R4.08 – Virtualisation

Conteneurs et Docker

Département Informatique

IUT2, UGA

2024/2025

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

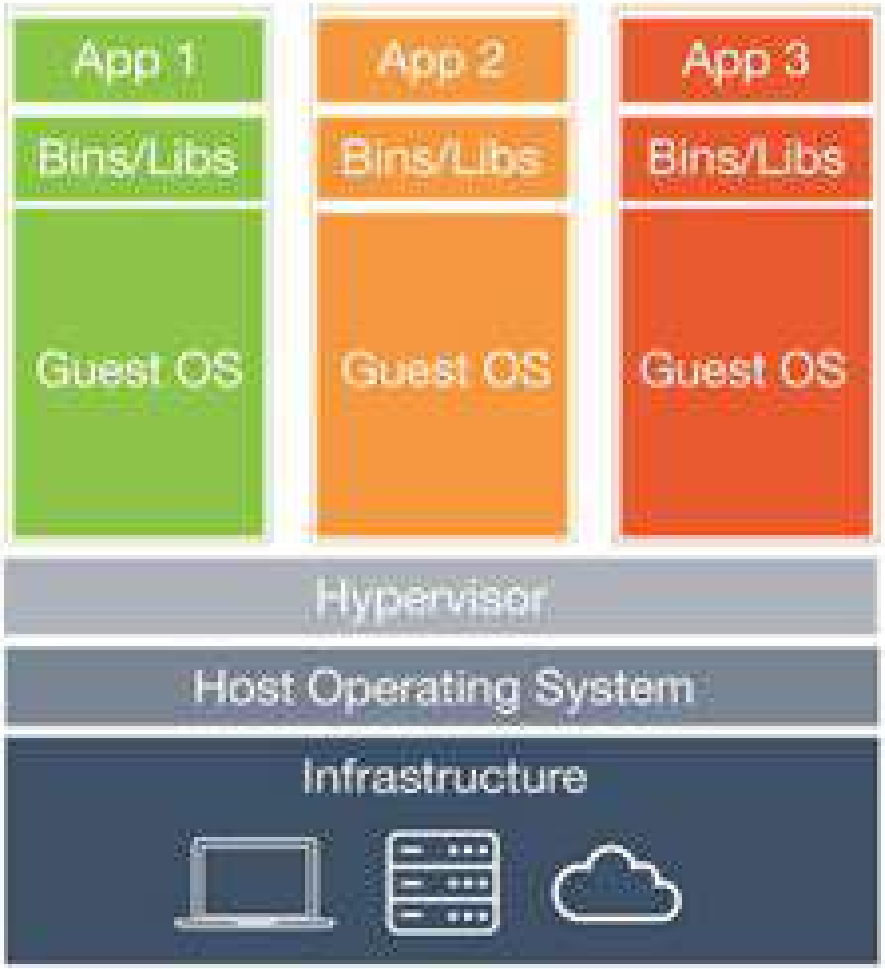
Plan du cours

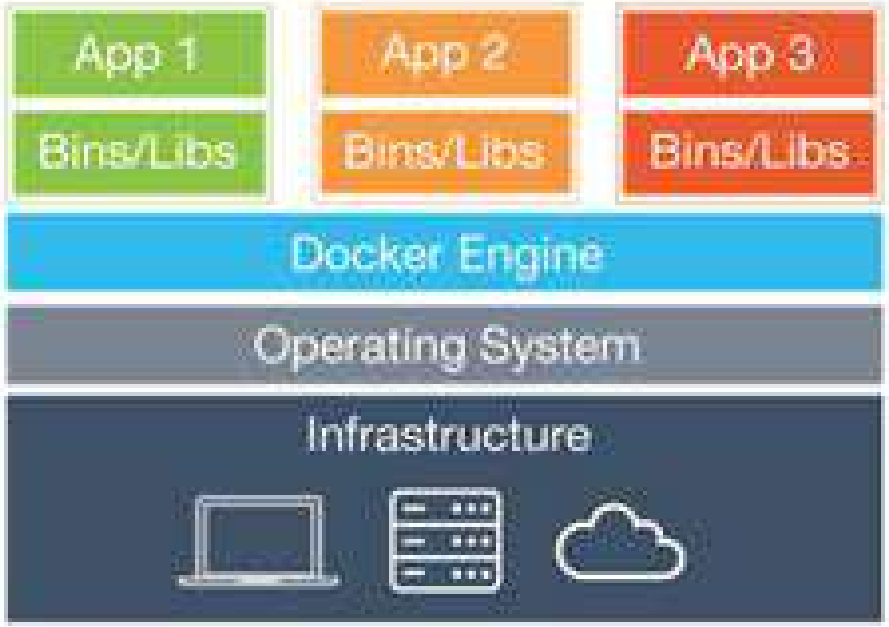
1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Définition d'un conteneur

- Processus ou ensemble de processus tournant de façon **isolée** sur un système hôte
- Le conteneur est l'environnement d'exécution qui contient tout ce qui est nécessaire pour faire tourner et isoler ce/ces processus

Architecture logicielle des VM et conteneurs





Source : NetApp

Comparaison entre VM et conteneurs

	Machine Virtuelle	**Conteneur**
Isolation	Complète (en principe)	Partielle
Sécurité	Plus sécurisé	Moins sécurisé
Système d'exploitation	Complet	Partagé
Ressources CPU et RAM	Allouées indépendamment	Partagées avec l'hôte et autres conteneurs
Stockage nécessaire	Plus grand	Plus petit
Gestion du stockage	Manuelle	Automatique
Temps de démarrage	Court	Très court

Utilité des conteneurs

- Facilité d'accès à tous les outils de **développement**
  - Langages (compilateurs, interpréteurs, ...)
  - Serveurs (Web, SGBD, ...)
  - ...
- Facilité pour le **déploiement** d'applications
  - Déploiement automatisable à 100%
  - Déploiement reproductible
    quel que soit l'environnement cible (OS/version, ...)
  - La fin du "ça marche chez moi"

Approche DevOps

- Tests d'un logiciel en mode **intégration continue**
- *Continuous Integration (CI)*
- Déploiement d'un logiciel en mode **livraison continue**
- *Continuous Delivery (CD)*
- Tests et déploiement se font dans le **même environnement logiciel** grâce à des conteneurs
- On parle alors de *CI/CD*
- Tout peut être automatisé

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Plan du cours

1 Introduction et rappels
2 Mécanismes système nécessaires aux conteneurs
- Besoins
- Mécanismes d'isolation
- Mécanisme de gestion des ressources

3 Principaux gestionnaires de conteneurs
4 Présentation de Docker
5 Utilisation de Docker
- Installation de Docker
- Catalogue d'images disponibles
- Gestion des images
- Gestion des conteneurs
- Variables d'environnement
- Création d'images

6 Pour aller plus loin avec Docker et les conteneurs
7 Résumé

Isolation des conteneurs

- Fichiers
  un processus dans un conteneur ne doit pas voir les fichiers en dehors de son conteneur
- Processus
  un processus dans un conteneur ne doit pas voir ni agir sur les autres processus (OS hôte et autres conteneurs)
- Réseau
  un conteneur doit être isolé au niveau réseau
  (ex : avoir sa propre adresse IP)
- Utilisateurs
  un conteneur peut avoir ses propres utilisateurs,
  distincts des utilisateurs de l'OS hôte

Gestion des ressources

- Types de ressources
  - RAM
  - CPU
  - débit des E/S disque
  - réseau
- Possibilité de
  - imposer des limites
  - définir des priorités
  - faire une comptabilité
    (pour faire payer à la consommation)

Plan du cours

1 Introduction et rappels
2 Mécanismes système nécessaires aux conteneurs
- Besoins
- Mécanismes d'isolation
- Mécanisme de gestion des ressources

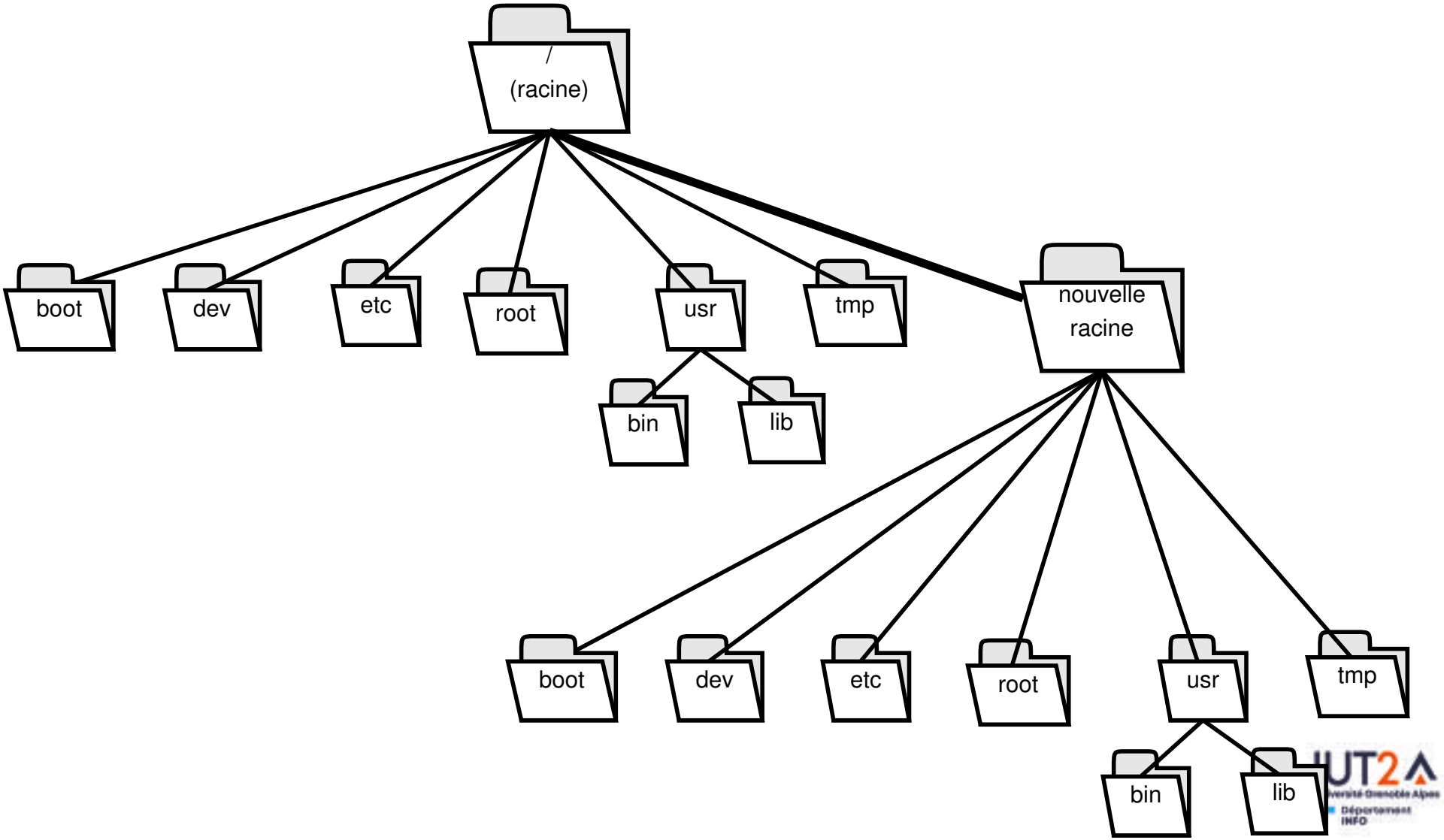
3 Principaux gestionnaires de conteneurs
4 Présentation de Docker
5 Utilisation de Docker
- Installation de Docker
- Catalogue d'images disponibles
- Gestion des images
- Gestion des conteneurs
- Variables d'environnement
- Création d'images

6 Pour aller plus loin avec Docker et les conteneurs
7 Résumé

Isolation des fichiers et `chroot()`

- `chroot()` est un appel système qui permet à un processus de s'isoler dans un répertoire
- Pour ce processus la racine du SGF est changée par le noyau Linux
- Il perd alors l'accès au reste du SGF
- Cette restriction est imposée par le noyau Linux
- La présence d'une mini-arborescence Linux est nécessaire au lancement d'un processus dans un *chroot*

Schéma d'un `chroot()`



Isolation des autres aspects du système

- Assurée par les *Linux namespaces*
- Il existe 8 types de *namespaces*
  - *Mount namespaces*
  - *PID namespaces*
  - *Network namespaces*
  - *User namespaces*
  - ...
- Appels système pour créer un *namespace*
  `unshare()`, `clone()`, `setns()`
- Utilitaire pour créer un *namespace* : `unshare`

User namespaces

- Dans un *namespace*, les utilisateurs peuvent être différents de ceux de l'OS hôte, y compris `root`
- Isolation des privilèges dans le *namespace*

```
bonnaudl@vougeot:~$ unshare --map-root-user
root@vougeot:~# whoami
root
```

- Mais ce n'est pas "pour de vrai"
- C'est simulé dans le *namespace*, et ne donne pas de privilèges supplémentaires sur le système hôte

Mount namespaces

- Version plus moderne et générale de `chroot()`
- Chaque processus peut avoir ses propres montages
- Très utile pour les SGF virtuels `/proc` et `/sys`
- Indispensable pour fabriquer la sous-arborescence d'un conteneur

```
# mount
/dev/nvme0n1p1 on /boot/efi type vfat
/dev/nvme0n1p2 on / type ext4
proc on /proc type proc (rw,nosuid,nodev,noexec,relatime)
sysfs on /sys type sysfs (rw,nosuid,nodev,noexec,relatime)
...
```

PID namespaces

- Dans un *namespace*, un processus ne voit que la sous arborescence des processus du *namespace*
- Similaire à `chroot()`, mais pour l'arborescence des processus
- Dans un *namespace*, le 1er processus a le numéro 1

```
bonnaudl@vougeot:~$$ unshare -r --pid --fork --mount-proc
root@vougeot:~# ps aux
USER    PID %CPU %MEM    VSZ    RSS TTY      STAT  COMMAND
root      1  1.5  0.0  13644   7872 pts/32   S     -bash
root     12  0.0  0.0  12528   3676 pts/32   R+    ps aux
```

Network namespaces

- Virtualisation des fonctionnalités réseau
- Initialement, un processus n'a accès qu'à une interface *loopback*
- Un *namespace* peut être complètement isolé au niveau réseau et/ou avoir en propre
  - réseau IP
  - table de routage
  - processus serveur attaché à un port TCP/UDP
  - pare-feu
  - ...

```
bonnaudl@vougeot:~$$ unshare -r --net
root@vougeot:~# ip addr
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group defaul
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

Plan du cours

1 Introduction et rappels
2 Mécanismes système nécessaires aux conteneurs
- Besoins
- Mécanismes d'isolation
- Mécanisme de gestion des ressources

3 Principaux gestionnaires de conteneurs
4 Présentation de Docker
5 Utilisation de Docker
- Installation de Docker
- Catalogue d'images disponibles
- Gestion des images
- Gestion des conteneurs
- Variables d'environnement
- Création d'images

6 Pour aller plus loin avec Docker et les conteneurs
7 Résumé

Mécanisme de gestion des ressources : *control groups*

- Un *control group* est un groupe de processus pour lesquels on peut
  - imposer des limites
  - définir des priorités
  - faire une comptabilité (pour faire payer à la consommation)
- Utilitaires de `cgroup-tools`
  - `cgcreate`, `cgdelete`
  - `cgget`, `cgset`
  - `cgexec`, ...
- Utilitaires de `systemd`
  - `systemd-cgls`
  - `systemd-cgtop`

Notes sur la sécurité

- Certains appels systèmes étaient initialement réservés à `root` (∼600 appels systèmes au total)
- Grâce aux *namespaces*, ils ont ensuite été autorisés pour n'importe quel utilisateur non `root`
- Cela a énormément augmenté la **surface d'attaque** du noyau Linux
- De nombreuses failles de sécurité du noyau Linux ont été exposées aux utilisateurs non `root`
- Elles étaient de simples bugs tant que ces fonctionnalités étaient réservées à `root`
- Ces failles ont été corrigées au fur et à mesure, mais il est encore très probable qu'on en trouve encore dans le noyau Linux
- Il existe aussi un gain en sécurité offert par ces appels système
- Ils permettent de créer des *sandboxes* pour certains logiciels
  - navigateurs Web
  - snap et flatpak
  - ...

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Petit panorama historique

- 1982 : appel système `chroot()` dans systèmes Unix
- 2000 : appel système `jail()` dans FreeBSD
- 2001 : Linux-VServer (utilise un noyau Linux modifié)
- 2005 : OpenVZ (utilise un noyau Linux modifié)
- 2008 : LXC/LXD/Incus : noyau Linux **standard**
  (combinaison *namespaces* + *cgroups*)
- 2010 : `systemd-nspawn`
  utilisé à l'IUT2 pour maintenir l'image des stations Linux
- 2013 : Docker (nécessite des privilèges `root`)
- 2015 : Singularity/Apptainer (ne nécessite pas de privilèges `root`)
- 2018 : Podman : alternative à Docker
  (pas de démon, pas de privilèges `root` nécessaires)

Plan du cours

1 Introduction et rappels
2 Mécanismes système nécessaires aux conteneurs
- Besoins
- Mécanismes d'isolation
- Mécanisme de gestion des ressources

3 Principaux gestionnaires de conteneurs
4 Présentation de Docker
5 Utilisation de Docker
- Installation de Docker
- Catalogue d'images disponibles
- Gestion des images
- Gestion des conteneurs
- Variables d'environnement
- Création d'images

6 Pour aller plus loin avec Docker et les conteneurs
7 Résumé

Présentation du logiciel

- Docker est un gestionnaire de conteneurs
- Docker est un logiciel libre, écrit en Go
- Docker est un logiciel multi-plateformes
  - Linux
  - MacOS
  - Windows
- Sur les autres OS que Linux, Docker utilise en fait une VM dans laquelle tourne un noyau Linux
- Docker est très lié aux *namespaces* et aux *cgroups* de Linux

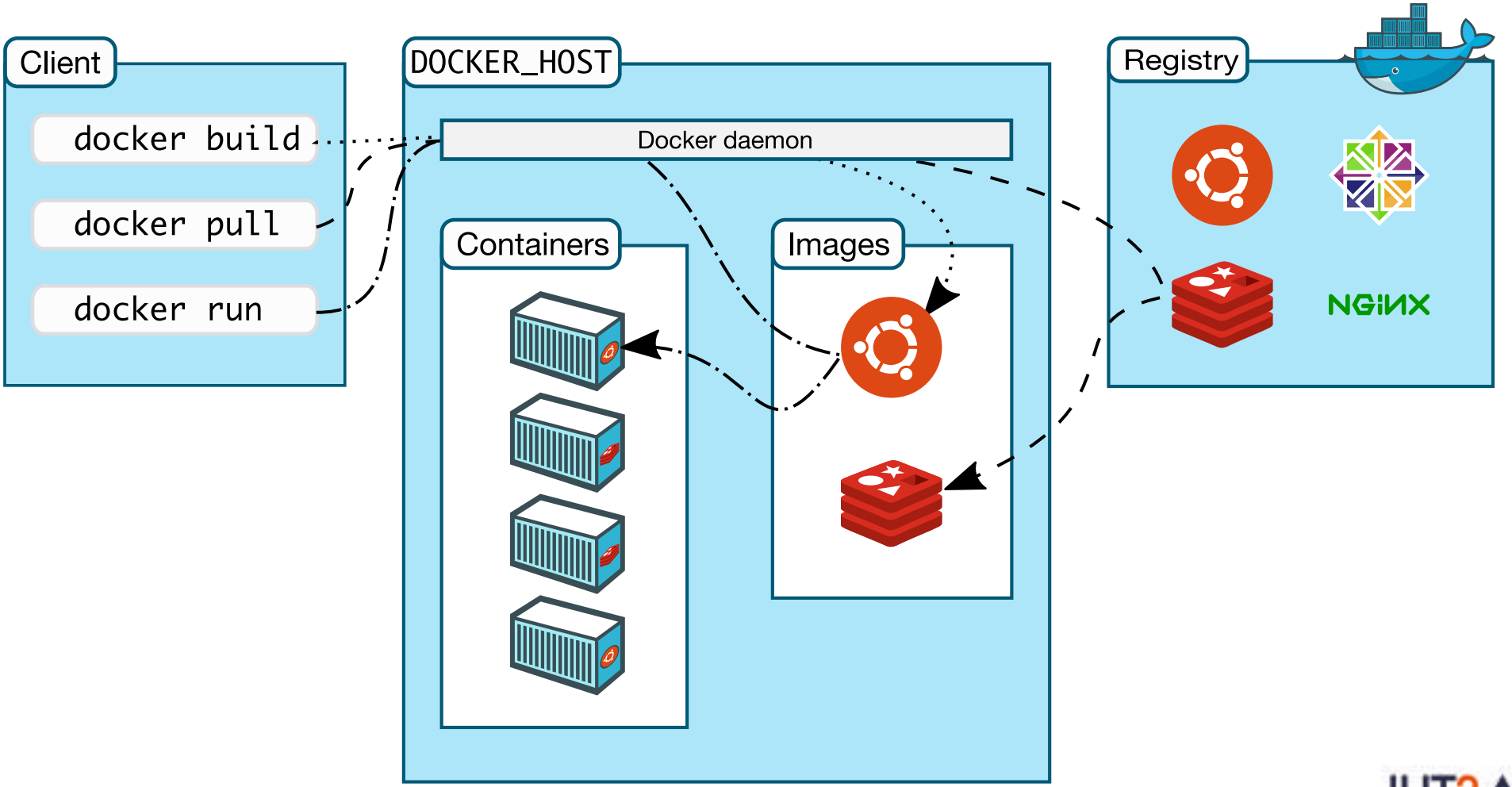
Concepts de base

- **Images**
  - Contiennent les fichiers nécessaires au fonctionnement des conteneurs
- **Conteneurs**
  - C'est une instance d'une image en train de s'exécuter
  - Un conteneur contient un ou plusieurs processus
  - Ils peuvent être créés, démarrés, arrêtés, supprimés
- **Volumes**
  - Servent à stocker des données que l'on veut garder de façon persistante
  - Exemple : BD
  - Détails plus loin

Logiciels et architecture générale

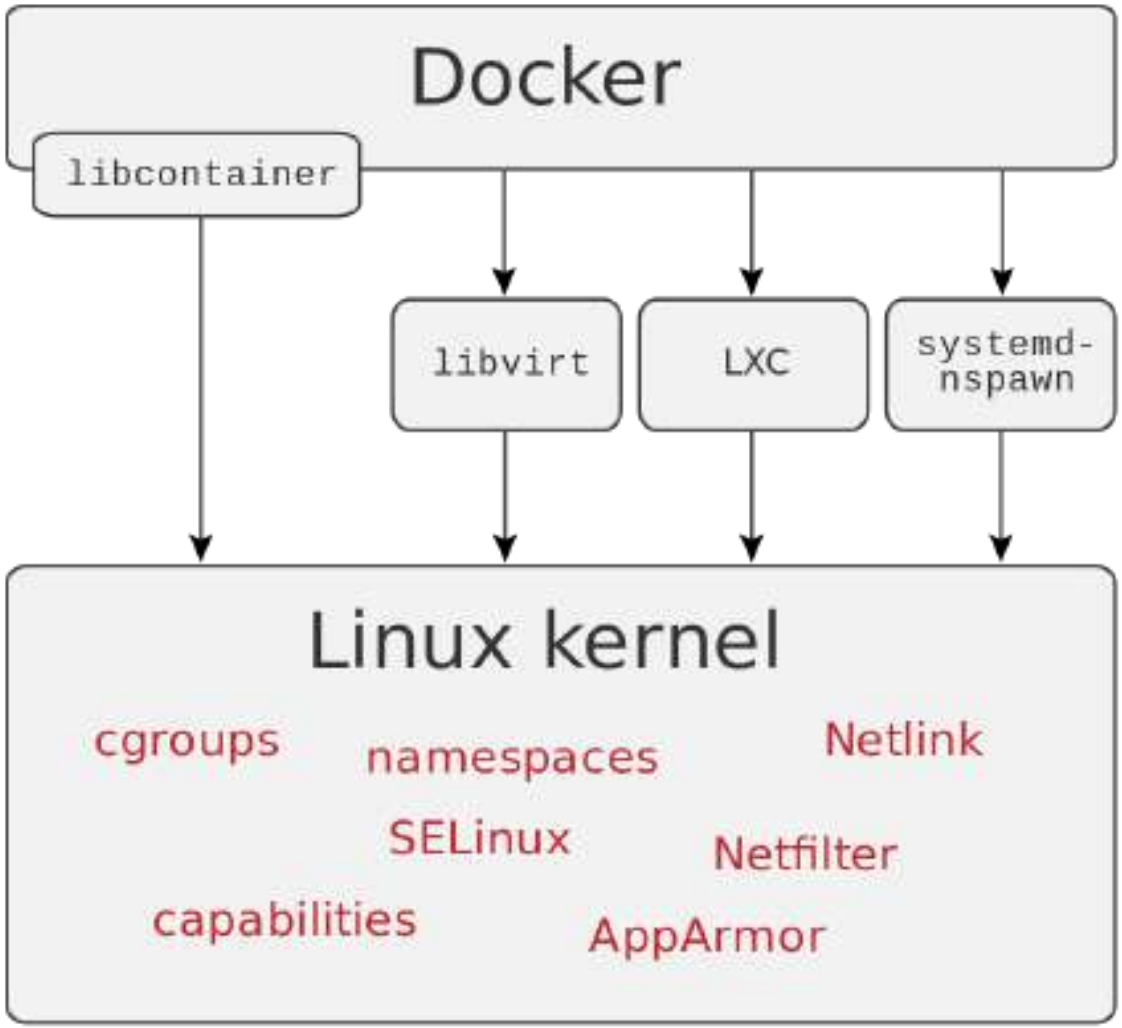
- Logiciels
  - Démons Docker : `dockerd`, `containerd`
  - Client Docker : `docker`
- Serveur(s) d'images *(Registry)*
  - Par défaut le *Docker Hub* est utilisé (`hub.docker.com`)
  - D'autres serveurs existent et peuvent être utilisés

Architecture générale de Docker



Source : Docker – Souvent, client et serveur sont sur la même machine

Architecture logicielle de Docker

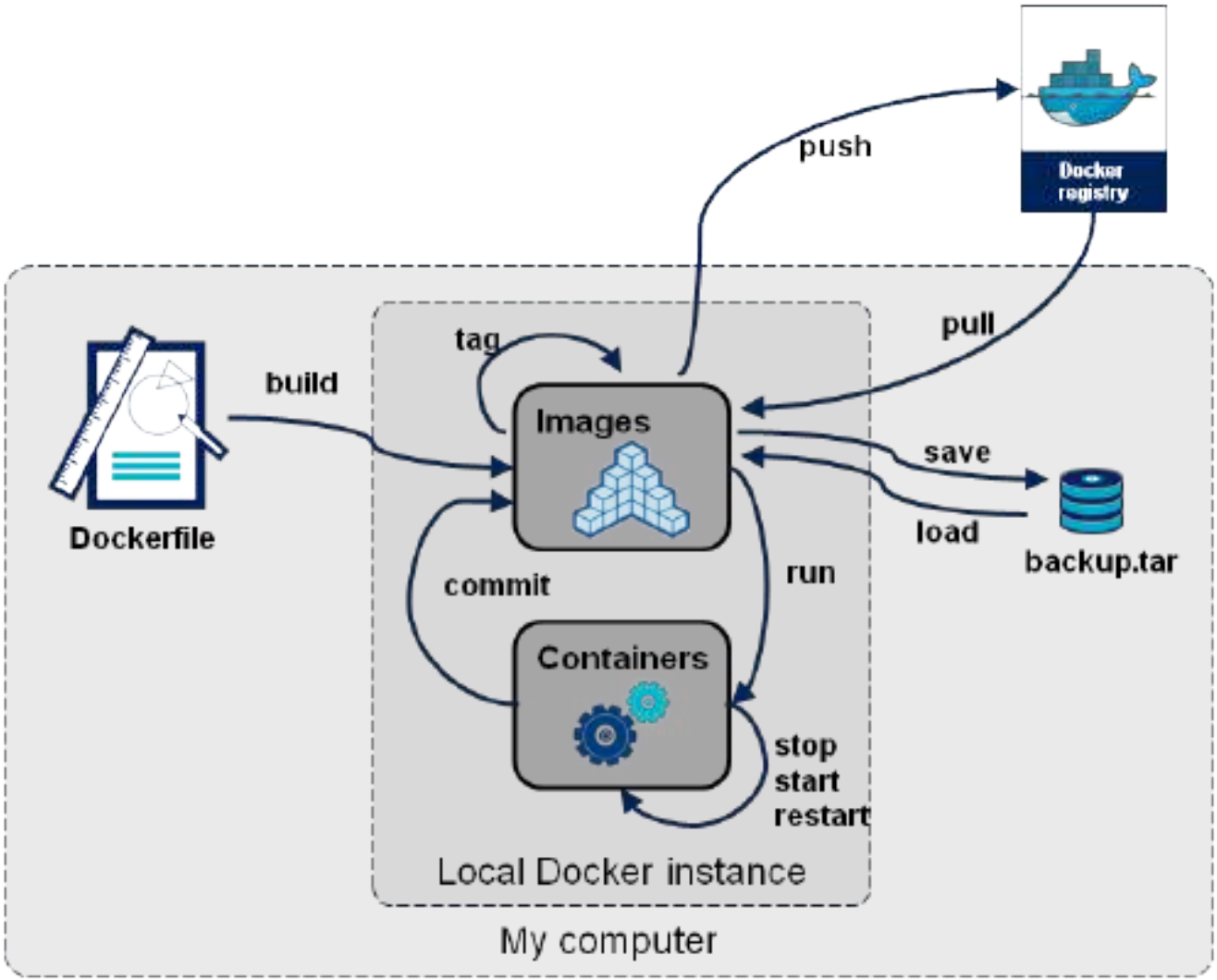


Source : Wikipedia

Relations entre images et conteneurs

- Plusieurs conteneurs peuvent utiliser la même image
- L'espace disque n'est utilisé qu'une seule fois
- L'image est montée automatiquement dans l'arborescence des fichiers des conteneurs tournant sur cette image

Travailler avec Docker



Source : octo.com

Images de Docker

- Les images sont stockées dans un format appelé OCI *(Open Container Initiative)*
- Les images sont immuables *(read-only)*
- Si on a besoin de modifier une image, on empile les modifications dans une couche qui se superpose à l'image de départ
- Les images sont souvent constituées de plusieurs couches
- On peut
  - utiliser des images standard
  - fabriquer ses propres images (avec des `Dockerfile`)
- Les images utilisées localement sont téléchargées et stockées dans un cache local
- On peut uploader ses images sur son propre serveur ou sur le *Docker Hub*

Persistance des données

- Il faut **faire attention** aux données écrites dans un conteneur
- Elles peuvent facilement être perdues !
- Si des processus écrivent dans une image, les écritures ne sont conservées que tant que le conteneur n'est pas détruit
- Le problème se pose quand on veut démarrer un conteneur sur une nouvelle version d'une image
- Attention en particulier aux SGBD
- Solutions : Volumes et *bind mounts*

Types d'images pour Docker

- Images de distributions Linux
  - Debian
  - Ubuntu
  - Fedora
  - Alpine
  - ...
- Images de logiciels (avec une distribution Linux sous-jacente)
  - httpd (Apache seul)
  - PHP CLI
  - PHP intégré dans Apache
  - PostgreSQL
  - ...
- Exemples : liste des images téléchargées

Intégrité des images et confiance dans les créateurs

- L'intégrité des images est vérifiée lors de leur téléchargement
- Empreinte SHA-256
- Il faut bien faire la différence entre
  - images **officielles**
  - images **non officielles**

Images et *tags*

- Chaque image a un *tag* (étiquette)
- Pour un nom d'image donné, on trouve souvent sur le Docker Hub plusieurs *tags*
- Le *tag* correspond à une version du logiciel contenu dans l'image
- Quand on récupère une image, c'est par défaut l'image avec le *tag* `latest` qui est téléchargée

Conteneurs et services

- Certains gestionnaires de conteneurs permettent de lancer plusieurs services dans un conteneur. Exemple :
  - `systemd` comme lanceur de services dans le conteneur
  - un serveur Apache
  - un serveur PosgreSQL
  - dans le **même conteneur**
- Docker incite à ne lancer qu'**un seul service par conteneur**
  - un conteneur pour les processus du serveur Apache
  - un conteneur pour les processus du serveur PosgreSQL

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Installation dans distribution Debian

- Package Docker disponible : `docker.io`
- Par défaut seul `root` est autorisé à utiliser Docker
- On peut autoriser d'autres utilisateurs
  en les ajoutant au **groupe** `docker`
- Les **images téléchargées** sont communes à tous les utilisateurs, **sans propriétaire**, ni contrôle d'accès
- Les **conteneurs lancés** sont communs à tous les utilisateurs, **sans propriétaire**, ni contrôle d'accès

Plan du cours

1 Introduction et rappels
2 Mécanismes système nécessaires aux conteneurs
- Besoins
- Mécanismes d'isolation
- Mécanisme de gestion des ressources

3 Principaux gestionnaires de conteneurs
4 Présentation de Docker
5 Utilisation de Docker
- Installation de Docker
- Catalogue d'images disponibles
- Gestion des images
- Gestion des conteneurs
- Variables d'environnement
- Création d'images

6 Pour aller plus loin avec Docker et les conteneurs
7 Résumé

Types d'images disponibles

- Distributions Linux
  Exemples : Debian, Alpine, ...
- Logiciels CLI
  Exemples : PHP, Node.js, ...
- Logiciels serveur
  Exemples : Apache/httpd, PostgreSQL, ...

```
$ docker images
REPOSITORY    TAG       IMAGE ID       CREATED          SIZE
node          latest    1b9d5f3b36bf   21 hours ago     1.1GB
php           latest    43fca8d539d4   3 days ago       485MB
postgres      latest    387fe63603d1   2 weeks ago      379MB
httpd         latest    6e794a483258   4 weeks ago      145MB
alpine        latest    05455a08881e   2 weeks ago      7.38MB
debian        latest    5c8936e57a38   5 weeks ago      117MB
```

Images pour PHP CLI

Tags pour

- versions majeures de PHP (ex : 8)
- branches de PHP (ex : 8.4)
- versions de PHP (ex : 8.4.x)

```
$ docker images | grep php
REPOSITORY      TAG           IMAGE ID       CREATED         SIZE
php             8.3           a6e20873c731   11 days ago     484MB
php             8.3-cli       a6e20873c731   11 days ago     484MB
php             8             43fca8d539d4   12 days ago     485MB
php             8-cli         43fca8d539d4   12 days ago     485MB
php             8.4           43fca8d539d4   12 days ago     485MB
php             8.4-cli       43fca8d539d4   12 days ago     485MB
php             8.4.4-cli     43fca8d539d4   12 days ago     485MB
php             latest        43fca8d539d4   12 days ago     485MB
```

Images pour PHP CLI

Tags pour

- distribution Linux sous-jacente
- version de cette distribution

```
$ docker images | grep php
REPOSITORY    TAG            IMAGE ID       CREATED         SIZE
php           8-alpine       a7a276ed9be7   12 days ago     94.6MB
php           8.4-alpine     a7a276ed9be7   12 days ago     94.6MB
php           8-bookworm     43fca8d539d4   12 days ago     485MB
php           8.4-bookworm   43fca8d539d4   12 days ago     485MB
```

Images pour PHP intégré dans Apache

- *Tags* pour version majeure, branche ou version de PHP
- Images basées sur Debian, pas Alpine

```
$ docker images | grep php | grep apache
REPOSITORY    TAG                        IMAGE ID       CREATED
php           8.3-apache-bookworm        97b6a0c3b265   11 days ago
php           8-apache-bookworm          fc602075292e   12 days ago
php           8.4-apache-bookworm        fc602075292e   12 days ago
php           8.4.4-apache-bookworm      fc602075292e   12 days ago
php           apache-bookworm            fc602075292e   12 days ago
php           apache                     fc602075292e   12 days ago
```

Images pour Apache/httpd

Tags pour

- branche ou version de Apache
- distributions Linux sous-jacentes

```
$ docker images | grep httpd
REPOSITORY      TAG              IMAGE ID       CREATED         SIZE
httpd           2.4-alpine       2717589de964   6 days ago      56.9MB
httpd           alpine           2717589de964   6 days ago      56.9MB
httpd           2.4              3a4ea134cf8e   8 days ago      145MB
httpd           2.4-bookworm     3a4ea134cf8e   8 days ago      145MB
httpd           2.4.63           3a4ea134cf8e   8 days ago      145MB
httpd           bookworm         3a4ea134cf8e   8 days ago      145MB
httpd           latest           3a4ea134cf8e   8 days ago      145MB
```

Images pour Debian

Tags pour version ou branche

- noms de code / noms de branche : `bookworm`, `bullseye`, ...
- numéros de branche : `12`, `11`, ...
- numéros de version : `12.x`, ...

```
$ docker images | grep debian
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
debian              12                  54e726b437fb        8 days ago          117MB
debian              12.9                54e726b437fb        8 days ago          117MB
debian              bookworm            54e726b437fb        8 days ago          117MB
debian              latest              54e726b437fb        8 days ago          117MB
debian              bullseye            56c4616e8399        8 days ago          124MB
debian              11                  56c4616e8399        8 days ago          124MB
```

Images pour Debian

Tags pour variante *"slim"*

- taille réduite
- doc et pages de manuel supprimées

```
$ docker images | grep debian
REPOSITORY      TAG              IMAGE ID       CREATED        SIZE
debian          bullseye-slim    a120a1536061   8 days ago     80.6MB
debian          bullseye         56c4616e8399   8 days ago     124MB
debian          12-slim          a36a86fb63b1   8 days ago     74.8MB
debian          bookworm-slim    a36a86fb63b1   8 days ago     74.8MB
debian          12               54e726b437fb   8 days ago     117MB
debian          bookworm         54e726b437fb   8 days ago     117MB
debian          latest           54e726b437fb   8 days ago     117MB
```

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Opérations sur les images

```
$ docker image
Usage:  docker image COMMAND
Manage images
Commands:
  build       Build an image from a Dockerfile
  history     Show the history of an image
  import      Import the contents from a tarball to create a filesystem image
  inspect     Display detailed information on one or more images
  load        Load an image from a tar archive or STDIN
  ls          List images
  prune       Remove unused images
  pull        Pull an image or a repository from a registry
  push        Push an image or a repository to a registry
  rm          Remove one or more images
  save        Save one or more images to a tar archive (streamed to STDOUT by defau
  tag         Create a tag TARGET_IMAGE that refers to SOURCE_IMAGE
```

Recherche d'images

- Commande

  `docker search MOT-CLÉ`

- Web

  `https://hub.docker.com/search`

```
$ docker search debian
NAME                                DESCRIPTION                                     STARS     OFFICIAL   AUTOMA
ubuntu                              Ubuntu is a Debian-based Linux operating sys…   15613     [OK]
debian                              Debian is a Linux distribution that's compos…   4582      [OK]
bitnami/minideb                     A minimalist Debian-based image built specif…   126
neurodebian                         NeuroDebian provides neuroscience research s…   98        [OK]
ustclug/debian                      Official Debian Image with USTC Mirror          2
bitnami/debian-base-buildpack       Debian base compilation image                   2                    [OK]
```

Téléchargement d'images

- Télécharger une image

```
docker pull NOM-IMAGE
docker pull NOM-IMAGE:NOM-TAG
```

- Sert aussi à la mise à jour
- Pas de commande (simple) pour mettre à jour toutes les images ☹

```
$ docker pull debian:12
12: Pulling from library/debian
Digest: sha256:43ef0c6c3585d5b406caa7a0f232ff5a19c1402aeb415f68bcd1
Status: Image is up to date for debian:12
docker.io/library/debian:12
```

Lister les images locales

- Lister les images

```
docker images
docker image ls
```

- Formatter l'affichage des images

```
docker images --format "{{.Repository}}:{{.Tag}}"
```

- Mettre à jour toutes ses images

```
docker images --format "{{.Repository}}:{{.Tag}}" |
    xargs -L1 docker pull
```

Suppression d'images

- Supprimer une image

```
docker image rm NOM-IMAGE
docker rmi NOM-IMAGE
```

- Certaines images n'ont pas de nom *(dangling images)*

```
$ docker images
REPOSITORY   TAG     IMAGE ID       CREATED          SIZE
<none>       <none> dc298aa4864e   11 hours ago   460MB
```

- Vieilles images locales, supprimées du *Docker Hub*
- Supprimer une *dangling image*

```
docker rmi IMAGE_ID
```

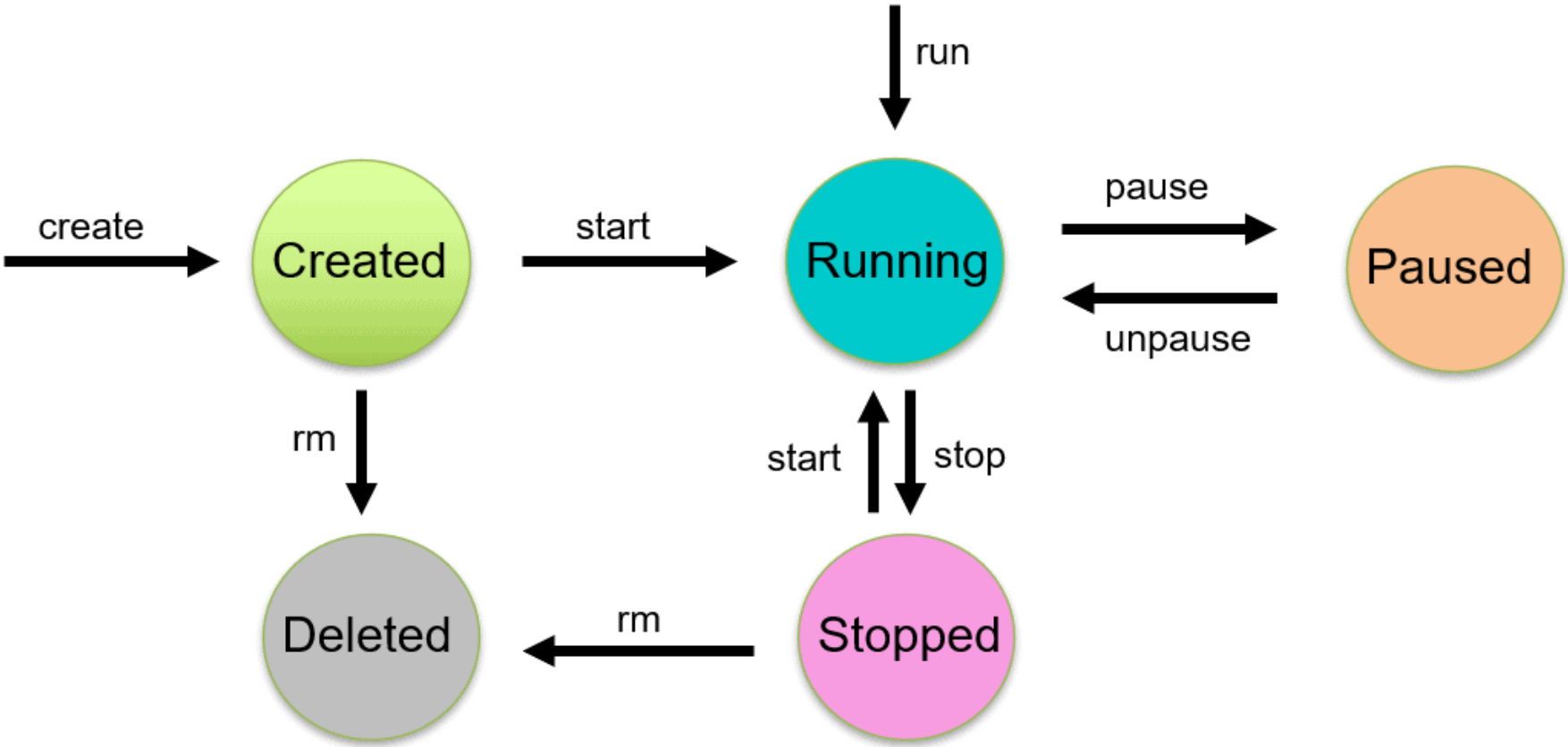
- Supprimer toutes les *dangling images*

```
docker image prune
docker rmi $(docker images -f "dangling=true" -q)
```

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Cycle de vie d'un conteneur



Source : K21Academy

Opérations sur les conteneurs

```
$ docker container
Usage:  docker container COMMAND
Manage containers
Commands:
  attach      Attach local standard input, output, and error streams to a running container
  commit      Create a new image from a container's changes
  cp          Copy files/folders between a container and the local filesystem
  create      Create a new container
  diff        Inspect changes to files or directories on a container's filesystem
  exec        Run a command in a running container
  export      Export a container's filesystem as a tar archive
  inspect     Display detailed information on one or more containers
  kill        Kill one or more running containers
  logs        Fetch the logs of a container
  ls          List containers
  pause       Pause all processes within one or more containers
  port        List port mappings or a specific mapping for the container
  prune       Remove all stopped containers
  rename      Rename a container
  restart     Restart one or more containers
  rm          Remove one or more containers
  run         Run a command in a new container
  start       Start one or more stopped containers
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop one or more running containers
  top         Display the running processes of a container
  unpause     Unpause all processes within one or more containers
  update      Update configuration of one or more containers
  wait        Block until one or more containers stop, then print their exit codes
```

Lancement d'un conteneur

- En 1 étape

```
$ docker run [OPTIONS] NOM_IMAGE [COMMANDE] [ARGS]
```

- En 2 étapes

```
$ docker create [OPTIONS] NOM_IMAGE [COMMANDE] [ARGS]
$ docker start NOM_CONTENEUR | ID_CONTENEUR
```

- Quand un conteneur est lancé, une couche de stockage supplémentaire est ajoutée au dessus de l'image
- Sert à stocker les données transitoires du conteneur
- Attention
  - Un conteneur n'a pas vocation à stocker des données
  - Les **données** dans un conteneur sont considérées comme «**jetables**»

Lister les conteneurs

- Lister les conteneurs qui **tournent**

```
$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS   NAMES
40300fc89d5e   httpd      "httpd-foreground"       12 seconds ago   Up       silly_swanson
f0295cb51ab6   php:apache "docker-php-entrypoi…"   13 seconds ago   Up       funny_rhodes
```

- Lister **tous** les conteneurs

```
$ docker ps --all
$ docker ps -a
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS    NAMES
40300fc89d5e   httpd      "httpd-foreground"       12 seconds ago   Up        silly_swanson
f0295cb51ab6   php:apache "docker-php-entrypoi…"   13 seconds ago   Up        funny_rhodes
e837bf29df87   debian     "bash"                   20 seconds ago   Created   confident_fermi
d4b37860f2b9   php        "docker-php-entrypoi…"   15 seconds ago   Exited    boring_feynman
```

Nommage des conteneurs

- Nommage **automatique**

```
$ docker run ...
$ docker ps
CONTAINER ID   IMAGE   ...   NAMES
40300fc89d5e   httpd   ...   silly_swanson
```

- Nommage **manuel**

```
$ docker run --name NOM_CONTENEUR ...
$ docker ps
CONTAINER ID   IMAGE   ...   NAMES
40300fc89d5e   httpd   ...   mon-conteneur
```

Lancement en fonction du contenu de l'image

Chaque type d'image a une utilisation particulière

- Les logiciels en ligne de commande
  exemple : PHP CLI
- Les logiciels serveurs en tâche de fond
  exemple : Apache/httpd
- Les distributions Linux nues

Exemples d'exécution : point d'entrée et terminal

- Dans chaque image, un **point d'entrée** *(entrypoint)* a été défini par défaut

```
$ docker run php
Interactive shell
php >
[on ne peut rien taper et le conteneur se termine]
```

- Pour une utilisation **interactive** dans un **terminal**, ces 2 options sont nécessaires

```
$ docker run --tty --interactive php
Interactive shell
php > echo 2+3;
5
```

- Options courtes (- au lieu de --)

```
$ docker run -ti php
```

Exemples d'exécution : commandes et *tags*

- Pour une image, on peut préciser la commande que l'on souhaite exécuter

```
$ docker run php php -v
PHP 8.4.4 (cli) (built: Feb 14 2025 20:22:21) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.4.4, Copyright (c) Zend Technologies
```

- On peut préciser le *tag* d'une image à exécuter

```
$ docker run php:8.3 php -v
PHP 8.3.17 (cli) (built: Feb 14 2025 06:34:40) (NTS)
Copyright (c) The PHP Group
Zend Engine v4.3.17, Copyright (c) Zend Technologies
```

Exemples d'exécution : logiciel serveur

- Le point d'entrée par défaut lance Apache

```
$ docker run httpd
AH00558: httpd: Could not reliably determine the server's fully qual
AH00558: httpd: Could not reliably determine the server's fully qual
[Fri Feb 17 20:22:19.813513 2023] [mpm_event:notice] [pid 1:tid 139
[Fri Feb 17 20:22:19.814624 2023] [core:notice] [pid 1:tid 13989778
```

- Lancement en tâche de fond

```
$ docker run --detach httpd
40300fc89d5ee1ff66e23c21154e189ecbc5c5df9d9b70ab68e558154d73333e
```

- Observation du conteneur en fonctionnement

```
$ docker ps
CONTAINER ID   IMAGE   COMMAND                  CREATED      STATUS PORTS     NAMES
40300fc89d5e   httpd   "httpd-foreground"   7 min ago   Up     80/tcp    priceless_s
```

Exemples d'exécution : logiciel serveur avec port réseau

- Lancement avec l'option `--publish` (ou `-p`)

```
$ docker run --detach --publish 80:80 httpd
```

- Observation des ports réseau

```
$ docker ps
CONTAINER ID  IMAGE PORTS                                    NAMES
957b7fd596ea  httpd 0.0.0.0:80->80/tcp, :::80->80/tcp  relaxed
```

- Port interne au conteneur et port visible sur le système hôte
- IPv4 et IPv6

Exemples d'exécution : distribution Linux

- Le point d'entrée par défaut est le *shell* `bash`

```
$ docker run debian
[rien]
```

- `bash` est lancé, mais se termine tout de suite
- Commande complète

```
$ docker run -ti debian
root@22c08fb3a5f2:/# pwd
/
```

Arrêter un conteneur

- Arrêt «propre» (équivalent d'un `kill`)

```
docker stop NOM_CONTENEUR
```

- Arrêt brutal (équivalent d'un `kill -9`)

```
docker kill NOM_CONTENEUR
```

- Le conteneur continue à exister sur le stockage
(couche de stockage supperposée aux couches de l'image)

Supprimer un/des conteneur(s)

- Supprimer un conteneur

```
docker container rm NOM_CONTENEUR
docker rm NOM_CONTENEUR
```

- Supprimer tous les conteneurs qui ne tournent pas

```
$ docker container prune
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N]
```

- Attention aux données qui sont stockées dans la couche haute du conteneur !

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Rappels sur les variables d'environnement (VE)

- Existent pour tout processus (cf. /proc/PID/environ)
- Transmises d'un processus père à ses fils
- Nom généralement en majuscules
- Modifient le comportement de certains logiciels
- Afficher une variable d'environnement
  `echo "$TOTO"`
- Lister les variables d'environnement
  `env`

Passage d'une VE dans un conteneur

- Utilité : initialiser certains paramètres dans un conteneur
- Exemple : mot de passe principal d'un SBGD
- Option à utiliser
  - option longue : `--env`
  - option courte : `-e`
- Exemples

```
docker run --env POSTGRES_PASSWORD=mon-mot-de-passe
docker run -e POSTGRES_PASSWORD=mon-mot-de-passe
```

- Attention
  - mot de passe en clair sur la ligne de commande (voire dans un fichier)
  - mauvaise pratique !

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Pourquoi créer des images ?

- Personnaliser une image existante
  - ex : ajouter des pages Web dans une image de serveur Web (mais on peut faire autrement)
- Compléter une image existante
  - ex : ajouter un module PHP à une image PHP
- Fabriquer une image qui n'est proposée nulle part
  - ex : logiciel développé en interne

Pourquoi créer des images ?

- Exemple : un OS avec une application
- Schéma de la vie d'une machine virtuelle
  - installation initiale de l'OS
  - màj d'OS
  - màj de l'application
  - tout se fait dans le stockage de la VM
- Schéma de la vie d'un conteneur
  - màj d'OS $\longrightarrow$ nouvelle image
  - màj de l'application $\longrightarrow$ nouvelle image
  - un nouveau conteneur est démarré pour chaque nouvelle image

Comment créer des images ?

- Écrire un fichier appelé `Dockerfile`
  qui décrit la recette de construction de l'image
- Fabriquer l'image
  depuis le répertoire qui contient le `Dockerfile`

```
docker build --tag NOM_IMAGE .
docker build -t NOM_IMAGE .
```

- En option : publier son image sur le *Docker Hub*
  ou sur son propre serveur d'images (instance Gitlab, ...)

```
docker push NOM_IMAGE
```

- Lancer un (ou plusieurs) conteneur(s) exécutant cette image

```
docker run ... NOM_IMAGE
```

Exemple simple de `Dockerfile`

- Exemple : on veut créer une image Apache avec un fichier qui constitue un mini site Web
- Contenu du `Dockerfile`

`Dockerfile`

```
FROM php:apache
RUN echo "Bonjour" > /var/www/html/bonjour.txt
```

- Une image est toujours fabriquée à partir d'une image de base
  `FROM` `NOM-IMAGE:TAG`
- Puis on peut enchaîner une ou plusieurs autres instructions (`RUN` ou d'autres instructions)
- **`RUN`** `COMMANDE`
  permet d'exécuter une commande *shell*

`Dockerfile` et instruction `COPY`

- Exemple : on veut créer une image Apache avec un site Web

`Dockerfile`

```
FROM php:apache
COPY ./mon-site /var/www/html/
```

- Le répertoire `mon-site` doit être au même endroit que le `Dockerfile` (le *build context*)

Exemple de `Dockerfile`

- Exemple : on veut créer une image Debian avec des packages supplémentaires

`Dockerfile`

```
FROM debian:latest
RUN apt-get update
RUN apt-get -y upgrade
RUN apt-get -y install php
RUN apt-get -y install git
RUN apt-get clean
RUN git clone https://...
```

- On utilise `apt-get` au lieu de `apt` pour éviter d'avoir des messages d'erreur de la part de `apt`
- L'option `-y` permet d'éviter la confirmation
- Une couche est créée pour chaque commande `RUN`
- Un mécanisme de cache permet d'accélérer la création de l'image quand on modifie le `Dockerfile`

Exemple de `Dockerfile` optimisé

L'empilement de multiples couches est coûteux

- en stockage
- en performance
  - lors de la création de l'image
  - lors de l'exécution d'un conteneur

`Dockerfile`

```
FROM debian:latest
RUN apt-get update; apt-get -y upgrade
RUN apt-get -y install php git
RUN apt-get clean
RUN git clone https://...
```

- On peut regrouper plusieurs commandes avec `;`
- On peut installer plusieurs packages avec une seule commande

`Dockerfile` et variables d'environnement

- Variables d'environnement utilisée pour transmettre des informations aux logiciels tournant dans un conteneur

`Dockerfile`

```
FROM une-image
ENV MA_VARIABLE ma-valeur
```

- Cette valeur est une valeur par défaut de la variable
- Cette valeur peut être redéfinie lors de la création d'un conteneur (vu précédemment)

Plan du cours

1 Introduction et rappels
2 Mécanismes système nécessaires aux conteneurs
- Besoins
- Mécanismes d'isolation
- Mécanisme de gestion des ressources

3 Principaux gestionnaires de conteneurs
4 Présentation de Docker
5 Utilisation de Docker
- Installation de Docker
- Catalogue d'images disponibles
- Gestion des images
- Gestion des conteneurs
- Variables d'environnement
- Création d'images

6 Pour aller plus loin avec Docker et les conteneurs
7 Résumé

Application complexe

On peut être amené à utiliser un grand nombre de conteneurs

- Serveur(s) Web pour application Web
- Serveur(s) Web pour contenu statique (images, ...)
- *Reverse-proxy* pour équilibrage de charge
- Serveur SGBD
- Réplication(s) du SGBD
- *Reverse-proxy* pour HTTPS/TLS
- Serveur(s) de cache pour accélérer l'application
- Exécution de tâches de fond
- Exécution de tâches planifiées
- ...

Limitations de Docker

- Docker, utilisé seul, convient pour un petit nombre de conteneurs
- On peut avoir besoin de répartir ces conteneurs sur un grand nombre de systèmes hôtes
- Dans ces cas d'utilisation, Docker seul ne suffit plus

Orchestration de conteneurs

- Le principe est de manipuler un groupe de conteneurs comme un seul objet
- Les systèmes hôtes ne sont plus manipulés indépendamment, mais forment un *cluster*
- Les conteneurs sont automatiquement distribués sur le *cluster*

Logiciels d'orchestration

- Écosystème Docker
  - Docker Machine : automatise la gestion des machines hôtes
  - Docker Compose : conteneurs multiples, micro-services
  - Docker Swarm : machines hôtes multiples
- Kubernetes (K8S)
- OpenShift
- Nomad
- ...

Plan du cours

1. Introduction et rappels
2. Mécanismes système nécessaires aux conteneurs
   - Besoins
   - Mécanismes d'isolation
   - Mécanisme de gestion des ressources
3. Principaux gestionnaires de conteneurs
4. Présentation de Docker
5. Utilisation de Docker
   - Installation de Docker
   - Catalogue d'images disponibles
   - Gestion des images
   - Gestion des conteneurs
   - Variables d'environnement
   - Création d'images
6. Pour aller plus loin avec Docker et les conteneurs
7. Résumé

Résumé

- Intérêt de l'approche DevOps
- Les conteneurs comme un outil essentiel pour cette approche
- Outil pour le développement
- Outil pour les tests
- Outil pour le déploiement
- Outil à compléter avec la notion d'orchestration de conteneurs