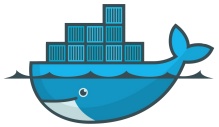
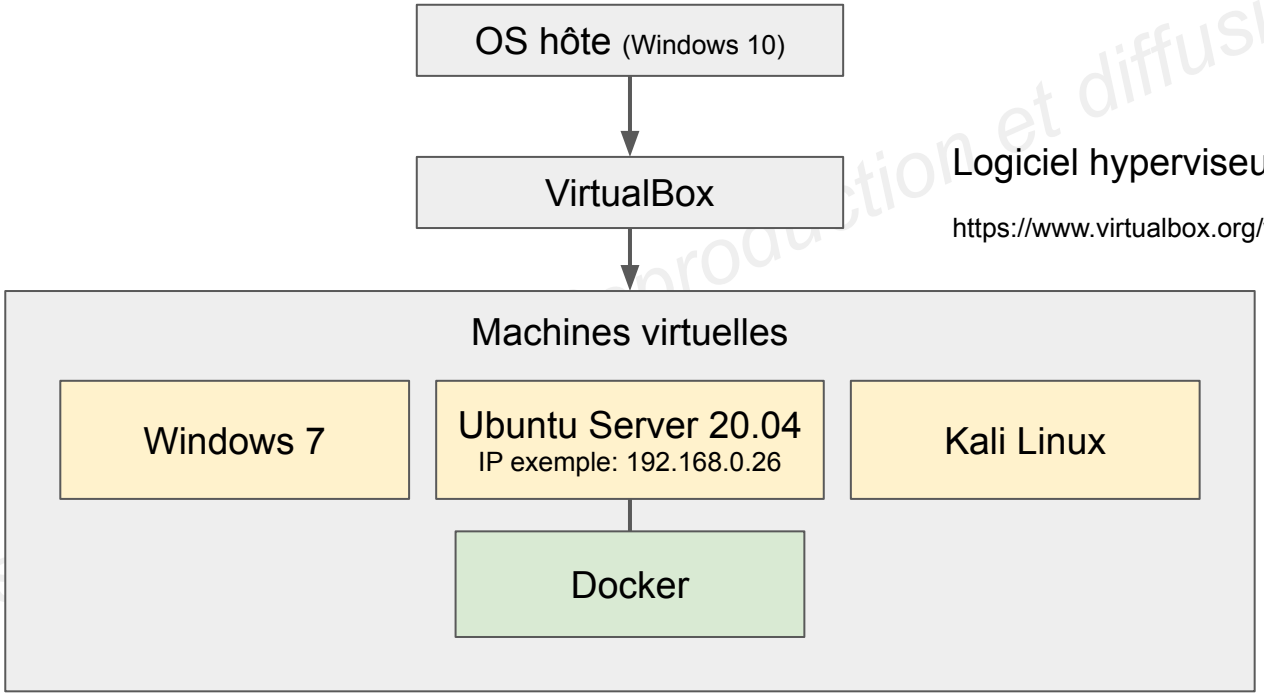
Docker



Christophe Dufour

Infrastructure (prof)



Logiciel hyperviseur

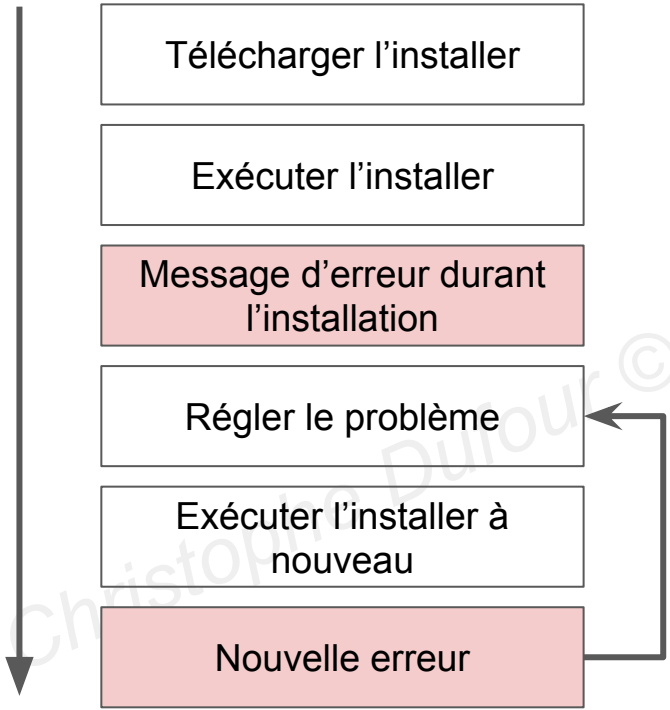
https://www.virtualbox.org/wiki/Downloads

Docker - intro

- Moteur de virtualisation/conteneurisation le plus populaire
- Logiciel libre, open source depuis 2013
- Développé au départ par Solomon Hykes pour un projet interne à la société française dotCLoud

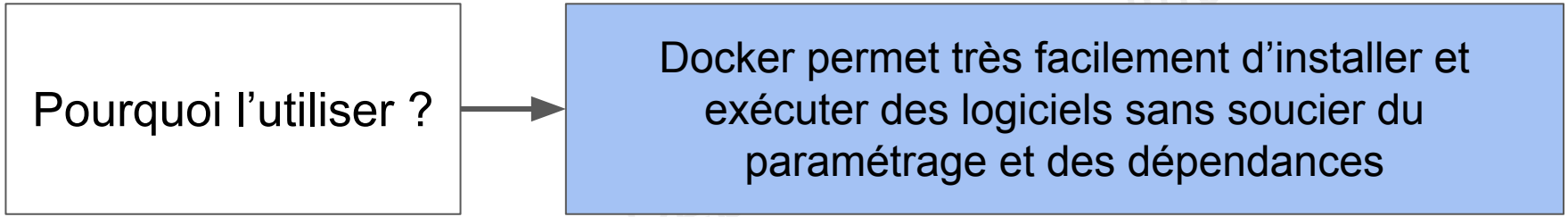
Pourquoi utiliser Docker ?

Installer un logiciel

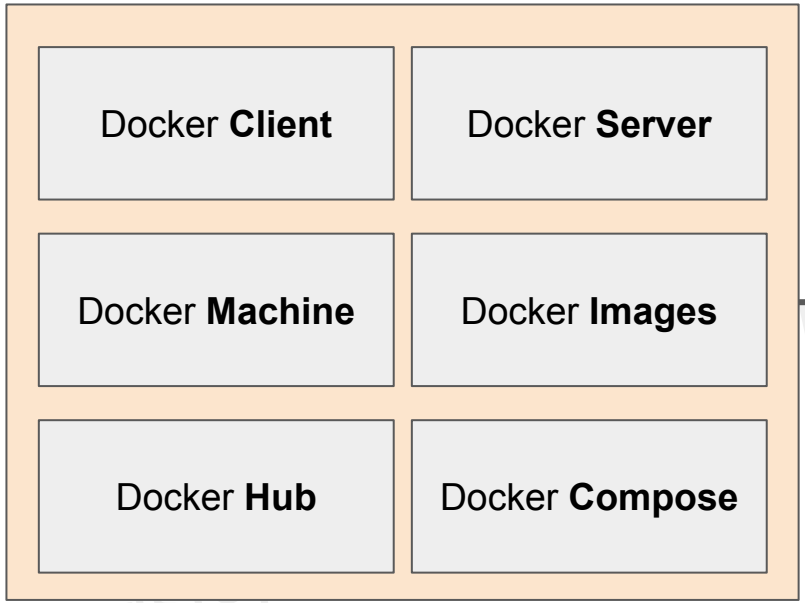


Ce à quoi Docker tente de remédier

Pourquoi utiliser Docker ?

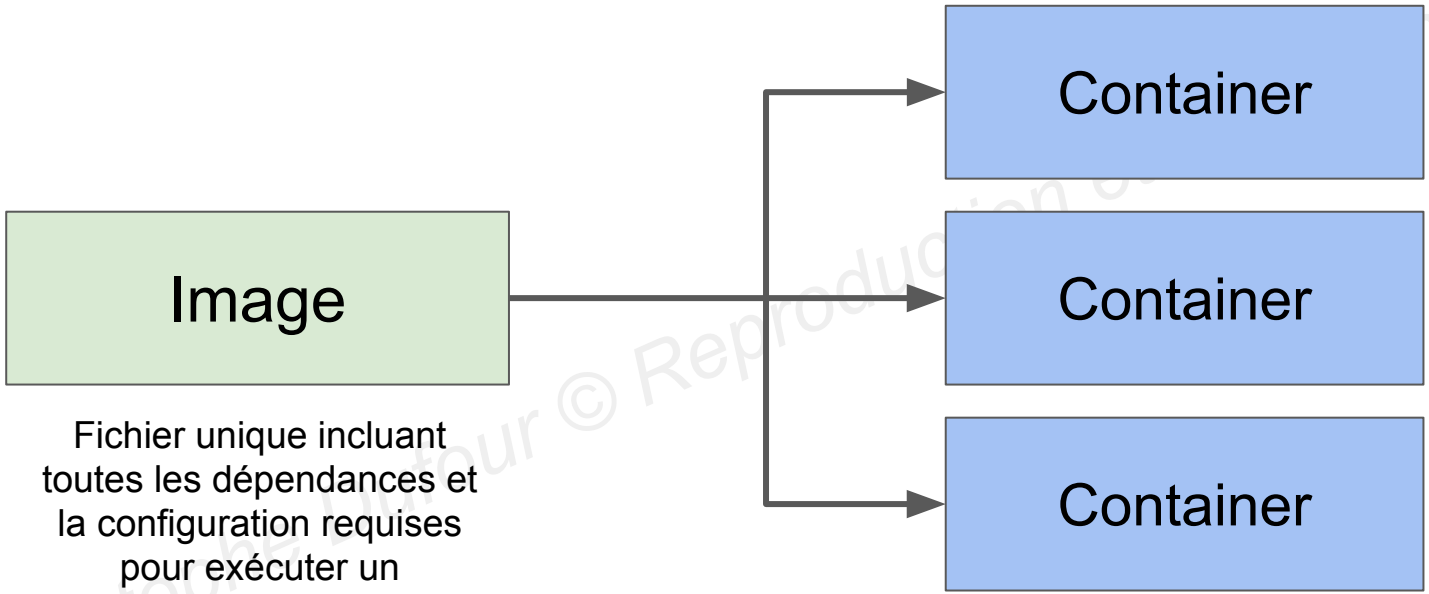


L’écosystème Docker



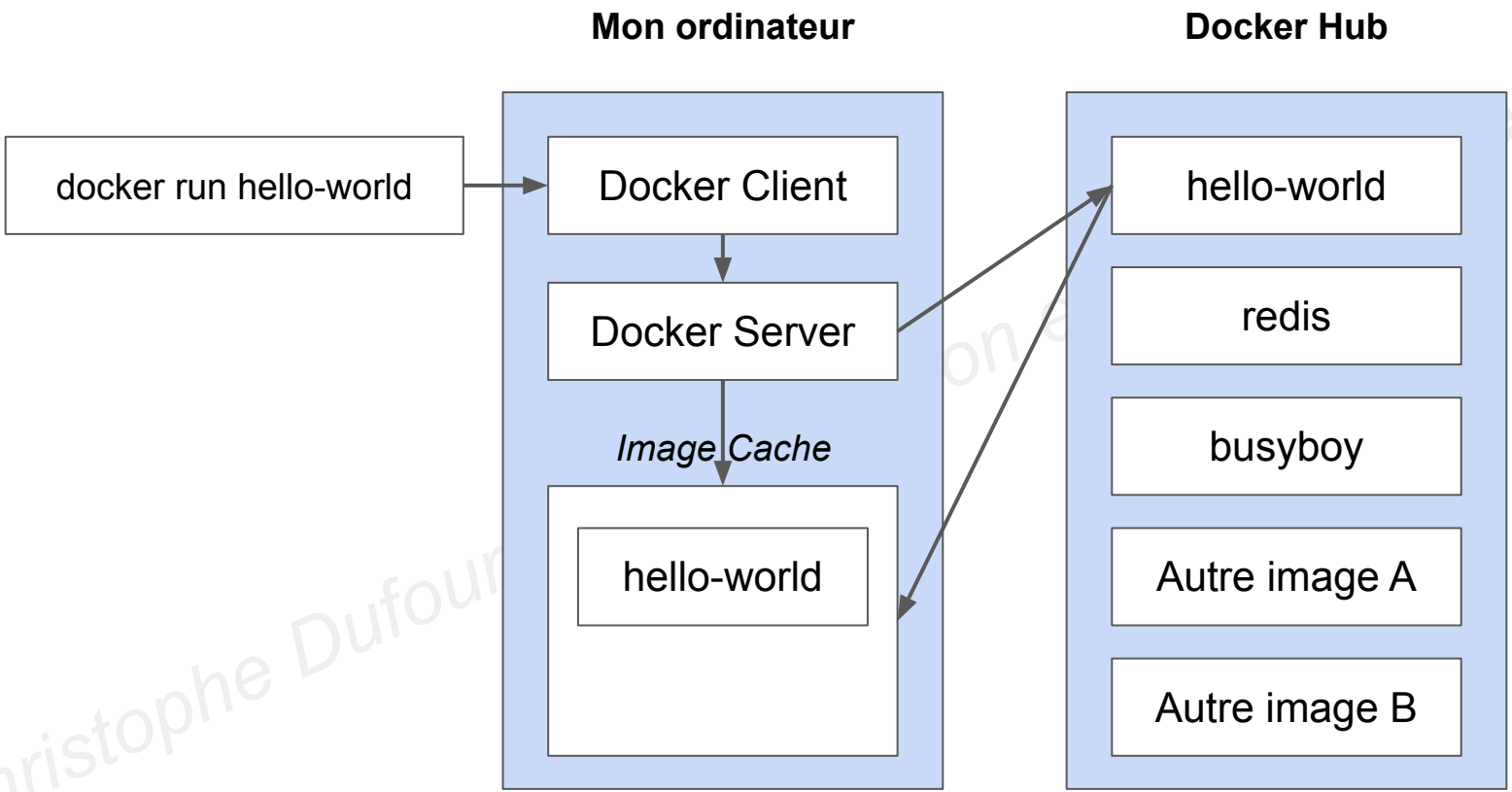
Docker est une plateforme ou écosystème destiné à créer et exécuter des **conteneurs**

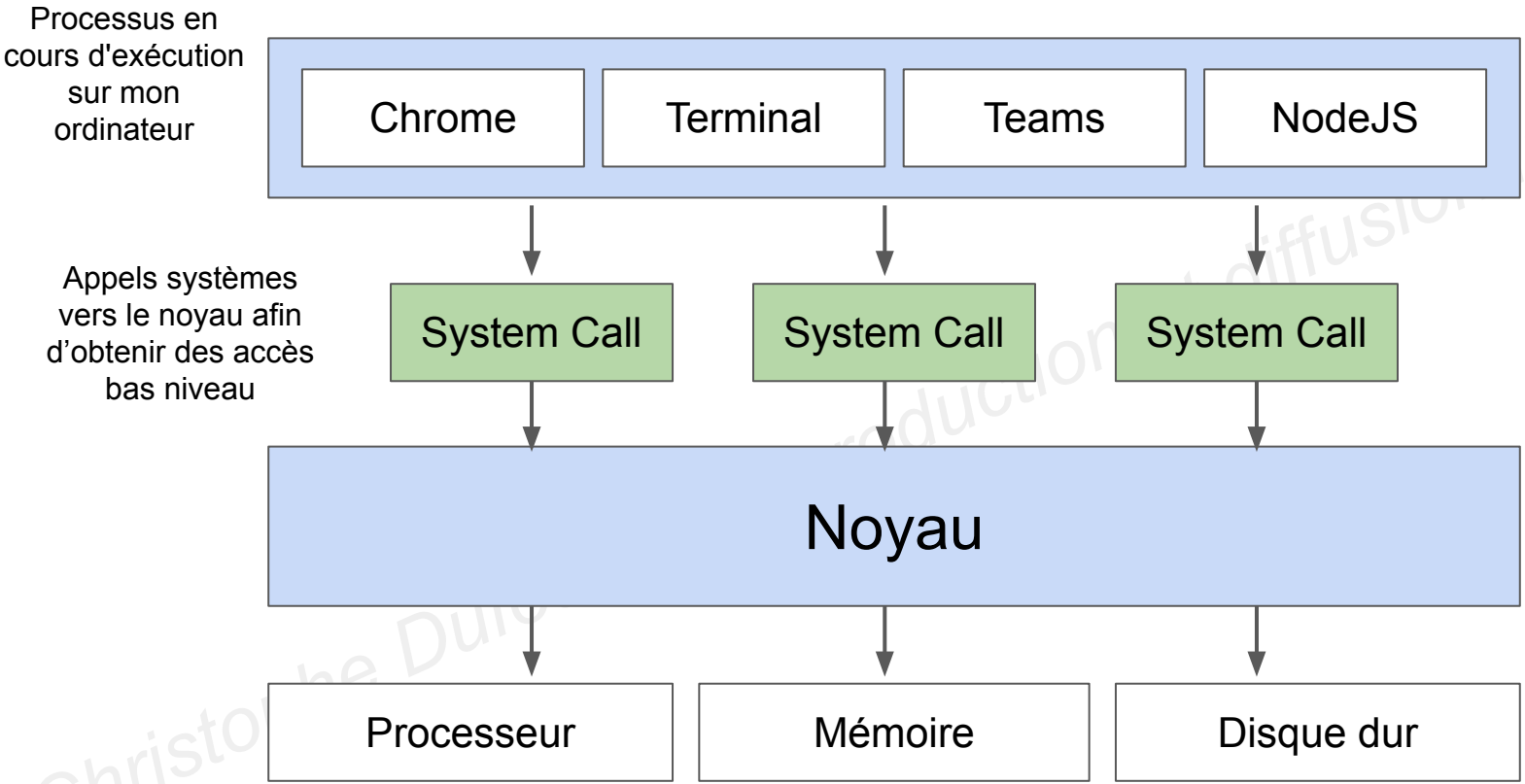
L’écosystème Docker

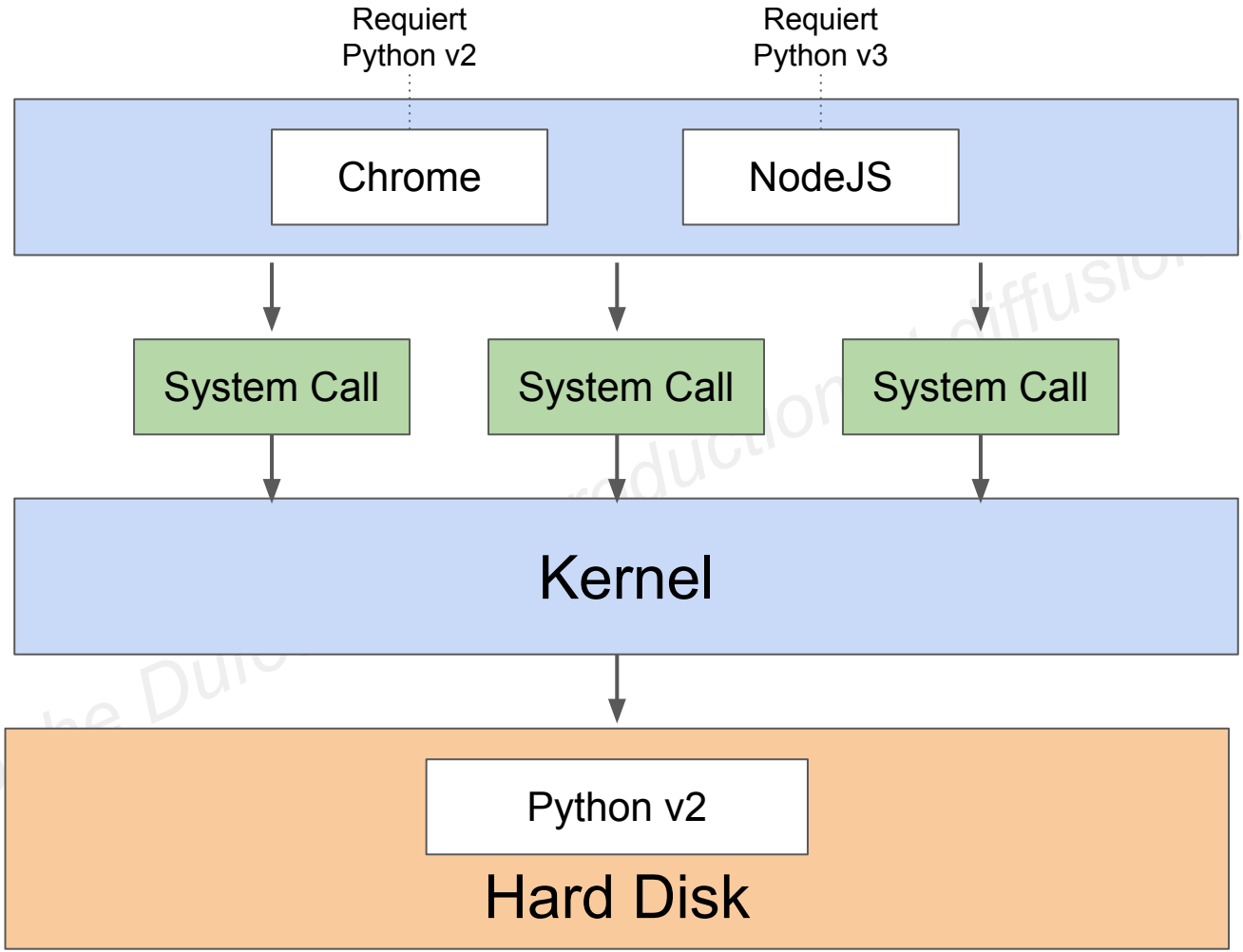


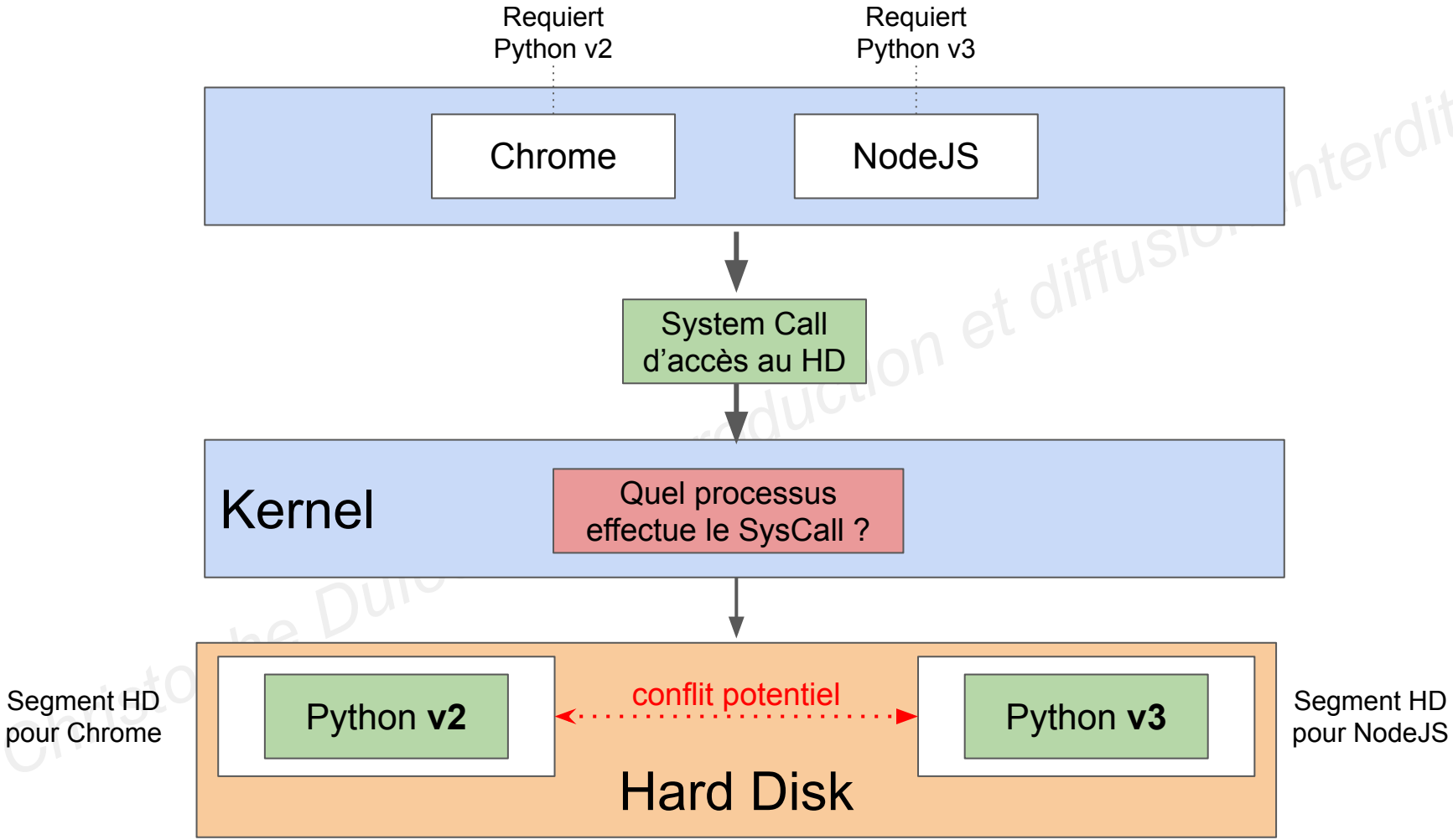
Fichier unique incluant toutes les dépendances et la configuration requises pour exécuter un programme

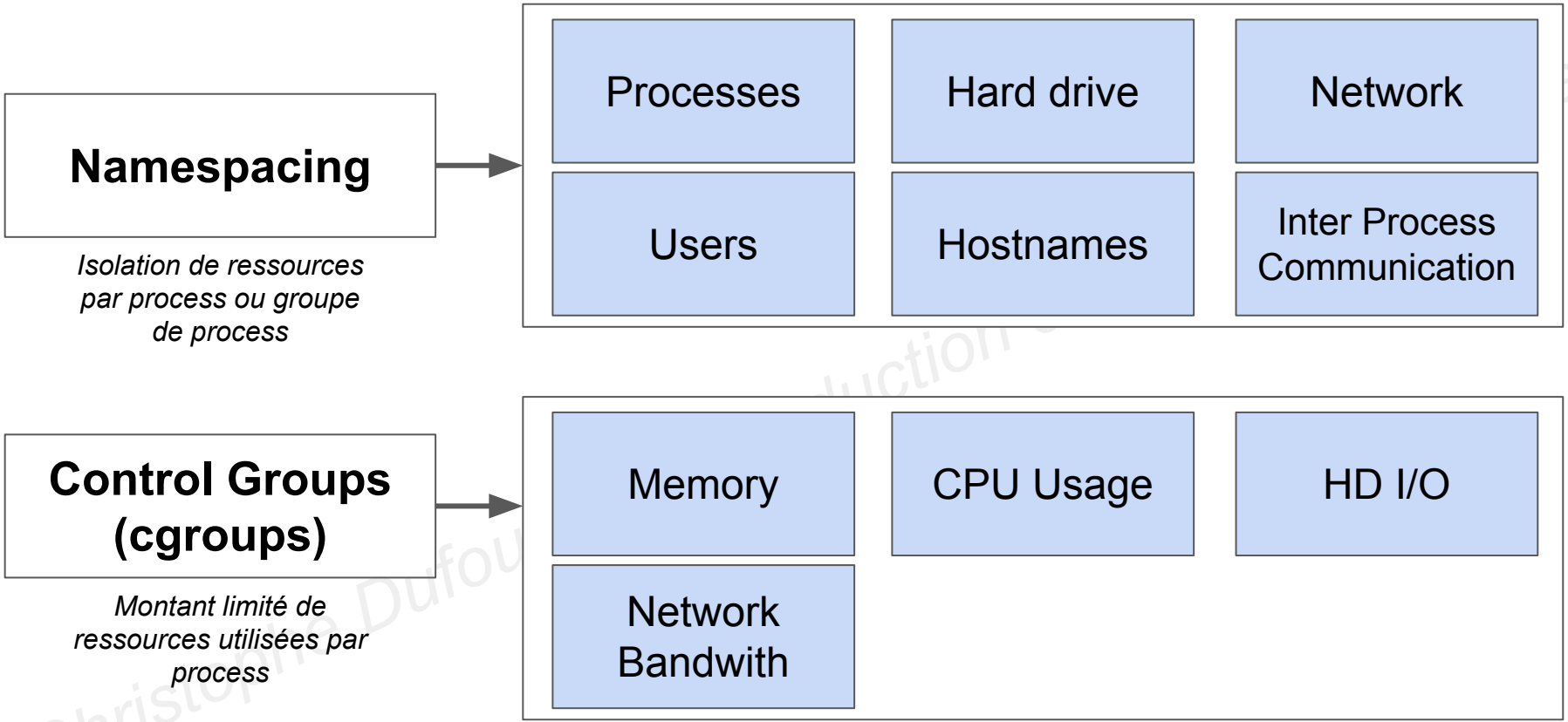
Instance d’une image. Execute un programme

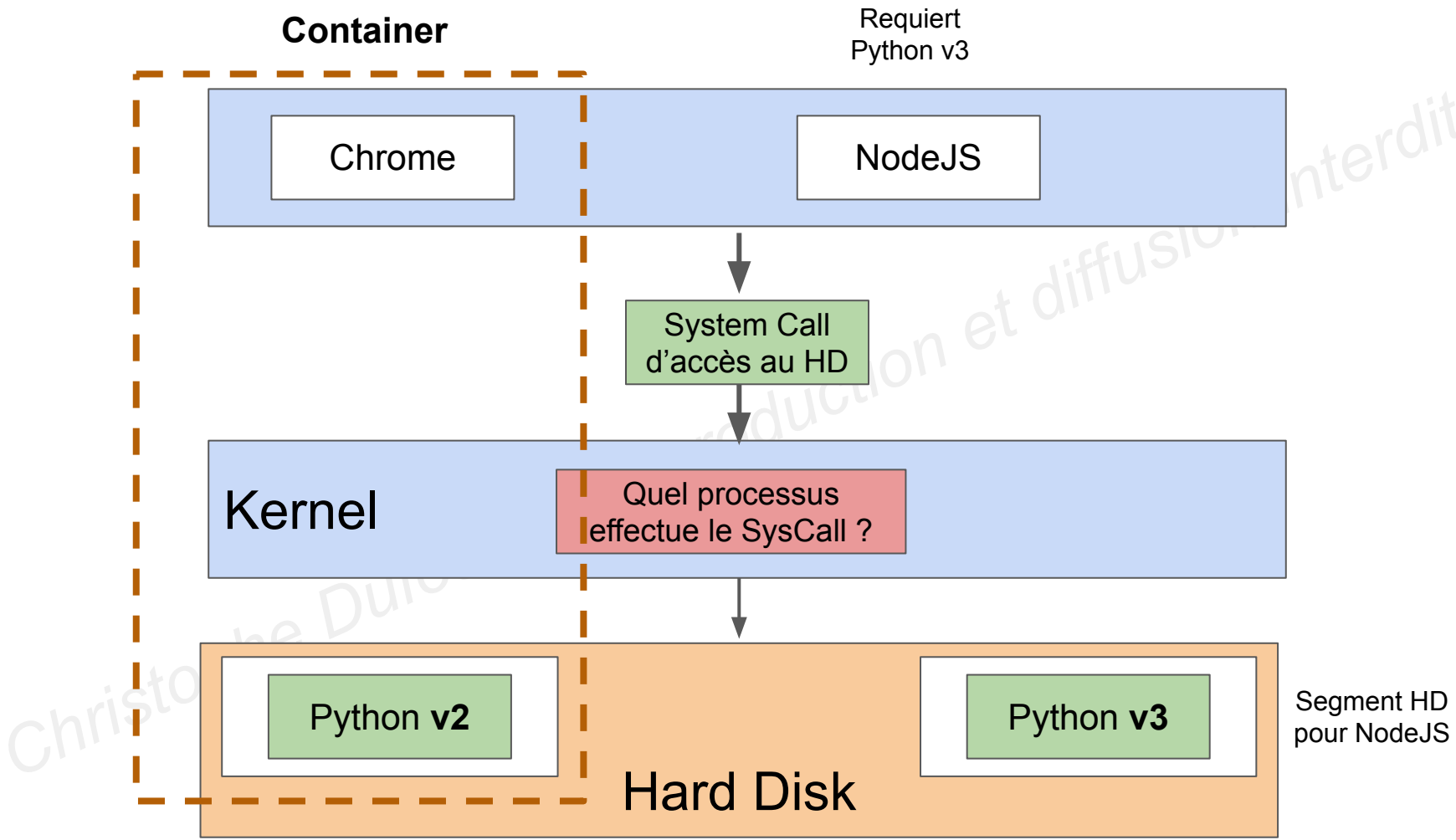


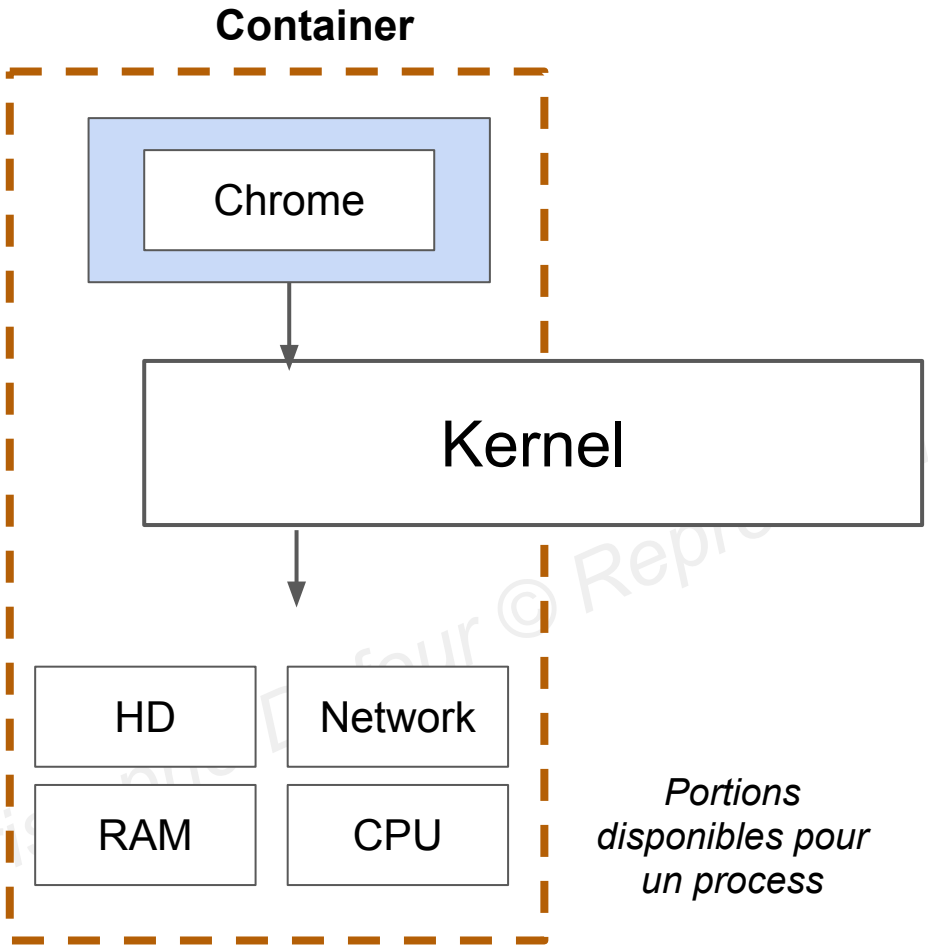




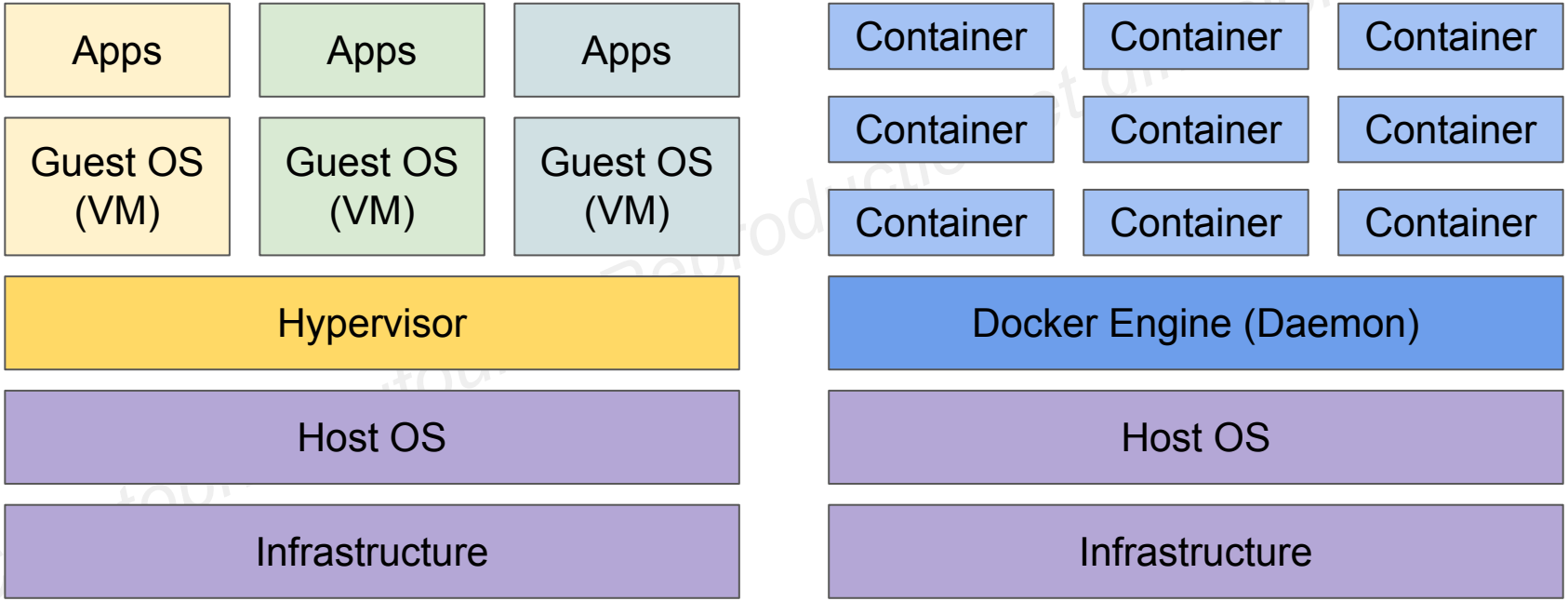








VM vs Docker



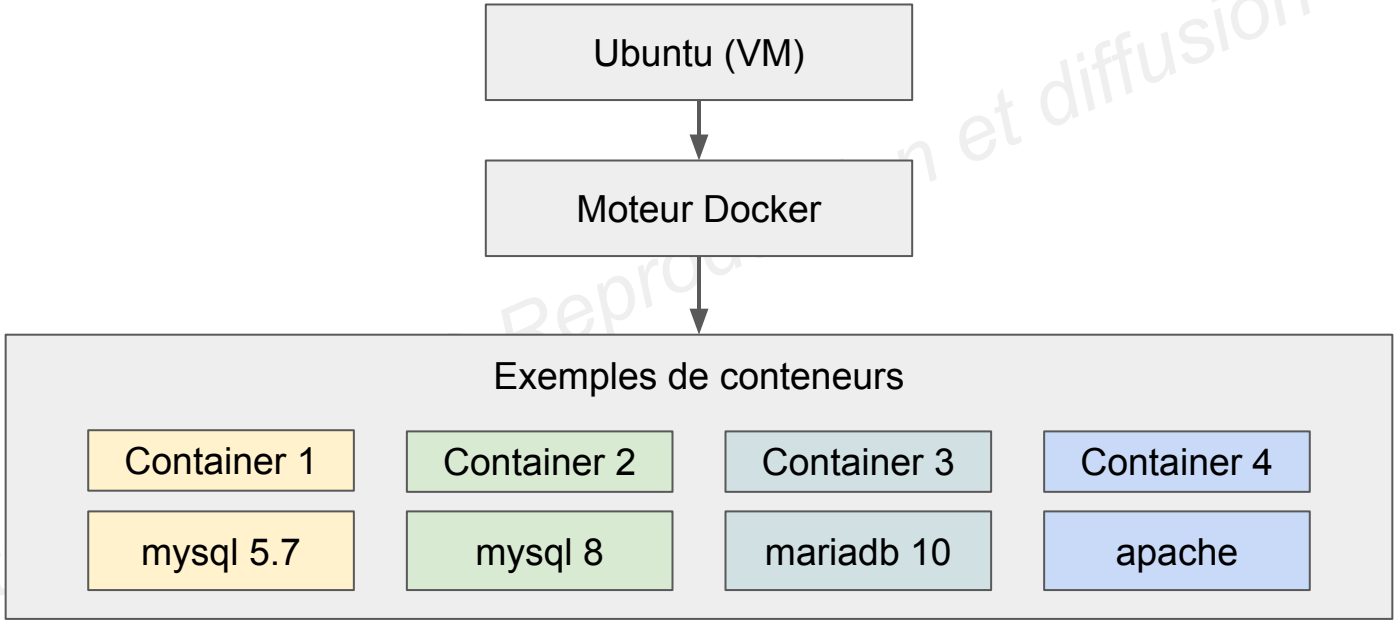
VM vs Docker

Isolation des processus au niveau matériel	Isolation des processus au niveau du système d'exploitation
Chaque machine virtuelle a un système d'exploitation distinct	Chaque conteneur peut partager le système d'exploitation
Démarrage en quelques minutes	Démarrage en quelques secondes
Poids en Go	Poids en Ko/Mo
Les VM prêtes à l'emploi sont difficiles à trouver	Les conteneurs Docker pré-construits sont facilement disponibles
Les VM peuvent facilement se déplacer vers un nouvel hôte	Les conteneurs sont détruits et recréés au lieu de se déplacer
Temps de création relativement long	Création en quelques secondes
Plus d'utilisation des ressources	Moins d'utilisation des ressources

VM vs Docker - que privilégier ?

- Docker s'il s'agit de faire tourner un grand nombre d'applications - potentiellement dans différentes versions - sur un nombre de serveurs limité
- VM s'il s'agit de faire tourner un nombre d'applications limité sur un grand nombre de serveurs (différents OS)
- VM si l'isolation maximale (sécurité) est prioritaire
- Dans la vie réelle, on trouve généralement des approches "hybrides" avec des conteneurs tournant dans des VMs

VM + Docker: approche hybride



Installation sur Ubuntu

- *Mise à jour apt*
  sudo apt update
- *Installation de paquets utiles*
  sudo apt install -y apt-transport-https ca-certificates curl gnupg lsb-release
- *Ajout de la clé GPG officielle de Docker*
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
- *Ajout du dépôt stable aux sources*
  echo "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
- *Mise à jour apt*
  sudo apt update
- *Installation des paquets Docker*
  sudo apt-get install -y docker-ce docker-ce-cli containerd.io
- *Ajout de l'utilisateur courant au groupe docker*
  sudo usermod -aG docker $USER

Docker run - exemple


```
vagrant@ubuntu-bionic:~$ docker run busybox echo Ciao !
Ciao !
vagrant@ubuntu-bionic:~$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS
1161506f27e5   busybox     "echo Ciao !"            39 seconds ago   Exited
```

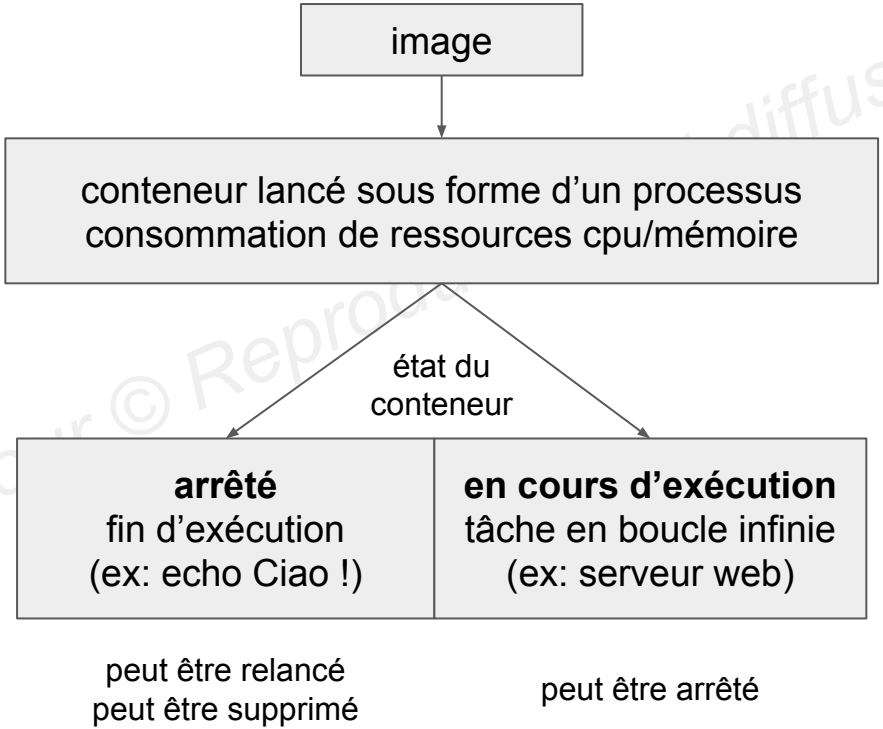
Docker run - exemple

docker run	=	docker create	+	docker start

```
vagrant@ubuntu-bionic:~$ docker create busybox echo Coucou !
b189fa6a73fe4ca09ea51e3e19e1405d19befc5e2de781e3b5d0f366a3a53376
vagrant@ubuntu-bionic:~$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED          STATUS
b189fa6a73fe   busybox     "echo Coucou !"          4 seconds ago    Created
```

```
vagrant@ubuntu-bionic:~$ docker start -a b189
Coucou !
vagrant@ubuntu-bionic:~$ docker start -a b189
Coucou !
vagrant@ubuntu-bionic:~$ docker ps -a
CONTAINER ID   IMAGE       COMMAND                  CREATED              STATUS
b189fa6a73fe   busybox     "echo Coucou !"          About a minute ago   Exited
```

Fonctionnement



Commandes - gestion des conteneurs

Commande	**Description**
docker create *image* [*commande*] docker run *image* [*commande*]	créer le conteneur = create + start
docker start *conteneur…* docker stop *conteneur…* docker kill *conteneur…* docker restart conteneur...	démarre le conteneur arrêt “doux” (SIGTERM) arrêt “brutal” (SIGKILL) = stop + start
docker pause *conteneur…* docker unpause *conteneur*...	suspend l’exécution du conteneur repren l’exécution du conteneur
docker rm [-f] *conteneur*...	détruit le conteneur = docker kill + docker rm

Commandes - inspection des conteneurs

Commande	**Description**
docker ps docker ps -a	affiche les conteneurs en cours d'exécution affiche tous les conteneurs (all)
docker logs *conteneur*	affocje la sortie du conteneur (stdout + stderr)
docker top *conteneur*	liste les processus en cours dans le conteneur
docker diff *conteneur*	montre les différences avec l'image
docker inspect *conteneur*...	affiche les infos bas- niveau (format json)

Commandes - interaction avec les conteneurs

Commande	**Description**
docker attach *conteneur*	attache le terminal actif au conteneur (stdin/out/err)
docker cp *conteneur:chemin hôte:chemin* docker cp *hôte:chemin conteneur:chemin*	copie des fichiers depuis le conteneur copie des fichiers dans le conteneur
docker export *conteneur*	exporte le contenu du conteneur (archive tar)
docker exec *conteneur args...*	exécute une commande dans le conteneur
docker wait *conteneur*	attend que le conteneur se termine (exit code)
docker commit *conteneur image*	commit une nouvelle image docker (snapshot du conteneur)

Commandes - gestion des images

Commande	**Description**
docker images docker history *image* docker inspect *image*	liste toutes les images locales affiche l'historique de l'image (ses calques) affiche les infos bas-niveau (format json)
docker tag *image tag*	tague une image
docker commit *conteneur image*	crée une image à partir d'un conteneur
docker import *chemin*	créer une image à partir d'un tarball
docker rmi *image...*	supprime l'image

Commandes - transfert d'image

Commande	Description
docker pull *repo*[:tag]... docker push *repo*[:tag]... docker search *text*	télécharge une image/repo depuis un registre envoie une image/repo vers un registre recherche une image dans le registre officiel
docker login docker logout	se connecte à un registre se déconnecte d'un registre
docker save *repo*[:tag]... docker load	exporte une image/repo en tant que tarball charge une image depuis un tarball

Créer ses propres images

- Docker permet d'utiliser des images faites par d'autres (depuis le registre docker.hub par exemple)
- On peut aussi créer des images personnalisées
- Avantages: conteneurs "sur-mesure" disposant de fonctionnalités precises
- Etapes de création d'une image
  - rédaction d'un Dockerfile
  - construction de l'image (docker build -f dockerfile)

Anatomie d'une image: les couches (layers)

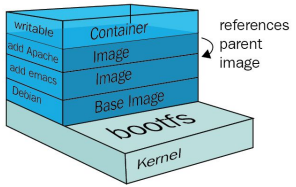


Image Couche v3 démarrée comme conteneur, accessible par les utilisateurs

0a4114a14ded

Lecture/écriture
Couche conteneur

1. Image Couche v1 démarrée comme conteneur
2. Serveur http configuré et démarré ("yum install…")
3. Nouvelle couche v2 produite(committed)

bf2feb98a4f5

Image en lecture seule
Couche v2

1. Image de base (docker.io/centos) démarrée comme conteneur
2. Paquets de l'image de base mise à jour par "yum update"
3. Nouvelle couche v1 porduite (committed)

d62409846594

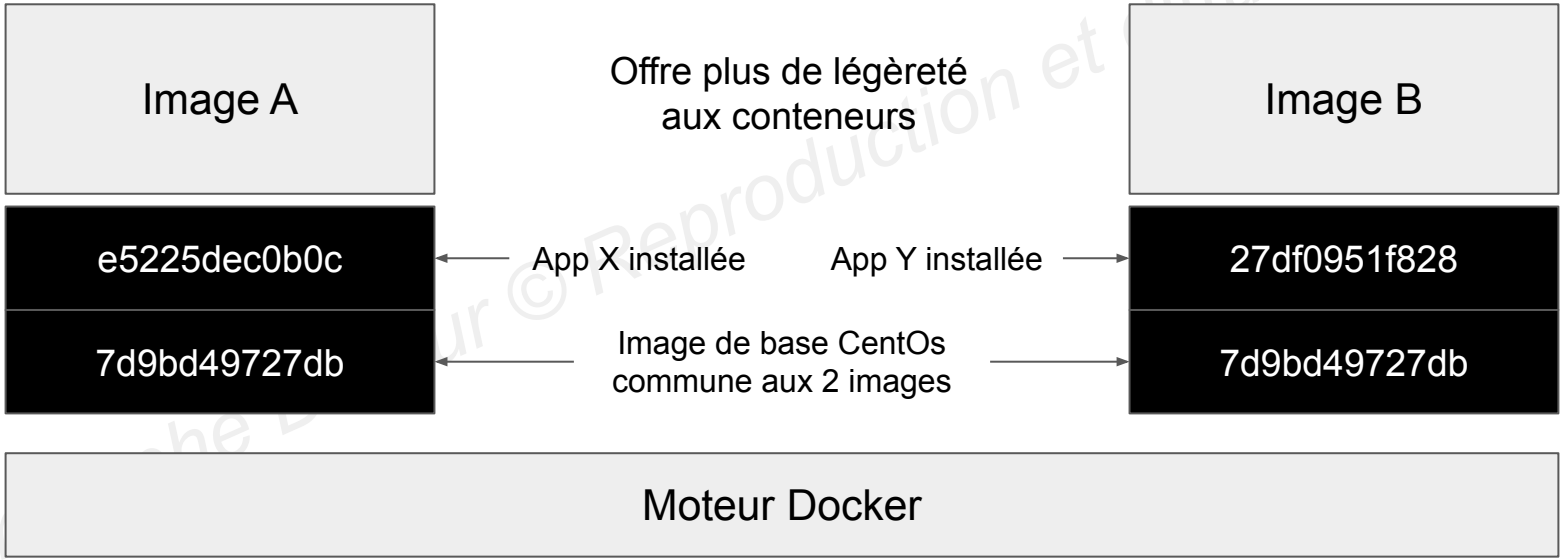
Image en lecture seule
Couche v1

Image CentOs téléchargée depuis Docker Hub par "docker pull".
Repo: docker.io/centos

7d9bd49727db

Image de base

Mutualisation des couches



Dockerfile: création


```
FROM alpine:3.12
RUN apk update
RUN apk add redis
CMD ["redis-server"]
```

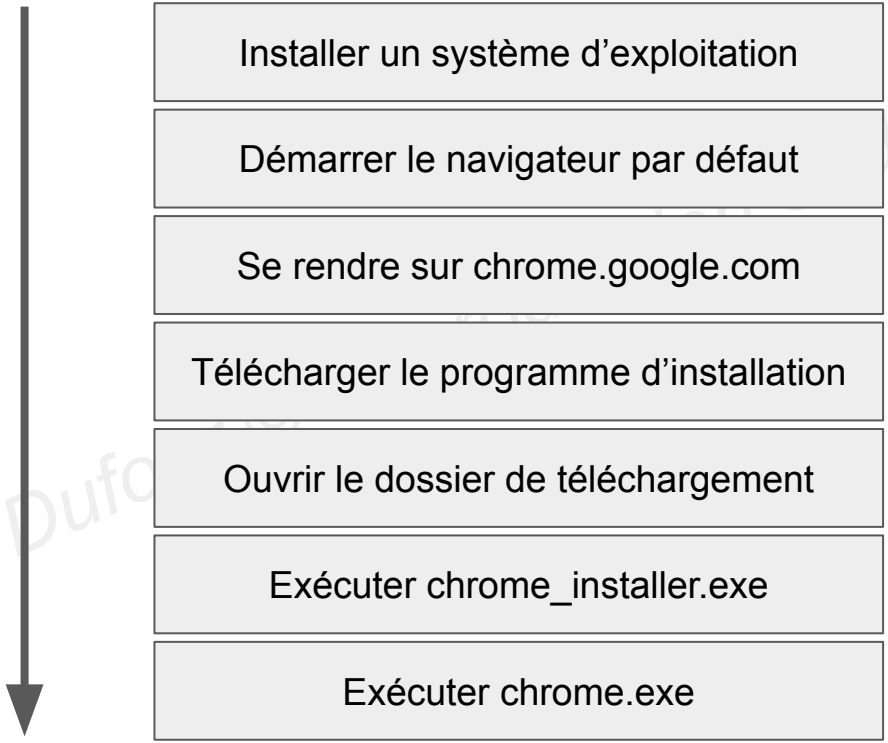
Exercice

Créer une image démarrant un serveur Redis et incluant le logiciel bash

Exercice: solution

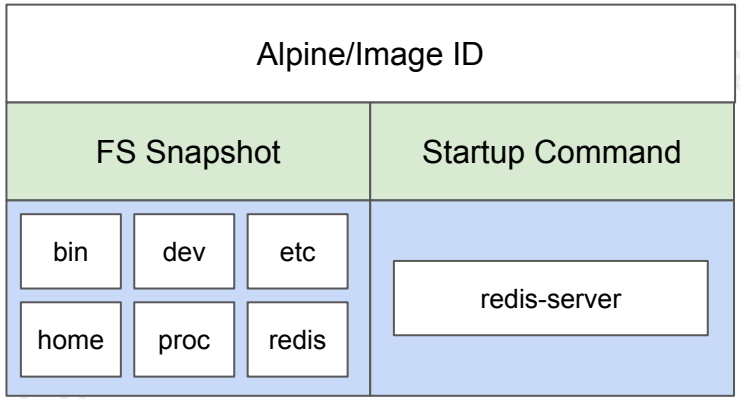
FROM	alpine
RUN	apk add --update redis
RUN	apk add bash
CMD	[“redis-server”]

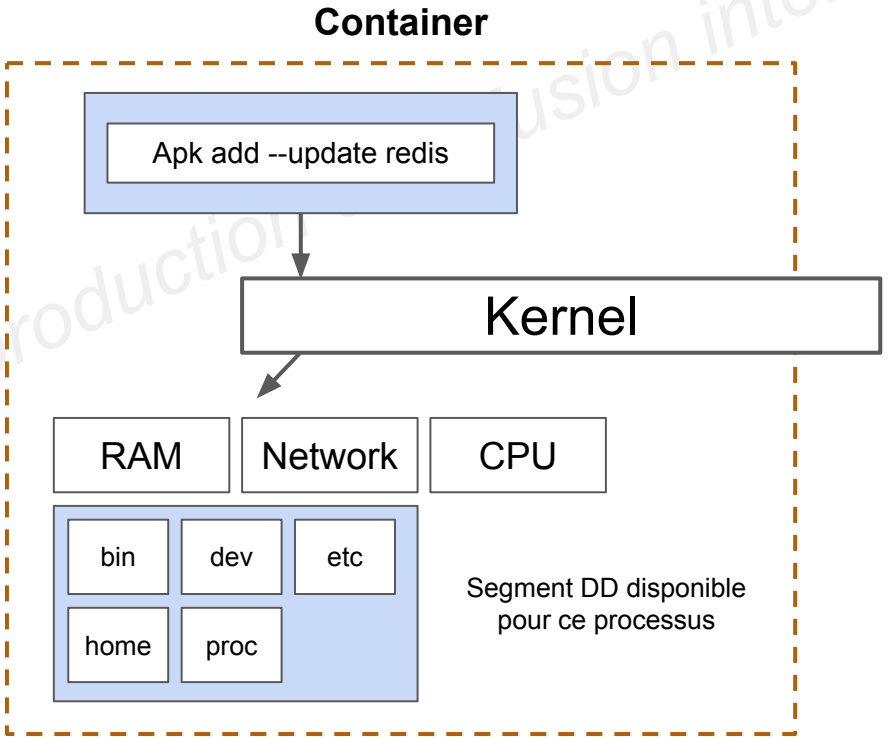
Exemple d’installation de chrome



Build process

FROM	alpine
RUN	Apk add --update redis
CMD	[“redis-server”]





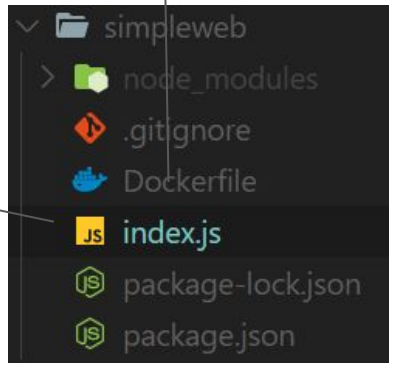
Exercice

“Conteneuriser” un serveur nodejs/express

Exercice: solution

```
import express from 'express';

const app = express();

app.get('/', (req, res) => {
  res.send('Coucou, je suis un serveur nodejs/express');
})

app.listen(3000, () => {
  console.log('Serveur écoutant le port 3000...');
})
```

```
FROM node:14.17-alpine

COPY index.js .
COPY package.json .

RUN npm install

CMD ["node", "index.js"]
```



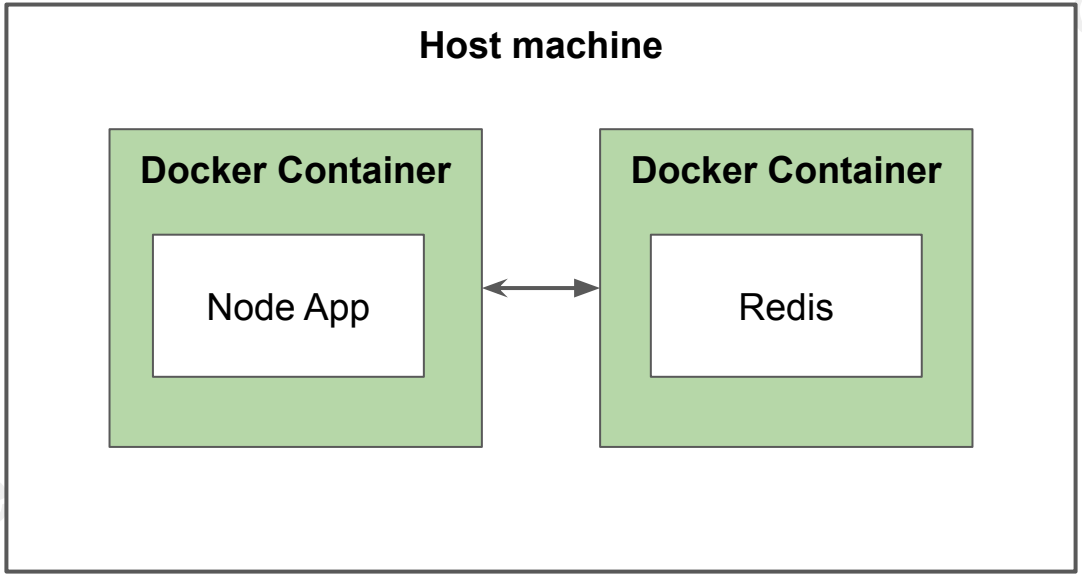

```
docker build . -t opusidea/simpleweb:v1
```

```
docker run --rm -p 3000:3000 opusidea/simpleweb:v1
```

Application multi-conteneurs

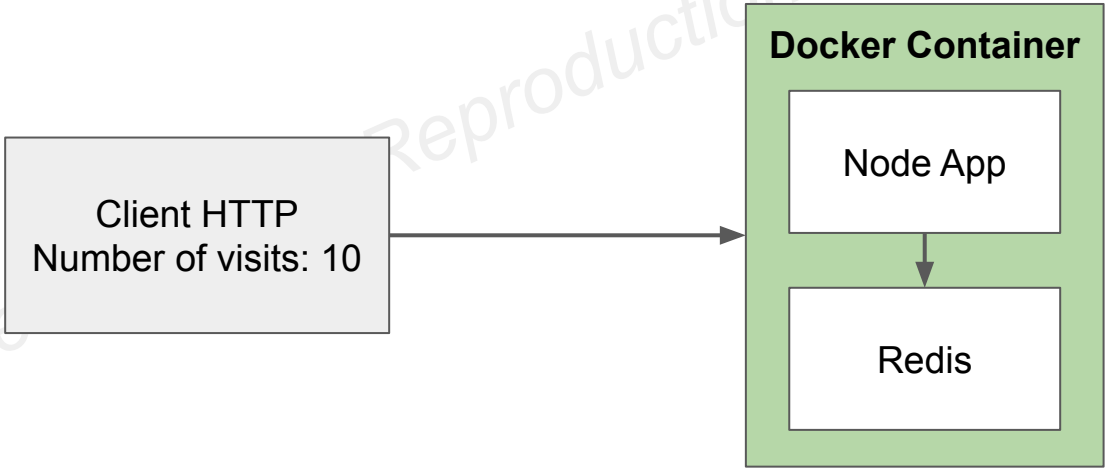
Créons une application nodejs communicant avec un serveur Redis

Application multi-conteneurs



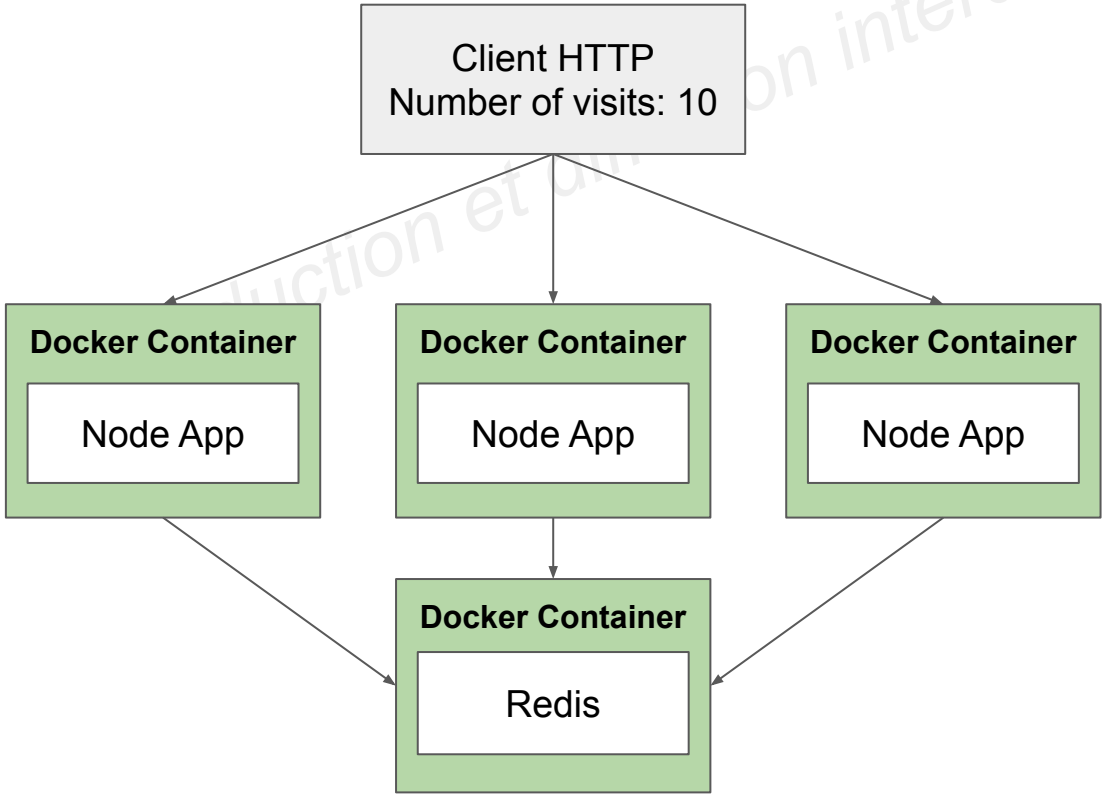
Application multi-conteneurs

Les serveurs nodejs et redis sont encapsulés dans le même conteneur. Ce manque de découplage est dangereux en cas de problème rencontré par une application. Limite la mise à échelle (scalability).



Application multi-conteneurs

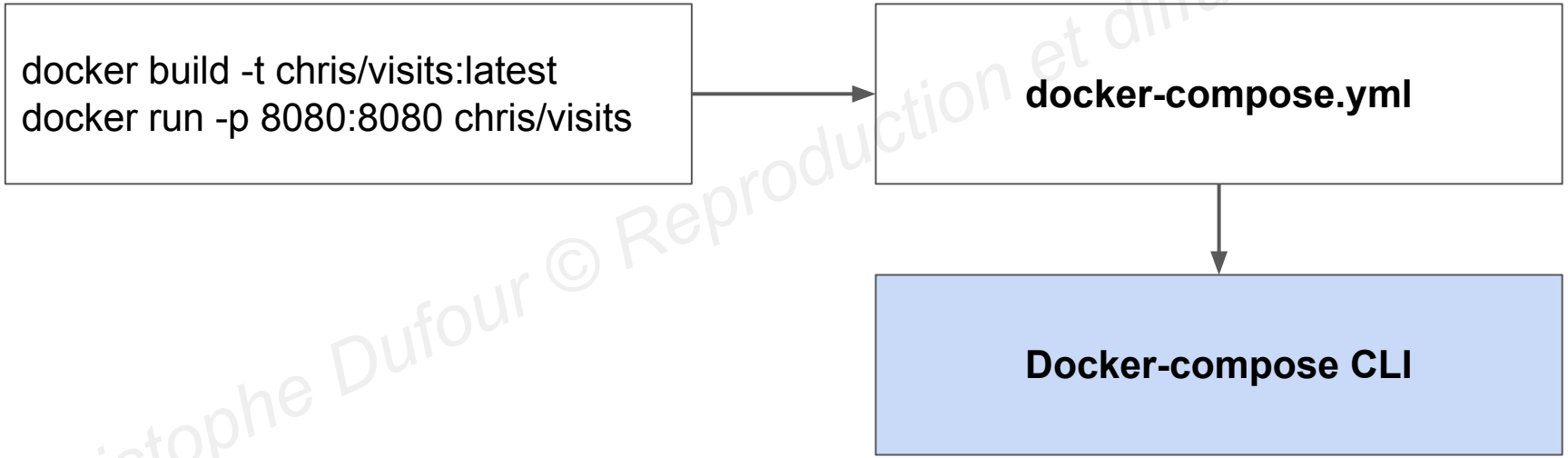
Les serveurs nodejs et redis sont encapsulés dans des conteneurs différents. Ce découplage permet une meilleure résilience à la faille, une mutualisation de la couche donnée, il favorise aussi la mise à échelle (scalability)



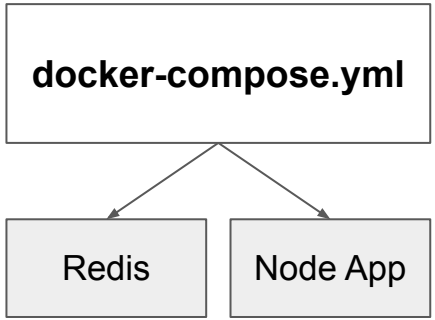
Docker Compose

- CLI distinct du docker CLI
- Utilisé pour gérer de multiples conteneurs en même temps
- On parle se service
- Permet de configurer des infrastructures multi-services complexes
- Fichier texte au format YAML (diffusable, partageable, versionable, etc.)
- Toute les fonctionnalités Docker sont configurables:
  - mapping de port
  - variables d'environnement
  - création de volume
- Docker-compose crée un réseau privé auquel les conteneurs (services) sont attachés

Docker Compose



Docker Compose

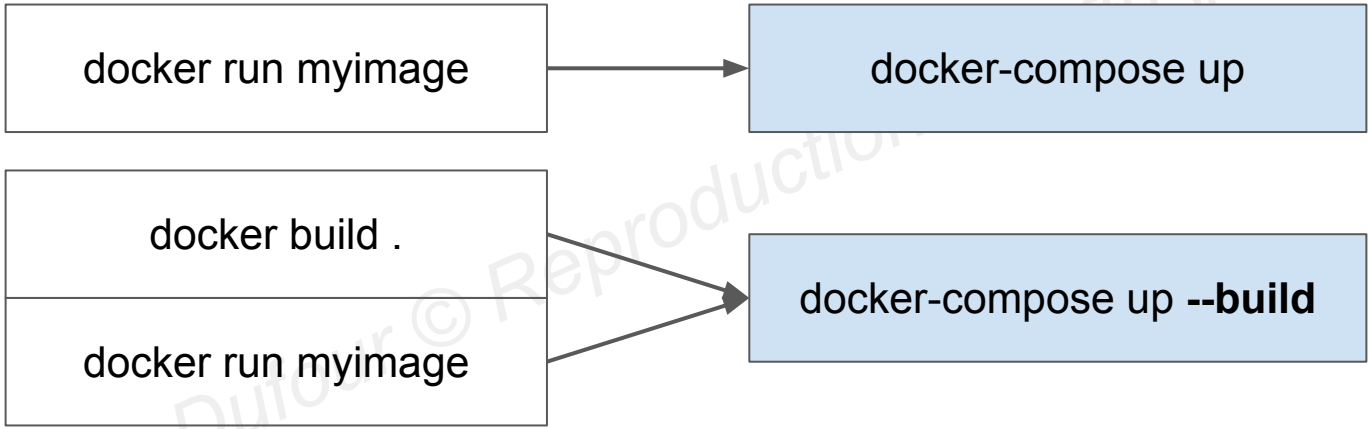



```yaml
version: '3'
services:

  redis-app: # dns name pour l'application node
    image: 'redis'

  node-app:
    build: . # cible le Dockefile (dossier courant)
    ports:
      - "4001:8081"
```

```yaml
const client = redis.createClient({
    host: 'redis-app',
    port: 6379
});
```

Le ficher docker-compose définit deux services.
Le service node-app est joignable de l'extérieur sur le port 4001.
Le service redis-app n'est pas joignable de l'extérieur. L'application node y accède par le nom du service (host: redis-app)

Docker Compose



Docker Compose

Démarrage en tâche de fond

```
docker-compose up -d
```

Arrêts des conteneurs

```
docker-compose down
```

Liste (depuis dossier contenant .yml)

```
docker-compose ps
```

Docker Volumes

```
docker run -p 3000:3000 -v /app/node_modules -v $(pwd):/app cid
```

Docker Network

```
docker network create mynetwork --subnet 172.50.0.0/16
```

```
docker run -tid --name toto--network mynetwork redis:5
```

```
docker run -tid --name tata --network mynetwork redis:5
```

TP “Complex”: infrastructure multi-services