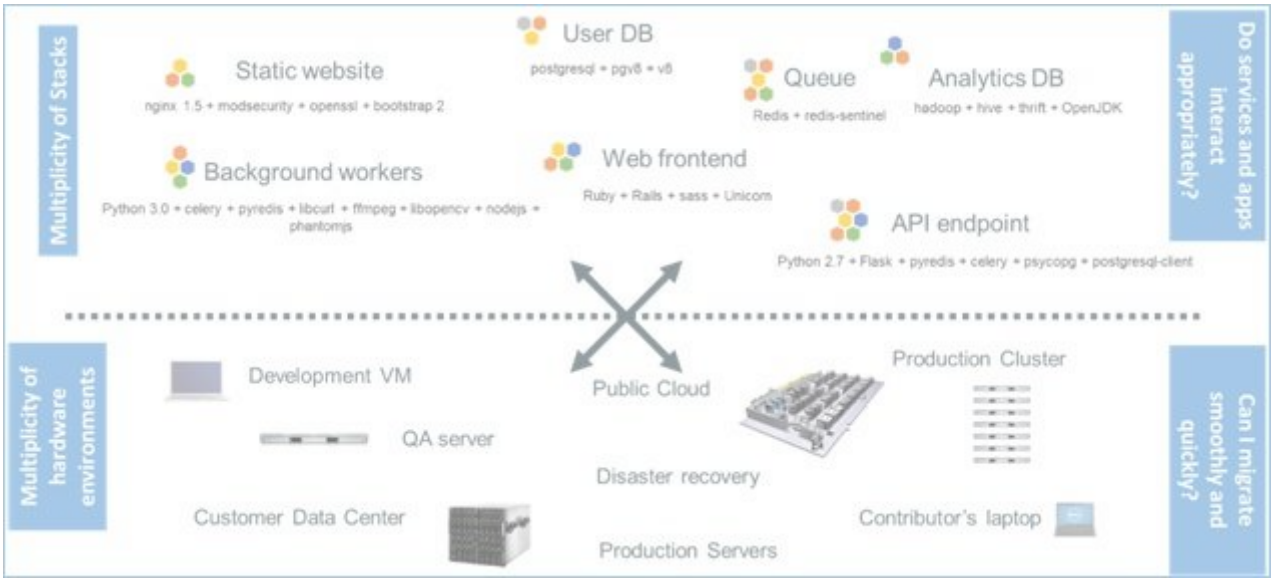
Quel est le problème ?



Besoins liés aux usages

Différentes attentes en fonction du cycle de vie

* ...en développement, * ...en test, * ...en production

Faciliter le travail

* Rapidité de déploiement * Exécuter facilement du code * Changer facilement de systèmes d’exploitation

Maitriser l'environnement

* Réduire les risques au déploiement * Garantir la reproductibilité * Sécurisation et/ou isolation des données * Isoler des utilisateurs (simultanéité, risques, etc.)

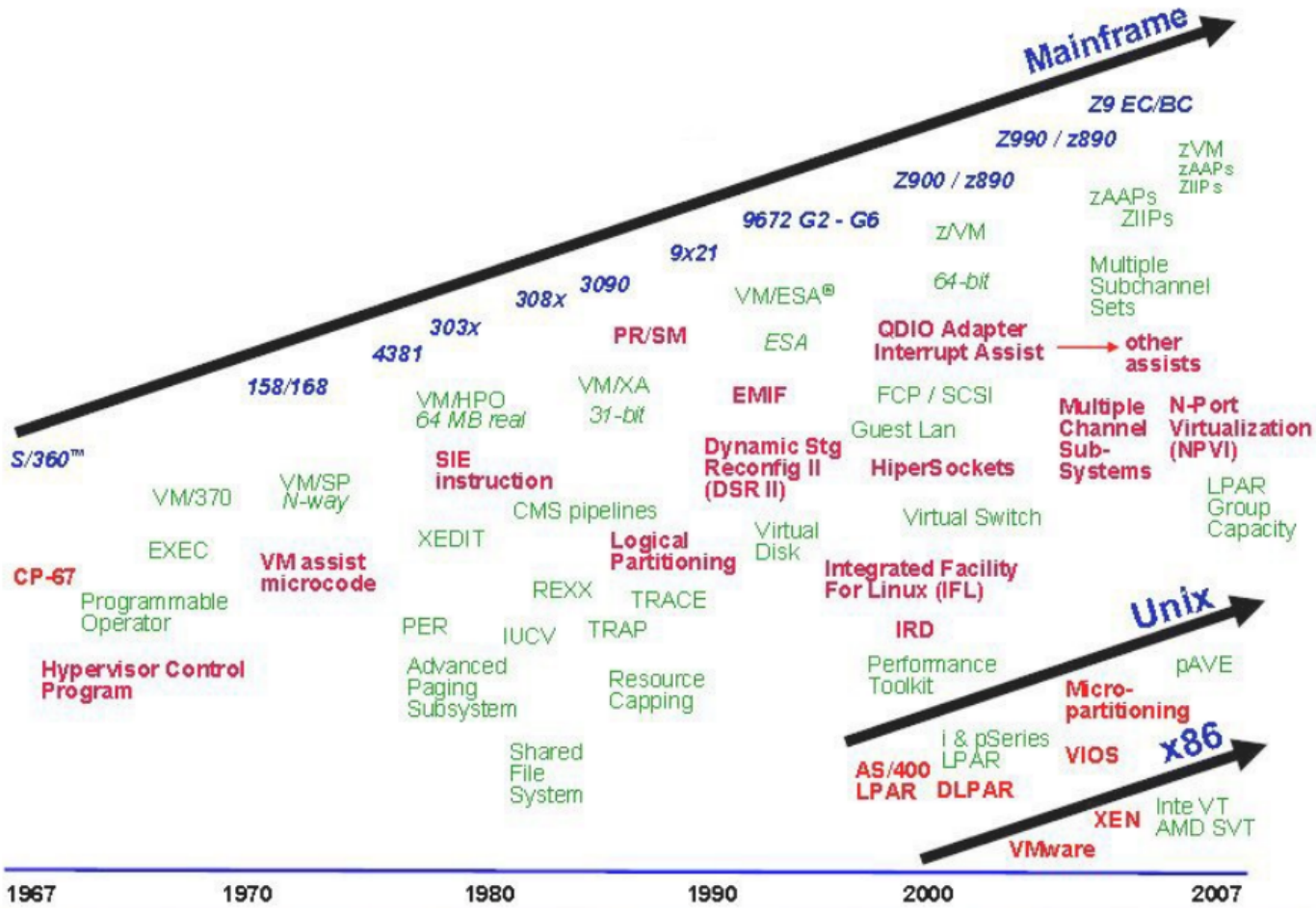
Besoins liés à l'infrastructure

- Utiliser plus intelligemment (de façon optimale?) des ressources
- Allouer des ressources (puissance de calcul, mémoire, balancing, etc.)
  - De façon dynamique
  - En fonction des besoins applicatifs à un instant donné

- Réduire son coût de possession du matériel (électricité, leasing serveur, climatisation)
- Economiser ou rentabiliser son infrastructure et ses ressources
  - par un meilleur dimensionnement
  - par mutualisation
- Faciliter l’évolution
  - du matériel
  - du logiciel : Installer, déployer, migrer une ou plusieurs applications d'une machine à l'autre
- Faciliter la gestion
  - Administration logique/physique centralisée
  - Cloisonner, sécuriser et réduire les risques liés (dimensionnement, etc.)

Virtualisation

Historique



Années 1970

- 197x : Centre sciendtifique de Cambridge d'IBM + MIT
  - Travaux sur la virtualisation,
  - Mise au point du système expérimental CP/CMS qui deviendra l'Hyperviseur VM/CMS.
- 1979 : IBM annonce les IBM 4331 et 4341
  - Mainframes contenant un accélérateur VM optionnel et microcodé capables de virtualiser leurs systèmes d’exploitation
  - Technologies spécifiques et propriétaires à la fois logicielles et matérielles
- 198x : Embryons de virtualisation sur les ordinateurs personnels

Années 1980-1990

- 198x : Création d'embryons de virtualisation sur des ordinateurs personnels.
  - solutions purement logicielles, soit couplées à du matériel additionnel (ajout de processeur, carte réseau, etc.).
  - Sur des ordinateurs Amiga équipé de processeur hétérogène comme le 80386 et 80486, 68xxx, et le PPC, il était possible de lancer d’autres OS comme un Windows, Mac OS, voire des solutions Linux. Le tout en multitâche sous AmigaOS.
  - Pour les PC, il y avait des émulateurs comme le SideCar et PC Task. Sur Macintosh, Emplant et ShapeShifter.

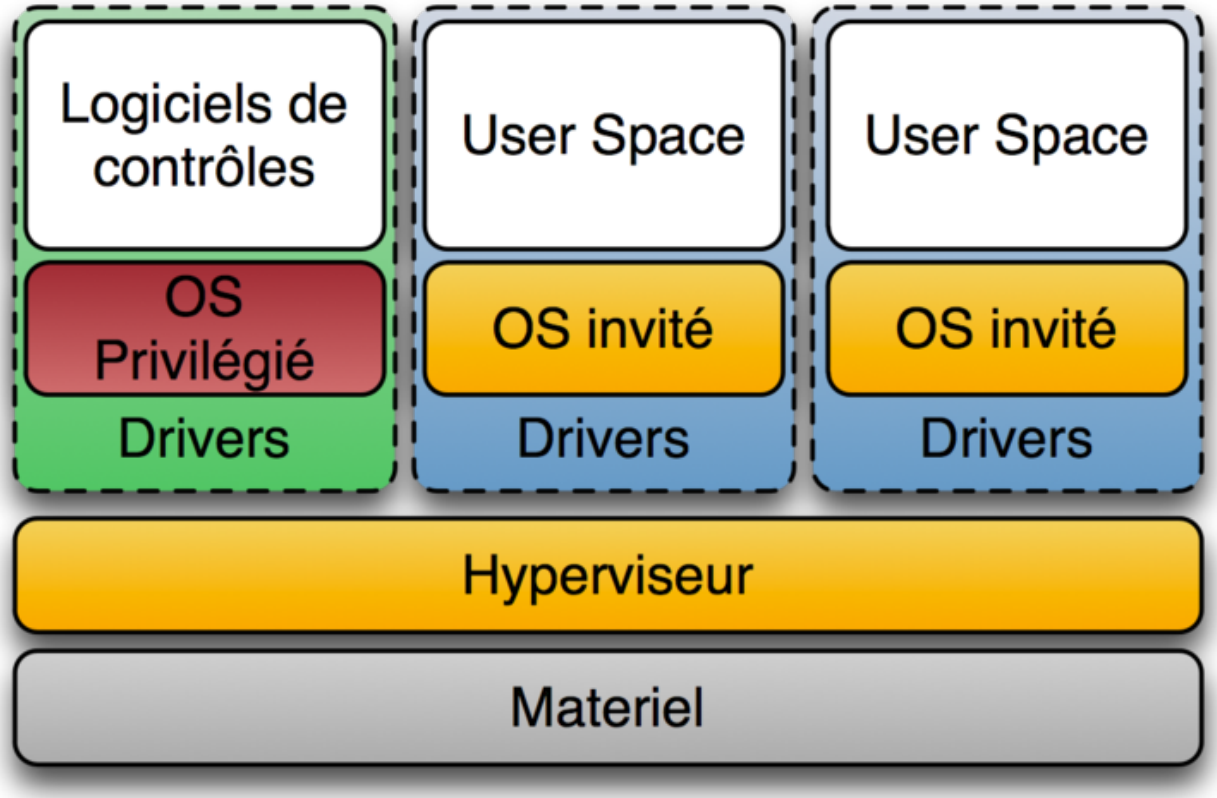
Années 1990-2000

- 1995+ : Apparition sur x86 d'émulateurs pour de vieilles machines des années 1980
  - ex: pour les ordinateurs Atari, Amiga, Amstrad et les consoles NES, SNES, Neo-Geo AES.
  - VMware popularisa un système de virtualisation logicielle des architectures de type x86 pour les architectures de type x86.
  - Apparition de nombreux autres logiciels (libres et propriétaires gratuits) pour le monde x86.

Hyperviseurs

Hyperviseur de type I (bare-metal)

- L’hyperviseur fournit un partage fin dans le temps de l’ensemble des ressources
- S'exécute directement sur le hardware
- Noyau hôte optimisé pour gérer les noyaux d'OS invités
- Les OS invités sont adaptés pour la virtualisation
- Méthode la plus performante
- Beaucoup de contraintes

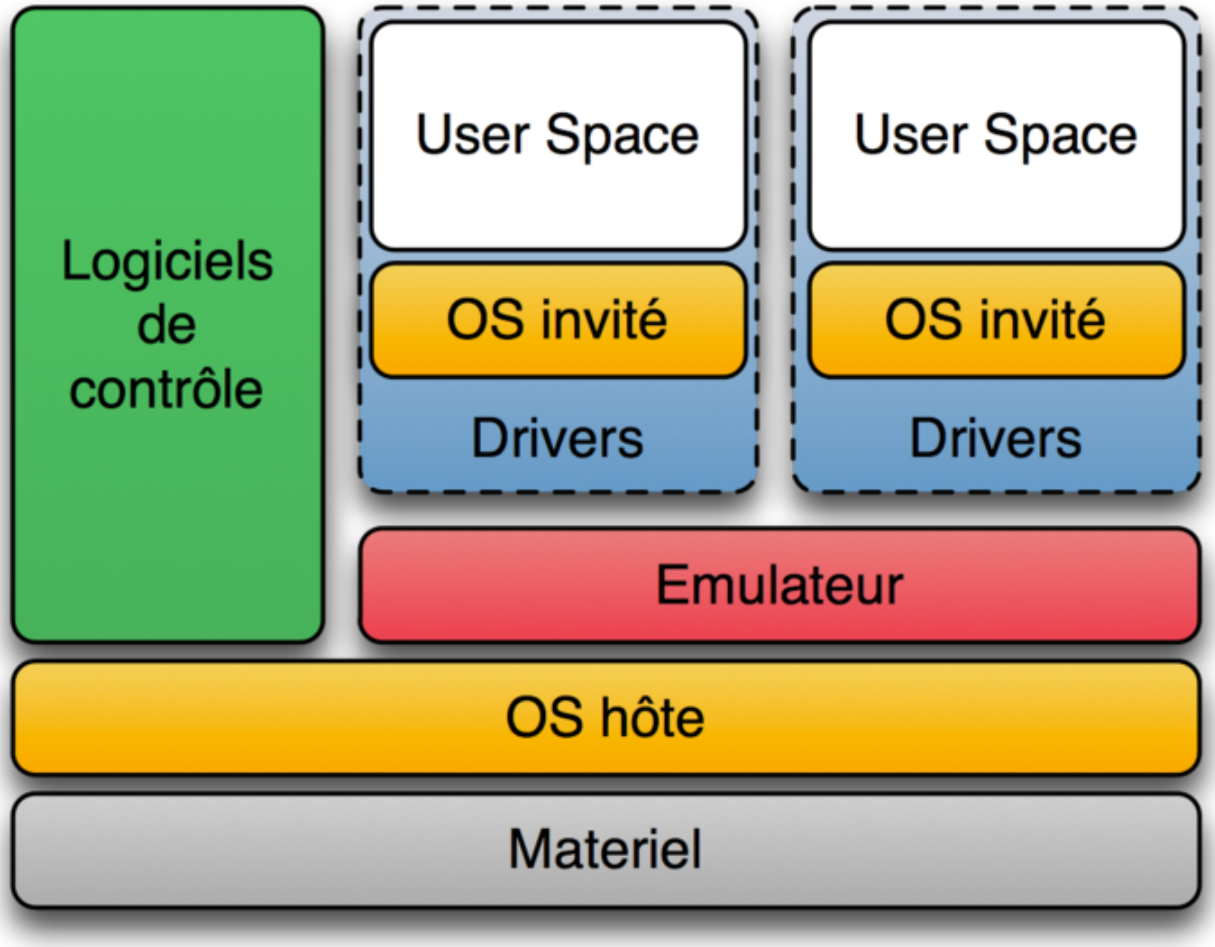


Example

- VMWare vSphere (ESXi / ESX)
- Microsoft HyperV Server, KVM
- System z PR/SM et z/VM
- POWER Hypervisor
- Xen Hypervisor

Hyperviseur de type II (hosted)

- S'execute à l'intérieur d'un autre autre système d'exploitation
- Les OS invités n'ont pas conscience d'être virtualisés



Example

- QEMU / KVM
- VMWare (Fusion, Player, Server, Workstation)
- VirtualPC
- Parallels
- VMware GSX
- Microsoft Virtual Server – Hyper-V
- HP Integrity VM
- Oracle VM VirtualBox

Para-virtualisation

Coté logiciel

- Présentation du matériel réel à l'OS invité
  par une interface logicielle spéciale
- L'OS invité doit être optimisé pour ce fonctionnement
- Moins lourd que d'émuler l'ensemble du matériel
  - L'hyperviseur est simplifié
  - Les performances de l'OS invité sont proches de celles du matériel réel
- Fonctionne sur les hyperviseurs de type I et II

Coté matériel

- Support de la virtualisation coté matériel
  - Intégré au processeur
  - Ou assisté par celui-ci
  - ex: le matériel se charge de virtualiser les accès mémoire ou de protéger le processeur physique des accès les plus bas niveaux
- Simplifier la virtualisation logicielle et améliore les performances.

Example

- Hyperviseur IBM Power10 & Micro-partitionnement AIX
- Mainframes VM/CMS
- Sun LDOM (hyperviseur pour la gestion de « logical domains »)
- Sun E10k/E15k
- HP Superdome
- AMD-V (anciennement Pacifica)
- Intel VT (anciennement Vanderpool)

Quelques démos

QEMU/KVM

- Inclus dans le noyau linux standard
- KVM = QEMU + para-virtualisation

```
$ wget https://cdimage.debian.org/debian-cd/current/amd64/iso-cd/debian-11.0.0-amd64-netinst.iso
$ sudo qemu-system-x86_64 -cdrom debian-11.0.0-amd64-netinst.iso
```

ou bien

```
$ vagrant init debian/bullseye64
$ vagrant up --provider=libvirt
```

Virtualbox + Vagrant

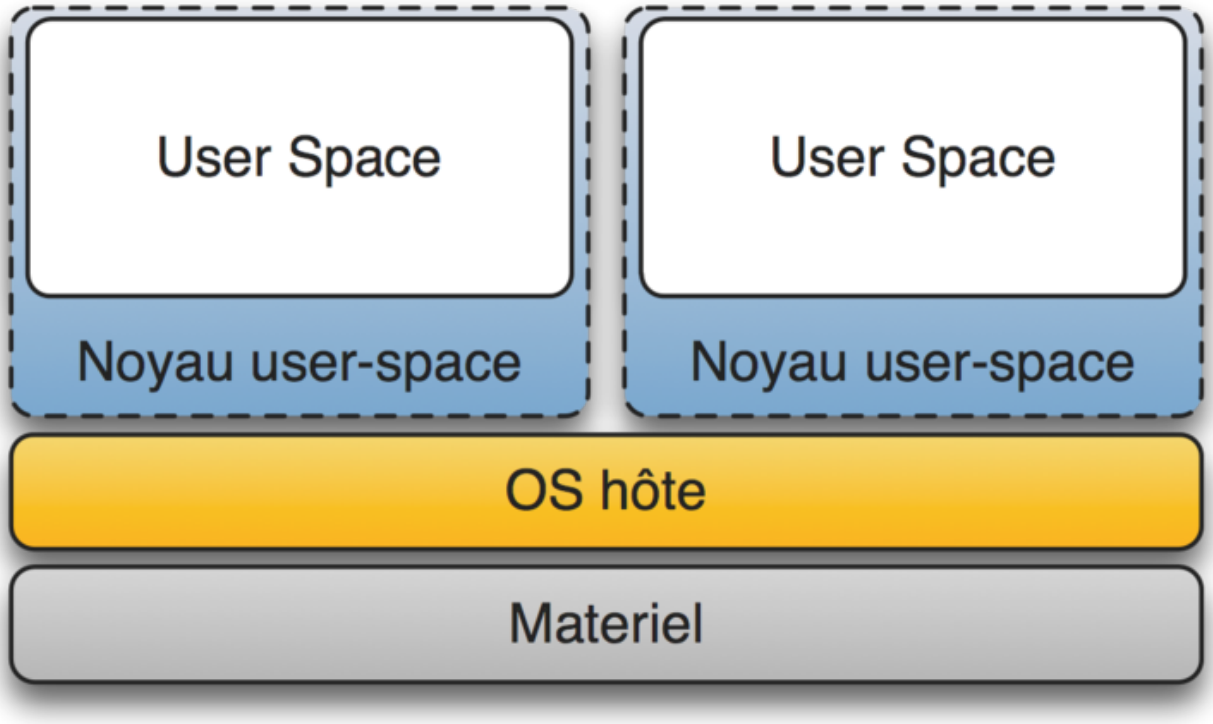
```
$ vagrant init debian/bullseye64
$ vagrant up --provider=virtualbox
```

Références

- Run Debian 11 (Bullseye) on KVM using Qcow2 Cloud Image

Noyau en espace utilisateur

- Noyau invité qui comme une simple application de l'OS hôte
- Le noyau invité possède son propre espace utilisateur
- Très peu performant
- Concû pour des tests fonctionnels (drivers, etc.)



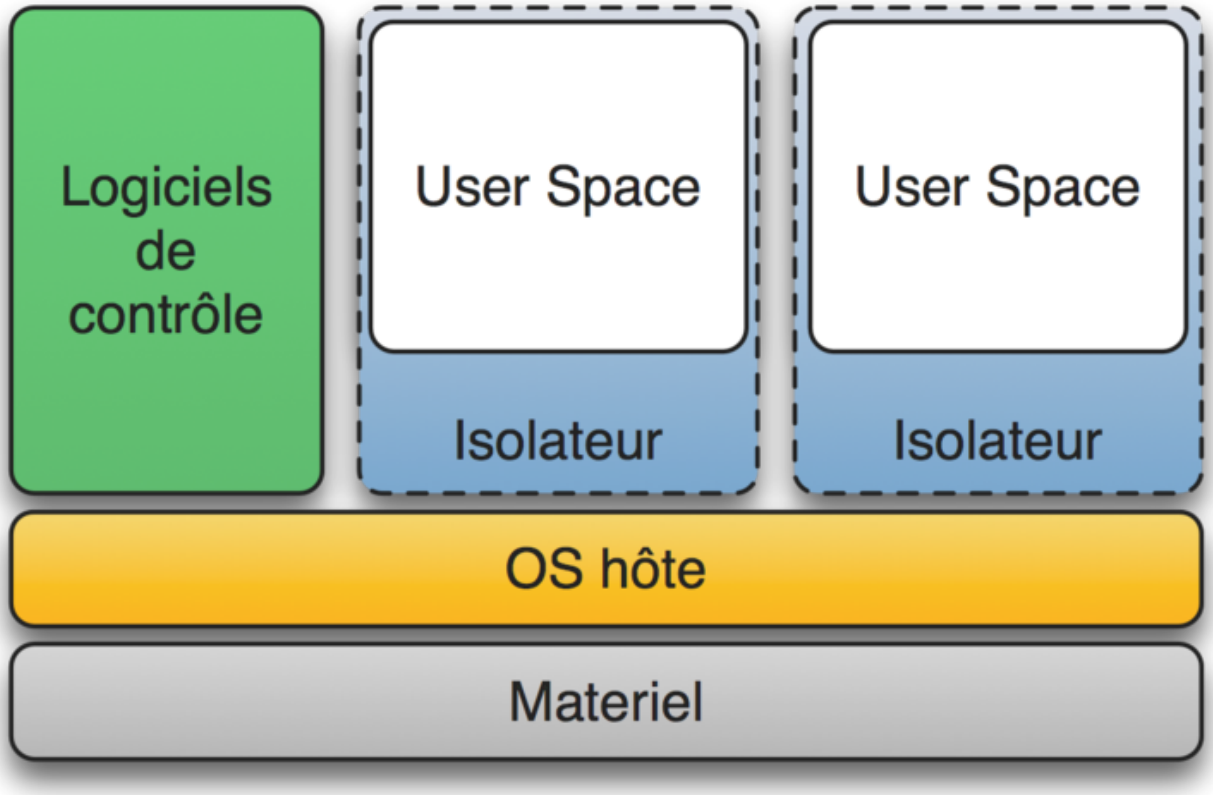
Example

- User Mode Linux
- coLinux
- Adeos

Conteneurs d'applications

Fonctionnement

- Isolation des applications dans des "contextes"
- Très bonnes performances (car peu d'overhead)
- Un seul noyau partagé entre l'hôte et les différents contextes



Historique des isolateurs

-- FS

- 1979 : Unix V7 Chroot
- 1982 : BSD Chroot
- 1996 : GNU + Linux Chroot, in coreutils

-- FS + Proc

- 2000 : BSD Jails

-- FS + Proc + NumProc + Réseau

- 2001 : Linux-Vserver
- 2004 : Solaris Container et Zones, by Sun
- 2005 : OpenVZ
- 2007 : Process Containers, by Google
    - 2008 : merged in Linux as Linux CGroups + namespaces
- 2008 : LXC, based on Cgroups
- 2011 : Warden, by CloudFoundry
- 2013 : Docker, by dotCloud
- 2014 : LMCTFY, by Google
- 2014 : Rocket, by CoreOS
- 2018-2019 : Container Runtime Interface, by CNCF (Cloud Native Computing Foundation)
- 2019 : CRI-O, by ???
- 2019 : Podman, by RedHat

Chroot

- Isolateur:
    - filesystem
- Patch kernel sur les fonctions d'accès aux fichiers
- Outils userland

ton programme dans le chroot	**le kernel**
open('/')	open('/mnt/chroot/')
open('/titi')	open('/mnt/chroot/titi')
pwd()	lechemin - '/mnt/chroot'

Jail

- Isolateur
  - filesystem
  - processus
- Patch kernel sur
  - les fonctions d'accès aux fichiers
  - l'espace des processus
- Outils userland

VServer

- Isolateur
  - filesystem
  - routage segmenté
  - quotas
- Possibilité d'unifier les programmes ou fichiers
- Patch kernel + outils userland
- Pas intégré aux sources officielles
- Pas d'accès direct aux matériel
- Migration d'hôte à hôte difficile

OpenVZ

- Isolateur
  - filesystem
  - utilisateurs
  - processus
  - réseau *
  - périphériques
  - ipc (inter process communication)
- Patch kernel + outils userland

```
# vzctl create 100 --ostemplate debian-7.0-x86
# vzctl create 102 --ostemplate suse-12.3-x86
# vzctl enter 100
```

LXC

- Inclus dans le noyau linux standard
- Vérifier avec `lxc-checkconfig`
- Un `rootfs` + un fichier de configuration

```
# lxc-create -n test1 -t debian
# lxc-start -n test1
```

Important

Désormais on parle de conteneurs

- virtualisation légere
- isolation du reste du systeme

Références

- Virtualisation sur Wikipedia
- Hyperviseur sur Wikipedia
- Paravirtualisation sur Wikipedia
- Linux-VServer sur Wikipedia
- OpenVZ sur Wikipedia
- Référence OpenVZ
- Référence LXC

Les namespaces

- Services fournis par le noyau Linux pour gérer l'isolation
- Permettent le partitionnement des ressources et des processus du noyau.
- Blocs "primitifs" de construction des conteneurs sous Linux
- Les processus dans un namespace n'accèderont qu'aux ressources intégrées à ce namespace, créant ainsi une couche d'isolation.

Lorsqu'un conteneur Docker est lancé, Docker crée un ensemble d'espaces de noms pour ce conteneur particulier et son accès est limité à cet namespaces. Des exemples d'espaces de noms utilisés dans Docker peuvent être trouvés dans [2] et sont généralement utilisés pour gérer les processus, la mise en réseau, le montage de volumes, la gestion de l'accès aux ressources de communication interprocessus et l'isolation des identifiants de noyau/version (partage du temps Unix). L'environnement d'exécution est ainsi isolé car il ne peut pas voir les autres espaces de noms, conteneurs et applications sur le système.

Le Mount namespace (Linux 2.4.19)

Gère l'isolation des points de montage du système de fichier vus par un groupe de process :

- les points de montage ne sont plus globaux mais spécifiques au namespace
- les points de montage peuvent être propagés
- racine propre (chroot)

Se manipule via `/proc/$PID/mounts`

Le UTS namespace (Linux 2.6.19)

Gère l'isolation des identifiants de nom et de domaine.

Le processus a une copie séparée du nom d'hôte et du nom de domaine NIS (obsolete), de sorte qu'il peut le définir à une autre valeur sans affecter le reste du système.

Le nom d'hôte est défini via `sethostname` et est le membre `nodename` de la structure retournée par `uname`. Le nom de domaine NIS est défini par `setdomainname` et est le nom de domaine membre de la structure retournée par `uname`.

Se manipule via `via uname -n`, `hostname -f`

Note

UTS vient de la structure "utsname" passée à l'appel système uname().

UTS voulant dire "UNIX Time-sharing System".

Le IPC namespace (Linux 2.6.19-2.6.30)

Gère l'isolation des ressources IPC :

- semaphores
- message queues
- shared memory segments

Note

Depuis Linux 2.6.30, il gère aussi les files de messages POSIX

Se manipule via `/proc/sys/fs/mqueue`, `/proc/sys/kernel`, `/proc/sysvipc`

Le PID namespace (Linux 2.6.24)

Gère l'isolation des ID de process :

- PID 1 init-like par namespace
- chaque namespace a sa propre numération PID (isolation de l'hôte)
- le process d'un namespace ne peut envoyer de systemcall sur un autre process d'un autre PID namespace
- gestion de pseudo-filesystem (ex : /proc) vu par le PID namespace qui le monte
- un process possède plusieurs PID : un dans le namespace, un en dehors (process vu par l'hôte), davantage en cas de namespaces imbriqués

Se manipule via `/proc/$PID/status`

Le Net namespace (Linux 2.6.19-2.6.24 et +)

Gère l'isolation du réseau. Chacun possèdant :

- ses interfaces

- ses ports
- sa table de routage
- ses règles de firewall (iptables)
- son répertoire /proc/net
- INADDR_ANY (0.0.0.0)

Se manipule via `ip netns list`, `/proc/net`, `/sys/class/net`

> **Note**
>
> Il est possible de créer des «pair interfaces» (visibles de l'intérieur et de l'extérieur)

Le User namespace (Linux 2.6.23-3.8)

Gère l'isolation des utilisateurs et des groupes en :

- séparant les droits selon les différents namespaces
- rendant sûr le partage de namespace à des process sans privilège

Se manipule via `id`, `/proc/$PID/uid_map`, `/proc/$PID/gid_map`

Le CGroup namespace (Linux 4.6)

Se manipule via `/sys/fs/cgroup/`, `/proc/cgroups`, `/proc/$PID/cgroup`

Références

- `namespaces(7)` manpage
- http://containerz.info/

Control Groups (cgroups)

Fonctionnalité du noyau Linux pour gérer les ressources.

Il permet de :

- suivre finement des processus (ex: systemd l'utilise)
- comptabiliser l'utilisation des ressources (CPU, mémoire, E/S de disque, réseau, etc.)
- appliquer des limites et des contraintes sur les processus
- apporter les garanties qui permettent l'isolation

Ainsi chaque service conteneurisé n'utilisera pas plus de ressources qu'il ne le devrait :

- empeche les containers consommer toutes les ressources système
- ... et de rendre le systeme hôte indisponible.

➡ Prévention des denis de service (attaques ou processus gourmands)

> **Note**
>
> Comme un `ulimit` mais pour un groupe de processus, et avec plus de critères

Pseudo-fichiers pour manipuler le groupe

```
# root@server1:/sys/fs/cgroup# ls -1
blkio/
cpu@
cpuacct@
cpu,cpuacct/
cpuset/
devices/
freezer/
memory/
net_cls@
net_cls,net_prio/
net_prio@
perf_event/
pids/
rdma/
systemd/
unified/
```

Références

- lizrice/containers-from-scratch - Writing a container in a few lines of Go code, as seen at DockerCon 2017 and on O'Reilly Safari
- mhausenblas/cinf - Command line tool to view namespaces and cgroups, useful for low-level container prodding

Systèmes de fichiers

Système de fichiers COW (Copy-On-Write)

L'idée fondamentale :

- si de multiples appelants demandent des ressources initialement impossibles à distinguer, vous pouvez leur donner des pointeurs vers la même ressource.
- si un appelant modifie sa « copie » de la ressource. À ce moment-là, une copie privée est créée. Cela évite que le changement soit visible ailleurs.

Example

- ReiserFS
- Ext4
- Btrfs
- LVM

Superpositions de couches

L'idée fondamentale : le FS fusionne de l'accès de répertoire lorsque deux systèmes de fichiers sont présents dans un répertoire du même nom.

Le systeme de fichier présente les objets fournis par l'un ou l'autre, le "plus haut" système de fichiers prenant le pas sur l'autre.

Example

- Basé historiquement sur AUFS
- Aujourd'hui plutôt OverlayFS
- Il en existe d'autres (mergerfs, mhddfs, ...)

Le cas AUFS / OverlayFS

Supporte le mode union (ro)

Avec AUFS

```
# mount -t aufs -o dirs=/lower1:/lower2:/lower3 none /merged
```

Avec OverlayFS

```
# mount -t overlay overlay \
      -olowerdir=/lower1:/lower2:/lower3 \
      /merged
```

Supporte le Copy On Write (COW)

Avec AUFS

```
# mount -t aufs -o  br=/upper=rw:/lower=ro /srv
```

Avec OverlayFS

```
# mount -t overlay overlay \
      -olowerdir=/lower,upperdir=/upper,workdir=/work \
      /merged
```

Le cas LXC

Installation de LXC

```
# apt-get install lxc lxc-templates
```

Templates

Scripts d'installation pour différents types d'OS.

Leur exécution est plus ou moins longue.

```
$ dpkg -L lxc-templates |grep '/templates/'
/usr/share/lxc/templates/lxc-alpine
/usr/share/lxc/templates/lxc-altlinux
/usr/share/lxc/templates/lxc-archlinux
/usr/share/lxc/templates/lxc-centos
/usr/share/lxc/templates/lxc-cirros
/usr/share/lxc/templates/lxc-debian
/usr/share/lxc/templates/lxc-fedora
/usr/share/lxc/templates/lxc-fedora-legacy
/usr/share/lxc/templates/lxc-gentoo
/usr/share/lxc/templates/lxc-openmandriva
/usr/share/lxc/templates/lxc-opensuse
/usr/share/lxc/templates/lxc-oracle
/usr/share/lxc/templates/lxc-plamo
/usr/share/lxc/templates/lxc-pld
/usr/share/lxc/templates/lxc-sabayon
/usr/share/lxc/templates/lxc-slackware
/usr/share/lxc/templates/lxc-sparclinux
/usr/share/lxc/templates/lxc-sshd
/usr/share/lxc/templates/lxc-ubuntu
/usr/share/lxc/templates/lxc-ubuntu-cloud
/usr/share/lxc/templates/lxc-voidlinux
```

Verification du noyau

```
# lxc-checkconfig

Kernel configuration not found at /proc/config.gz; searching...
Kernel configuration found at /boot/config-5.0.0-trunk-amd64
--- Namespaces ---
Namespaces: enabled
Utsname namespace: enabled
Ipc namespace: enabled
Pid namespace: enabled
```

```
User namespace: enabled
Network namespace: enabled

--- Control groups ---
Cgroups: enabled

Cgroup v1 mount points:
/sys/fs/cgroup/systemd
/sys/fs/cgroup/memory
/sys/fs/cgroup/freezer
/sys/fs/cgroup/net_cls,net_prio
/sys/fs/cgroup/cpuset
/sys/fs/cgroup/rdma
/sys/fs/cgroup/perf_event
/sys/fs/cgroup/cpu,cpuacct
/sys/fs/cgroup/blkio
/sys/fs/cgroup/devices
/sys/fs/cgroup/pids

Cgroup v2 mount points:
/sys/fs/cgroup/unified

Cgroup v1 clone_children flag: enabled
Cgroup device: enabled
Cgroup sched: enabled
Cgroup cpu account: enabled
Cgroup memory controller: enabled
Cgroup cpuset: enabled

--- Misc ---
Veth pair device: enabled, not loaded
Macvlan: enabled, not loaded
Vlan: enabled, not loaded
Bridges: enabled, loaded
Advanced netfilter: enabled, loaded
CONFIG_NF_NAT_IPV4: enabled, loaded
CONFIG_NF_NAT_IPV6: enabled, not loaded
CONFIG_IP_NF_TARGET_MASQUERADE: enabled, not loaded
CONFIG_IP6_NF_TARGET_MASQUERADE: enabled, not loaded
CONFIG_NETFILTER_XT_TARGET_CHECKSUM: enabled, not loaded
CONFIG_NETFILTER_XT_MATCH_COMMENT: enabled, not loaded
FUSE (for use with lxcfs): enabled, loaded

--- Checkpoint/Restore ---
checkpoint restore: enabled
CONFIG_FHANDLE: enabled
CONFIG_EVENTFD: enabled
CONFIG_EPOLL: enabled
CONFIG_UNIX_DIAG: enabled
CONFIG_INET_DIAG: enabled
CONFIG_PACKET_DIAG: enabled
CONFIG_NETLINK_DIAG: enabled
File capabilities:

Note : Before booting a new kernel, you can check its configuration
usage : CONFIG=/path/to/config /usr/bin/lxc-checkconfig
```

Création de LXC

```
# lxc-create -n test -t debian
debootstrap est /usr/sbin/debootstrap
Checking cache download in /var/cache/lxc/debian/rootfs-stable-amd64 ...
Downloading debian minimal ...
I: Target architecture can be executed
I: Retrieving InRelease
I: Checking Release signature
I: Valid Release signature (key id 6D33866EDD8FFA41C0143AEDDCC9EFBF77E11517)
I: Retrieving Packages
I: Validating Packages
I: Resolving dependencies of required packages...
I: Resolving dependencies of base packages...
I: Checking component main on http://deb.debian.org/debian...
I: Retrieving libacl1 2.2.53-4
...
'debian' template installed
'test' created
```

Lancement d'une LXC

```
# lxc-start -n test
```

Pour voir le démarrage :

```
# lxc-start -n test -F
systemd 241 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSVINIT
+UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL +XZ +LZ4 +SECCOMP +B
LKID +ELFUTILS +KMOD -IDN2 +IDN -PCRE2 default-hierarchy=hybrid)
Detected virtualization lxc.
Detected architecture x86-64.

Welcome to Debian GNU/Linux 10 (buster)!

Set hostname to <test>.
[  OK  ] Reached target Slices.
[  OK  ] Listening on Journal Socket (/dev/log).
[  OK  ] Listening on Journal Audit Socket.
[  OK  ] Started Forward Password Requests to Wall Directory Watch.
[  OK  ] Listening on Journal Socket.
         Starting Remount Root and Kernel File Systems...
         Mounting Huge Pages File System...
         Starting Journal Service...
[  OK  ] Started Dispatch Password Requests to Console Directory Watch.
[  OK  ] Reached target Paths.
[  OK  ] Created slice system-getty.slice.
[  OK  ] Reached target Remote File Systems.
[  OK  ] Reached target Local Encrypted Volumes.
         Starting Apply Kernel Variables...
[  OK  ] Reached target Swap.
```

```
         Mounting POSIX Message Queue File System...
...
```

Entrer dans une LXC

```
# lxc-attach -n test
root@test:/#
root@test:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  root  run  sbin  selinux  srv  sys  tmp
usr  var
root@test:/# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.1  21492  9600 ?        Ss   17:09   0:00 /sbin/init
root        40  0.0  0.1  27440  8420 ?        Ss   17:09   0:00 /lib/systemd/systemd-journald
root        73  0.0  0.0   5380  1904 console  Ss+  17:09   0:00 /sbin/agetty -o -p -- \u --noclear --keep-baud
console 115200,38400,9600 vt220
root        74  0.0  0.0  15832  6636 ?        Ss   17:09   0:00 /usr/sbin/sshd -D
root        76  0.0  0.0   6856  3568 ?        Ss   17:10   0:00 /bin/bash
root        78  0.0  0.0  10628  3168 ?        R+   17:10   0:00 ps aux
root@test:/#
```

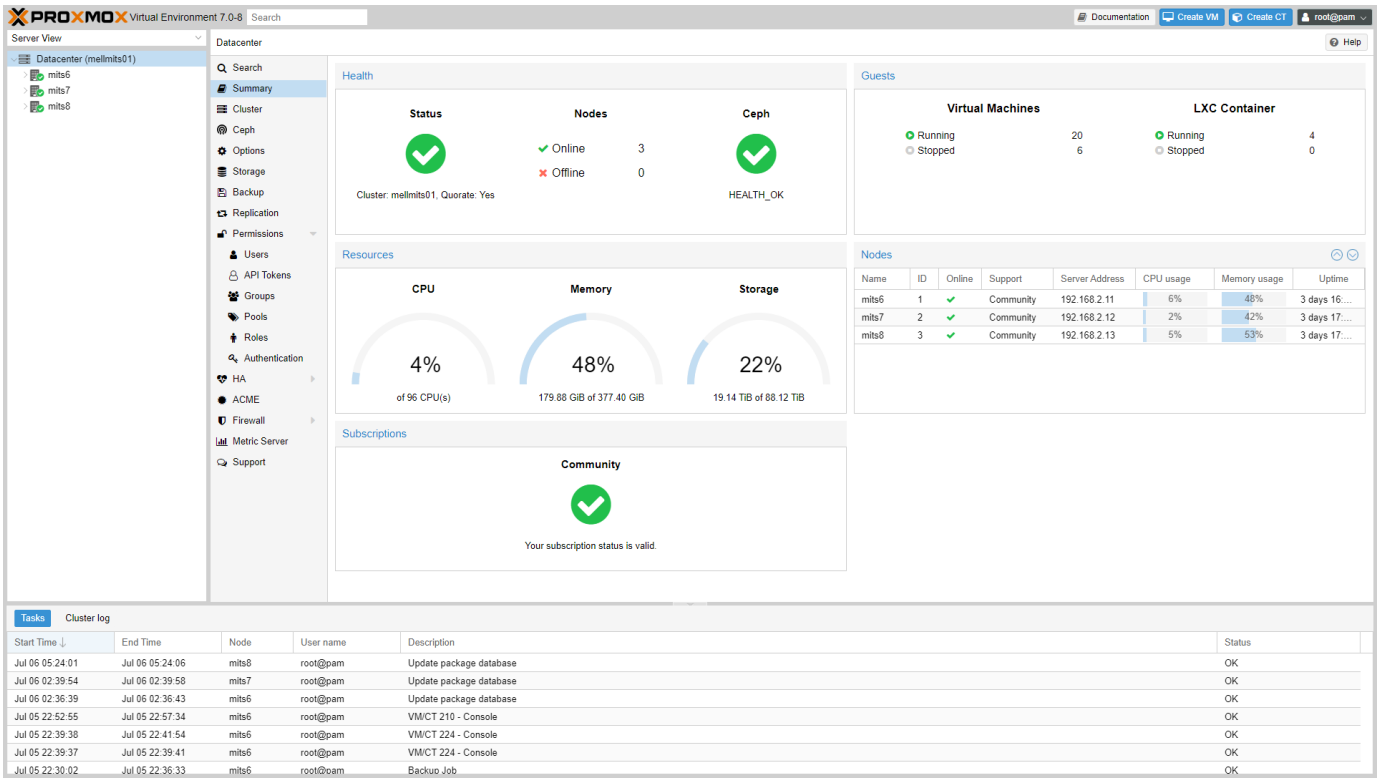
Gestion des isolateurs

Outils de gestion

Hyperviseur / Isolateur	Outil de gestion
OpenVZ	Proxmox
QEMU / KVM	libvirt / oVirt
Virtualbox	RemoteBox / Vagrant
LXC	LXD / Proxmox

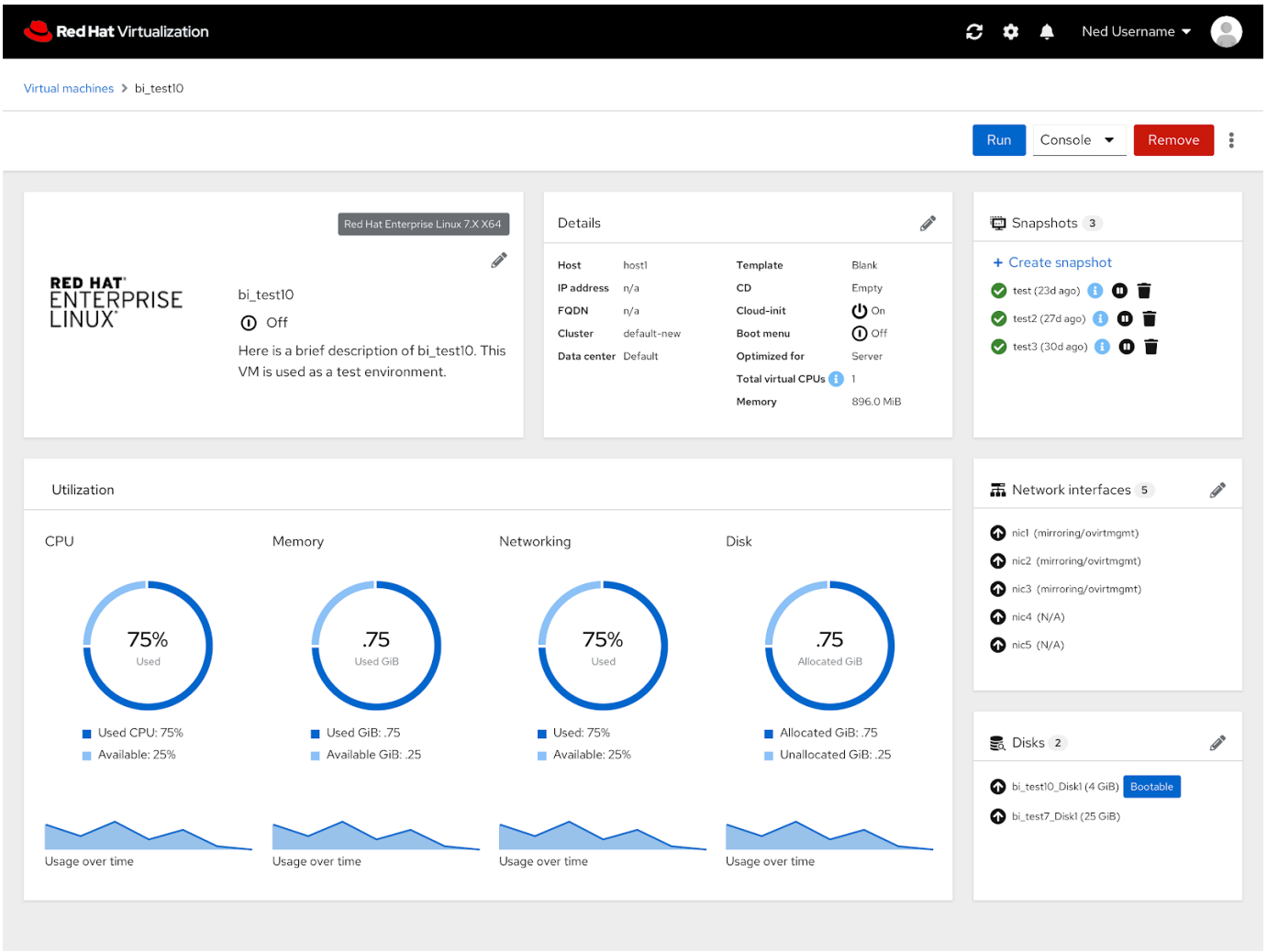
Proxmox

- Web UI
- Historiquement basé sur OpenVZ et KVM
- Utilise maintenant LXC
- Gestion de data center



oVirt

- Web UI
- Basé sur OpenVZ et libvirt
- Gestion de data center



Vagrant

- Ligne de commande
- Basé initialement sur VirtualBox (par défaut) et VMWare
- Accepte des "providers" (LXC, OpenVZ, Docker, AWS, etc.)

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/bullseye64"
  config.vm.box_check_update = false

  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4000"
  end

  3.times do |i|
    config.vm.define "server#{i}" do |machine|
      machine.vm.hostname = "server#{i}"
      machine.vm.network "private_network", ip: "192.168.50.#{10+i}"

      if i == 0
        machine.vm.network "forwarded_port", guest: 80, host: 1080
```

```
      machine.vm.network "forwarded_port", guest: 8080, host: 8080
    end
  end
end

config.vm.provision "shell", path: "provision.sh"
end
```

Des points de friction

Paramétrage sur l'hôte

Comment gérer :

- Lancement du container au démarrage du système ?
- Réseau :
  - bridge sur l'hote ?
  - carte réseau virtuelle par container ?
- Droits sur les périphérques matériels ?
- Préparation spécifique du filesystem
  - sur mesure ?
  - avec des scripts de provisionning ? (ex: templates LXC)
  - avec clonage d'images plutot que re-création from scratch ?

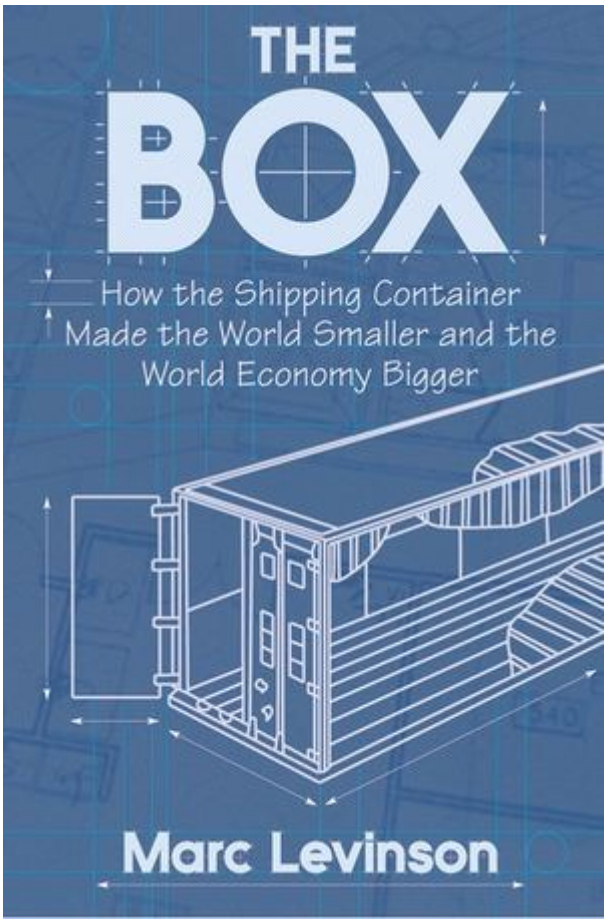
Paramétrage sur le système guest (invité)

- Installation spécifique de logiciels ?
- Configuration spécifique ?

Besoin de standardisation ?

Histoire des conteneurs maritimes

- Malcom McLean (1914-2001) est magnat dans le transport routier.
- Il constate que les routes sont saturées et investit dans le transport maritime.
- Il se rend rapidement compte :
  - que l'espace inutilisé est trop important !
  - que le temps de chargement et déchargement est long…
  - …à cause de l'hétérogénéité de la marchandise transportée !



- McLean invente la notion de "conteneur" (container en anglais)
  - Indépendant du contenu
  - La taille est uniforme (ainsi que les outils (grues) pour charger et décharger)
  - La taille est standard (35 pieds, c-a-d le standard USA des camions)
  - Le chargement est dense

    - Les conteneurs réduisent les risques :
        - d'erreur liés aux manipulations répétées des marchandises
        - de vols
- Il fonde la société SeaLand, rachetée en 1999 par Maersk, leader mondial du conteneur.

- Le premier bateau porte-conteneur, le Ideal-X
    - Part le 26 avril 1956 du port de Newark (New Jersey) vers le port de Houston (Texas).
    - A son bord, 58 conteneurs
    - McLean avait ansi calculé que le coût de transport passerait de $5.83 la tonne à $0.16
- Les containeurs ont donc **réduit les coûts de transport cargo de 90%**.

Parallele avec les isolateurs

- Agile methodologies → more deliverables → more pressure on Ops
- Pets VS Cattle
- CI/CD and DevOps methodologies

Références

- Wikipedia: Malcom McLean
- Les Echos: Malcom McLean
- FEE Stories: Malcom McLean: Truck Driver, Entrepreneur, Billionaire
- ISO: Une mise en boîte intelligente – Comment la normalisation a construit l’économie mondiale
- DevOps Concepts: Pets vs Cattle

- The History of Pets vs Cattle and How to Use the Analogy Properly
- Cattle vs Pets - DevOps Explained

Qu'est-ce que Docker ?

Un outil pour manipuler des conteneurs

Docker est une plate-forme open-source pour :

- le développement,
- la production
- et l'exécution d'applications.

Princpe de Docker :

- séparer vos applications de votre infrastructure
- ET traiter votre infrastructure comme une application gérée.

Docker aide :

- à livrer le code plus rapidement,
- à tester plus rapidement,
- à déployer plus rapidement
- et à raccourcir le cycle entre l'écriture du code et le code en exécution.

Projet jeune… mais actif !

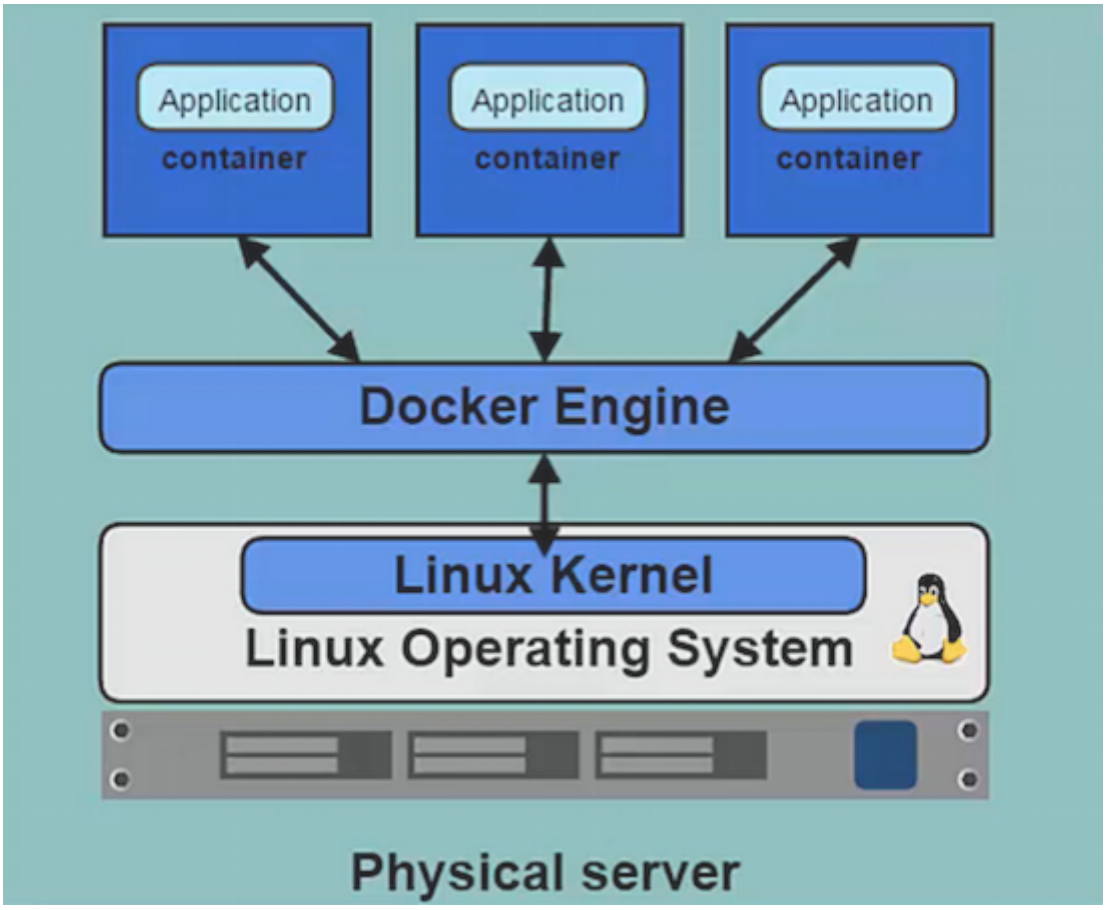
- 2013-01 : 1er commit chez dotCloud
- 2013-02 : 1ere démo en ligne
- 2013-03 : Version 0.1.
  - 1ere démo à Pycon US.
  - Docker devient open-source. Ouverture du dépôt GitHub
- 2013-04 : Version 0.2

- 2013-05 : Version 0.3
- 2013-06 : Version 0.4
    - Docker rejoint la fondation Linux
- 2013-05 : Version 0.5 (top, mount)
- 2013-08 : Version 0.6 (-privileged, LXC conf)
- 2013-09 : Partenariat avec Red Hat
- 2013-10 :
    - Société renommée en Docker
    - Version 0.7 (noyau standard, device-mapper, name, links)
- 2014-02 :
    - Levée de 15M$
    - Version 0.8 (MacOSX, BTRFS experimental, ONBUILD)
- 2014-03 : Docker remplace LXC par libcontainer, framework construit sur-mesure
- 2014-06 :
    - Début de Kubernetes. Plus Docker croit en popularité, plus la demande d'outils d'orchestration augmente.
- 2014-11 : Amazon’s EC2 Container Service (ECS) premier hébergement container-as-a-service.
- 2015-06 : Lancement de l'Open Container Initiative, qui promeut les standards ouverts liés aux containers.
- 2016-01 : Docker Inc. acquiert Unikernels, une petite société spécialisée sur kes unikernels
- 2016-05 : Les principaux contributeurs de Docker sont Cisco, Google, Huawei, IBM, Microsoft, and Red Hat.
- 2016-06 : Docker introduit Swarm, un orchestrateur désormais inclus par défaut.
- 2016-09 : Docker fonctionne nativement sous windows.
- 2017-01 : Le nombre de profils mentionnant Docker sur Linkedin augmente de 160% par rapport à l'année précédente.

Concepts & vocabulaire

Docker Engine

- le programme qui permet de construire et de faire fonctionner des conteneurs.



Images and Conteneurs

Images

- Modèle en lecture seule utilisé pour créer des conteneurs
- Construit par vous ou d'autres utilisateurs de Docker
- Stocké dans le Docker Hub ou dans votre registre local

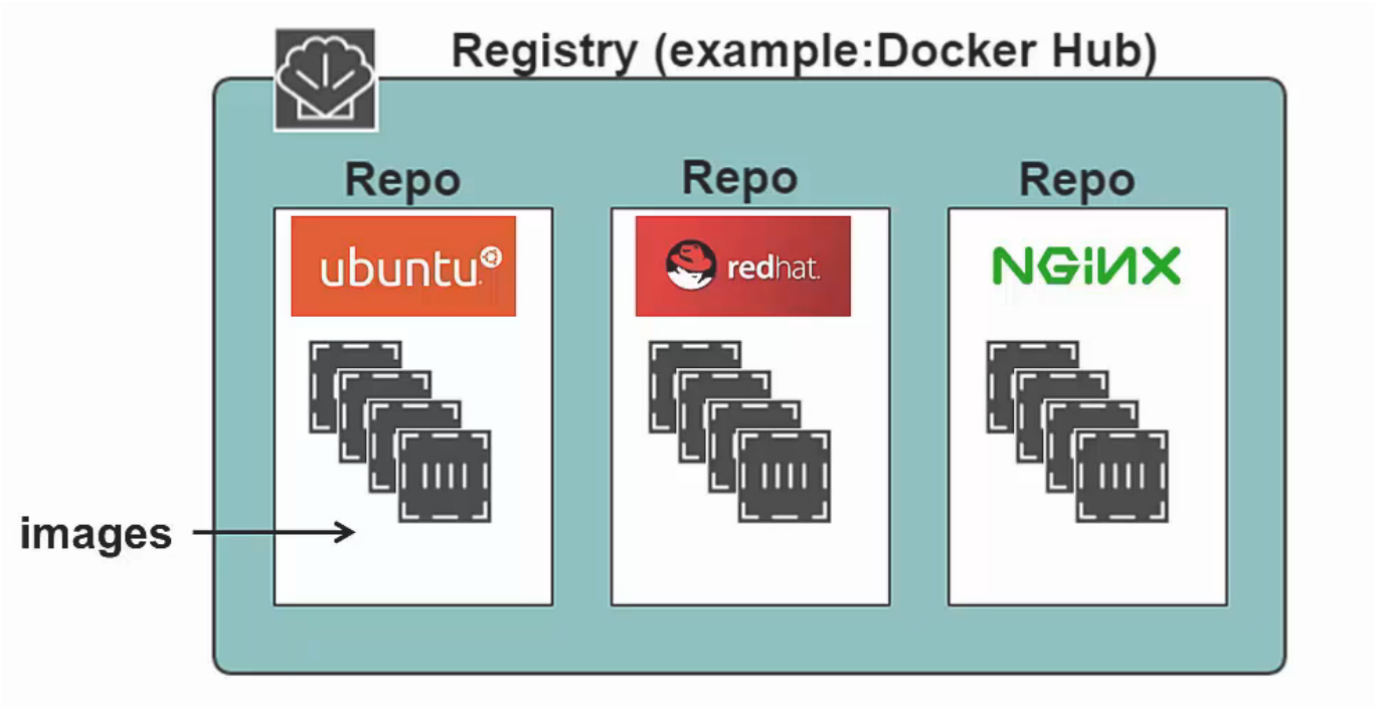
Conteneurs

- Plate-forme d'application isolée
- Contient tout ce dont vous avez besoin pour exécuter votre application

- Basé sur des images

Registre and Dépôts

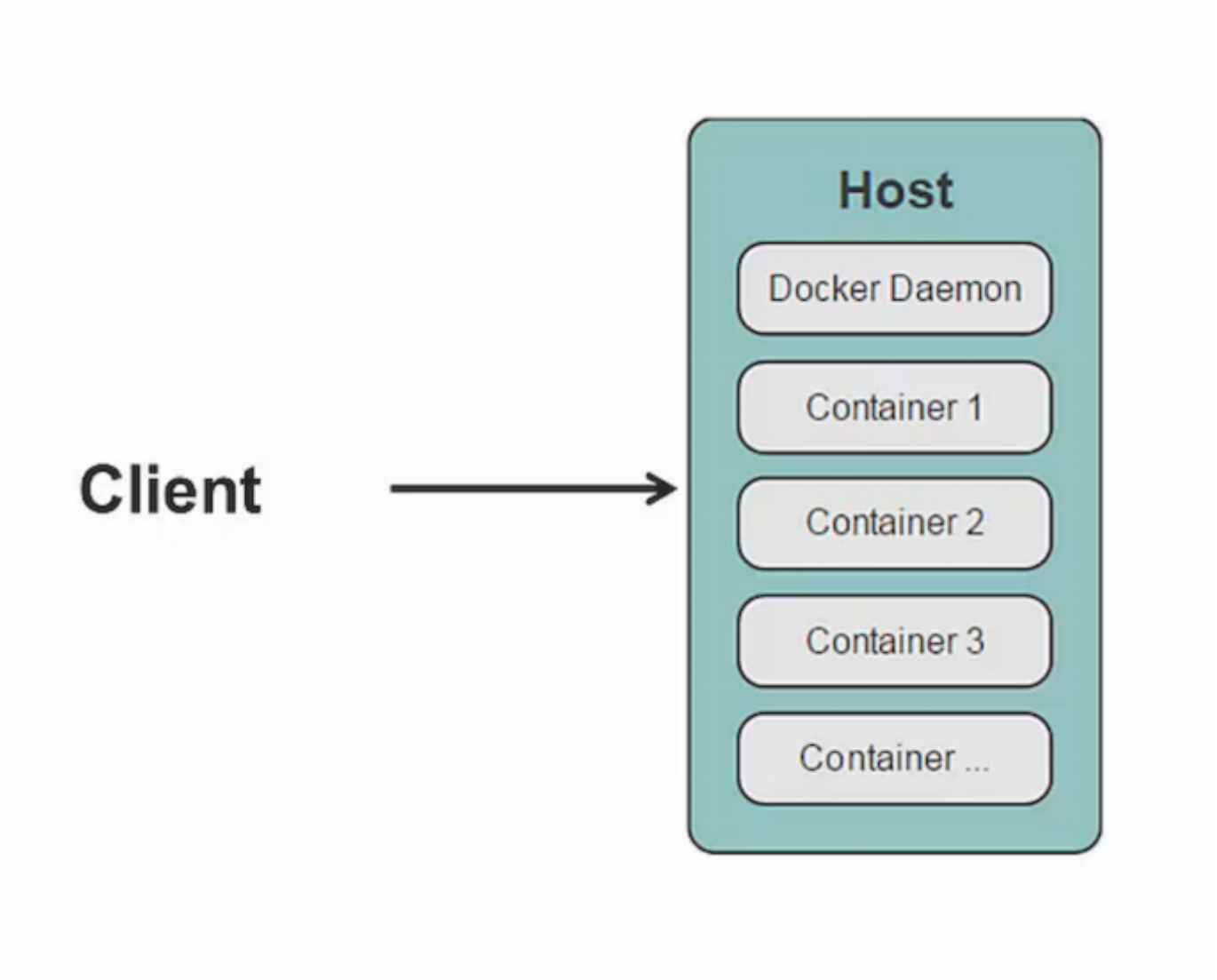
- Le registre est l'endroit où sont sauvegardées les images.
- Le registre peut être privé ou public (Docker Hub)
- Les dépôts sont à l'intérieur du Registre



L'architecture de Docker

Docker Client et Daemon

- Architecture client / serveur
- Le client prend les entrées de l'utilisateur et les envoie au daemon.
- Le daemon construit, lance et dispatche des conteneurs
- Le client et le démon peuvent fonctionner sur des hôtes identiques ou différents.



Images Docker

- Les images de Docker sont des modèles
  - Elles sont en lecture seule
  - Elle serviront de base au lancement de conteneurs
- Chaque image se compose d'une série de couches superposées
  - Chaque image part d'une image de base
    - E.g. ubuntu, Apache.
  - Le processus de construction suit ensemble d'étapes, des instructions
  - Chaque instruction crée une nouvelle couche (comme un calque) dans notre image.
  - Ces couches sont des filesystems, chacune des couche "complète" celle du dessous (ex: fichiers en plus, en moins, déplacés, complétés, etc.)
  - Docker utilise un système permettant de combiner ces couches en un seul système de fichiers
- Type d'instructions :
  - Exécutez une commande.
  - Ajouter un fichier ou un répertoire.
  - Créer une variable d'environnement.
  - Définir un processus à exécuter au lancement de cette image
  - etc.

Conteneurs Docker

- Chaque conteneur est construit à partir d'une image.
- Un conteneur se compose :
  - d'un système d'exploitation,
  - de fichiers ajoutés par l'utilisateur
  - et de métadonnées.
- L'image indique à Docker :
  - ce que contient le conteneur,
  - le processus à exécuter lorsque le conteneur est lancé
  - et diverses autres données de configuration.
- A l'exécution du conteneur Docker ajoute un calque en lecture-écriture par-dessus l'image

Containers Docker vs. VMs

- S'exécute en l'espace utilisateur plutôt qu'en espace noyau
- Les VM dupliquent l'OS et les bibliotheques du système hôte
- Docker réutilise les fichiers grace à UnionFS
- Les petites images du docker facilitent la gestion
    - Créer une image immuable par build
    - Le déploiement de l'image est plus rapide

Le registre d'images Docker

- Les images sont stockées localement, mais elle peuvent être extraites d'un Registre d'images.
- Le Registre est un serveur qui stocke et permet de distribuer des images Docker.
- Le Registre stocke et distribue chaque couche de l'image séparément

Pré-requis

- Kernel 3.8+ (avec Cgroups et namespaces) ou RHEL 2.6.32
- AUFS (ou device-mapper / VFS)
- LXC
- CPU 64 bits

> **Note**
>
> Il est possible de faire fonctionner Docker sur 32 bits, mais aucune des images publiques ne sera adaptée.

Vous n'avez pas Linux ?

Docker a besoin du noyau Linux.... donc vous aurez besoin d'une machine virtuelle linux.

Certaines installations pour Windows / MacOS prévoient directement l'installation d'une VM.

Installation sous Linux

Installation de Docker

Méthode propre

- Utiliser les paquets de votre distribution ou bien installer le dépot de paquets de Docker pour votre distribution (pour les versions les plus récentes)
- Installer le logiciel avec votre gestionnaire de paquets

Exemples :

- Installation de Docker sous Debian
- Installation de Docker sous CentOS
- Installation de Docker sous Fedora

Méthodes sales

- Installation depuis les binaires
- Installation depuis un shellscript téléchargé et exécuté à la volée

```
$ wget -qO- https://get.docker.com/ | sh
```

Installation de Docker-Compose

```
$ sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(unam
-m)" -o /usr/local/bin/docker-compose
$ sudo chmod +x /usr/local/bin/docker-compose
```

Références

- Docker Documentation: Install Docker Compose

Installation de Docker sur MacOS

- Mac => virtualisé dans VBOX mais bien packagé + wrapper

Installation de Docker sous Windows

Windows and Mac OS X:

Docker need linux kernel... so you will need a linux Virtual Machine. Docker toolbox contains everything needed.

À propos de l'installation

- Windows
- Windows 7 => virtualisé dans VBOX
- Windows 10 => WSL + libcontainer en quasi natif (il faut activer WSL...)
- Le plus simple : Vagrant + Docker

Préparation du système

- Windows : https://gist.github.com/glenux/92dbcefff3024e0a286774d88f2c5c47

Vérifier l'installation

Coté client

```
$ docker --version
Docker version 18.09.7, build 2d0083d
```

Coté serveur

```
$ docker info
Containers: 75
 Running: 0
 Paused: 0
 Stopped: 75
Images: 292
Server Version: 18.09.7
Storage Driver: overlay
 Backing Filesystem: extfs
 Supports d_type: true
Logging Driver: json-file
Cgroup Driver: cgroupfs
Plugins:
 Volume: local
 Network: bridge host macvlan null overlay
 Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
Swarm: inactive
Runtimes: runc
Default Runtime: runc
Init Binary: docker-init
containerd version: 894b81a4b802e4eb2a91d1ce216b8817763c29fb
runc version: 425e105d5a03fabd737a126ad93d62a9eeede87f
init version: fec3683
Security Options:
 apparmor
 seccomp
  Profile: default
Kernel Version: 5.0.0-trunk-amd64
Operating System: Debian GNU/Linux 10 (buster)
OSType: linux
Architecture: x86_64
CPUs: 4
Total Memory: 7.677GiB
Name: dilong
ID: 5UKL:GVGR:CYXM:ETFH:YPHB:7JPG:7STX:D7G5:FSCU:QZDR:Z52K:TB5W
Docker Root Dir: /var/lib/docker
Debug Mode (client): false
Debug Mode (server): false
Username: glenux
Registry: https://index.docker.io/v1/
Labels:
```

```
Experimental: false
Insecure Registries:
 127.0.0.0/8
Live Restore Enabled: false
Product License: Community Engine

WARNING: No swap limit support
WARNING: the overlay storage-driver is deprecated, and will be removed in a future release.
```

Important

Si vous avez une erreur de type permission denied :

1. vérifiez que votre utilisateur fait bien partie du groupe docker
2. relancez votre session avant de relancer la commande

Vérifier le bon fonctionnement

```
$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
1b930d010525: Pull complete
Digest: sha256:6540fc08ee6e6b7b63468dc3317e3303aae178cb8a45ed3123180328bcc1d20f
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Cette commande télécharge et lance un conteneur `hello-world`

Références

- Docker Documentation: Orientation and setup

Créer un conteneur

Avec la commande `docker run`

Syntaxe :

```
$ docker run [options] [image] [command] [args]
```

- est spécifiée par `repository:tag`

Example

```
$ docker run ubuntu:14.04 ps aux
```

Containeur et terminal

Utilisez quelques options :

- `-i` indique à docker de connecter STDIN sur le conteneur
- `-t` demande à Docker d'obtenir un pseudo-terminal

Example

```
$ docker run -i -t ubuntu /bin/bash
```

Les processus des conteneurs

- Un conteneur ne fonctionne que tant que le processus de votre commande est en cours d'exécution
- Le processus de votre commande est toujours PID 1 à l'intérieur du conteneur

Tip

Pour quitter le conteneur sans l'arrêter `CTRL+P` `Q`.

Identifiant des conteneurs

- Les conteneurs peuvent être spécifiés à l'aide de leur ID ou de leur nom
- ID long et ID court
- L'ID court et le nom peuvent être obtenus en utilisant la commande `docker ps` pour lister les conteneurs
- ID long est obtenu par l'inspection d'un conteneur

Retrouver les conteneurs

- Utiliser `docker ps` pour lister les conteneurs en cours d'exécution
- Utiliser `docker ps -a` pour lister tous les conteneurs (y compris les conteneurs qui sont arrêtés)

Inspecter les containers

`docker inspect CID`

Commandes, arrière-plan et logs

Commande "Hello world"

```
$ docker run ubuntu:14.04 /bin/echo 'Hello world'
```

- Nous avons d'abord appelé le binaire docker et l'action que nous voulions exécuter, run.
- Ensuite, nous indiquons une image : ubuntu:14.04.
- Ensuite, nous explicitons quelle commande exécuter dans notre nouveau conteneur

Example

```
$ docker run ubuntu:14.04 /bin/echo 'Hello world'
Unable to find image 'ubuntu:14.04' locally
14.04: Pulling from library/ubuntu
a7344f52cb74: Pull complete
515c9bb51536: Pull complete
e1eabe0537eb: Pull complete
4701f1215c13: Pull complete
Digest: sha256:2f7c79927b346e436cc14c92bd4e5bd778c3bd7037f35bc639ac1589a7acfa90
Status: Downloaded newer image for ubuntu:14.04
Hello world
```

Daemon (service) "Hello world"

```
$ docker run -d ubuntu:14.04 \
    /bin/sh -c "while true; do echo 'hello world'; sleep 1; done"
```

- `docker run -d` indique à Docker de lancer le conteneur et de le mettre en arrière-plan, pour le démoniser
- Nous avons spécifié la même image : ubuntu:14.04.
- La commande à exécuter est un script shell qui fait écho à'Hello world' pour toujours.

Pour rechercher notre service

```
$ docker ps
CONTAINER ID  IMAGE         COMMAND             CREATED        STATUS       PORTS NAMES
1e5535038e28  ubuntu:14.04  /bin/sh -c 'while tr  2 minutes ago  Up 1 minute        insane_babbage
```

Logs

```
$ docker logs insane_babbage
hello world
hello world
hello world
...
```

Démarrage et arrêt des conteneurs

- Trouvez d'abord vos conteneurs avec `docker ps` et notez l'ID ou le nom

Arrêt d'un container

```
$ docker stop <container ID>
```

Démarrage d'un container

```
$ docker start <container ID>
```

Execution de commandes

Avec `docker exec` on peut démarrer un nouveau processus au sein d'un conteneur

Utile notamment pour :

- Obtenir un shell (et débugger)
- Lancer une commande de maintenance

```
$ docker exec -i -t [container ID] commande
```

> **Note**
>
> Quitter le terminal ne terminera pas le conteneur

Supression de conteneurs

Commande `docker rm <container>`.

- Spécifiez l'ID ou le nom du conteneur

> **Note**
>
> Ne peut effacer que les conteneurs qui ont été arrêtés

Images et registres

De nombreuses d'images sont disponibles pour l'utilisation dans docker

Elle peuvent être créées par l'utilisateur ou bien téléchargées depuis un registre distant.

Images dans le registre local

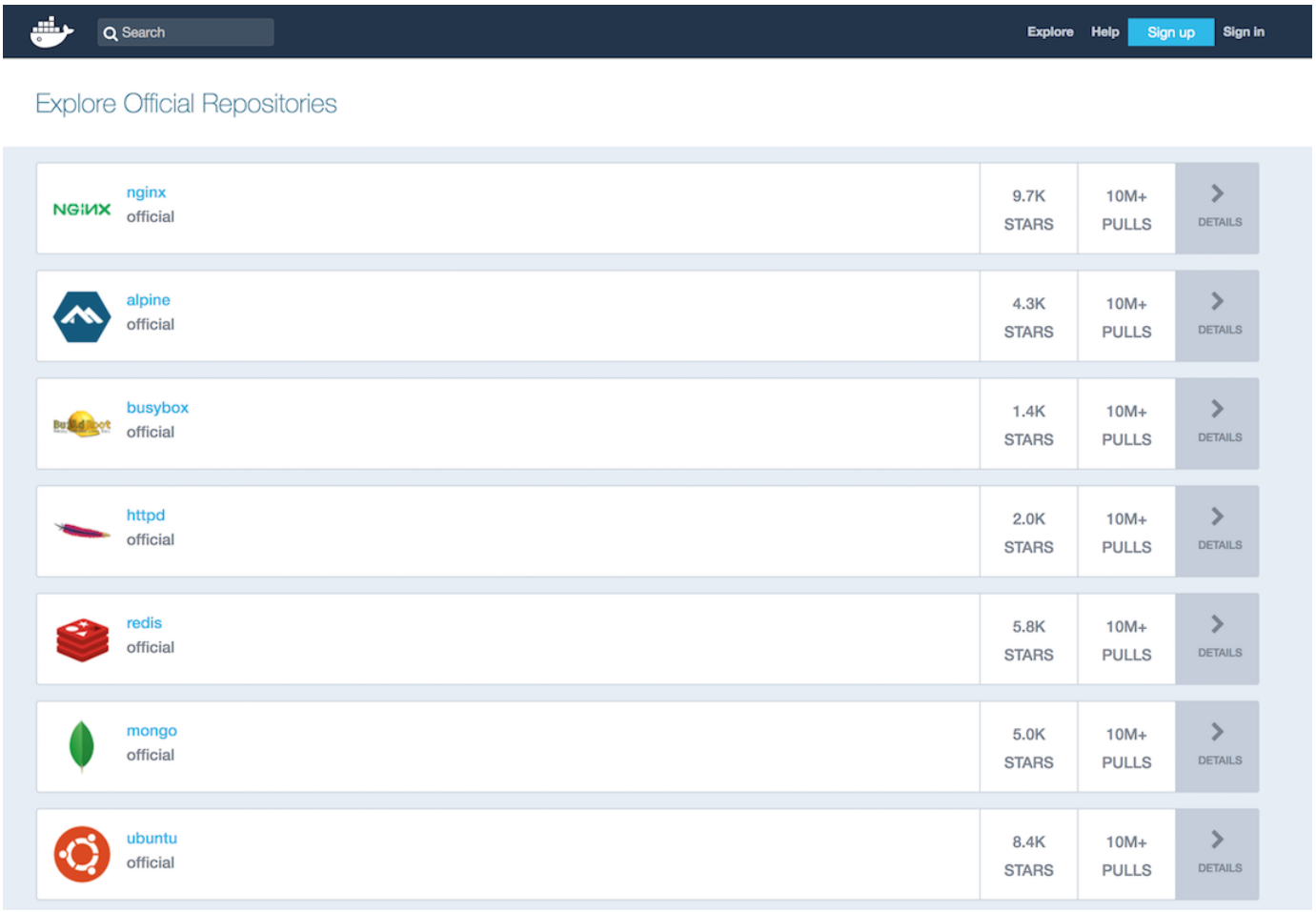
```
$ docker images
```

- Lors de la création d'un conteneur, Docker tentera d'utiliser d'abord une image locale.
- Si aucune image locale n'est trouvée, Docker cherchera dans Docker Hub
  - …sauf si un autre registre est spécifié

Images dans un registre distant

N'importe qui peut mettre en place un registre pour héberger des images docker.

Les registres les plus connus sont DockerHub (par défaut dans docker) et Quay.io (proposé par RedHat).



Pour rechercher des images dans ces registres, il suffit d'utiliser le moteur de recherche intégré au registre.

Recherche en ligne de commandes

```
docker search TERMS
```

Exemple:

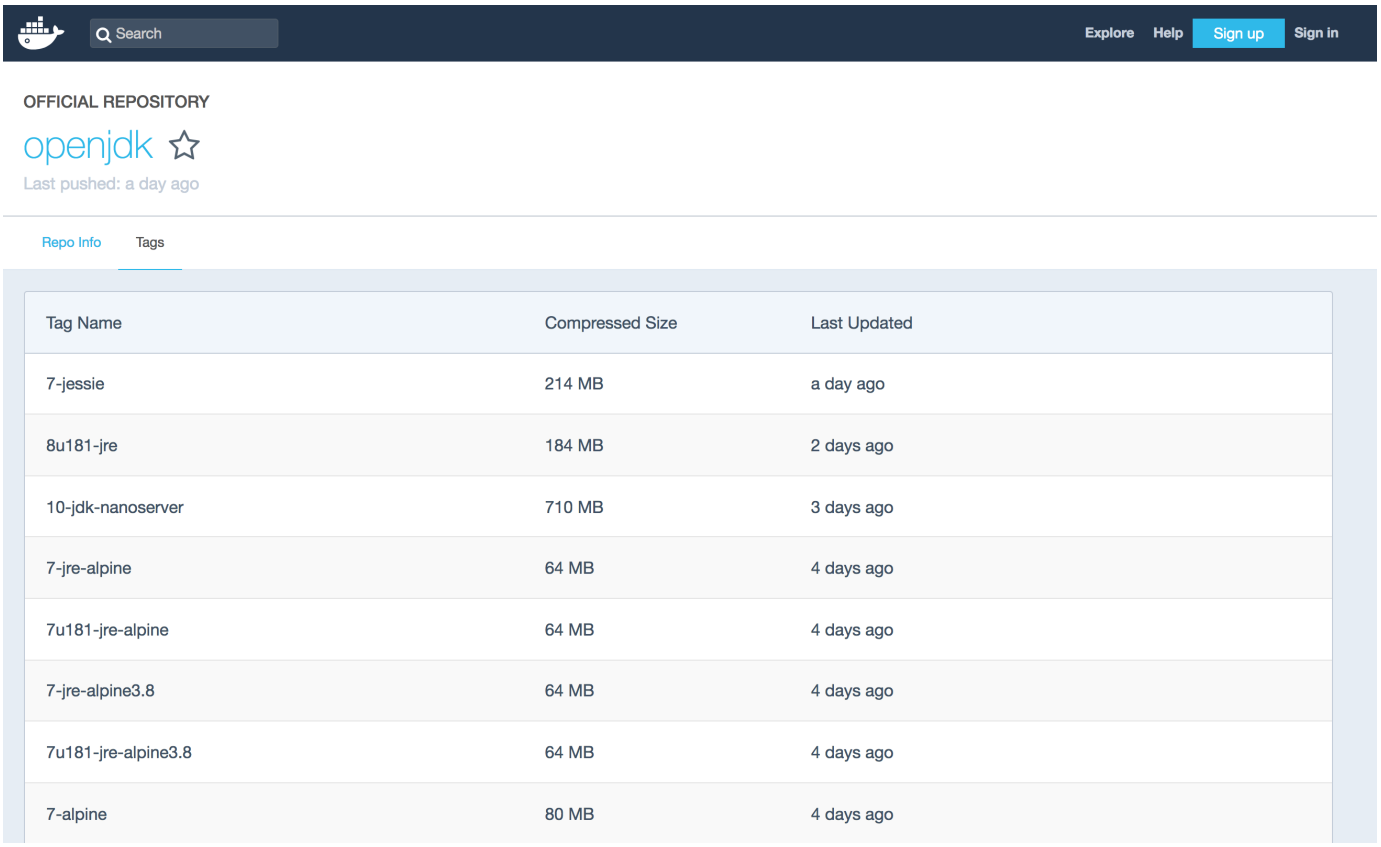
```
$ docker search tomcat
NAME                                     DESCRIPTION                                     STARS     OFFICIAL   AUTOMATED
tomcat                                   Apache Tomcat is an open source implementati…   3306      [OK]
tomee                                    Apache TomEE is an all-Apache Java EE certif…   96        [OK]
bitnami/tomcat                           Bitnami Tomcat Docker Image                     44                   [OK]
arm32v7/tomcat                           Apache Tomcat is an open source implementati…   11
rightctrl/tomcat                         CentOS , Oracle Java, tomcat application ssl…   7                    [OK]
arm64v8/tomcat                           Apache Tomcat is an open source implementati…   7
amd64/tomcat                             Apache Tomcat is an open source implementati…   4
tomcat2111/pisignage-server              PiSignage Server                                3                    [OK]
jelastic/tomcat                          An image of the Tomcat Java application serv…   3
cfje/tomcat-resource                     Tomcat Concourse Resource                       2
oobsri/tomcat8                           Testing CI Jobs with different names.           2
tomcatling/jupyterhub_aws                                                                1
chenyufeng/tomcat-centos                 tomcat基于centos6的镜像                              1                    [OK]
tomcat2111/bitbucket-pipelines-elasticsearch Elasticsearch for Bitbucket's Pipelines     0
tomcat2111/phpredisadmin                 This is a Docker image for phpredisadmin        0                    [OK]
softwareplant/tomcat                     Tomcat images for jira-cloud testing            0                    [OK]
tomcat2111/papercut-mf                   PaperCut MF Application Server                  0
tomcat2111/redaxo                        Redaxo                                          0
store/microsoft/defaultpublisher         Zulu for Azure build of OpenJDK                 0
tomcatengineering/docker_swarm_exporter  Prometheus metrics exporter for Docker Swarms   0
tomcatengineering/pg_backup_rotated      Clone of martianrock/pg_backup_rotated but w…   0
tomcat2111/nextcloud                     Nextcloud container with environment variabl…   0                    [OK]
s390x/tomcat                             Apache Tomcat is an open source implementati…   0
tomcat2111/piwik                         Matomo (formerly Piwik) image                   0
tomcat0823/auto1                                                                         0
```

Labels d'images

- Les images sont spécifiées par `repository:tag` (en français: `dépôt:label`)
- Les `repository`
  - peuvent contenir un nom de domaine et un chemin
  - sont assimilables à des noms projets
- Les `tag` sont assimilables à des numéros de versions
- La même image peut avoir plusieurs `repository:tag`
- Le label par défaut est `latest` quand celui-ci n'est pas précisé.

> **Important**
>
> Le label `latest` est une valeur qui n'est pas automatique, elle doit être définie spécifiquement par ceux qui mettent à disposition des images.

Par exemple, pour l'image openjdk



Références

- https://hub.docker.com/explore/
- Docker Documentation: docker search

Téléchargement d'images

```
$ docker pull REPOSITORY:TAG
```

Références

- Docker Documentation: docker pull

Suppression d'images

Utilisez la commande `docker rmi`.

```
docker rmi <image ID>
```

ou

```
docker rmi <repo:tag>
```

Note

Si une image est labelisée plusieurs fois, il faudra supprimer chaque balise

Renommage d'images

```
docker tag [image ID] [repo:tag]
```

ou bien

```
docker tag [local repo:tag] [Docker Hub repo:tag]
```

Références

- Docker Documentation: docker rename

Téléversement d'images

Création d'un dépôt

- Il faut avoir accès à un `registry` (registre) en écriture
- Il faut avoir le droit de créer un `repository` et d'écrire des images dedans
- Les utilisateurs peuvent créer leurs propres dépôts sur Docker Hub.
- Public et privé (un gratuit)
- Poussez les images locales vers un référentiel

Pousser une image sur Docker Hub

Utiliser la commande `docker push`

```
$ docker push [repo:tag]
```

- Le repo local doit avoir le même nom et le même tag que le repo Docker Hub.

Archivage et restauration

Archivage avec docker save

```
$ docker save IMAGE_ID > archive.tar
```

Restauration avec docker load

```
$ docker load < archive.tar
```

Intérêt

Analyse des images

- `docker save` produit une archive tar
  - Celle-ci contient un dossier pour chaque couche de l'image
  - Chacun de ces dossiers contient des méta-données et archives tar pour chaque couche
- Cela permet l'exploration et l'analyse du contenu des images docker (pour leur sécurité, pour les optimiser, etc.)

```
$ mkdir dokuwiki
$ docker save dokuwiki:latest > archive.tar
$ tar xavf archive.tar
28c062ccf7f2f886ae5ea38d01c00b5de4dcc421c6053eb09d2fd9cd7936c561/
28c062ccf7f2f886ae5ea38d01c00b5de4dcc421c6053eb09d2fd9cd7936c561/VERSION
28c062ccf7f2f886ae5ea38d01c00b5de4dcc421c6053eb09d2fd9cd7936c561/json
28c062ccf7f2f886ae5ea38d01c00b5de4dcc421c6053eb09d2fd9cd7936c561/layer.tar
4de9349b11d6ab1a32ec1ad2683b9469f1804710f70e588542a91080b0efda19/
4de9349b11d6ab1a32ec1ad2683b9469f1804710f70e588542a91080b0efda19/VERSION
4de9349b11d6ab1a32ec1ad2683b9469f1804710f70e588542a91080b0efda19/json
4de9349b11d6ab1a32ec1ad2683b9469f1804710f70e588542a91080b0efda19/layer.tar
5fb87d35b556fdd3bc08288d45e32c1f3b466d0f2c9deedbd66c000d312f68b7/
5fb87d35b556fdd3bc08288d45e32c1f3b466d0f2c9deedbd66c000d312f68b7/VERSION
5fb87d35b556fdd3bc08288d45e32c1f3b466d0f2c9deedbd66c000d312f68b7/json
5fb87d35b556fdd3bc08288d45e32c1f3b466d0f2c9deedbd66c000d312f68b7/layer.tar
7de849f9519543e9f515e4f10c2df42674a5f619e62855a3c999d1b679b00271.json
manifest.json
repositories
```

Transfert d'images sans registry

- La combinaison `docker save` + `docker load` avec ssh permet les transferts entre deux serveurs sans passer par un registry
- En plus `docker load` est capable de décompresser automatiquement les archives au format gz, bzip2 et xz

Exemples:

```
$ docker save app:1.0 | bzip2 \
   | ssh johndoe@example.com docker load
```

```
$ docker save app:1.0 | bzip2 \
   | DOCKER_HOST=ssh://johndoe@example.com docker load
```

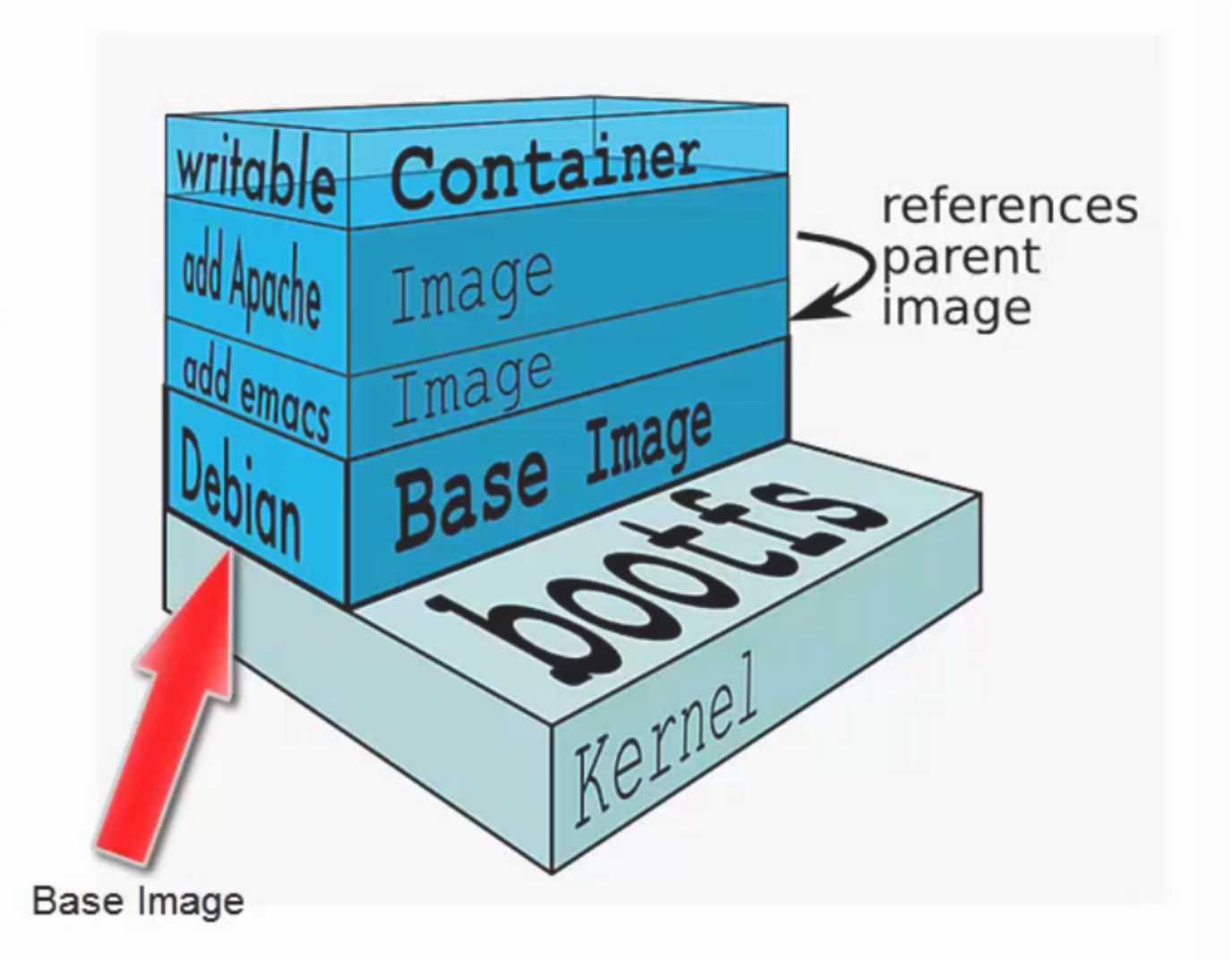
Références

- Docker Documentation: docker save
- Docker Documentation: docker load

Création d'images

Couches de l'image

- Les images sont composées de plusieurs couches
- Les couches sont des dossiers contenant un morceau de filesystem
- Les couches sont en lecture seule
- Chaque couche référence une couche parente, qu'elle complète
- Chaque image contient une couche de base (sans parent)
- Docker utilise un système de copie en écriture (copy on write)



La couche inscriptible du conteneur

- Docker crée une couche supérieure inscriptible pour les conteneurs.
- Les images mères sont en lecture seule
- Tous les changements sont effectués au niveau de la couche inscriptible

Docker commit

La commande **docker commit** crée une image d'apres un container.

Elle "fige" les changements qui ont été faits dans la couche inscriptible sous forme d'une nouvelle couche d'image.

Usage:

```
$ docker commit [options] [container ID] [repository:tag]
```

- Le nom du référentiel doit être basé sur le nom d'utilisateur/application.
- Peut référencer le conteneur avec le nom du conteneur au lieu de l'ID

Exemple:

```
$ docker commit <id> glenux/dokuwiki:2
```

Note

Quand le tag n'est pas précisé, sa valeur par défaut est `latest`

Références

- Docker Documentation: docker commit

Création d'images avec Dockerfile

Construction d'une image avec docker build

Syntax

```
$ docker build [options] [path]`
```

```
$ docker build -t [repository:tag] [path]`
```

Options:

- `-t` pour définir le nom de l'image
- `--file` pour spécifier le chemin du `Dockerfile`
- `path` pour le chemin du contexte de construction

Contexte de build

Fichier Dockerfile

A **Dockerfile** is a configuration file that contains instructions for building a Docker image

- Provides a more effective way to build images compared to using docker commit
- Easily fits into your continuous integration and deployment process

Example:

```
FROM debian:11
RUN apt-get update \
 && apt-get install -y apache2 php7.4 \
 && apt-get clean

COPY --chown=www-data:www-data . /var/www/html

CMD apachectl -D FOREGROUND
```

Dockerfile instructions

- Instructions specify what to do when building the image
- FROM instruction specifies what the base image should be

- RUN instruction specifies a command to execute

Build context

- When you reference files inside instructions it must be file in your build context.
- When building docker client will tar files from the context and send it to the docker daemon.
- Thats also where docker will look for your Dockerfile.
  By default its the file named Dockerfile in the root of the context (for a different one use -f option)

Example

```
$ docker build -t lodelestra/vim:1.2 .
```

ou :

```
$ docker build -t lodelestra/vim:1.2 vim_project
```

output

```
bash-3.2$ cat Dockerfile
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y vim

bash-3.2$ docker build -t lodelestra/buildproject:1.0 .
Sending build context to Docker daemon 2.048 kB
Step 0 : FROM ubuntu:14.04
---> 91e54dfb1179
Step 1 : RUN apt-get update && apt-get install -y vim
---> Running in 54c85fe3bed4
...
#install logs
...
---> 58c42776c9b7
Removing intermediate container 54c85fe3bed4
Successfully built 58c42776c9b7
```

Références

- https://github.com/Sudneo/dockerfile-audit
- https://github.com/iamabhishek-dubey/dockerlinter
- https://www.grottedubarbu.fr/docker-healthcheck/

Instruction FROM

Références

Instruction RUN

Run Instruction

- Chaque instruction RUN exécutera la commande sur la couche supérieure inscriptible et effectuera une validation de l'image.
- Vous pouvez regrouper plusieurs instructions RUN en utilisant `&&`.

Exemple :

```
RUN apt-get update && apt-get install -y curl vim
```

Instruction CMD

Instruction CMD

- CMD définit une commande par défaut à exécuter lors de la création d'un conteneur.
- CMD n'effectue aucune action pendant la construction de l'image
- Format shell et format EXEC
- Ne peuvent être spécifiés qu'une seule fois dans un fichier Docker.
- **Peuvent être remplacés au moment de l'exécution.**

```
#Shell format
CMD ping 127.0.0.1 -c 30
#Exec format (json)
CMD ["ping", "127.0.0.1", "-c", "30"]
```

Exemple

```
bash-3.2$ cat Dockerfile
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y vim
CMD ["ping", "127.0.0.1", "-c", "30"]
```

```
bash-3.2$ docker build -t lodelestra/buildprojectt:1.1 .
Sending build context to Docker daemon 2.048 kB
Step 0 : FROM ubuntu:14.04
---> 91e54dfb1179
Step 1 : RUN apt-get update && apt-get install -y vim
---> Using cache
---> 58c42776c9b7
Step 2 : CMD ping 127.0.0.1 -c 30
---> Running in 13829d762189
---> 8637e2fd41fd
Removing intermediate container 13829d762189
Successfully built 8637e2fd41fd
bash-3.2$
```

L'ajout d'un calque est beaucoup plus rapide que la construction de la première image.

Références

Image

Pratique

- Transformer un conteneur en une image (avec COMMIT)
- Création d'un fichier de recette Dockerfile
- Gestion des services Docker avec supervisor
- Automatiser la création d'une image et publier sur le store
- Créer une image Docker pour GLPI :
  - à partir d'un Debian, puis

    ```
    docker commit
    ```

  - avec un Dockerfile

Instruction ENTRYPOINT

- Définit la commande qui sera exécutée lors de l'exécution d'un conteneur.
- Les arguments d'exécution et l'instruction CMD sont transmis en tant que paramètres à l'instruction ENTRYPOINT.
- Forme shell et EXEC
- La forme EXEC est préférable car la forme shell ne peut pas accepter d'arguments au moment de l'exécution.
- Le conteneur s'exécute essentiellement comme un exécutable
- L'instruction ENTRYPOINT ne peut pas être remplacée au moment de l'exécution.

Exemple

Dockerfile

```
bash-3.2$ cat Dockerfile
FROM ubuntu:14.04
RUN apt-get update && apt-get install -y vim
ENTRYPOINT ["ping"]
```

Construire l'image

```
bash-3.2$ docker build -t lodelestra/buildproject:1.2 .
Sending build context to Docker daemon 2.048 kB
Step 0 : FROM ubuntu:14.04
---> 91e54dfb1179
Step 1 : RUN apt-get update && apt-get install -y vim
---> Using cache
---> 58c42776c9b7
Step 2 : ENTRYPOINT ping
---> Running in d8ffe9f42f9d
---> 4f940463520e
Removing intermediate container d8ffe9f42f9d
Successfully built 4f940463520e
```

Comment l'utiliser

```
bash-3.2$ docker run lodelestra/buildproject:1.2
Usage: ping [-aAbBdDfhLnOqrRUvV] [-c count] [-i interval] [-I interface]
        [-m mark] [-M pmtudisc_option] [-l preload] [-p pattern] [-Q tos]
        [-s packetsize] [-S sndbuf] [-t ttl] [-T timestamp_option]
        [-w deadline] [-W timeout] [hop1 ...] destination
bash-3.2$ docker run lodelestra/buildproject:1.2 127.0.0.1 -c 10
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
```

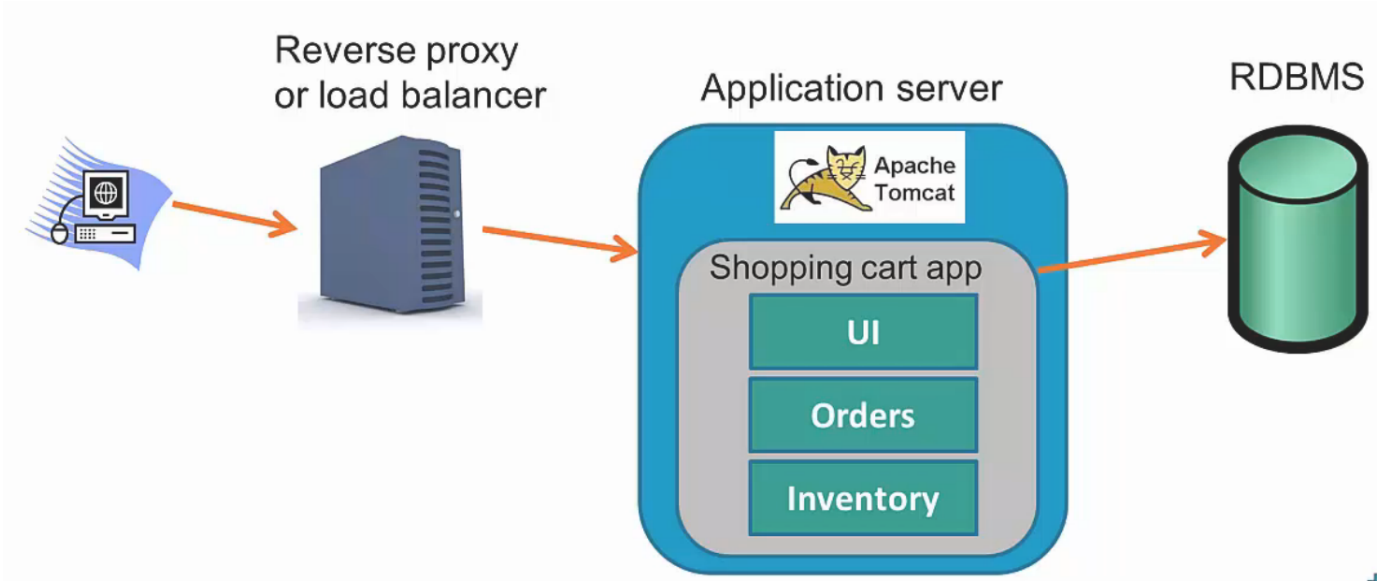
```
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.032 ms
#...
64 bytes from 127.0.0.1: icmp_seq=10 ttl=64 time=0.038 ms

--- 127.0.0.1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8996ms
rtt min/avg/max/mdev = 0.032/0.041/0.050/0.004 ms
```

Références

- https://dockerlabs.collabnix.com/beginners/dockerfile/entrypoint-vs-run.html

Concept

Architecture monolithique



- conçues pour traiter de multiples tâches connexes
  - généralement d'applications complexes
  - englobent plusieurs fonctions étroitement couplées.
- ex: une application SaaS e-commerce monolithique
  - un serveur web
  - un équilibreur de charge
  - un catalogue qui propose des images de produits
  - un système de commande
  - une fonction de paiement
  - une composante d'expédition.

Avantages

- tout en un
- plus facile à développer (au début)

Inconvénients

- énormes bases de code

- nombreux effets de bords
  - une petite modification à une seule fonction peut obliger à compiler et tester toute la plateforme

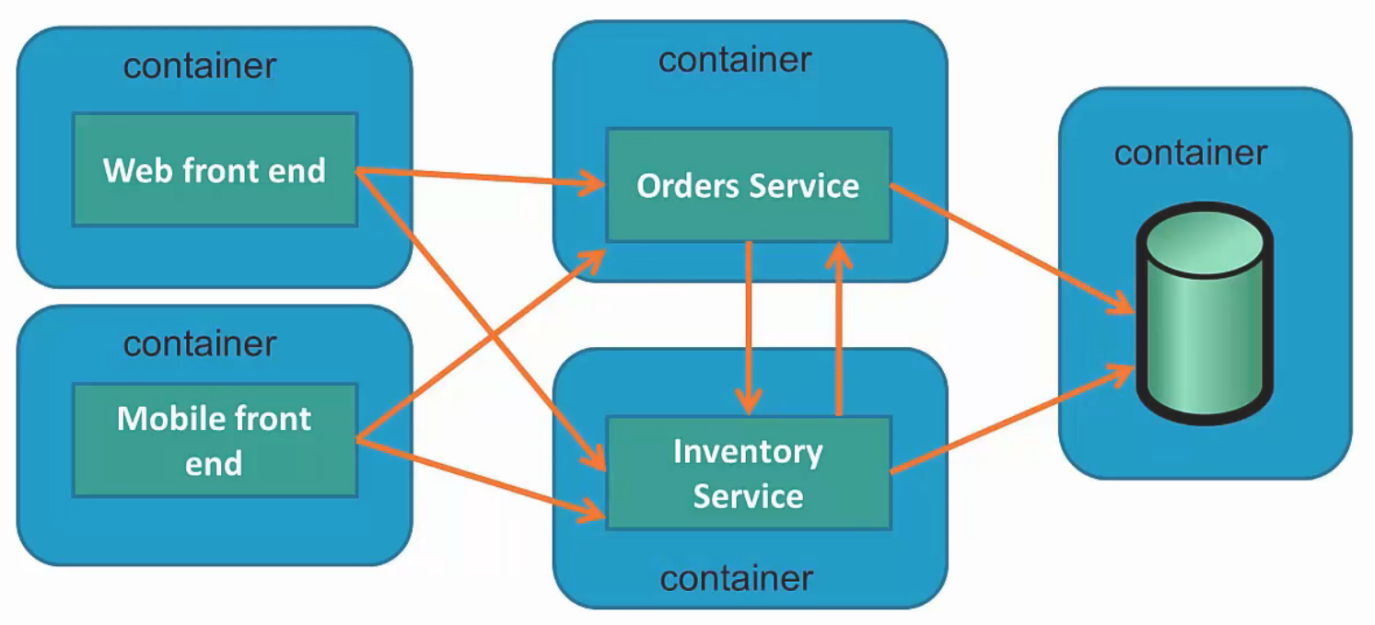
Architecture modulaire

- découplage des composants de l'application
  - sous forme de packages, ou de bibliothèques
  - mais toujours liées au sein d'une même application
- application de "bonnes pratiques" de développement
  - principes SOLID
  - méthodes agiles

Avantages

Inconvénients

Architecture en micro-services



- Services en couplage lâche
  - Reliées entre eux à travers des moyens de communication (IPC, socket, API, Bus, Message Queue, etc.)

    - Portée est moins large (donc leur taille est donc plus petite)

- Il n'est pas nécessaire que l'ensemble de l'application soit consacré à une seule pile technologique
    - Chaque micro-service peut être codé dans n'importe quel langage de programmation
    - Chacun communique avec ses "voisins" à travers une API ou un BUS

Avantages

- Chaque service peut être développé et mis à jour indépendamment
- Plus facile à comprendre pour les développeurs
    - Permet à chacun de se concentrer sur leur domaine de compétence
    - Plus simples à développer
    - Plus rapide à améliorer
    - Les pannes sont plus facile à réparer
- Facilite l'intégration et une livraison en continu (approche CI/CD)
- Si un service tombe en panne, l'application devrait continuer à fonctionner, bien qu'avec des fonctions réduites

Inconvénients

- Déployer plusieurs conteneurs simultanément.
- Lier tous les conteneurs de l'application
    - ancienne méthode (link + ENV)
    - nouvelle méthode (depend_on)
    - pb du aux container qui démarrent quand ils veulent
    - (ex: "app" a besoin de "db" mais "db" n'est pas pret => "app" crashe)
    - wait-for-port

Reference:

- Wikipedia: SOLID
- Docker Documentation: docker run
- Monolithique vs microservices : un guide sur l'architecture des applications

Travaux pratiques

- Mettre en oeuvreune application multi-conteneurs
- Haproxy
- Pool de Apache avec un script simple accédant à la BDD
- MySQL

Vue d'ensemble

Plusieurs type de stockages

- Bind mounts
- Volumes
- Tmpfs

Références

- https://docs.docker.com/storage/

Volumes

Un volume est un répertoire désigné dans un conteneur, qui est conçu pour conserver les données, indépendamment du cycle de fichier du conteneur.

- Les modifications dans les volumes sont indépendants de la mise à jour d'une image
- Persiste lorsqu'un conteneur est supprimé
- Peut être attaché à un dossier hôte
- Peut être partagé entre les conteneurs

Deux types de volumes

- chemins
- volumes nommés

Avantages

- Détacher les données stockées du conteneur qui a créé les données (par exemple, les journaux)
- Bon pour le partage des données entre les conteneurs
- Peut mettre en place un conteneur de données dont le volume se monte dans d'autres conteneurs
- Le montage de dossiers à partir de l'hôte est bon pour les essais mais n'est généralement pas recommandé pour la production

Instruction VOLUME dans le Dockerfile

🚧

Références

🚧

Bind mounts

Monter un volume

- Les volumes sont montés lors de la création ou de l'exécution d'un conteneur
- Peut être mis en correspondance avec un répertoire d'hôtes
- Les chemins de volume spécifiés doivent être absolus

Exécutez un nouveau conteneur et montez le volume nommé "testvolume" sur le dossier /myvolume dans son système de fichiers

```
docker run -d -P -v testvolume:/myvolume nginx:1.9.4
```

Exécuter un nouveau conteneur et mapper le dossier /data/src de l'hôte dans le dossier /test/src du conteneur

```
docker run -i -t -v /data/src:/test/src nginx:1.9.4
docker run -d -v "$(pwd)"/target:/app nginx
```

Volumes dans le Dockerfile

- L'instruction VOLUME crée un point de montage
- Peut spécifier des arguments JSON array ou string
- Impossible de faire correspondre les volumes aux répertoires d'accueil
- Les volumes sont initialisés lors de l'exécution du conteneur

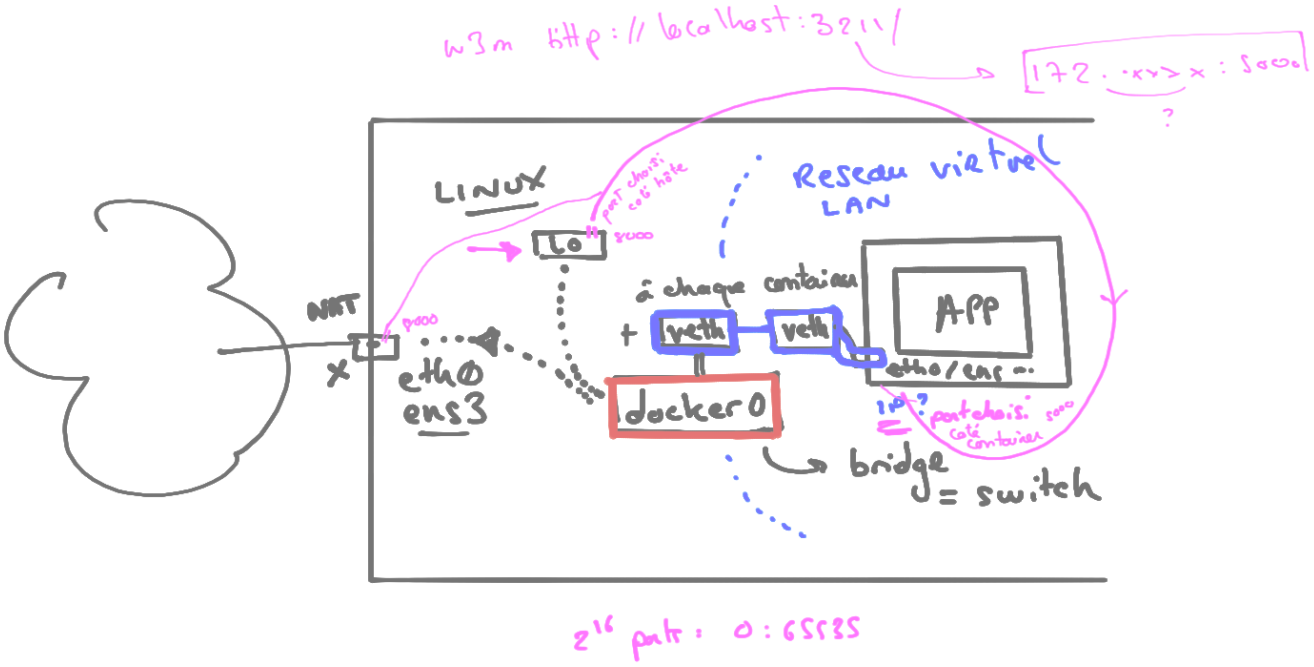
```
#String example
VOLUME /myvol
#String example with multiple volumes
VOLUME /www/website1.com /www/website2.com
#JSON example
VOLUME ["myvol", "myvol2"]
```

Tmpfs

Références

- https://docs.docker.com/storage/tmpfs/

Réseaux



- Par défaut, quand un conteneur est créé par `docker create` ou `docker run`, il ne publie aucun de ses ports vers le monde extérieur.

Théorie

- Les différents types de réseau
- Séparation des réseaux
- Support des VLANs
- Publication de ports réseau
- Principes du reverse proxy

Exemple de réseaux isolés sous Docker

🚧 FIXME: Illustration pour réseaux isolés

Références

🚧 FIXME

Port Mapping

- Pour rendre un port disponible aux services extérieurs à Docker, ou aux conteneurs Docker qui ne sont pas connectés au réseau du conteneur, utilisez l'indicateur --publish ou -p.
- Cela crée une règle de pare-feu qui mappe un port de conteneur à un port sur l'hôte Docker vers le monde extérieur.
- Rappel : les conteneurs ont leur propre réseau et leur propre adresse IP.
- Mettez en correspondance les ports exposés du conteneur avec les ports de la machine hôte.
- Les ports peuvent être mappés manuellement ou automatiquement.
- Utilise les paramètres -p et -P dans `docker run`.

Mappage du port 80 du conteneur vers le port 8080 de l'hôte :

```
$ docker run -d -p 8080:80 nginx:1.9.4
```

Voici quelques exemples.

| Valeur du paramètre | Description | | -p 8080:80 | Mapper le port TCP 80 dans le conteneur au port 8080 sur l'hôte Docker. | | -p 192.168.1.100:8080:80 | Mappage du port TCP 80 dans le conteneur au port 8080 sur l'hôte Docker pour les connexions à l'hôte IP 192.168.1.100.| | -p 8080:80/udp | Mappage du port UDP 80 dans le conteneur au port 8080 sur l'hôte Docker. | | -p 8080:80/tcp -p 8080:80/udp | Mappage du port TCP 80 du conteneur au port TCP 8080 de l'hôte Docker, et mappage du port UDP 80 du conteneur au port UDP 8080 de l'hôte Docker. |

Automapping ports

- Use the -P option in **docker run**
- Automatically maps exposed ports in the container to a port number in the host
- Host port numbers used go from 49153 to 65535
- Only work for ports defined in the EXPOSE instruction

Auto map ports exposed by the NGINX container to a port value on the host :

```
docker run -d -P nginx:1.9.4
```

EXPOSE instruction

- Configures which ports a container will listen on at runtime
- Ports still need to be mapped when container is executed

```
FROM ubuntu:14.04
RUN apt-get update
RUN apt-get install -y nginx

EXPOSE 80 443

CMD ["nginx","-g","daemon off;"]
```

Lien entre containers (obsolete)

Le lien est une méthode de communication entre les conteneurs qui leur permet de transférer des données de manière sécurisée de l'un à l'autre.

- Conteneurs source et destinataire
- Les conteneurs destinataires ont accès aux données des conteneurs sources.
- Les liens sont établis sur la base des noms des conteneurs

Créer un lien

1. Créez d'abord le conteneur source (ex: base de données)
2. Créer le conteneur destinataire et utiliser l'option --link (ex: application)

```
# Create the source container using the postgres
docker run -d --name database postgres

# Create the recipient container and link it
# Note: "db" is an alias added in /etc/hosts inside the container
docker run -d -P --name website --link database:db nginx
```

Utilisation du lien

- Les conteneurs peuvent communiquer entre eux sans avoir à exposer les ports à l'hôte.
- Essentiel pour l'architecture des applications de micro-services
- Exemple :
  - Conteneur avec Tomcat en cours d'exécution
  - Conteneur avec MySQL en cours d'exécution
  - L'application sur Tomcat doit se connecter à MySQL.

Réseaux définis par l'utilisateur

Types de réseaux

Aucun réseau (none)

- Désactive tous les réseaux
- Généralement utilisé en conjonction avec un pilote réseau personnalisé.

> **Warning**
>
> Le pilote `none` n'est pas disponible pour les services swarm

Réseau hôte (host)

- Pour les conteneurs autonomes,
- Supprimez l'isolation réseau entre le conteneur et l'hôte Docker (utilise directement le réseau de l'hôte)

Réseau par pont (bridge)

- Permet aux conteneurs connectés au même réseau de pont de communiquer
- Aassure l'isolation des conteneurs qui ne sont pas connectés à ce réseau de pont

Réseau IPVLAN

- donne aux utilisateurs un contrôle total sur l'adressage IPv4 et IPv6
- donnant le contrôle total du marquage VLAN de la couche 2 et du routage IPVLAN L3

Réseau MACVLAN

- Permet d'attribuer une adresse MAC à un conteneur, le faisant apparaître comme un périphérique physique sur votre réseau.
- Le démon Docker achemine le trafic vers les conteneurs en fonction de leur adresse MAC.
- C'est le meilleur choix lorsqu'il s'agit d'applications anciennes qui s'attendent à être directement connectées au réseau physique, plutôt que d'être acheminées par la pile réseau de l'hôte Docker.

Utilisation des réseaux

Réseau par défaut (systeme)

Create by default when you start Docker and newly-started containers connect to it unless otherwise specified.

- Containers can only access each other by IP addresses, unless you use the --link option

Réseaux définis par l'utilisateur

- Les réseau définis par l'utilisateur assurent une meilleure isolation et interopérabilité entre les applications conteneurisées.
- Les conteneurs peuvent être attachés et détachés à la volée.
- Les conteneurs connectés au même réseau de ponts définis par l'utilisateur exposent automatiquement tous les ports les uns aux autres, et aucun port au monde extérieur.
- Les réseaux définis par l'utilisateur fournissent une résolution DNS automatique entre les conteneurs connectés au même réseau
    - Les conteneurs peuvent se résoudre par nom, par CID ou par alias.

La commande docker network

Lister les réseau

```
$ docker network ls
NETWORK ID     NAME                      DRIVER    SCOPE
3653c0ea6aef   bridge                    bridge    local
73a1f0aa37ce   host                      host      local
dcd57b259f55   none                      null      local
```

Créer un réseau utilisateur en bridge

```
$ docker network create my-net
```

Afficher les options de `create`

```
$ docker network create --help
```

Supprimer un réseau utilisateur

```
`$ docker network rm my-net`
```

Connecter un nouveau container à un réseau utilisateur

```
`$ docker create --name my-nginx --network my-net --publish 8080:80 nginx:latest`
```

Connecter un container déjà démarré à un rseau utilisateur

```
`$ docker network connect my-net my-nginx`
```

Déconnecter un container d'un réseau utilisateur

```
$ docker network disconnect my-net my-nginx
```

Références

- https://docs.docker.com/network/
- https://docs.docker.com/network/macvlan/
- https://docs.docker.com/network/ipvlan/

Présentation

🚧 FIXME: illustration pour compose

Docker **Compose** est un outil permettant de créer et de gérer des applications multi-conteneurs.

- Les conteneurs sont tous définis dans un seul fichier appelé `docker-compose.yml`.
- Chaque conteneur exécute un composant/service particulier de votre application.
  - Par exemple : Front-end web / authentification de l'utilisateur / paiements / base de données.
- Les liens entre conteneurs sont définis (via des links (obsolete) ou networks)
- Compose va faire démarrer tous vos conteneurs en une seule commande.

Objectifs

- Définir les services qui composent votre application
- Chaque service contient des instructions pour construire et exécuter un conteneur.

En pratique ?

- Déployer plusieurs conteneurs simultanément
- Lier tous les conteneurs de l'application
- Mise en commun de stockage interconteneur
- Mise en commun de port TCP interconteneur
- Publication de ports réseau
- Utilisation de Docker Compose
- Création d'un fichier de configuration au format YAML

Références

- Docker Docs: Compose
- Github: docker/compose
- OpenClassrooms: Découvrez et installez Docker Compose
- Alsacreations: Docker compose

Syntaxe YAML

Vue d'ensemble

- Equivalence avec INI, XML ou JSON
- Structure par l'indentation
- Des dictionnaires, des tableaux et des scalaires

Note: Attention aux valeurs "true" / "false" / "yes" / "no" ou numériques, qui sont interprétées par YAML avant docker-compose.

Clés, valeurs et commentaires

```
---
# ceci est un commentaire
cléA: valeur
cléA: valeur
```

Tableaux

Forme étendue

Les éléments sont au même niveau d'indentation et commençent par un tiret et un espace :

```
---
fruits:
  - Apple
  - Orange
  - Pear
```

Forme abbrégée

On utilise des crochets et les éléments sont séparés par des virgules :

```
fruits: ['Apple', 'Orange', 'Pear']
```

Dictionnaires

Structures clé/valeur

Forme étendue

```yaml
---
# An employee record
martin:
  name: Martin Dean
  job: Developer
```

Forme abbrégée

```yaml
---
martin: {name: "Martin Dean", job: "Developer"}
```

Tableaux de tables de hachage

```yaml
---
# Employee records
-  martin:
    name: Martin D'vloper
    job: Developer
    skills:
      - python
      - perl
      - pascal
-  tabitha:
    name: Tabitha Bitumen
    job: Developer
    skills:
      - lisp
      - fortran
      - erlang
```

Replis de lignes

En respectant le format

```yaml
include_newlines: |
  exactly as you see
```

```
  will appear these three
  lines of poetry
```

En pliant les lignes

```
fold_newlines: >
  this is really a
  single line of text
  despite appearances
```

Note

- l'indentation est supprimée dans les deux cas
- dans le cas des lignes pliées, il faut sauter deux lignes pour créer un retour chariot

Attention : guillemets et typage automatique

Le contenus suivants sont considérées comme des booléens :

```
create_key: yes
needs_agent: no
knows_oop: True
likes_emacs: TRUE
uses_cvs: false
```

Références

- YAML.org: The Official YAML Web Site
- Wikipedia: YAML
- Ansible : YAML Syntax

Structure de la configuration

Vue d'ensemble

```yaml
---
version: '3'

services:
  # ...

networks:
  # ...

volumes:
  # ...
```

> **Note**
>
> Il existe d'autres sections, mais elles sont moins importantes que celles-ci pour le moment :-)

Numéro de version

- Indique à l'outil Docker Compose la version de la syntaxe utilisée dans ce fichier.
- Selon le numéro indiqué, l'outil active sa compatibilité avec les mot clés spécifiques à ladite version.

Section "services"

- Déclaration des services (~containers) et leur configuration.
  - Environnement
  - Utilisation des réseaux
  - Utilisation des volumes
  - etc.

Section "réseaux"

- Déclare les réseaux à créer et/ou rendre disponible à ce projet

Section "volumes"

- Déclare les volumes à créer et/ou rendre disponible à ce projet

Exemple

```yaml
---
version: '3'

services:
  javaclient:
    build: .
    image: glenux/javaclient:latest
    command: java HelloWorld
    networks:
      - app_net

  redis:
    image: redis
    networks:
      - app_net

volumes: {}

networks:
  app_net:
```

Références

- Docker Docs: Overview od Docker Compose
- OpenClassroom: Découvrez et installez Docker Compose
- Xavki: Docker compose

Utilisation

Démarrer l'application

- Utiliser `docker-compose up`
- La commande Up va
  - Construire l'image pour chaque service (si elle n'existe pas)
  - Créer et démarrer les conteneurs

> **Warning**
>
> Penser à utiliser `docker-compose up --build` pour reconstruire l'image

Arreter l'application

- Utiliser `docker-compose stop` - arret des services
- Utiliser `docker-compose kill` - arret des services (violent)
- Utiliser `docker-compose down` - suppression des services & réseaux, etc.

> **Question**
>
> Quelle différence entre `docker-compose stop` et `docker-compose kill` ?

Redimensioner l'application

```
docker-compose scale ...
```

Références

🚧

Section "services"

Build et/ou image

- **Build** définit le chemin vers le Dockerfile qui sera utilisé pour construire l'image.
- Le conteneur sera exécuté en utilisant l'image construite
- **Image** définit l'image qui sera utilisée pour exécuter le conteneur.
- Tous les services doivent avoir une instruction de construction ou d'image.

```
---
version: '3'
services:
 javaclient:
   build: . # Build image using Dockerfile in current directory
 redis:
   image: redis # Use the latest redis image from DockerHub
```

Volumes

🚧

Environnement

🚧

Ports

🚧

Réseaux

🚧

Links

- Même concept que la liaison de conteneur
- Spécifiez `<nom du service>:<alias>`.

- Si aucun alias n'est spécifié, le nom du service sera utilisé comme alias.
- Crée une entrée pour l'alias dans le fichier /etc/hosts du conteneur.

```
---
version: '3'
services:
 javaclient:
   build: .
   command: java HelloWorld
   links:
   - redis
 redis:
   image: redis
```

Références

🚧

Section "volumes"

🚧

Références

🚧

Section "networks"

🚧

Références

🚧

Fichier d'environnement

Objectif

- Adapter le `docker-config.yml` sans avoir à le modifier.
  - Ex: parametres différents en dev, test, prod

Fichiers .env

```
SOME_VARIABLE=some_content
```

N.B: attention aux espaces et aux guillemets !

Injection de variables

```
value: "${SOME_VARIABLE}"
```

Valeurs par défaut

```
value: "${SOME_VARIABLE:-default_value}"
```

Vérifier la configuration

```
docker-compose config
```

Notes

=> variables d'environnement / priorité (+ pb des credentials dans le YML + injection de variables du user)

=> blague du restart: always (et pb qui vont avec)

- différentier le docker-compose.yml de test et de prod ?
  - docker-compose.override.yml pour le dev

Références

🚧

Travaux pratiques

- Faire inter-agir plusieurs conteneurs
- Ex: Dolibarr en micro-services * MariaDB * Apache/PHP (voir chap. "Without a Dockerfile")

Concepts

References

- How to run your own docker registry with password, SSL and S3 backend :: Devopsian | Just another blog about Devops

Utilisation (rappel)

Récupération d'une image

```
$ docker pull IMAGE_ID
```

Envoi d'une image

```
$ docker tag IMAGE_ID IMAGE_ID
$ docker push IMAGE_ID
```

Trusted Registry

- la solution de stockage et de gestion d'images par Docker
- offre de Docker pour l'entreprise
- aussi appelé DTR et UCP (universal control plane) dans Docker Enterprise Edition
  - pour que le DTR fonctionne, l'UCP doit être installé
  - pour que l'UCP soit installé, vous avez besoin de Docker Enterprise Edition

> **Note**
>
> Une fois que vous avez installé Docker EE, vous pouvez obtenir une licence gratuite sur DockerHub.

Fonctionnalités

Gestion des images et des tâches

- peut être installé sur n'importe quelle plateforme
- permet de stocker les images Docker en toute sécurité dans le SI de l'organisation
- dispose d'une interface utilisateur de parcourir les images Docker
  - Permet d'examiner les événements du référentiel
  - Permet de voir quelles lignes de Dockerfile ont été utilisées pour produire l'image
  - Permet de voir une liste de tous les logiciels installés dans vos images (via scan de sécurité)

Disponibilité

- dispose de plusieurs répliques de conteneurs en cas de défaillance.

Efficacité

- capacité de nettoyer les manifestes non référencés
- met en cache les images pour accélérer leur extraction.

Contrôle d'accès intégré

- mécanismes d'authentification tels que RBAC et la synchronisation LDAP
- utilise la même authentification que UCP.

Scanner de sécurité

- l'analyse d'image est une fonction intégrée fournie par STR.

Signature d'image

- DTR a un "notaire" intégré, vous pouvez utiliser Docker Content Trust pour signer et vérifier les images.

Références

- Le Labo #12 2/2 | Docker Trusted Registry // A Geek's Lab
- Docker Registry vs Docker "Trusted" Registry - Stack Overflow
- An Introduction to the Docker Trusted Registry - DZone DevOps
- Content trust in Docker
- Notary

Portus

Références

- SUSE/Portus: Authorization service and frontend for Docker registry (v2)

Harbor

Références

- Harbor : la docker registry d’entreprise open source de VMware – part 1

Quay

Références

- https://github.com/quay/quay

Autres outils

- Portus
- DockerRegistryUi
- Trusted Registry
- Harbor
- Redhat Quay
- Gitlab
- Openshift Container Registry (OCR)
- ... et autres services

Références

- Objectif-Libre: Registres Docker à héberger
- https://github.com/atcol/docker-registry-ui

https://www.google.com/url?sa=t&source=web&rct=j&url=https://www.docker.com/sites/default/files/Docker%2520Trusted%2520Registry.pdf&ved=2ahUKEwjKgcKit-b0AhUG14UKHaPhDacQFnoECAcQAQ&usg=AOvVaw36gUJopyIBWseeE3gVzu5C

- Docker Pricing & Monthly Plan Details | Docker

Architecture

Architecture. Containers et volumes propres au DTR

Mise en place

- Docker Registry : construire et utiliser son propre hub.
- Exercice Construire et utiliser son propre hub

Mise en place d'un registre avec registry:2

```
$ docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

```
$ docker tag ubuntu:16.04 localhost:5000/my-ubuntu
$ docker push localhost:5000/my-ubuntu
```

```
$ docker image remove ubuntu:16.04
$ docker image remove localhost:5000/my-ubuntu
```

Travaux pratiques

- Déploiement d'un registry simple
- Utilisation de certificats
- Utilisation de l'authentification
- Configuration

Références

- Docker Docs: Deploy a registry server
- Docker Docs: Configuring a registry
- Docker Blog: How to Use Your Own Registry
- Digital Ocean

Pilotage d'un registry

Enjeux

- authentification
- certificats & chiffrement
- gestion des repository
  - gestion des accès (read/write)
  - protection des tags
- conservation
  - durée de conservation des images
  - quantité d'images à garder
  - taille sur le disque
- tracabilité
  - scan d'images
  - dockerfile
  - lien au git repository
  - build automatique
  - re-build automatique (en cas d'update d'image FROM)
- etc.

Démos

- Demo de DockerHub
- Demo de harbor

Références

Notes

- Principe d'un registry
- Création d'un registry
- Création d'un registry sécurisé
- Création d'un registry authentifié
- Configuration d'un client Docker et les certificats

Docker daemon

Rôle

Configuration des principales options

Références

- Everything You want to know about Docker Daemon | dockerlabs
- dockerd | Docker Documentation
- Configure and troubleshoot the Docker daemon | Docker Documentation

Option socket

... pour les accès en réseau

Variables d’environnement

- DOCKER_HOST
- DOCKER_TLS_VERIFY

Option storage-driver

Définition des formats de stockage des images

Gestion de nœuds

avec l’option -cluster-advertise

Travaux pratiques

- configuration des accès réseau
- de clusters Docker

Introduction

Concepts clés

🚧

Services clés

🚧

Références

🚧

Les limites de Docker

Des questions ouvertes

- Comment gérer et planifier le cycle de vie des containers ?
- Comment monter en charge ?
  - ex: redimensionner (rescale) l'infrastructure automatiquement ? (ex: php-fpm)
- Comment gérer tous ces containers
  - ex: faire communiquer tout ce beau monde si j'ai un conteneur par processus ?
  - ex: configurer les reverse proxy sait où envoyer les requêtes
- Comment faire la maintenance ?
  - gérer les pannes ?
  - comment mettre à jour mon application ?
- Comment gérer les task queue / les workers ?

Notion d'orchestration (voir wikipedia)

- Arrangement automatisé
- Coordination automatisée
- Gestion automatisée
- De SI et middleware et service

L'écosystème docker

En construction et très changeant (pire que l'écosystème JS ?)

- Docker Compose
- Docker Swarm
- Docker Universal Control Plane (UCP)
- Docker Trusted Registry
- Apache Mesos
- Kubernetes
- Rancher
- etc.

Quel orchestrateur choisir ?

- Choix difficile il y a encore 2 ans
  - de très nombreuses solutions
  - certains FLOSS, d'autres non
- Qui va survivre jusqu'à l'année prochaine ?
- Docker vient d'annoncer que Kubernetes était désormais géré nativement en plus de son orchestrateur maison (Swarm)
- pas une bonne nouvelle pour Swarm ...
- une bonne nouvelle pour la standardisation de Kubernetes ?

Docker Compose

Enjeux et contraintes

- Qu'appelle-t-on la production ? Quel niveau de service souhaite-t-on ?
  - Pour mettre simplement en ligne , Docker Compose est ok
  - Pour un service en haute-disponibilité (sur plusieurs serveurs)
    - soit utiliser un autre orchestrateur
    - soit gérer vous-même le balancing et le heartbeat entre vos serveurs
- Pas de gestion multi-host
- Pas de scaling automatique

Méthodologie

- Supprimer toute liaison de volume pour le code de l'application, afin que le code reste à l'intérieur du conteneur et ne puisse pas être modifié de l'extérieur.
- Mettre en écoute sur un port différent sur l'hôte
  - Et utiliser un reverse proxy pour le port public (ex: Caddy)
- Définition de variables d'environnement différentes
  - Par exemple : réduire la verbosité de la journalisation, spécifier les paramètres de services externes, des informations d'authentification
  - Utilisation d'un fichier `.env`
  - Injection de variables sous la forme `${...}` dans le `docker-compose`
- Spécifier une politique de redémarrage telle que `restart : always` (redémarrer : toujours) pour éviter les temps d'arrêt.
- Ajouter des services supplémentaires, comme un agrégateur de journaux.

Utilisation d'un fichier d'environnement

🚧 Exemple de `docker-compose.yml`

🚧 Fichier `.env`

Utilisation d'un second docker-compose.yml pour la production

- On peut définir un fichier Compose supplémentaire (ex: production.yml)
- Ce fichier de configuration doit uniquement inclure **les modifications** que vous souhaitez apporter au fichier Compose d'origine
- Le fichier Compose supplémentaire peut être appliqué par-dessus le fichier original docker-compose.yml pour créer une nouvelle configuration.

Note

Si vous avez besoin d'un second fichier de configuration, dites à Compose de l'utiliser avec l'option -f :

```
docker-compose -f docker-compose.yml -f production.yml up -d
```

Spécification de profils sur les services

Définition du parametre profiles dans le `docker-compose.yml`

```yaml
version: "3.9"
services:
  frontend:
    image: frontend
    profiles: ["frontend"]

  phpmyadmin:
    image: phpmyadmin
    depends_on:
      - db
    profiles:
      - debug

  backend:
    image: backend

  db:
    image: mysql
```

Utilisation du profil debug, à la demande:

```
$ docker-compose --profile debug up
$ COMPOSE_PROFILES=debug docker-compose up
```

Déploiement

- Penser à re-construire les images dont le code a changé
- Redéployer uniquement le service concerné, sans détruire/recréer les autres

```
docker-compose build web
docker-compose up --no-deps -d web
```

Exécution de Compose sur serveur distant

- Définir les variables d'environnement `DOCKER_HOST`, `DOCKER_TLS_VERIFY` et `DOCKER_CERT_PATH` de manière appropriée.
- Ensuite toutes les commandes de docker-compose fonctionnent sans autre configuration.

Références

- Docker Docs: Use Compose in production
- "Is Docker-Compose Suited For Production?", by Vladislav Supalov
- Docker Docs: Using profiles with Compose

Heroku / Dokku

Enjeux et contraintes

- Pas de support pour docker-compose
- Des plugins pour accompagner automatiquement vos services (postgres, maria, letsencrypt, etc.)

Principe

- Une solution "tout intégrée"
  - Un dépot git
  - Un build + déploiement + test lors du push

Utilisation

🚧 demonstration

TP

- Installation de dokku sur une VM.
- Push et déploiement automatisé d'un projet

Docker swarm

Présentation

- Un Docker Engine "virtuel"
  - Publie l'API standard de Docker pour le Docker Engine
  - Mais utilise plusieurs moteurs Docker (en arriere plan)
- Extrêmement facile à mettre en œuvre

En pratique

- Créer un cluster
- Ajouter des nœuds à un groupe
- Démarrer l'essaim

```
$ swarm create
```

```
$ swarm join --add=<node_ip> token://<token>
```

```
$ swarm manage --addr=<swarm_ip> token://<token>
```

Références

- Docker Docs: Getting started with swarm mode

Kubernetes

Présentation

En bref

- Des manifests en YAML
- La commande `kubectl`

Références

🚧

Vue d'ensemble

Architecture

🚧

Fonctionnement

🚧

References

Installation de Swarm

Paramétrage du master

Advertise address

`docker swarm init --advertise-addr <MANAGER-IP>`

Options supplémentaires

Récupération des tokens

`docker swarm join-token worker`

`docker swarm join-token manager`

Note: Selon le TOKEN, l'effet sera différent

Rotation des secrets

`docker swarm join-token  --rotate worker`

Ajout de noeuds

Worker

`docker swarm worker-token worker`

Manager

`docker swarm manager-token worker`

ou alors

`docker swarm worker-token worker` `docker node upate --role manager nodename`

Références

- https://docs.docker.com/engine/swarm/swarm-mode/
- https://www.grottedubarbu.fr/introduction-docker-swarm/

Gestion des nodes

Opérations

Liste

`docker node ls`

Inspection

`docker node inspect self --pretty`

Mise à jour

`docker node update --availability drain node-1`

Options des noeuds

Disponibilité

Meta-données

Promotion

Références

- Docker Docs: Manage Nodes in a Swarm

Déploiement d'un service

Références

- https://docs.docker.com/engine/swarm/services/

Fonctionnement des services et tâches

Services, tasks, and containers

Tasks and scheduling

Replicated and global services

Références

- https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/
- https://docs.docker.com/engine/swarm/how-swarm-mode-works/swarm-task-states/

Redimensionnement et répartition de charge

Network

- https://docs.docker.com/network/
- https://docs.docker.com/network/network-tutorial-overlay/
- https://docs.docker.com/network/network-tutorial-standalone/
- https://docs.docker.com/network/network-tutorial-host/
- https://docs.docker.com/network/network-tutorial-overlay/
- https://docs.docker.com/network/network-tutorial-macvlan/

Extending Docker

Network

Volumes

Références

- https://docs.docker.com/engine/extend/
- https://github.com/rancher/convoy

Stacks

Références

- https://docs.docker.com/engine/reference/commandline/stack_deploy/
- https://www.javainuse.com/devOps/docker/docker-swarm2
- https://buildvirtual.net/how-to-use-docker-stack-to-deploy-docker-containers/

Config et Secrets

Références

- https://www.grottedubarbu.fr/docker-secrets/
- https://www.grottedubarbu.fr/introduction-vault-hashicorp/

Contextes

Gestion

🚧

Utilisation

Avec docker

🚧

Avec docker-compose

Nécessite une version > 1.23

🚧

Références

- https://birthday.play-with-docker.com/context/
- https://www.grottedubarbu.fr/docker-context/
- https://www.grottedubarbu.fr/docker-compose-hote-distant/
- https://www.docker.com/blog/how-to-deploy-on-remote-docker-hosts-with-docker-compose/

Plan de controle

Outils usuels

- Docker Universal Control Plane (UCP)
- Crane

References

- Docker Technical Brief: Universal Control Plane
- Github: Dataman-Cloud/crane
- https://www.shurenyun.com/product-crane.html

Démarrage des containers

Ordre de démarrage

- impossible de prévoir l'ordre de démarrage des service liés entre eux
- aussi bien dans Docker Compose que dans les orchestrateurs
  - l'option `depend_on` (Docker Compose) garantit seulement l'ordre de démarrage des containers, pas la disponibilité du service lui-même
  - et pas de démarrage séquentiel (évidemment)

Init containers

Voir https://stackoverflow.com/questions/70322031/does-docker-compose-support-init-container

```
---
x-common-env: &cenv
    MYSQL_ROOT_PASSWORD: totopipobingo

services:
    db:
        image: mysql:8.0
        command: --default-authentication-plugin=mysql_native_password
        environment:
            <<: *cenv
    init-db:
        image: mysql:8.0
        command: /initproject.sh
        environment:
            <<: *cenv
        volumes:
            - ./initproject.sh:/initproject.sh
        depends_on:
            db:
                condition: service_started
    my_app:
        build:
            context: ./php
        environment:
            <<: *cenv
        volumes:
            - ./index.php:/var/www/html/index.php
        ports:
            - 9999:80
        depends_on:
            init-db:
                condition: service_completed_successfully
```

Instruction Healthcheck (Dockerfile)

Le principe

Mecanisme permettant à Docker comment tester un container pour vérifier qu'il fonctionne toujours correctement

> The HEALTHCHECK instruction tells Docker how to test a container to check that it is still working. This can detect cases such as a web server that is stuck in an infinite loop and unable to handle new connections, even though the server process is still running.

Fonctionnement

Il est possible de déclarer un HEALTHCHECK de deux façons :

- Dans votre fichier Dockerfile avant de build votre image,
- Lors de la déclaration du service dans le fichier docker-compose.

Syntaxe et options

```
HEALTHCHECK [options] CMD <command>:
```

- `--interval=<interval>` : interval entre deux vérifications (30 secondes par défaut)
- `--timeout=<time length>` : la durée maximum de la commande de vérification (30 sefondes par défaut)
- `--start-period=<time length>` : le temps d'attente avant de lancer la premiere vérification (0 secondes par défaut)
- `--retries=<number>` : nombre d'échecs consécutifs de la commande pour déclarer un container en mauvaise santé

ou bien `HEALTHCHECK NONE` pour désactiver la vérification.

Exemple de Dockerfile:

```
FROM nginx:1.13

# le HEALTHCHECK contient un CMD sur la même ligne que lui
HEALTHCHECK --interval=30s --timeout=3s \
```

```
  CMD curl -f http://localhost/ || exit 1
EXPOSE 80
```

Réferences

- Docker Docs: HealthCheck
- https://www.grottedubarbu.fr/docker-healthcheck/
- ScoutAPM: How to Use Docker’s Health Check Command
- Lab #14: Create a Docker Image with HEALTHCHECK instruction
- "How to Add a Health Check to Your Docker Container", by Tyler Jones

Analyse des points à risques

le noyau, le service Docker, les containers, … et des types de dangers : déni de service, accès réseau non autorisés, …

Références

- "Attacking & Auditing Docker Containers Using Open Source tools", by Madhu Akula
  - https://github.com/appsecco/defcon-26-workshop-attacking-and-auditing-docker-containers
  - https://www.youtube.com/watch?v=ru7GicI5iyI
- https://alexander.holbreich.org/docker-components-explained/
- https://www.sans.org/white-papers/37437/
- https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html
- https://github.com/OWASP/Docker-Security
- https://man7.org/linux/man-pages/man7/capabilities.7.html
- https://www.cyberark.com/resources/threat-research-blog/how-i-hacked-play-with-docker-and-remotely-ran-code-on-the-host
- https://www.cyberark.com/resources/threat-research-blog/the-route-to-root-container-escape-using-kernel-exploitation
- [](https://blog.trailofbits.com/2019/07/19/understanding-docker-container-escapes/
- CVE-2019-5736: Escape from Docker and Kubernetes containers to root on host
- Frichetten/CVE-2019-5736-PoC
- Github: singe/container-breakouts

Mécanismes de protection

- pile réseau propre à chaque container,
- limitations de ressources par les cgroups
- restrictions des droits d’accès sur les sockets
- politique de sécurité des containers
- namespaces & mapping des users

Sécurisation des clients

par des certificats Principe, et mise en oeuvre avec openssl

Reference

- https://www.grottedubarbu.fr/introduction-vault-hashicorp/
- https://www.grottedubarbu.fr/docker-touche-pas-groupe/

Travaux pratiques

- mise en évidence de failles de sécurité
- des bonnes pratiques à adopter

Fiabilité des images

- présentation de Content Trust pour signer les images
- outils d'audit

Exercices pratiques : activation de Content Trust, variable d’environnement DOCKER_CONTENT_TRUST

Création et déploiement d’images signées

Configuration réseau, sécurité et TLS

Gestion des logs

- Bien penser à utiliser STDOUT
- syslog ne sont pas présents par défaut dans les containers

Exemple avec CRON

```
##
## Cron
##
# Redirect log files to cron process file descriptors STDOUT & STDERR
rm -f /var/log/cron.log /var/log/cron.err.log
ln -sf /proc/1/fd/1 /var/log/cron.log
ln -sf /proc/1/fd/2 /var/log/cron.err.log
```

🚧

Références

- Top 5 Docker Logging Practices
- Docker Logging: Getting Started, Best Practices, and More

Audit

Syntaxe des Dockerfiles

- Sudneo/dockerfile-audit - Outil d'audit de conformité des fichiers Dockerfile
- [RedCoolBeans/dockerlint] - Outil de linting pour Dockerfiles
- Dockerlinter -
- https://github.com/opstree/OT-Dockerlinter - A linter designed in golang for checking Dockerfile best practices.
- https://github.com/buddy-works/dockerfile-linter
- https://hadolint.github.io/hadolint/ - Dockerfile linter, validate inline bash, written in Haskell

Struture des images

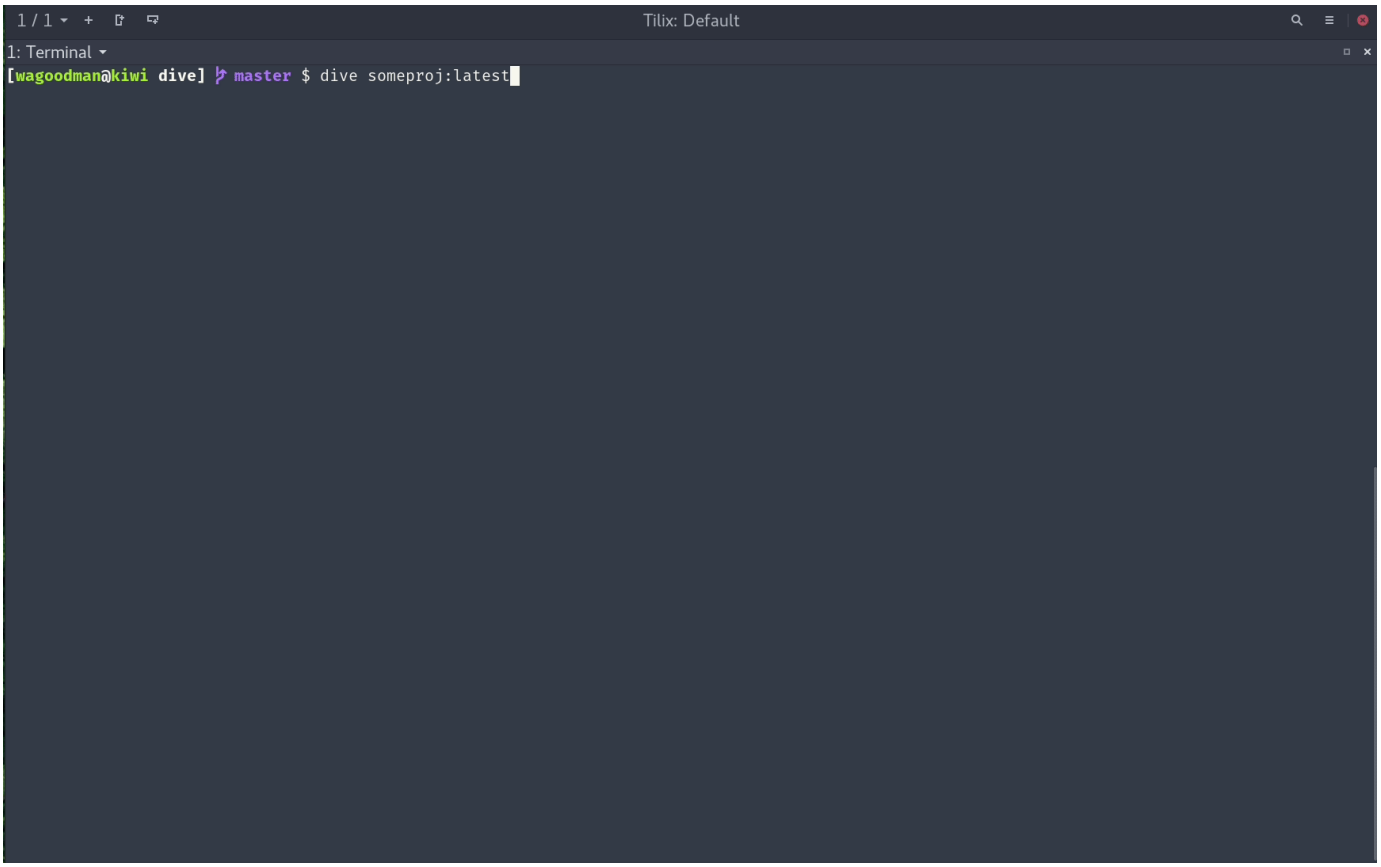
Il s'agit d'explorer une image Docker pour en voir :

- le contenu des couches,
- découvrir les moyens d'en réduire la taille

> **Note**
>
> La commande docker fournit déjà des moyens d'inspecter le contenu des images sans outils supplémentaires :
>
> - `docker save IMAGE_ID > archive.tar` suivi de l'extraction du contenu l'archive permet déjà d'explorer les couches d'une image docker sans avoir besoin d'outils spécifiques.
> - `docker history` montre l'historique des commandes qui ont permi la génération de l'image et la taille des couches qui en résultent.

Outils :

- `docker history` - Montre l'historique d'une image
- wagoodman/dive - Un outil pour explorer chaque couche dans une image docker



Sécurité

- Docker Bench...
- Drydock
- Blowhole

Références

- https://github.com/docker/docker-bench-security
- https://github.com/dockersecuritytools/batten
- https://github.com/zuBux/drydock
- https://github.com/Keramas/Blowhole
- https://github.com/Sudneo/dockerfile-audit
- https://github.com/iamabhishek-dubey/dockerlinter
- Fabricio Pautasso: How to Optimize Docker Images Using Dive

Outils de supervision

- Autoheal - Monitor and restart unhealthy docker containers automatically.
- Axibase Collector - Axibase Collector streams performance counters, configuration changes and lifecycle events from the Docker engine(s) into Axibase Time Series Database for roll-up dashboards and integration with upstream monitoring systems.
- cAdvisor - Analyzes resource usage and performance characteristics of running containers. Created by @Google
- Ctop
- Docker-Alertd - Monitor and send alerts based on docker container resource usage/statistics
- Dockerana: packaged version of Graphite and Grafana, specifically targeted at metrics from Docker
- Docker-Flow-Monitor - Reconfigures Prometheus when a new service is updated or deployed automatically by @docker-flow
- Docker-mon (Console-based Docker monitoring) by @icecrime
- Dockly
- DockProc - I/O monitoring for containers on processlevel.
- dockprom - Docker hosts and containers monitoring with Prometheus, Grafana, cAdvisor, NodeExporter and AlertManager by @stefanprodan
- Dozzle - Monitor container logs in real-time with a browser or mobile device. @amir20
- Dynatrace heavy_dollar_sign - Monitor containerized applications without installing agents or modifying your Run commands
- Glances - A cross-platform curses-based system monitoring tool written in Python by @nicolargo
- Grafana Docker Dashboard Template - A template for your Docker, Grafana and Prometheus stack @vegasbrianc
- InfluxDB, cAdvisor, Grafana - InfluxDB Time series DB in combination with Grafana and cAdvisor by @vegasbrianc
- LogJam - Logjam is a log forwarder designed to listen on a local port, receive log entries over UDP, and forward these messages on to a log collection server (such as logstash) by @gocardless
- Logspout - Log routing for Docker container logs by @gliderlabs
- monit-docker - Monitor docker containers resources usage or status and execute docker commands or inside containers. [@decryptus][decryptus]

- NexClipper - NexClipper is the container monitoring and performance management solution specialized in Docker, Apache Mesos, Marathon, DC/OS, Mesosphere, Kubernetes by @Nexclipper
- Out-of-the-box Host/Container Monitoring/Logging/Alerting Stack - Docker host and container monitoring, logging and alerting out of the box using cAdvisor, Prometheus, Grafana for monitoring, Elasticsearch, Kibana and Logstash for logging and elastalert and Alertmanager for alerting. Set up in 5 Minutes. Secure mode for production use with built-in Automated Nginx Reverse Proxy (jwilder's).
- Portainer
- Prometheus/Grafana
- Seagull (Friendly Web UI to monitor docker daemon.)
- SuperVisor CPM Frontend Service and Driver Service construction - A simple and accessible FOSS container performance monitoring service written in Python by @t0xic0der
- SwarmAlert - Monitors a Docker Swarm and sends Pushover alerts when it finds a container with no healthy service task running.
- Sysdig: An open source troubleshooting tool that provides a rich set of real-time, system-level information. It has container-specific features and is very useful in Docker environments. veggiemonk/awesome-docker
- Watchtower
- Zabbix Docker module - Zabbix module that provides discovery of running containers, CPU/ memory/blk IO/net container metrics. Systemd Docker and LXC execution driver is also supported. It's a dynamically linked shared object library, so its performance is (~10x) better, than any script solution.
- Zabbix Docker module: Zabbix module that provides discovery of running containers, CPU/ memory/blk IO/net container metrics. Systemd Docker and LXC execution driver is also supported. It's a dynamically linked shared object library, so its performance is (~10x) better, than any script solution.
- Zabbix Docker - Monitor containers automatically using zabbix LLD feature.

Références

- tobegit3hub/seagull: Friendly Web UI to manage and monitor docker
- dockerana/dockerana: Docker Monitoring with support for Grafana and Graphite
- icecrime/docker-mon: Console-based Docker monitoring
- Sysdig
- jangaraj/Zabbix-Docker-Monitoring: Docker/Kubernetes/Mesos/Marathon/Chronos/LXC/LXD/ Swarm container monitoring

- amir20/dozzle: Realtime log viewer for docker containers
- lirantal/dockly: Immersive terminal interface for managing docker containers and services
- portainer: Container Management Made Easy
- bcicen/ctop: Top-like interface for container metrics
- Docker Docs: Collect Docker metrics with Prometheus
- YoanDev: Superviser ses containers avec Prometheus/Grafana
- veggiemonk/awesome-docker