Conteneurisation avec Docker

Licence Professionnelle Métiers des Réseaux Informatiques et Télécommunications, Administration et Sécurité des Réseaux

Sebastien.Kramm@univ-rouen.fr

IUT R&T Rouen, site d'Elbeuf

2021-2022

(version du 25 avril 2022)

Licence

Production

Ce document est généré à partir du fichier source LaTeXen 3 versions :

- Une version "diapos" pour le cours lui-même
  → suffixée par "_B"
- Une version pour l'impression, avec 4 diapos par page A4
  → suffixée par "_P"
- Une version pour la lecture à l'écran, similaire à la première mais sans les animations
  → suffixée par "_H"

Information

- Ce document contient des liens vers des pages ressources, qui apparaissent avec une couleur distinctive.
- Page du cours : universitice.univ-rouen.fr/course/view.php?id=17569

Organisation de ce mini-module

- Thème : conteneurisation sous Docker
- 2021-2022 :
  - CM 1h30 : lundi 10/01/2022 - 13h
  - TP1 2h : lundi 10/01 - 14h30 & 16h30
  - TP2 3h : vendredi 14/01 - 8h30 & 13h
- Evaluation :
  - TP : aisance, respect des consignes, capacités à répondre aux questions, etc.
  - QCM en fin de TP 2

UNIVERSITÉ DE ROUEN

Sommaire

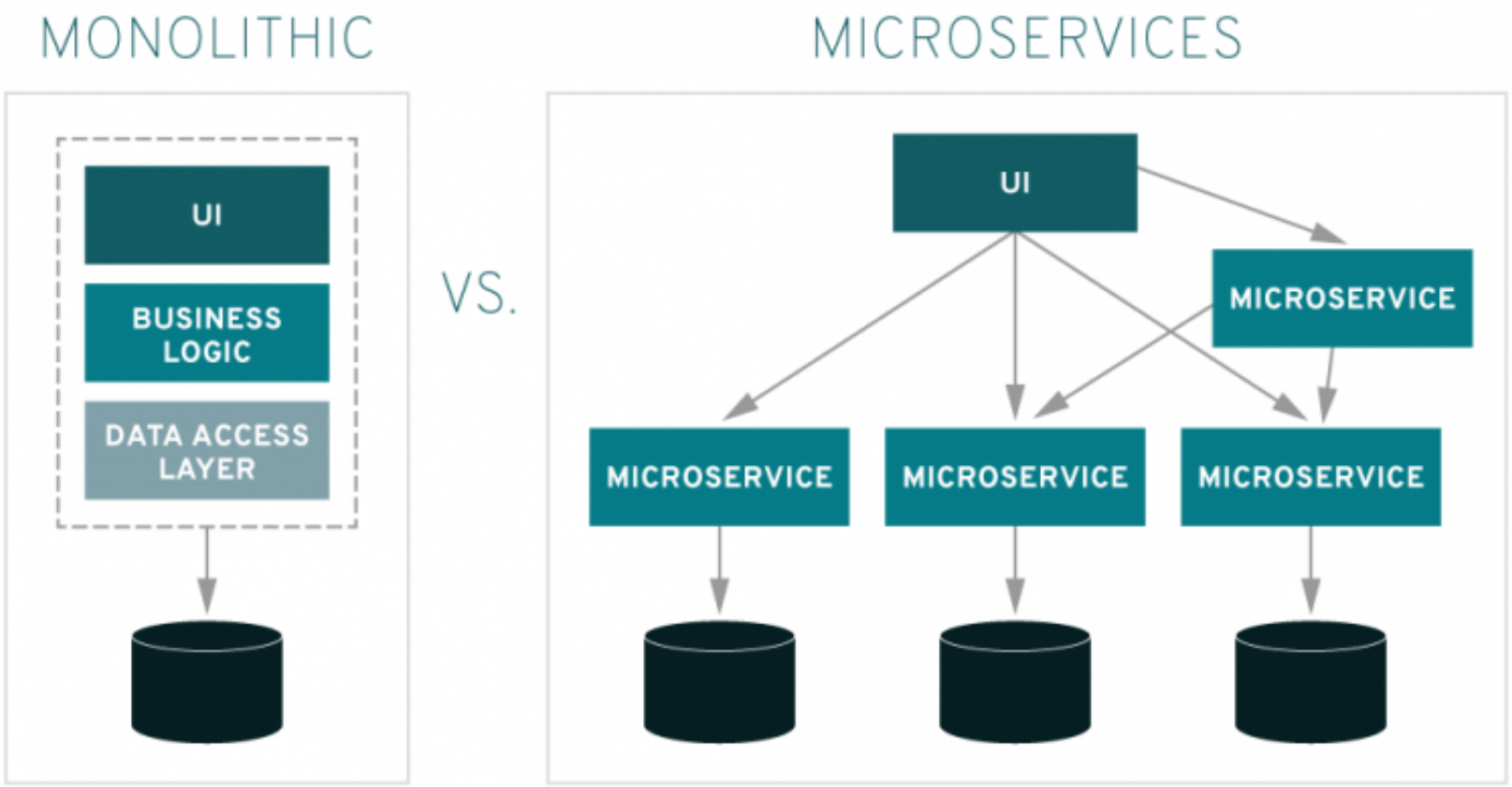
Contexte général

Sur un serveur :

- application web : frontal web + moteur (PHP, Java, Python,...) + appli métier + BDD (plusieurs) + ...
- des outils différents avec des dépendances différentes
- une équipe dont chaque membre travaille avec un env. différent

$\Rightarrow$ Difficile à maintenir et à faire évoluer.

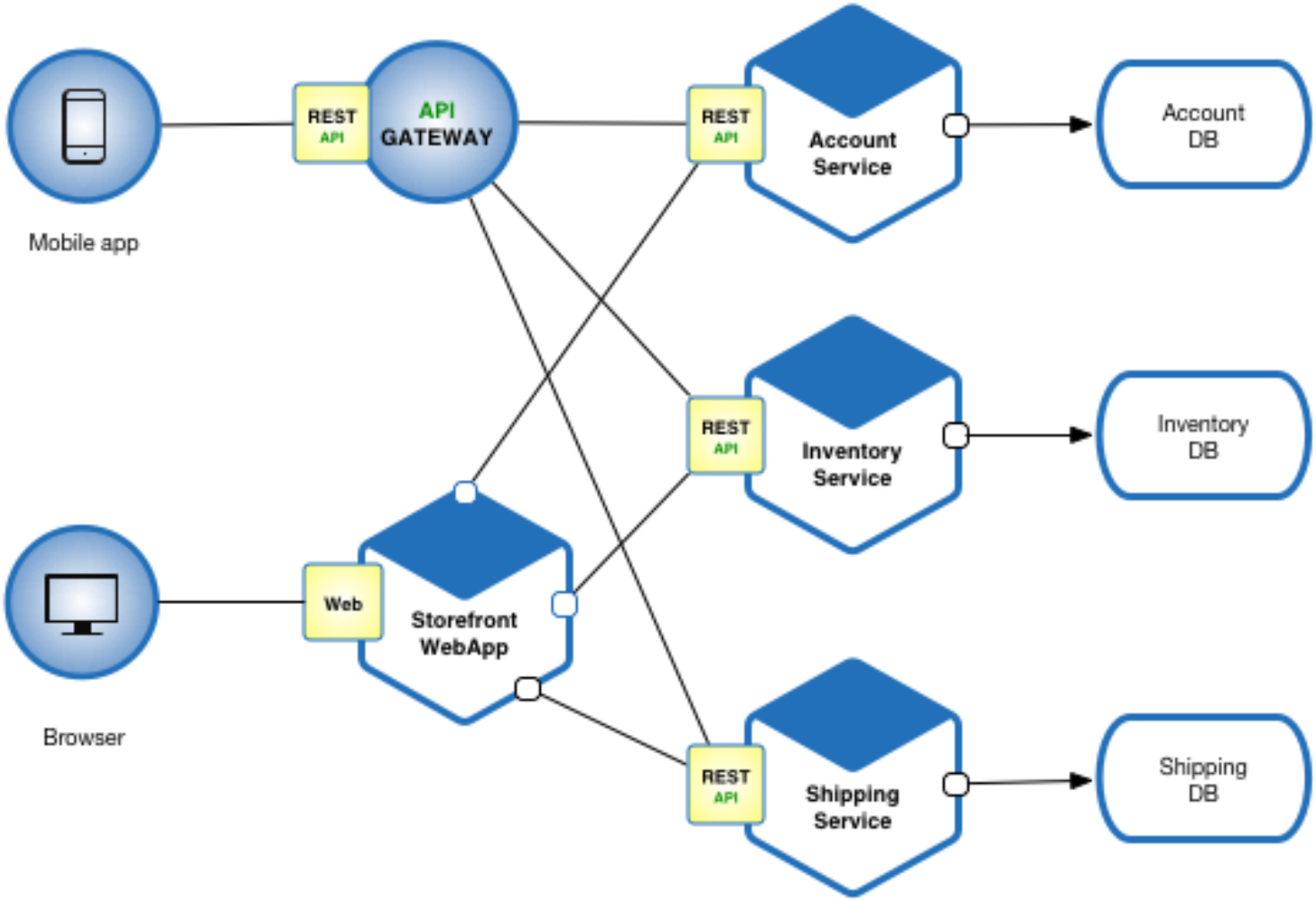
Architecture SI : deux approches



source : https://www.redhat.com/en/topics/microservices/what-are-microservices

Microservices : exemple

Un vendeur en ligne :



source : https://microservices.io/

Mais pas suffisant !

Exemple : on veut faire tourner 3 services A, B, C ayant les dépendances suivantes :

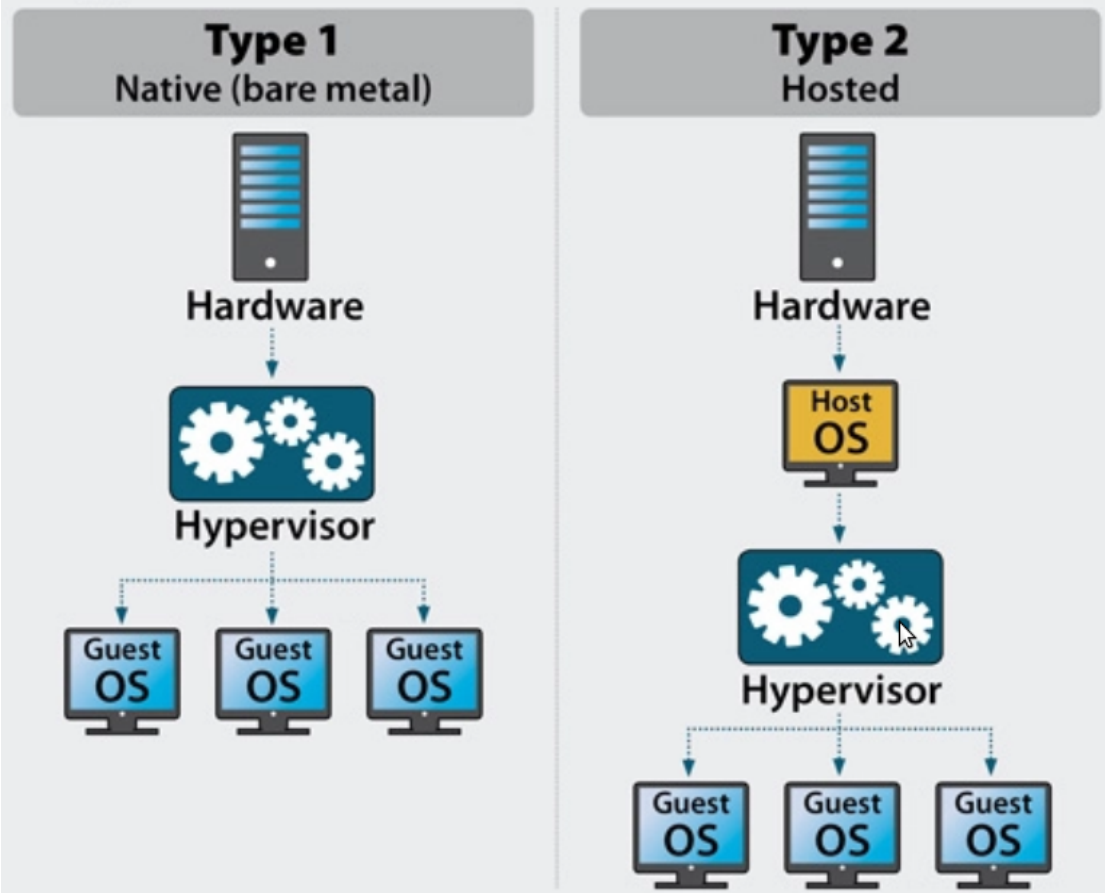
Table 3-2. *Service dependencies*

Service A	Service B	Service C
JavaScript v1.8.5	-	-
Python v2.7	-	Python v2.1
Flask v0.12.4	Flask v0.10.3	-

Problème ! : dépendances différentes...
→ très difficile à faire tourner sur la même machine...

Solution : virtualisation ?

Rappels : 2 types de virtualisation

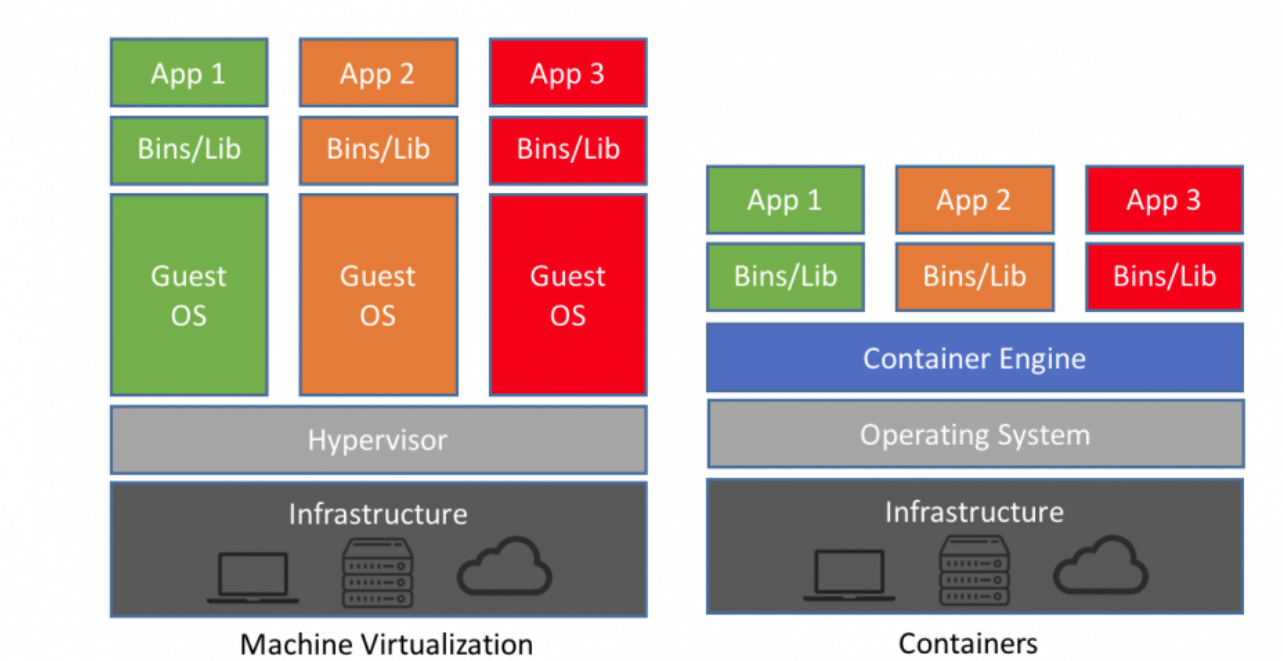


Dans les deux cas, pas mal d'inconvénients.

https://www.serverwatch.com/virtualization/hypervisor-server/

Solution contemporaine : conteneurisation

S'appuie sur les ressources locales (OS), mais avec une **totale isolation** entre les conteneurs.



Le leader ("the *de facto* standard") : docker

Avantages de la conteneurisation

Les conteneurs sont :

- Flexible : même les applications les plus complexes peuvent être conteneurisées.
- Léger : les conteneurs exploitent et partagent le noyau hôte.
- Interchangeable : vous pouvez déployer des mises à jour à la volée
- Portable : vous pouvez créer localement, déployer sur le cloud et exécuter n'importe où votre application.
- Évolutif : vous pouvez augmenter et distribuer automatiquement les réplicas (les clones) de conteneur.
- Empilable : Vous pouvez empiler des services verticalement et à la volée.

src : https://devopssec.fr/article/differences-virtualisation-et-conteneurisation

Avec/sans la conteneurisation

Exemple de situation : site à mettre à jour
- passage de PHP x.y à PHP x.z
- chgt version SGBD
- nouvelle bibli js
- etc.

Sans conteneur

- Se connecter sur le serveur
- Arretez le site, mettre en place une page "En travaux"
- Installer le nouveau moteur PHP
- Installer la nouvelle version SGBD
- Copier la lib JS
- tout redémarrer, faire des tests, corriger les bugs, etc...

→ Interruption du service : longue...

Avec conteneur

- créer un conteneur en local sur sa machine et tout tester dessus
- se connecter sur le serveur
- y copier les conteneurs pour remplacer les anciens
- le démarrer

→ Interruption du service : courte !

Sommaire

Pourquoi Docker ?

https://www.docker.com/why-docker

Docker delivers on its promise to "Build, Ship, and Run"
In 2013, Docker introduced what would become the industry standard for containers.
Containers are a standardized unit of software that allows developers to isolate their app from its environment, solving the "it works on my machine" headache.

Docker, c'est :

- une solution de conteneurisation "clés en main", robuste et fiable
  - s'appuie sur le *kernel* Linux... mais tourne aussi (depuis peu) sur Windows
- une société qui développe le produit
  - fondée à Paris en 2008 par Solomon Hykes + 2 amis (à 24 ans !)
  - départ Silicon Valley en 2010
- *pricing* : 0 pour usage perso, payant pour entreprises

Terminologie Docker

- Images : The blueprints of our application which form the basis of containers.
- Containers : Created from Docker images and run the actual application. A list of running containers can be seen using the docker ps command.
- Docker Daemon : The background service running on the host that manages building, running and distributing Docker containers. The daemon is the process that runs in the operating system which clients talk to.
- Docker Client : The command line tool that allows the user to interact with the daemon.
- Docker Hub : A registry of Docker images. You can think of the registry as a directory of all available Docker images.

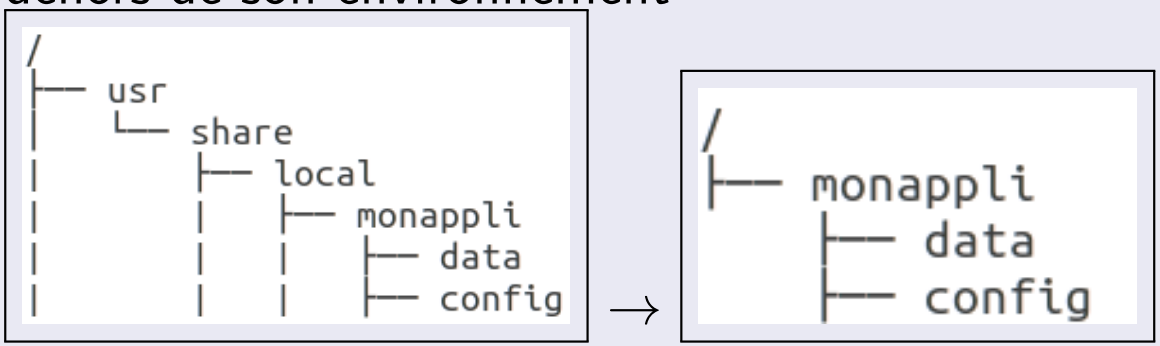
source : https://docker-curriculum.com/

Historique

Docker a été construit sur des briques pré-existantes

1982 : `chroot`

Idée : changer le répertoire racine apparent pour un processus
→ Un programme *chrooté* ne peut pas accéder à des fichiers et commandes en dehors de son environnement



2006 : `cgroups` (" *control groups*")

(WP)
Linux kernel feature that limits and isolates resource usage (such as CPU, memory, disk I/O, and network) to a collection of processes.
De multiples évolutions, avec à chaque itération plus de "features" et de "use cases"

Historique-2

2008 : containers LXC (LinuXContainers)

https://en.wikipedia.org/wiki/LXC

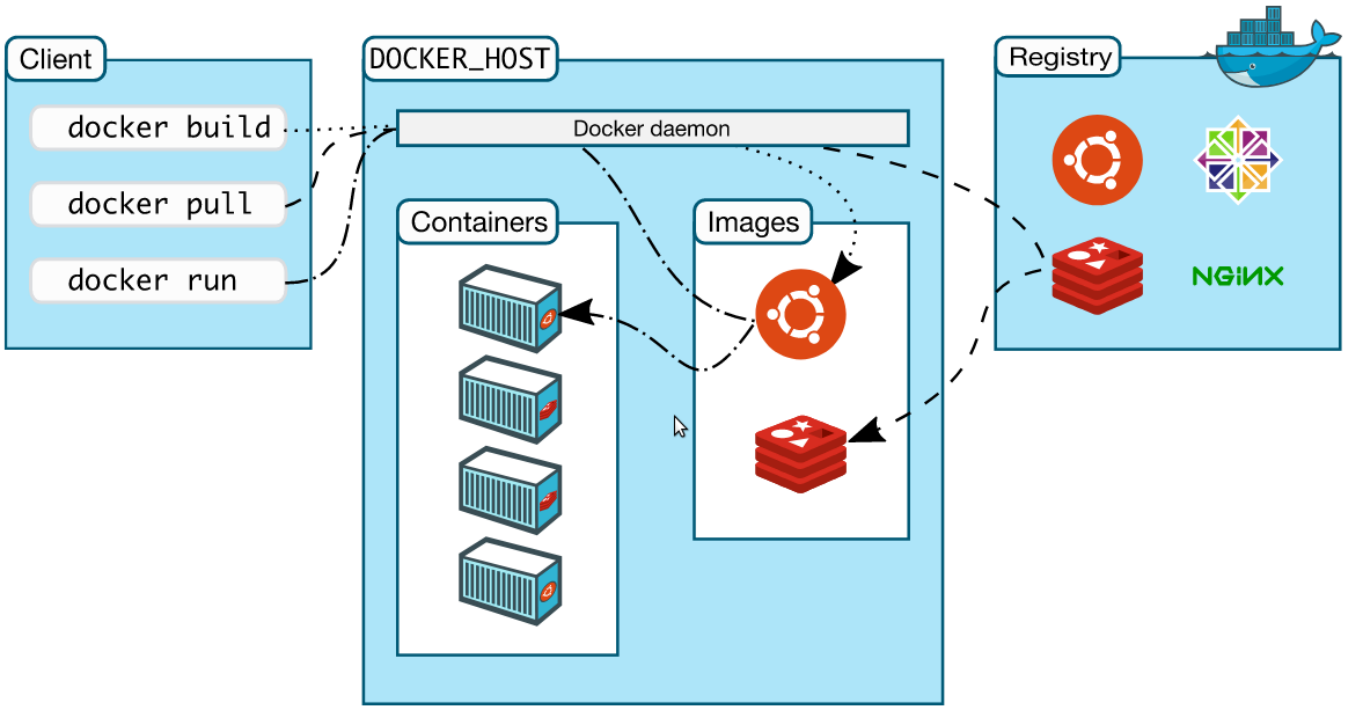
Operating-system level virtualization by combining Linux kernel's cgroups and support for isolated namespaces to provide an isolated environment for applications.

Utilisé initialement par Docker, avant que celui-ci ne développe son propre outil

Docker ?

Docker est composé de plusieurs éléments :

- le démon Docker qui s'exécute en arrière-plan et qui s'occupe de gérer les conteneurs (`dockerd`)
  - intègre une API REST pour communiquer avec le *daemon*
- le client en CLI (Command Line Interface) : commande `docker xxx`



src : https://docs.docker.com/get-started/overview/

Client CLI

- Un ensemble de (nombreuses) commandes :

```
$ docker <commande> [options] <sous-commande>
```

- La liste : `$ docker --help`
- Pour chaque commande, beaucoup d'options :

```
$ docker <commande> --help
```

- Mais une cohérence globale dans les commandes.
  Sous-commandes communes : `ls`, `rm`, `inspect`, etc.

API REST

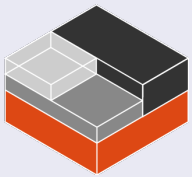
- Permet d'accéder au *daemon* à distance, via HTTP
  - via un client HTTP (`curl`)
    → use case : si on a pas l'accès SSH sur la machine
  - via un client lourd, qui fait les requetes HTTP
- ref : https://docs.docker.com/engine/api/
- Exemple (si l'API écoute le port 5555) :

```
$ curl http://localhost:5555/images/json
```

Alternatives ?

Principales alternatives à Docker

- LXD : s'appuie sur la techno LXC, développé par Canonical



https://linuxcontainers.org/

- Podman : https://podman.io/

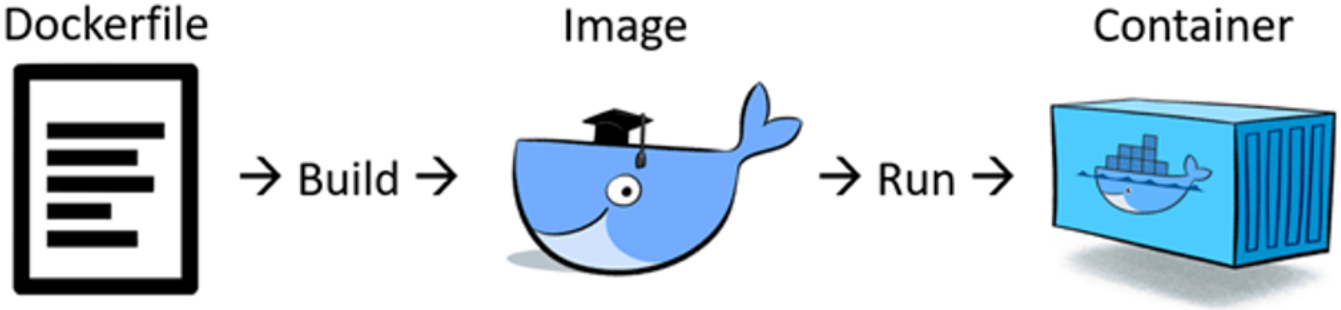
podman

Soutenu par https://opencontainers.org/

Sommaire

"Images" Docker

- Une image est la base qui va servir à la création d'un conteneur



- Contient les bibliothèques et le code dont votre application a besoin pour fonctionner
- Est en général construite à partir d'une autre image, et peut elle-même servir de base pour la création d'une autre image
- On trouve les images dans les "registry"
  - publics : docker registry : https://hub.docker.com/
    Janvier 2022 : 8,700,567 images dispos ! ! !
  - mais on peut créer des "registry" privées.

Gérer mes images

- Chercher une image :
  `$ docker search <nom-image>`
  Exemple : `$ docker search nginx | wc -l` → 26
- Recupérer une image (registry public docker) :
  `$ docker pull <nom-image>`
- Afficher la liste des images en local :
  `$ docker image ls` ou `$ docker images`
- Supprimer une image de la liste locale : `$ docker rmi <nom-ou-id>`
  (possible **uniquement** si l'image n'est pas utilisée dans un conteneur)
- Afficher les détails d'une image :
  `$ docker image inspect <nom-ou-id>`

Note : si de multiples images on le même nom, il faut soit préciser le tag, soit utiliser l'id

Tags d'images

Les images sont versionnées (façon "git")
→ Chaque image peut avoir des versions ("tags") différentes
https://hub.docker.com/_/debian?tab=tags
Exemple :

```
$ docker pull ubuntu:21.04
$ docker pull ubuntu:20.04
```

⇒

```
$ docker images
REPOSITORY   TAG       IMAGE ID         CREATED       SIZE
ubuntu        20.04          825d55fb6340   2 days ago      72.8MB
ubuntu        21.04          7cc39f89fa58   2 days ago      80MB
```

Création d'images : Dockerfile

- On part d'une image existante, à laquelle on ajoute des composants et commandes
- Utilisation d'un script ("Dockerfile") :

```
#This is a sample image
FROM ubuntu
LABEL maintainer="email@example.com"
LABEL version="1.0"
RUN apt update
RUN apt install -y nginx
CMD ["echo", "Hello World!"]
```

- Commandes :
  - `FROM` : image de départ (par défaut, prise sur le Docker Hub)
  - `RUN` : commandes executée lors de la construction de l'image
  - `CMD` : commande executée lors du lancement du conteneur (unique !)

Attention, les commandes utilisées par RUN et CMD doivent exister dans l'image de base.

Dockerfile : autres commandes utiles

- `LABEL` : ajout de métadonnées (optionnel)
- `ENV` : permet de définir des variables d'environnement
  Exemple : `ENV MY_NAME="John Doe"`
- `WORKDIR /path/to/workdir` : pour spécifier le dossier courant dans le conteneur, lors de l'exécution de `RUN` ou `CMD`
  (équivalent à `cd`)
- `COPY` : copie des fichiers/dossiers depuis la machine hôte vers l'image
  Les chemins dans le conteneur peuvent être absolus ou relatif (par rapport à `WORKDIR`)
  Exemples :

```
COPY test.txt relativeDir/
```

```
COPY test.txt /absoluteDir/
```

Ref : https://docs.docker.com/engine/reference/builder/

Création d'images : étape 2

Construction de l'image via le Dockerfile :

- si fichier nommé `Dockerfile`, et dans le dossier courant) :

```
$ docker build .
```

- option -f : permet de spécifier le nom/chemin du Dockerfile

```
$ docker build -f monDockerFile .
```

- option -t : pour donner un nom à l'image :

```
$ docker build -t NomDeMonImage .
```

"*Best Practices*"

- Chaque ligne du Dockerfile correspond à une "couche" (*Layer*) : chaque version modifiée va ajouter une couche
  → mieux vaut essayer de les minimiser
- Exemple :

```
RUN apt update && apt-get install -y nginx
```

préférable à :

```
RUN apt update
RUN apt install -y nginx
```

Sommaire

Lancer un conteneur

- `$ docker run <nom>` :
  - télécharge l'image si pas présente en local
  - démarre le conteneur, exécute la commande prévue, et arrête le conteneur
- Exemple :
  `$ docker run ubuntu` : rien ne se passe
- Pour avoir un shell sur le conteneur :
  `$ docker run -it ubuntu /bin/bash`
  → démarre le conteneur, et démarre bash
- options :
  - `-p 81:80` : map port 81 of the host to port 80 in the container
  - `-d` : run the container in detached mode (in the background)
  - `-e` : permet de spécifier des variables d'environnement
    exemple : `-e MA-VAR=abcd`

Visualiser les conteneurs

- `$ docker ps` : uniquement ceux qui sont actifs
- `$ docker ps -a` : tous ("all") ou : `docker container ls -a`
- Chaque conteneur se voit attribuer un **UID** et un **nom** :

```
CONTAINER ID   IMAGE           COMMAND                  CREATED         STATUS         NAMES
e90b8831a4b8   nginx           "/bin/bash -c 'mkdir "   11 weeks ago    Up 4 hours     my_nginx
00c6131c5e30   telegraf:1.5    "/entrypoint.sh"         11 weeks ago    Up 11 weeks    my_telegraf
```

- On pourra les gérer (supprimer, lancer, interagir, etc) via le nom ou l'ID
- Exemple : on peut obtenir les détails sur le conteneur indifférement avec : `$ docker inspect my_nginx` ou avec `$ docker inspect e90b`
- Pour avoir les logs d'un conteneur : `$ docker logs <id-ou-nom>`

Lancer un conteneur

Attention

- `$ docker run` crée un nouveau conteneur à chaque fois
- docker run = (docker pull +) docker create + docker start + docker attach

Pour lancer un conteneur déjà présent :

```
$ docker start -a <id-ou-nom>
```

Arreter/supprimer un conteneur

- Supprimer un conteneur :
  `$ docker rm <id-ou-nom>`
  (doit être arreté)
- `$ docker stop <id-ou-nom>` : arrête "proprement" le conteneur
  (envoie le signal SIGTERM)
- `$ docker kill <id-ou-nom>` : envoie un signal SIGKILL (terminaison immédiate)
  → risque de perte de données...
- Note : au bout d'un certain temps sans réponse, `docker stop` envoie également un SIGKILL...

```
$ docker stop --help
Usage:  docker stop [OPTIONS] CONTAINER [CONTAINER...]
Stop one or more running containers
Options:
  -t, --time int   Seconds to wait for stop before killing it (default 10)
```

Interagir avec un conteneur

- Quand un conteneur est actif, on peut "se connecter" dessus (=ouvrir un shell) avec `exec` :

```
$ docker exec -ti my_container sh
```

- On peut transférer des fichiers depuis/vers le conteneur avec `cp` :
  - Fonctionne de façon similaire à la commande Linux :

```
$ cp <SOURCE> <DEST>
```

  - Pour copier depuis le conteneur vers l'hôte :

```
$ docker cp cont-id:/chemin/vers/fichier /home/$USER
```

  - Pour copier depuis l'hôte vers le conteneur :

```
$ docker cp /home/$USER cont-id:/chemin/vers/fichier
```

- Pour monitorer les containers : `$ docker stats`

```
File  Edit  View  Search  Terminal  Help
CONTAINER ID   NAME              CPU %     MEM USAGE / LIMIT     MEM %     NET I/O       BLOCK I/O   PIDS
05692694f90d   jolly_tu          0.00%     1.309MiB / 15.25GiB   0.01%     3.79kB / 0B   0B / 0B     1
cc78feed3023   vibrant_thompson  0.00%     1.199MiB / 15.25GiB   0.01%     4.4kB / 0B    0B / 0B     1
```

Et dans le conteneur ?

- Un conteneur est pensé pour gérer un seul "service" (principe de l'architecture en "micro services")
- Si on veut deux tâches dans un conteneur, on va les gérer à la construction de l'image.
- Exemple générique :

`Dockerfile` :

```
FROM ubuntu:latest
COPY process_1 process_1
COPY process_2 process_2
COPY wrapper_script.sh wrapper_script.sh
CMD ./wrapper_script.sh
```

`wrapper_script.sh` :

```
#!/bin/bash

# Start the first process
./my_first_process &

# Start the second process
./my_second_process &

# Wait for any process to exit
wait -n

# Exit with status of process that exited first
exit $?
```

Sommaire

Information générales

Information générales : `$ docker system info`

Occupation disque : `$ docker system df`

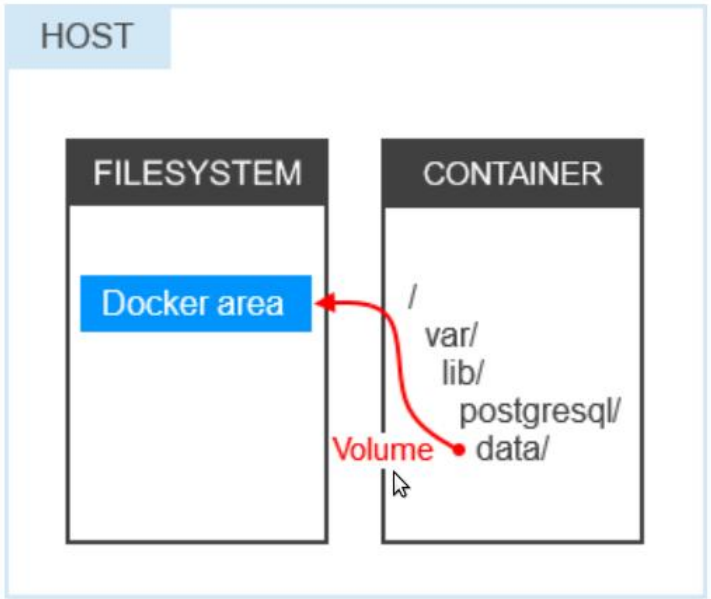
```
TYPE            TOTAL     ACTIVE    SIZE      RECLAIMABLE
Images          7         7         1.806GB   80.37MB (4%)
Containers      24        0         68.43kB   68.43kB (100%)
Local Volumes   5         4         297.5MB   0B (0%)
Build Cache     0         0         0B        0B
```

Sommaire

Docker volumes

"Volume" : **espace de stockage nommé** dans le conteneur.

- Identifié par un **nom**
- Sera stocké sur l'hôte, de façon séparée du conteneur
  → pas besoin de s'occuper de sa localisation
- Pourra être **partagé** entre plusieurs conteneurs



source : https://dominikbraun.io/blog/docker/docker-volumes-a-quick-summary/

Docker volumes : commandes

- Créer un volume : `$ docker volume create <nom>`
- Lister les volumes : `$ docker volume ls`
- Afficher les détails (JSON) : `$ docker volume inspect <nom>`
- Effacer un volume : `$ docker volume rm <nom>`
- Effacer tous les volumes : `$ docker volume prune`

Sommaire

Docker network

- Docker peut mettre en place des réseaux virtuels permettant de faire communiquer des applications conteneurisées.
- Chaque réseau crée se voit attribué un **nom** et un **id**.
  - Création d'un réseau :

    ```
    $ docker network create --driver <type> <nom>
    ```

    (différents types de réseaux possibles selon les besoins)
  - Afficher tous les réseaux crées : `$ docker network ls`

    ```
    $ docker network ls
    NETWORK ID     NAME           DRIVER    SCOPE
    a588f28a1a20   bridge         bridge    local
    3bd9b1514577   cracboumhue    bridge    local
    3574c02cf6ca   host           host      local
    829d35d0cb9f   none           null      local
    ```

  - Afficher les détails d'un réseau (JSON) :

    ```
    $ docker network inspect <networkname>
    ```

détails : https://docs.docker.com/network/

Sommaire

Nettoyage

"*to prune ...*" $\Leftrightarrow$ nettoyer, élaguer

1. Supprimer les images non utilisées `$ docker image prune`
2. Supprimer les conteneurs non utilisés `$ docker container prune`
3. Supprimer les réseaux non utilisés `$ docker network prune`
4. Supprimer les volumes non utilisés `$ docker volume prune`
5. 1-2-3 en une seule fois : `$ docker system prune`

Options disponibles, voir https://docs.docker.com/config/pruning/

Sommaire

Sécurité des conteneurs

- Docker est très utilisé en production
  → la sécurité des conteneurs est primordiale !
- Docker a un partenariat avec une sté spécialisée : snyk
  ⇒ on dispose d'un service de **scan** des images pour y trouver des vulnérabilités répertoriées (CVE)

  `$ docker scan <image-name>`

  - implique d'être authentifié sur le "docker-hub" → `$ docker login`
- Exemple : `$ docker scan getting-started`

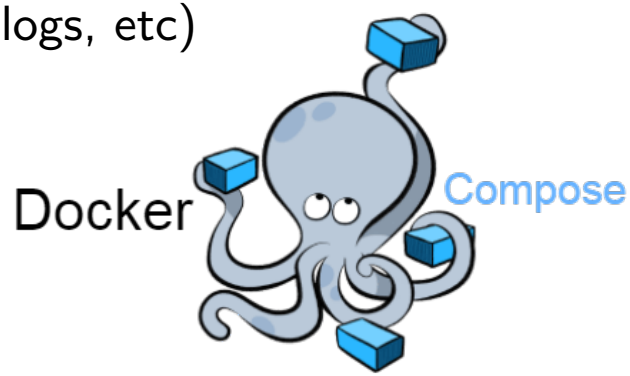
```
✗ Low severity vulnerability found in freetype/freetype
  Description: CVE-2020-15999
  Info: https://snyk.io/vuln/SNYK-ALPINE310-FREETYPE-1019641
  Introduced through: freetype/freetype@2.10.0-r0, gd/libgd@2.2.5-r2
  From: freetype/freetype@2.10.0-r0
  From: gd/libgd@2.2.5-r2 > freetype/freetype@2.10.0-r0
  Fixed in: 2.10.0-r1

✗ Medium severity vulnerability found in libxml2/libxml2
  Description: Out-of-bounds Read
  Info: https://snyk.io/vuln/SNYK-ALPINE310-LIBXML2-674791
  Introduced through: libxml2/libxml2@2.9.9-r3, libxslt/libxslt@1.1.33-r3, ngi
  From: libxml2/libxml2@2.9.9-r3
  From: libxslt/libxslt@1.1.33-r3 > libxml2/libxml2@2.9.9-r3
  From: nginx-module-xslt/nginx-module-xslt@1.17.9-r1 > libxml2/libxml2@2.9.9-
  Fixed in: 2.9.9-r4
```

UNIVERSITÉ DE ROUEN

Sommaire

Orchestration de conteneurs

- En pratique : on se retrouve très rapidement avec une grande quantité de conteneurs locaux sur sa machine.
- Dans un contexte de "mise en prod", les manipuler un par un est délicat (oubli d'une option dans un `docker run`, etc.)
- Docker propose un outil permettant de manipuler un **ensemble** de conteneurs sur le même hôte comme un tout : **Docker Compose**
  - création, démarrage, arrêt
  - supervision (status, logs, etc)



- Si configuration plus complexe (multiples machines) : Docker Swarm ou Kubernetes

Sommaire

Référence

- Commandes :
  https://docs.docker.com/engine/reference/commandline/cli/
- Format *Dockerfile* :
  https://docs.docker.com/engine/reference/builder/

Bibliographie

- Sathyajith Bhat - Practical Docker with Python (2018, Apress)
- https://dominikbraun.io/blog/docker/
- https://en.wikipedia.org/wiki/Docker_(software)
- https://xataz.developpez.com/tutoriels/utilisation-docker/ (2017)
- https://devopstuto-docker.readthedocs.io/en/latest/tutoriels/tutoriels.html