatif

Genèse de Docker

Docker a *probablement* été créé par des mutants hybrides mi-développeurs, mi-adminsys et du coup:

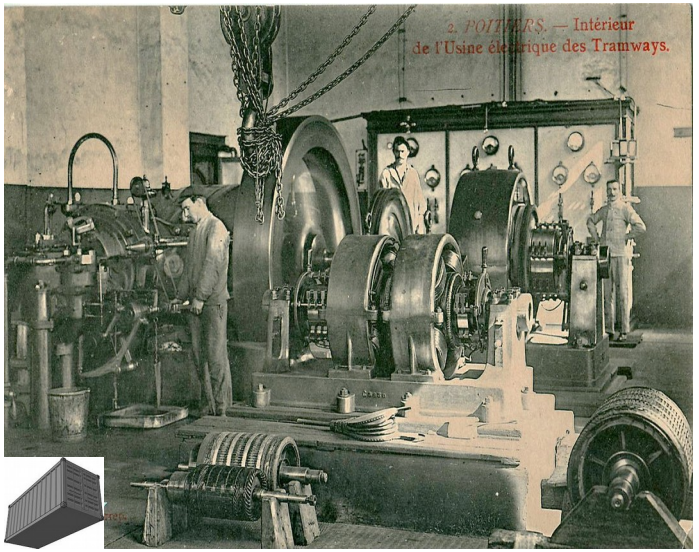
- La philosophie objet y est très présente (immutablilité, externalisation des parties variables comme les logs, encapsulation, division en micro-services …)
- On peut comitter un container comme un code. La persistance a été pensé dès le départ.
- C'est une combinaison de briques existantes comme on peut ré-utiliser du code.
- L'application est la brique de base de la solution et non pas la machine virtuelle. On cherche à répondre aux besoins métiers.
- La scalabilité a été prise en compte au départ. La gestion des MEP aussi.

Bref un gentil ? monstre très disruptif.

Cycle de vie simplifié d'un container

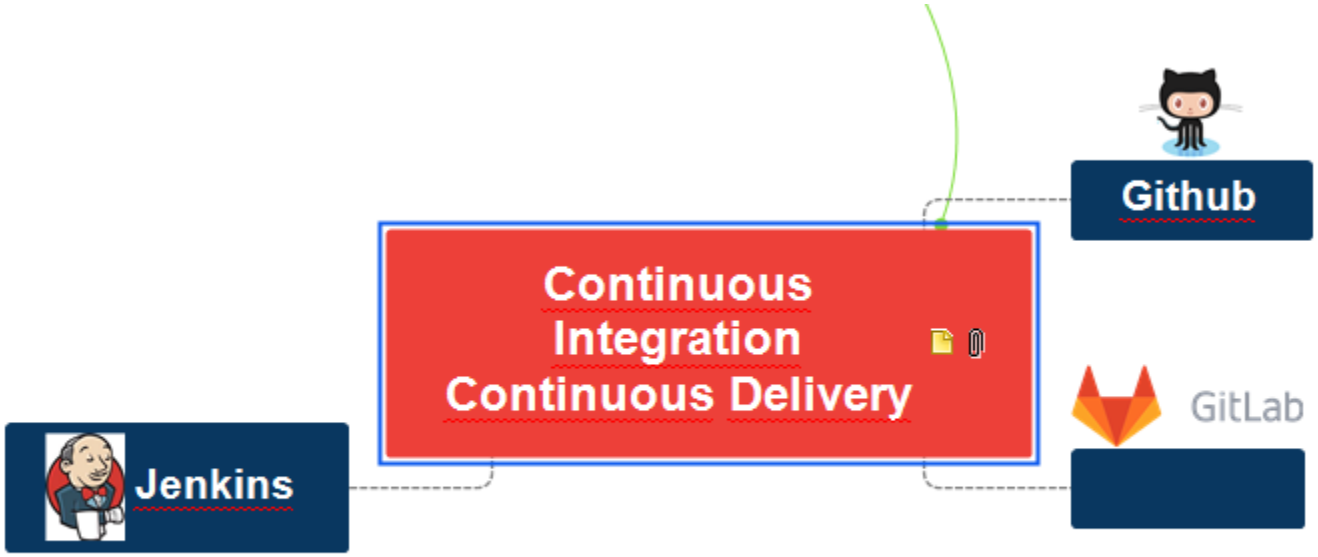
Dev

Stockage (registry)



Production (Ops)

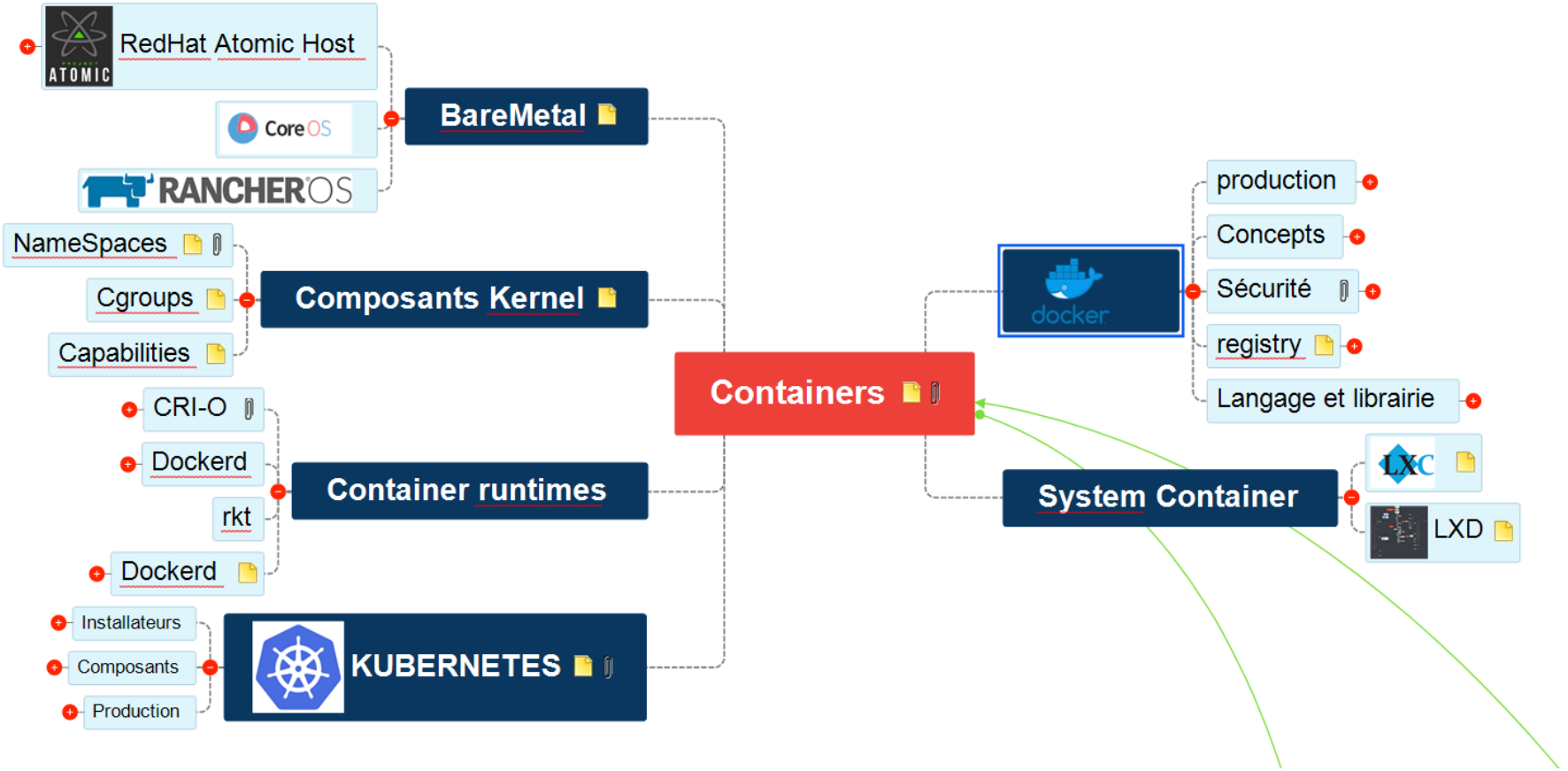
Cycle de vie simplifié d'un container



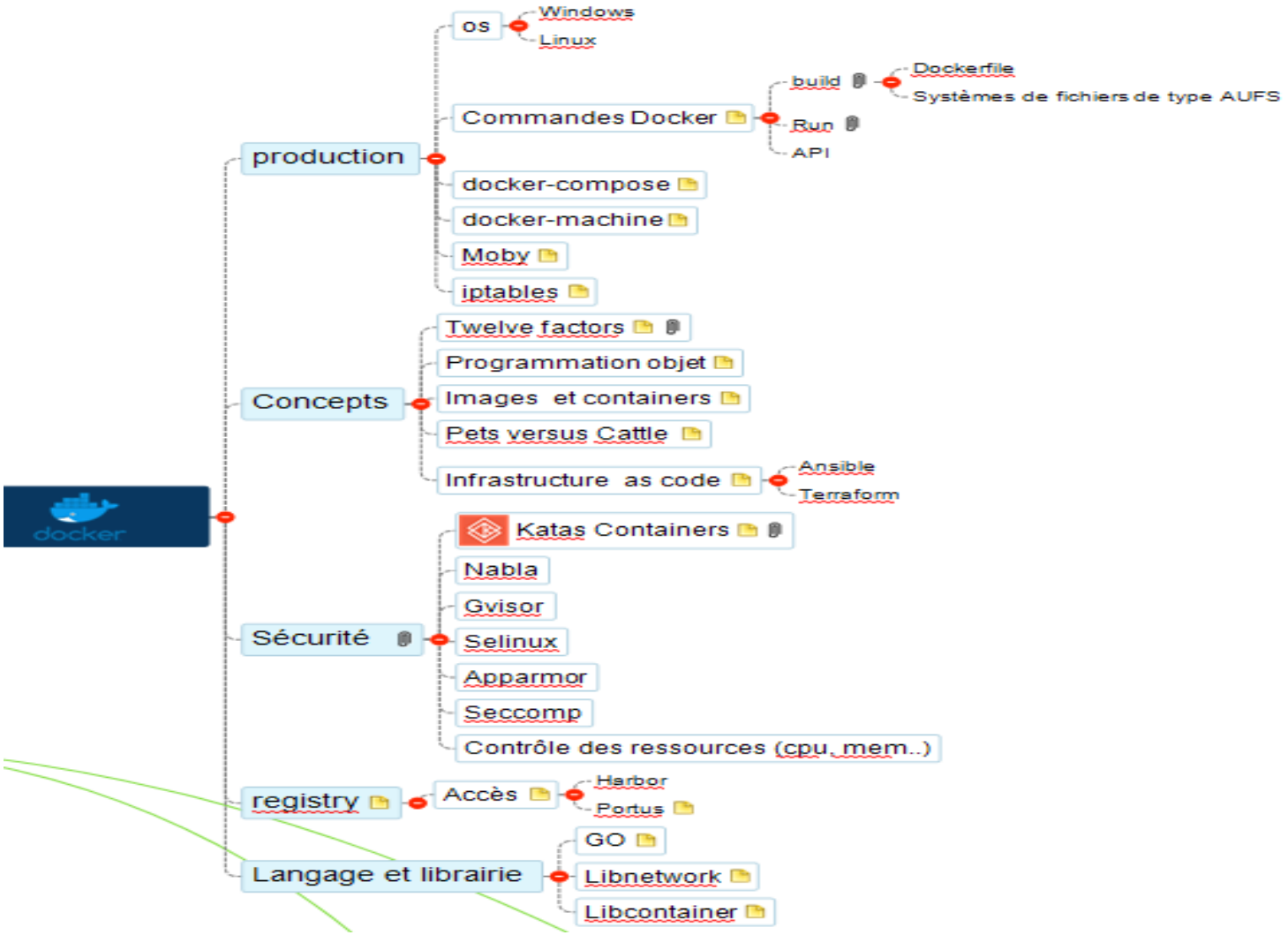
Eco-système «Container applicatif »

redhat.

Eco-système «Container »



« Docker »

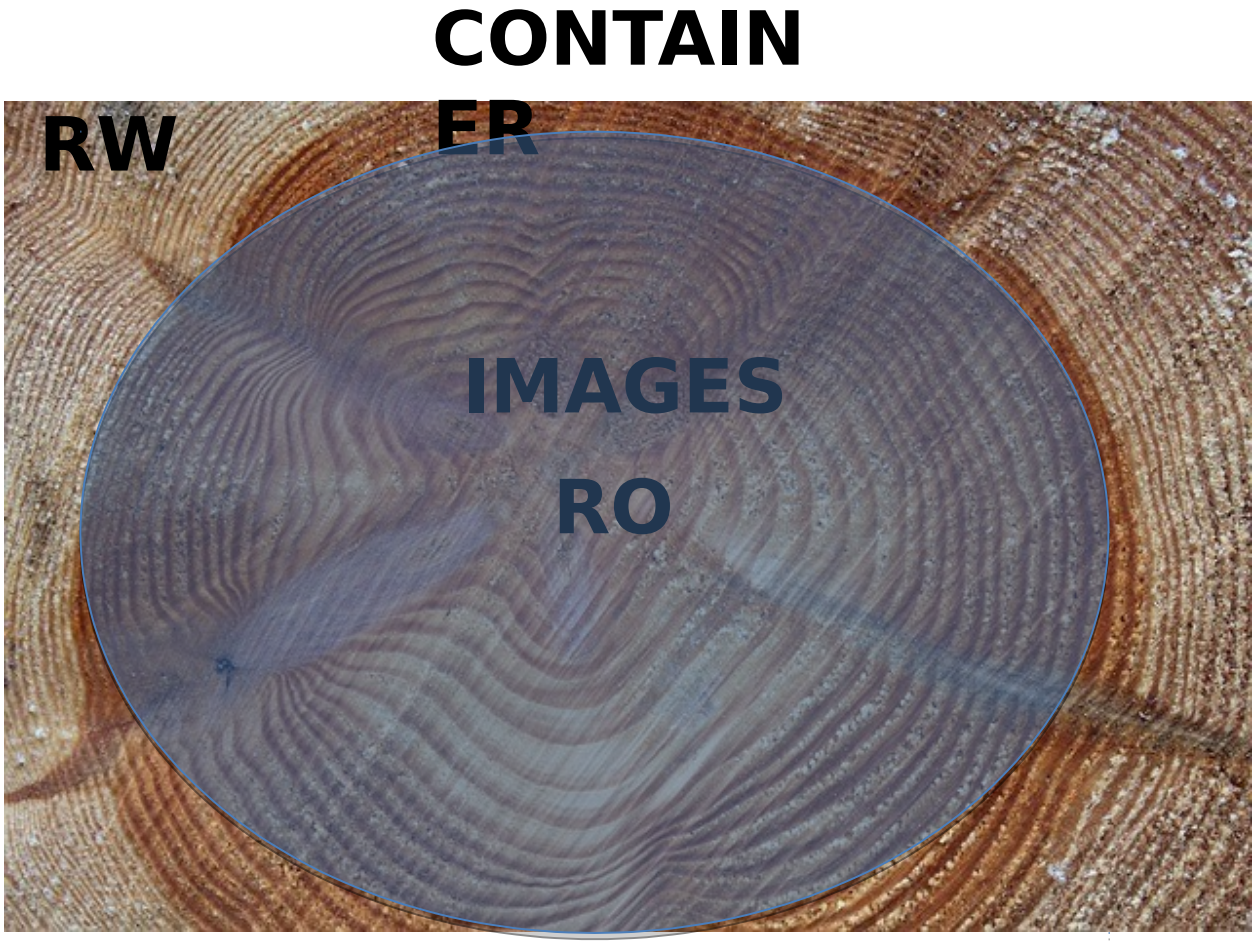

docker
production
os
Windows
Linux
Commandes Docker
build
Dockerfile
Systèmes de fichiers de type AUFS
Run
API
docker-compose
docker-machine
Moby
iptables
Concepts
Twelve factors
Programmation objet
Images et containers
Pets versus Cattle
Infrastructure as code
Ansible
Terraform
Sécurité
Katas Containers
Nabla
Gvisor
Selinux
Apparmor
Seccomp
Contrôle des ressources (cpu, mem..)
registry
Accès
Harbor
Portus
Langage et librairie
GO
Libnetwork
Libcontainer

Briques logicielles de base de la solution docker

Docker repose sur 3 composantes:

1. Docker Client : Elle a pour but de piloter le cycle de vie du container. Un client docker sur Mac ou Windows peut piloter des containers Windows ou Linux.
2. Docker Engine: C'est le moteur en Go qui va à partir des briques de bases du système Linux (CGROUPS-NAMESPACES) fabriquer les containers.
3. Docker Registry: C'est le point de stockage des containers. Il est essentiel dans le cycle de vie du logiciel.

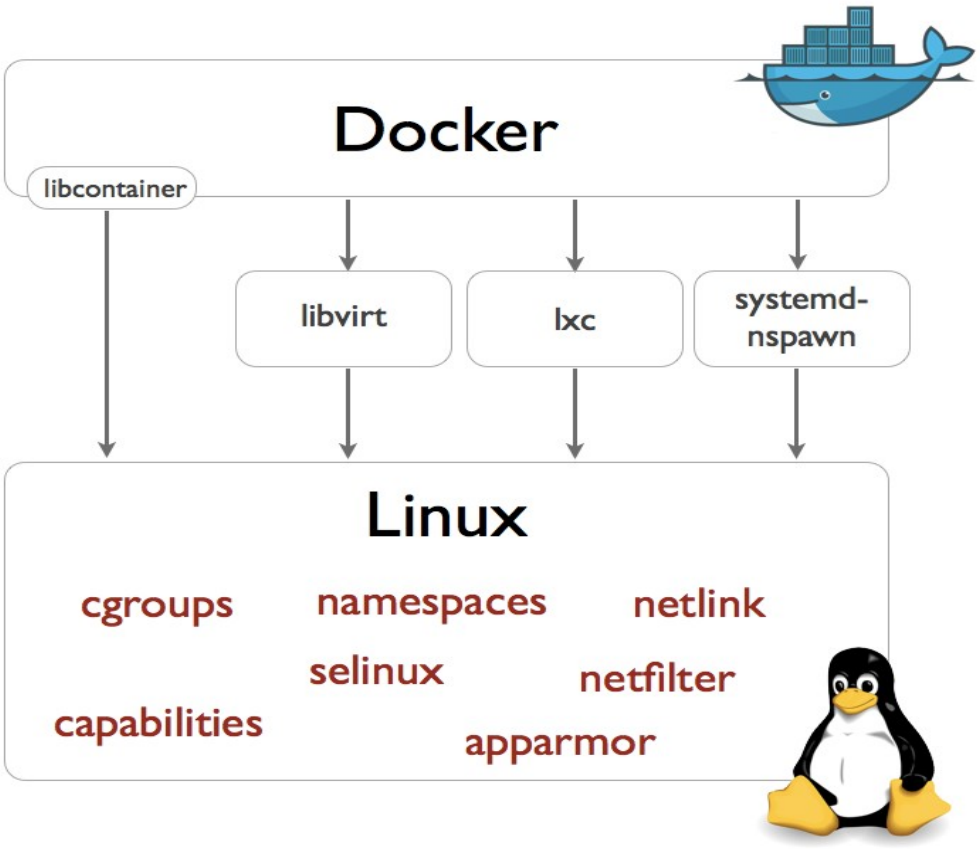
Docker et Union File System



Les fondements techniques de Docker

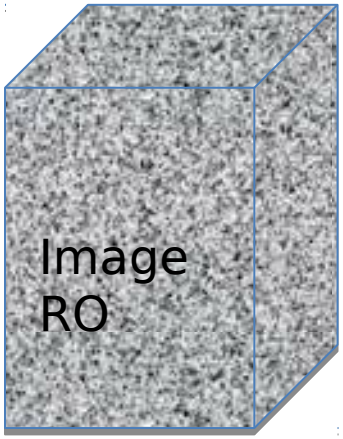
- Il permet de créer et de manager des containers basés sur le kernel Linux (Namespaces, Cgroups , Capabilities) tout comme LXC.
- Copy OnWrite Storage et UFS: Un container Docker est créé instantanément à partir d'une image en lecture seule. On écrit juste les différences entre l'image et le container sur le disque à part.
- Docker est développé en GO et se base sur la libcontainer pour créer les containers. (c'était LXC par défaut).
- Un format de container ouvert basé sur Docker est défini (www.opencontainers.org), réunissant de très nombreux acteurs de l'IT.

Architecture Docker

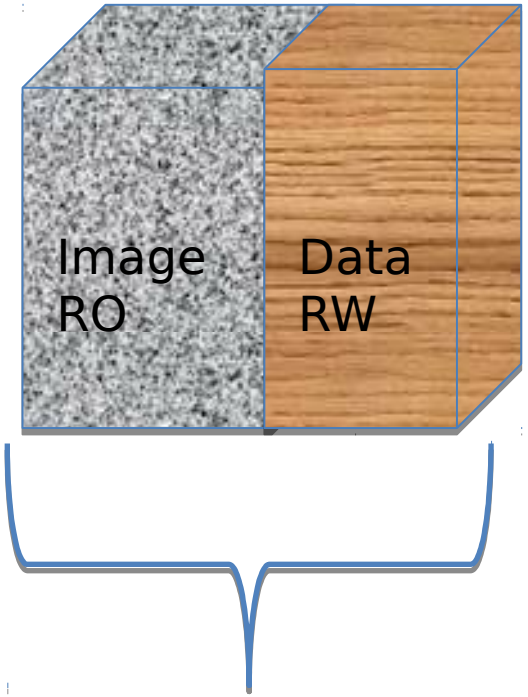


Source docker

Commandes client docker



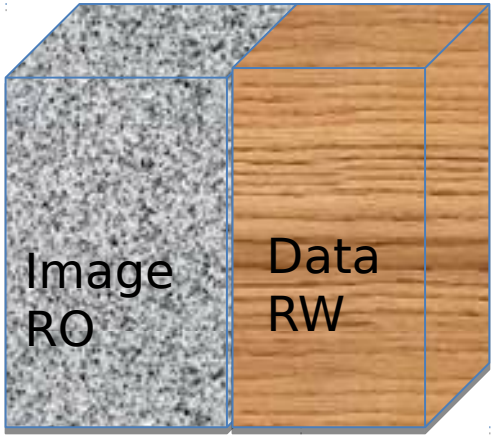
docker create
<imageid>



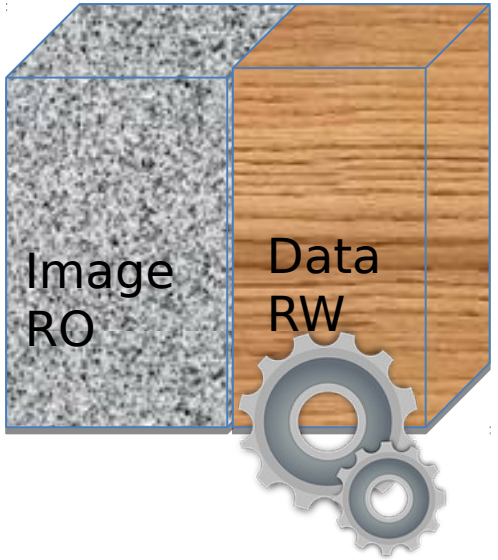
Création du container
sans le démarrer

Union File
System=>Containe
r

Commandes client docker



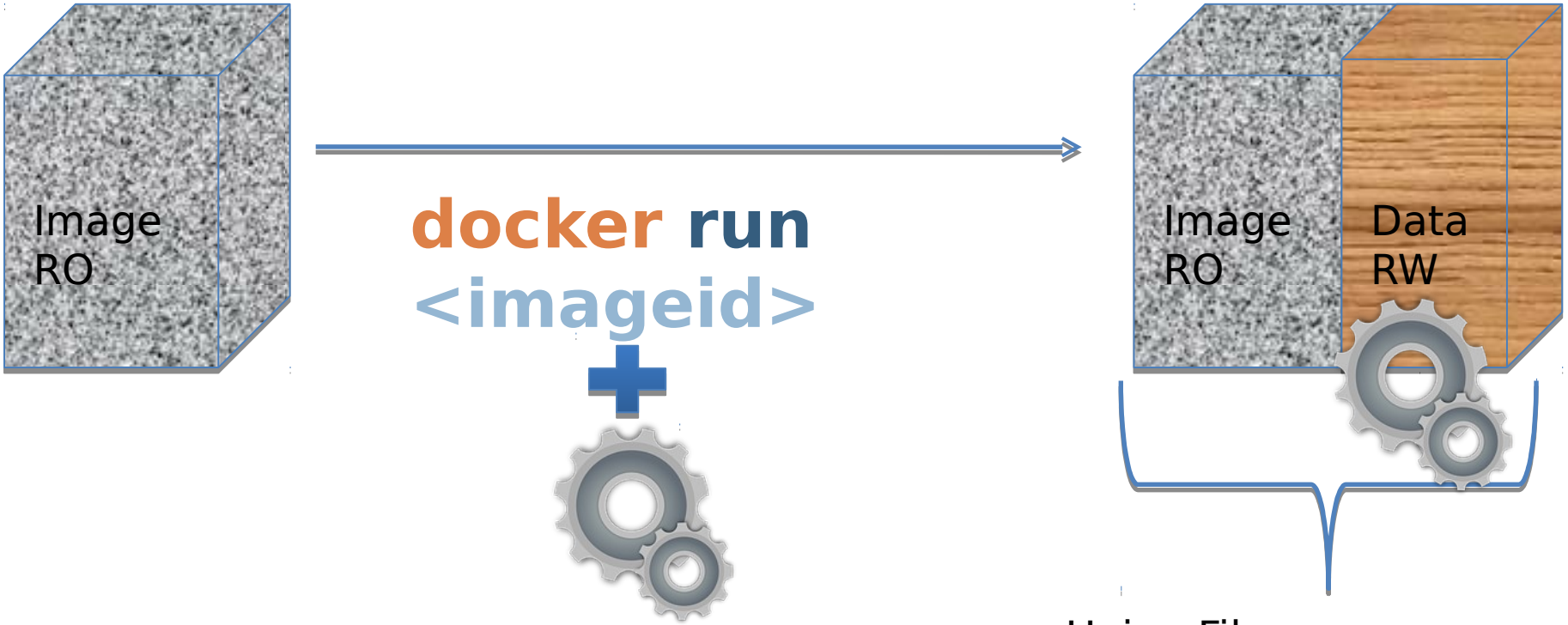
docker start
<containerid>



docker stop
<containerid>

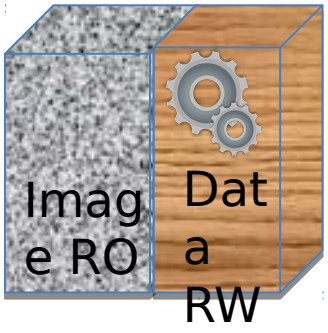
Envoi du signal SIGTERM d'arrêt par le client Docker. Le container transmet au process qui doit savoir l'interpréter sinon timeout et envoi du signal SIGKILL au container.

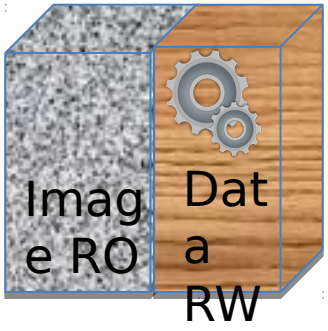
Commandes client docker

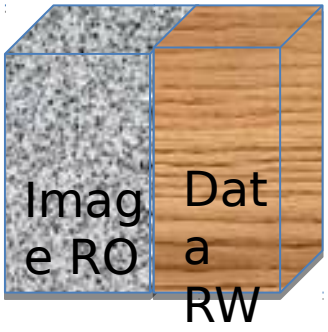


docker **run** = docker **create** + docker **start**

Commandes client docker

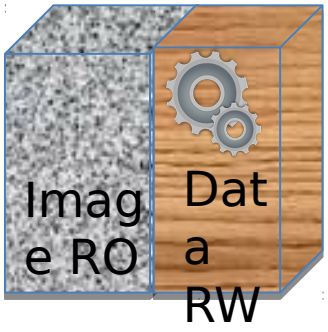


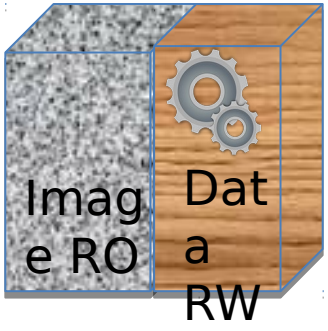




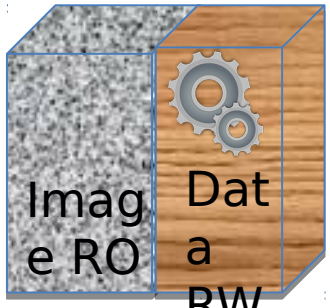
docker ps

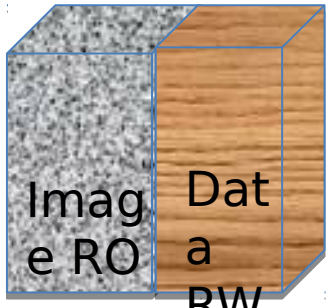
Liste les containers actifs

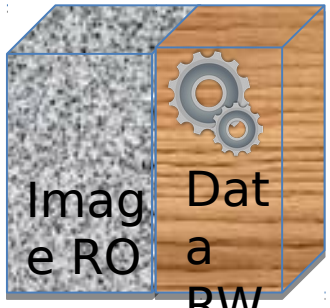




Commandes client docker

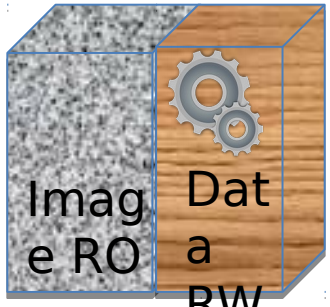


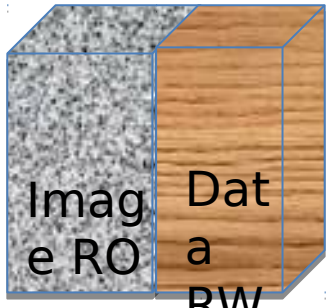


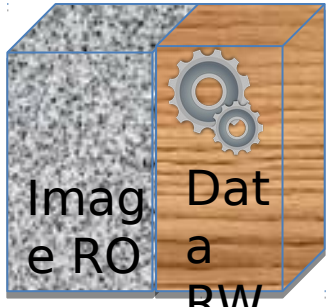


docker ps -a

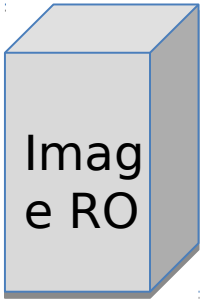
Liste tous les containers

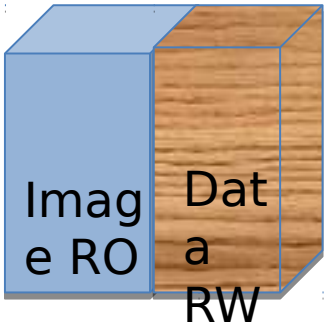


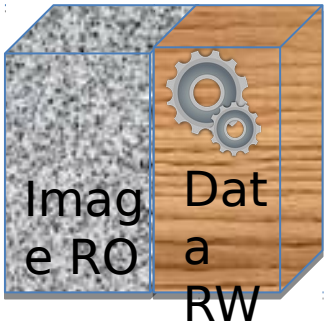




Commandes client docker

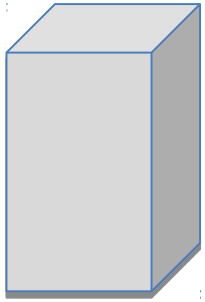


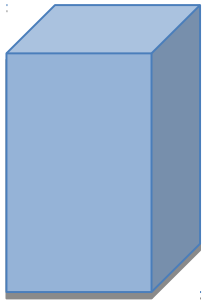


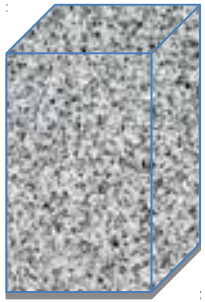


docker images

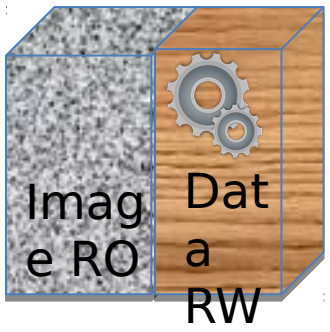
Liste tous les images







Commandes client docker



docker rm

<containerid>

Supprime le container

Commandes client docker

Docker rmi

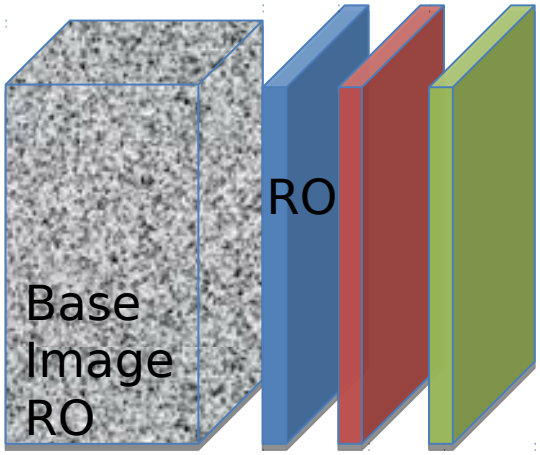
<imageid>

Supprime l’image

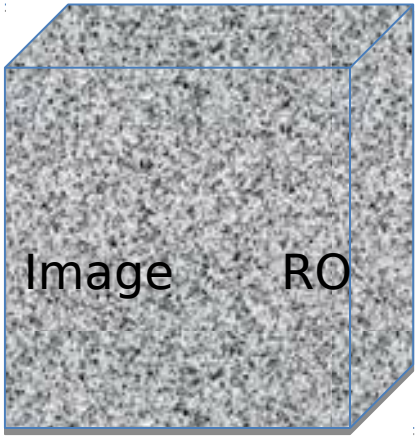
Commandes client docker

Commandes client docker

Commandes client docker



Docker build –
t nom-image .

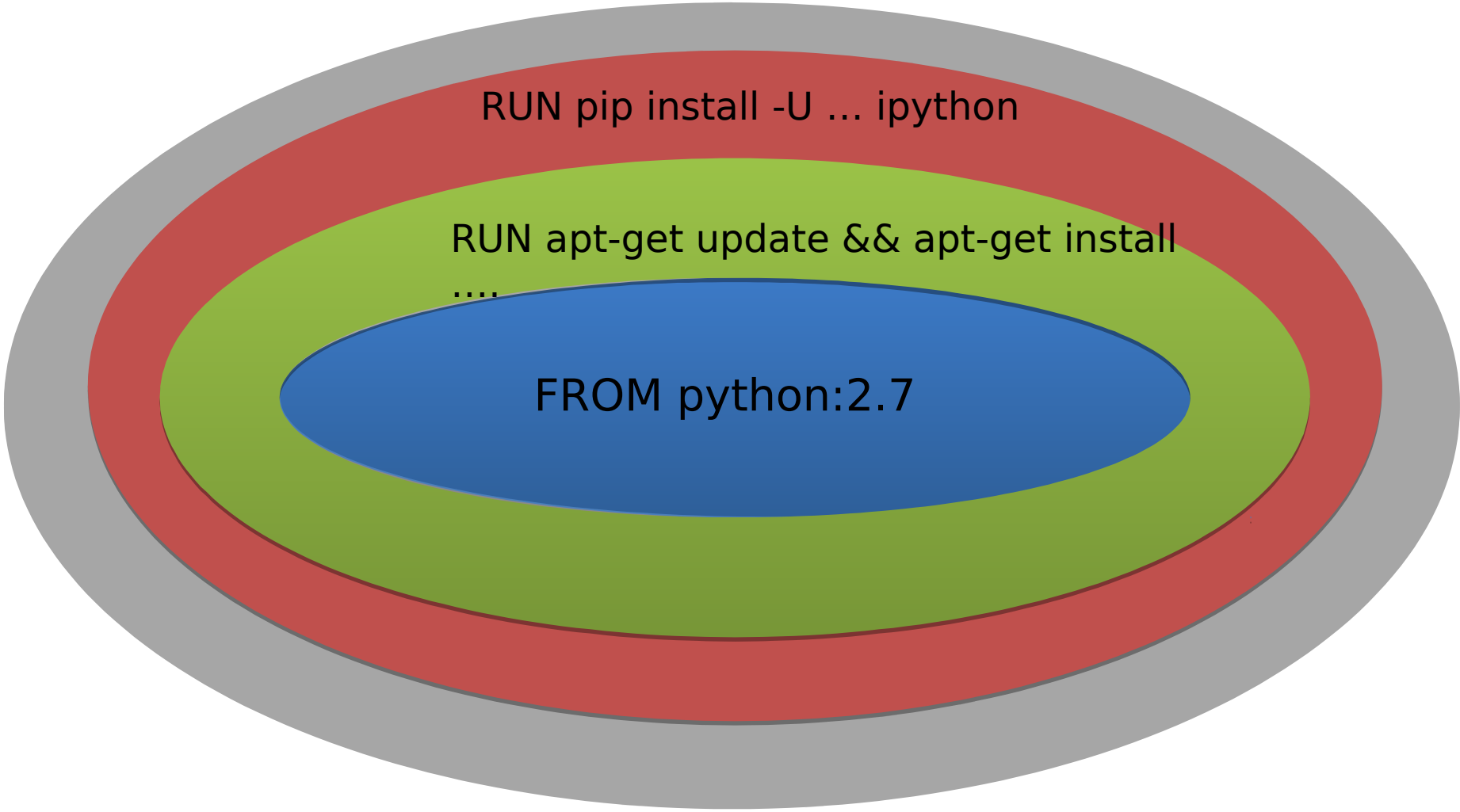


DockerFile

Création d'une image à partir du dockerfile

```
FROM python:2.7
#  d'après https://github.com/jfrazelle/dockerfiles.gi
MAINTAINER pouchou <pouchou@iutbeziers.fr>

# install ( incomplète ... )
RUN apt-get update && apt-get install -y \
    liblapack-dev \
    libzmq-dev \
    --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

RUN pip install -U \
    numpy \
    scipy \
    matplotlib \
    pandas \
    patsy \
    ipython

EXPOSE 8888

ADD notebook.sh /
RUN chmod u+x /notebook.sh \
&& mkdir -p /root/notebooks

WORKDIR /root/notebooks

CMD ["/notebook.sh"]
```

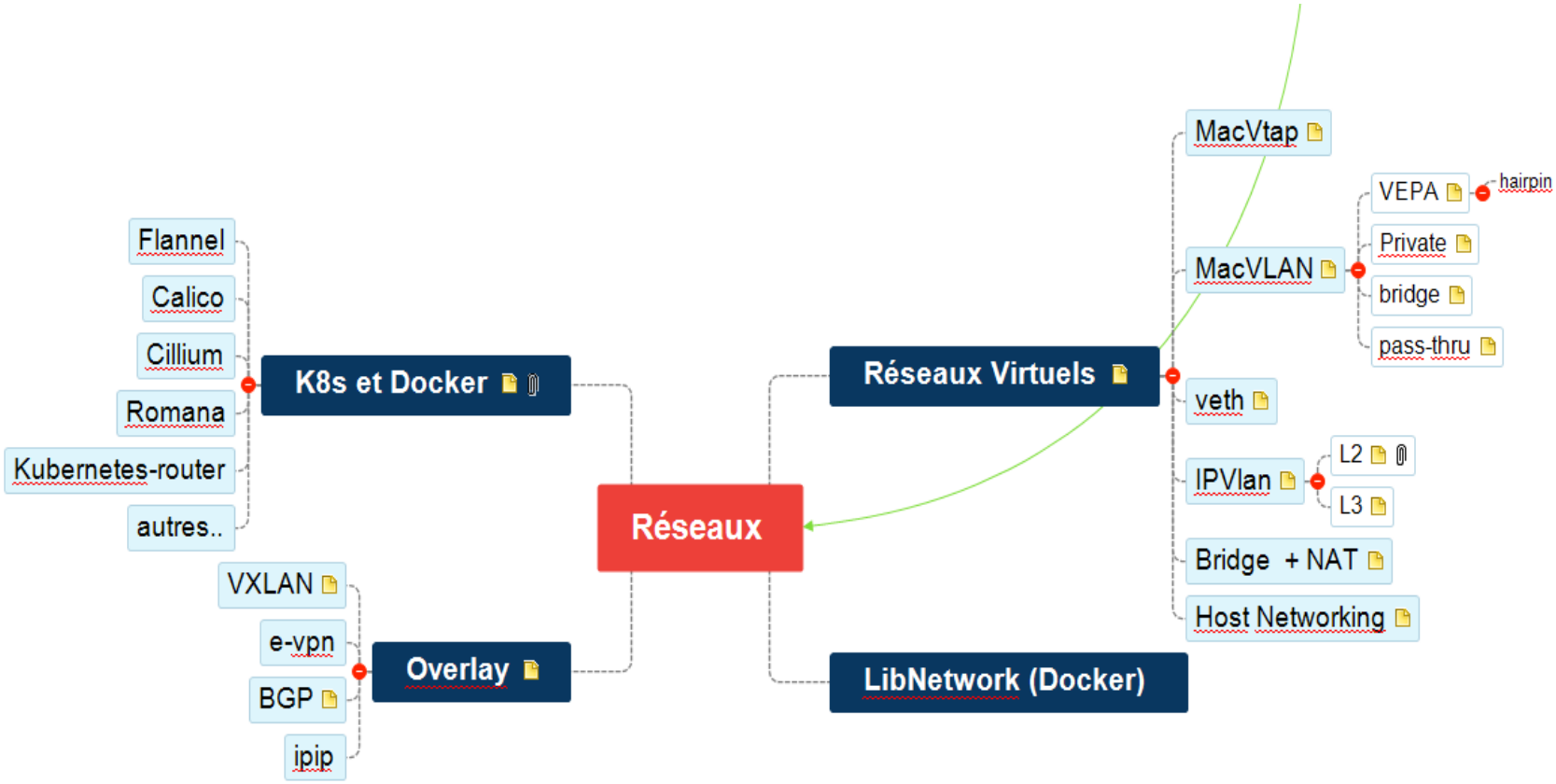
A chaque instruction du DockerFile une couche est rajoutée avec une mise en cache..



Multi-Stage Build .. and RUN

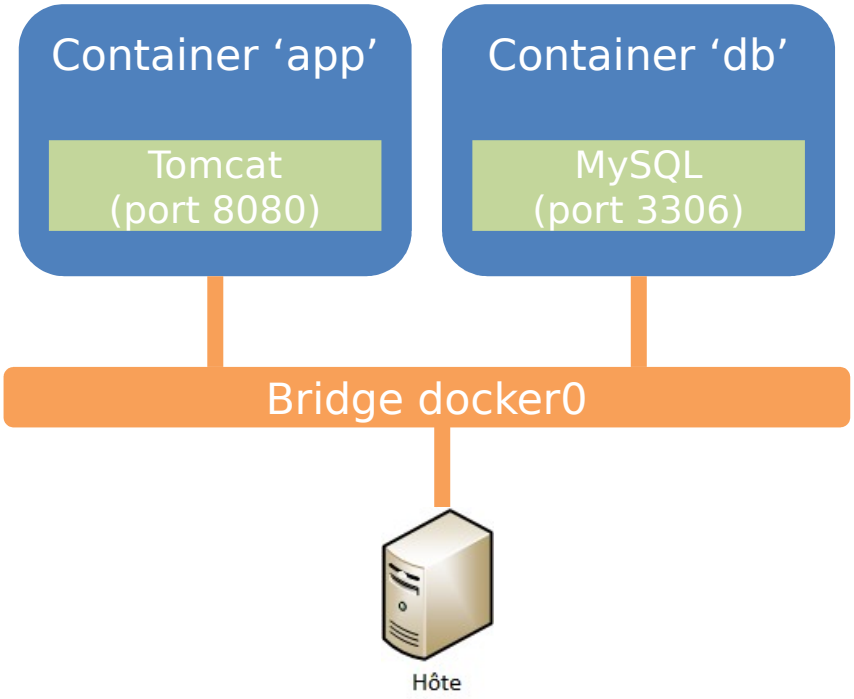
```
FROM golang:1.7.3
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .
FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/app .
CMD ["./app"]
```

Docker et le réseau



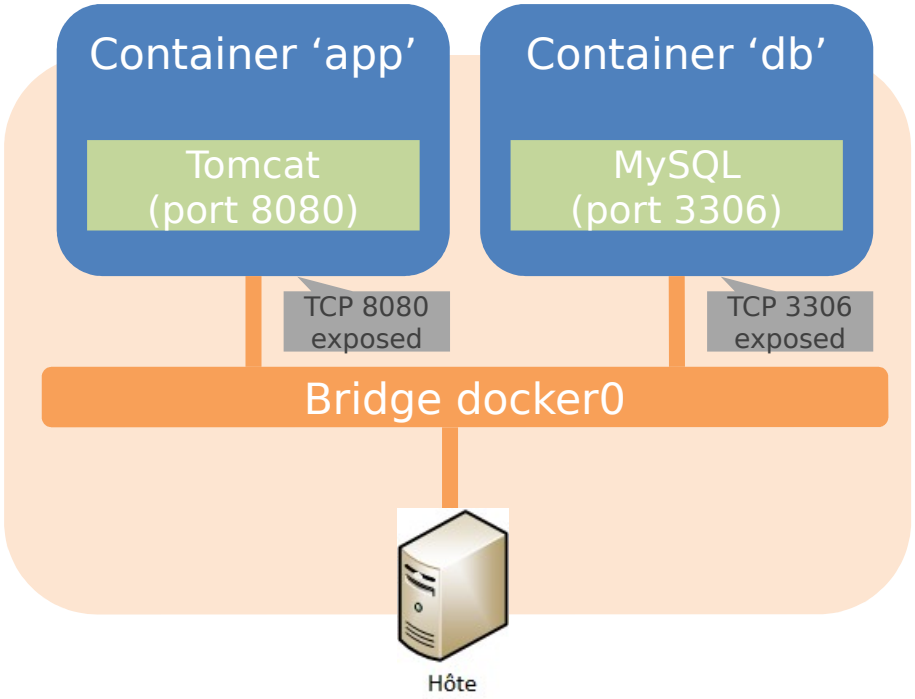
Docker et le réseau

Docker relie par défaut les containers à un bridge Linux lui même relié à l'hôte. (On peut avec l'option –net modifier ce comportement). La couche réseau est en refondation pour la version 1.9.



Expose port

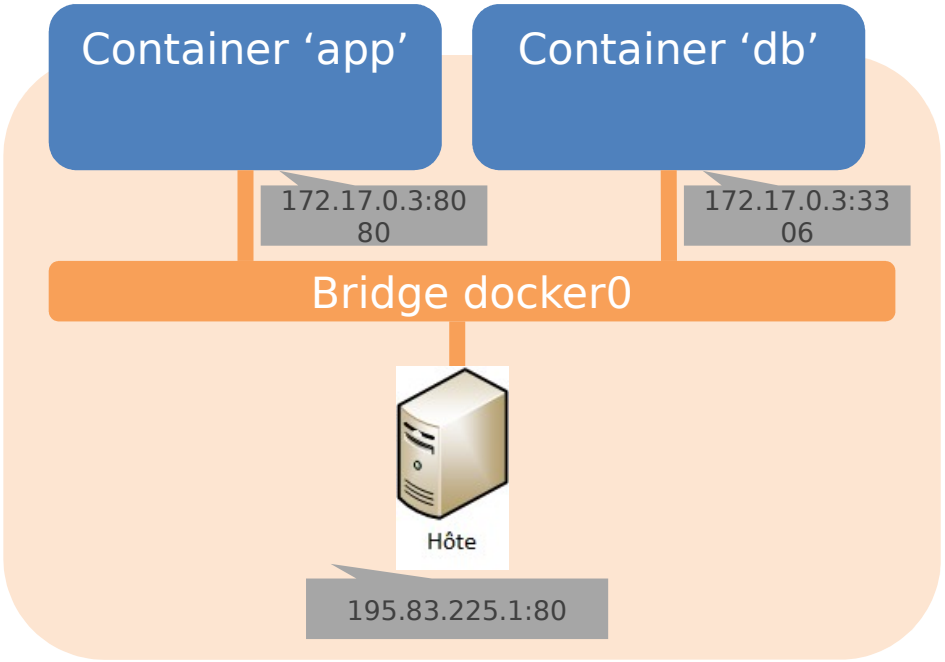
La commande expose port dans un Dockerfile permet de publier des ports accessibles aux autres containers (mais pas aux machines présentes sur le réseau) .



Accessibilité du container depuis l'extérieur.

On peut reprendre le port "exposé" et le natter (via iptables) sur un port de la machine hôte via un :

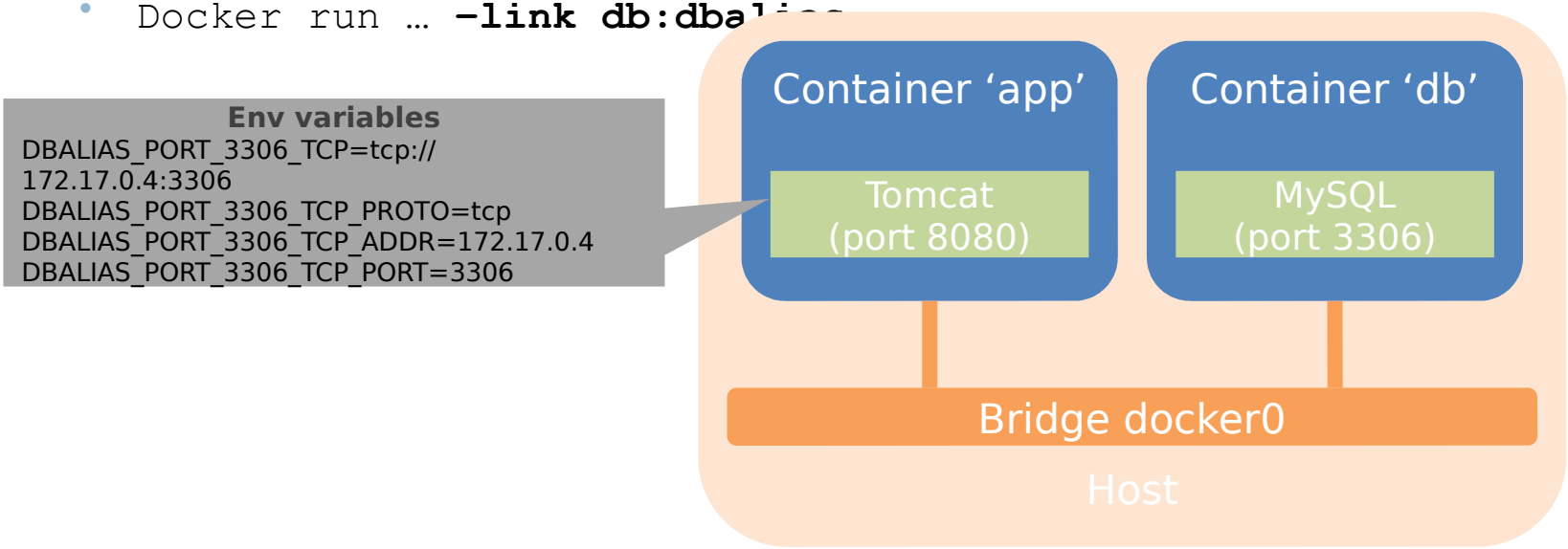
docker run -d **–p 80:8080** --restart always –name=app …



Expose 8080

Linking de containers

- **Le “Linking” permet à un container de positionner des variables accessibles au container “linké” et des entrées dans le fichier host.**
  - `docker run` **`-link <containername:alias>`**
  - `Docker run …` **`-link db:dbalias`**




```
docker run -d --link=db:database --name app ubuntu tail -f /dev/null
docker run -d --name db -h db ubuntu tail -f /dev/null
```

DNS et docker

- Par défault docker utilise le resolv.conf de l’hôte.

Externaliser les datas des containers

- Mettre des fichiers de base de données sur AUFS n’est pas recommandé.
- Comment sauvegarder des datas dans ce type de container ?

La solution c’est le volume. Un volume va échapper à AUFS et pourra être partager avec d’autres containers. Un volume peut être mappé avec une directory de l’hôte.

Volumes intercontainer

```
# --volumes pour partager des données depuis un autre container
docker create -v /dbdata --name dbdata ubuntu # create container data
docker run -d  --volumes-from dbdata --name db1 ubuntu tailf /dev/null
docker run -d  --volumes-from dbdata --name db2 ubuntu tailf /dev/null

docker exec db1 touch /dbdata/bonjour1.txt
docker exec db2 touch /dbdata/bonjour2.txt

# Sauvegarde
docker run --volumes-from dbdata -v $(pwd) :/backup ubuntu tar cvf /backup/backup.tar /dbdata
```

Persistence des containers.

- Docker fournit un container qui sert à stocker ses images : le registry.
- On peut utiliser le registry de Docker dans le cloud mais il est payant si voulez stocker des images privés.

```
docker run -d -p 5000:5000 --restart always --name=monregistry -h monregistry\
    -v $(pwd)/data:/var/lib/registry \
    registry:2

docker tag unboundsave localhost:5000/refunbound
docker push localhost:5000/refunbound
```

Persistence des containers.

On peut aussi sauvegarder les containers sous forme de tarball:

```
docker run  -d ubuntu
f2547a24e4c7b693fc275a640597a3faa14da163125e434d244a55c76262d224
docker export f2547a24e4c7b693fc2 > containersauve.tgz
pouchou@debian8:~$ ls -ltr c*
-rw-r--r-- 1 pouchou pouchou 196810240 oct.  29 23:26 containersauve.tgz
```

Ou mieux (portabilité) sauvegarder une image:

```
docker save fedora > fedora-all.tar
docker save --output=fedora-latest.tar fedora:latest
ls -sh fedora-all.tar
721M fedora-all.tar
ls -sh fedora-latest.tar
367M fedora-latest.tar
```

Commandes diverses

- docker log <containerID>
- docker exec –it <containerid> bash
- docker diff <containerID>
- docker pause <containerID>
- docker stats <containerID>
- docker tag <imageID> nomimg:Montag
- docker run -it --memory=256m ubuntu bash
- docker inspect -f '{{ .NetworkSettings.IPAddress }}’ <containerID>
- doman docker-pull

Commandes diverses

- docker log <containerID>
- docker exec –it <containerid> bash
- docker diff <containerID>
- docker pause <containerID>
- docker stats <containerID>
- docker tag <imageID> nomimg:Montag
- docker run -it --memory=256m ubuntu bash
- docker inspect -f '{{ .NetworkSettings.IPAddress }}’ <containerID>
- doman docker-pull

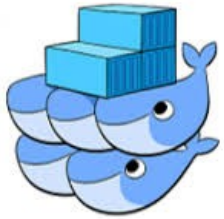
Cluster de containers

Reposer sa production sur un seul hôte est une mauvaise idée
On utilisera un cluster de container pour la scalabilité et la haute disponibilité.

kubernetes

RED HAT OPENSHIFT Container Platform

RANCHER



docker SWARM

MESOS

Supervision des containers

Sysdig