

# Devops

## Docker

2023/2024

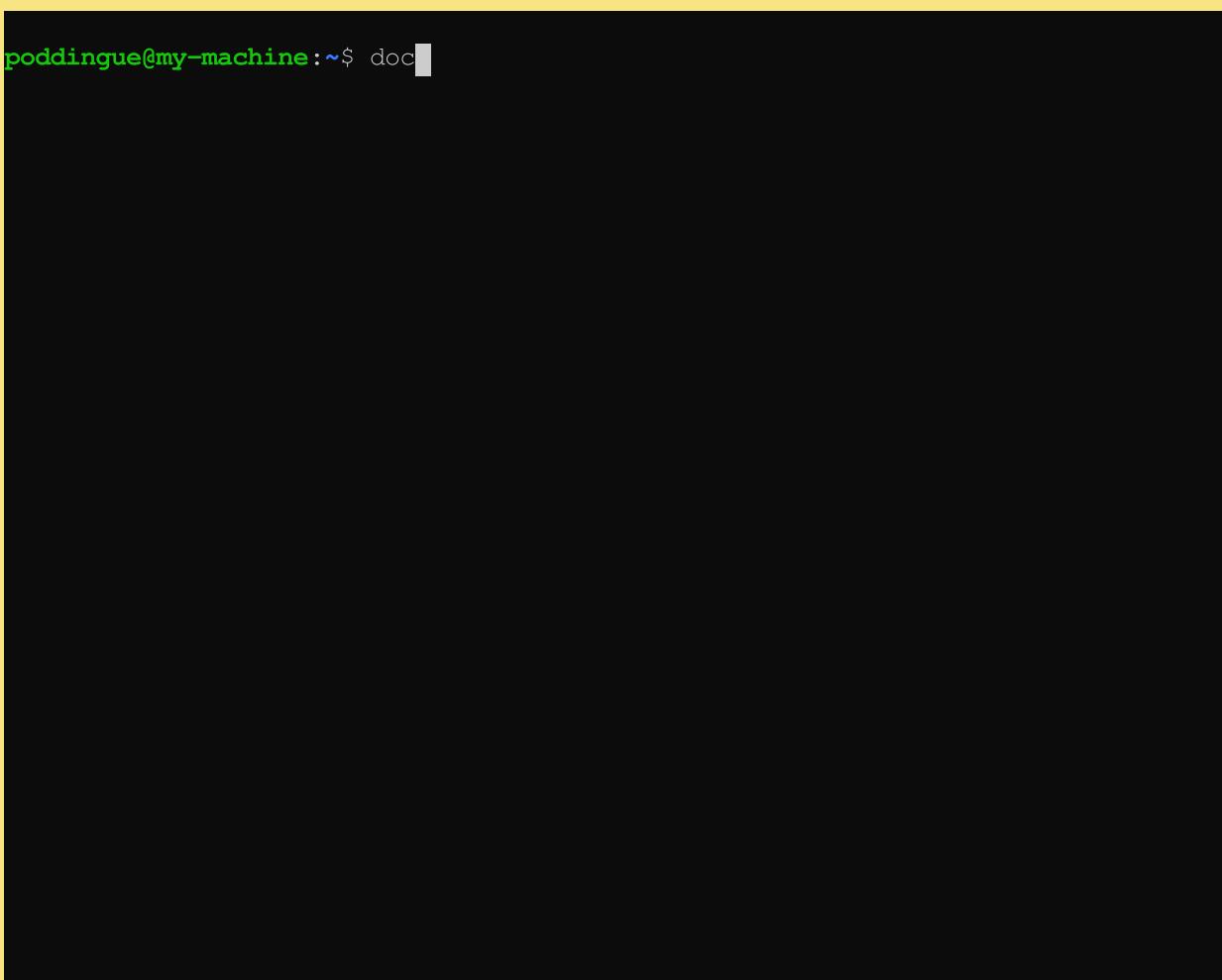


- Présentation disponible à l'adresse: <https://gounthar.github.io/gounthar/cours-devops-docker/updated-dependencies>
- Version PDF de la présentation :  [Cliquez ici](#)
- Contenu sous licence [Creative Commons Attribution 4.0 International License](#)
- Code source de la présentation: <https://github.com/gounthar/gounthar/cours-devops-docker>

# Comment utiliser cette présentation ?

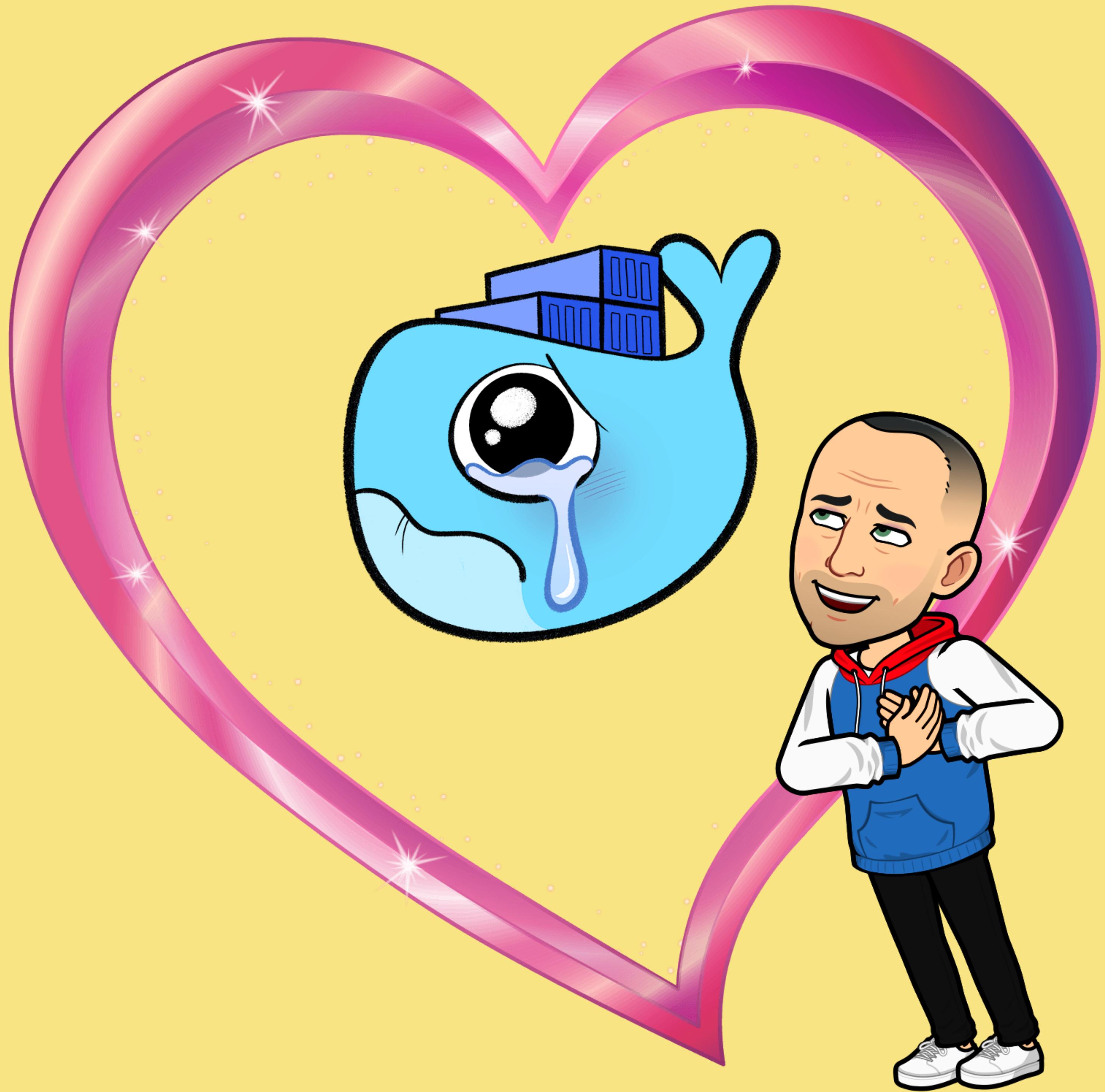
- Pour naviguer, utilisez les flèches en bas à droite (ou celles de votre clavier)
  - Gauche/Droite: changer de chapitre
  - Haut/Bas: naviguer dans un chapitre
- Pour avoir une vue globale : utiliser la touche "o" (pour "**Overview**")

# Bonjour !



poddinque@my-machine:~\$ doc

La suite: vers le bas ↓





## Bruno VERACHTEN

- Sr Developer Relations chez CloudBees pour le projet Jenkins 
- Me contacter :
  -  gounthar@gmail.com
  -  gounthar
  - <https://bruno.verachten.fr/>
  -  Bruno Verachten
  -  @poddigue

Et vous ?



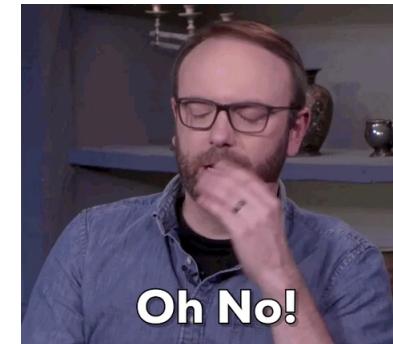
# A propos du cours

- Première itération d'une découverte Docker dans le cadre du DevOps
- Contenu entièrement libre et open-source
- Méchamment basé sur le travail de Damien Duportal et Amaury Willemant
  - N'hésitez pas ouvrir des Pull Request si vous voyez des améliorations ou problèmes: sur cette page (😉 wink wink)

# Calendrier

 Work in progress... 

# Evaluation



- Pourquoi ? s'assurer que vous avez acquis un minimum de concepts
- Quoi ? Une note sur 20, basée sur une liste de critères exhaustifs
- Comment ? Un projet GitHub (public) à me rendre (timing à déterminer ensemble)

# Plan

- Intro
- Base
- Containers
- Images
- Fichiers, nommage, inspect
- Volumes
- Réseaux
- Docker Compose
- Bonus

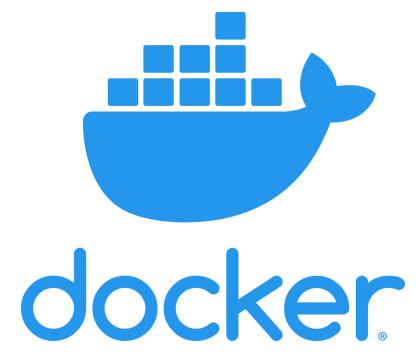
La suite: vers la droite ➔

Docker  
"La Base"

UNLOADING  
AREA

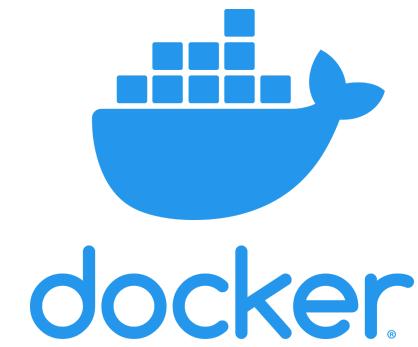


# Pourquoi ?



🤔 Quel est le problème ?

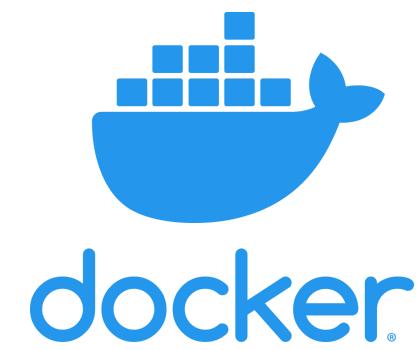
# Pourquoi commencer par un problème ?



🤔 Commençons plutôt par une définition:

*Docker c'est ...*

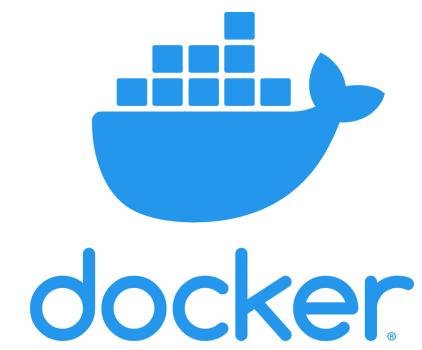
# Pourquoi commencer par un problème ?



🤔 Définition quelque peu datée (2014):

*Docker is ...*

# Pourquoi commencer par un problème ?

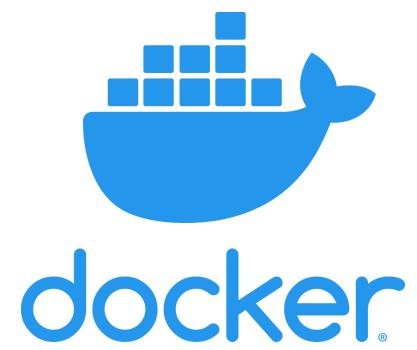


🤔 Définition quelque peu datée (2014):

*Docker is a toolset for Linux containers designed to 'build, ship and run' distributed applications.*

<https://www.infoq.com/articles/docker-future/>

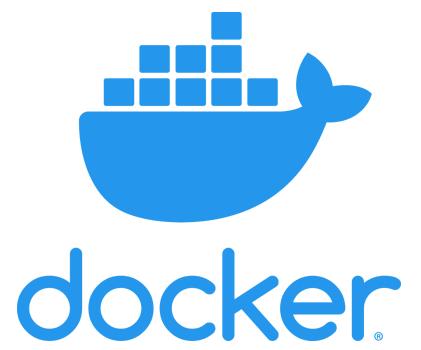
# Linux containers ?



🤔 Nous voilà bien...

*C'est quoi un container?*

# Linux containers ?



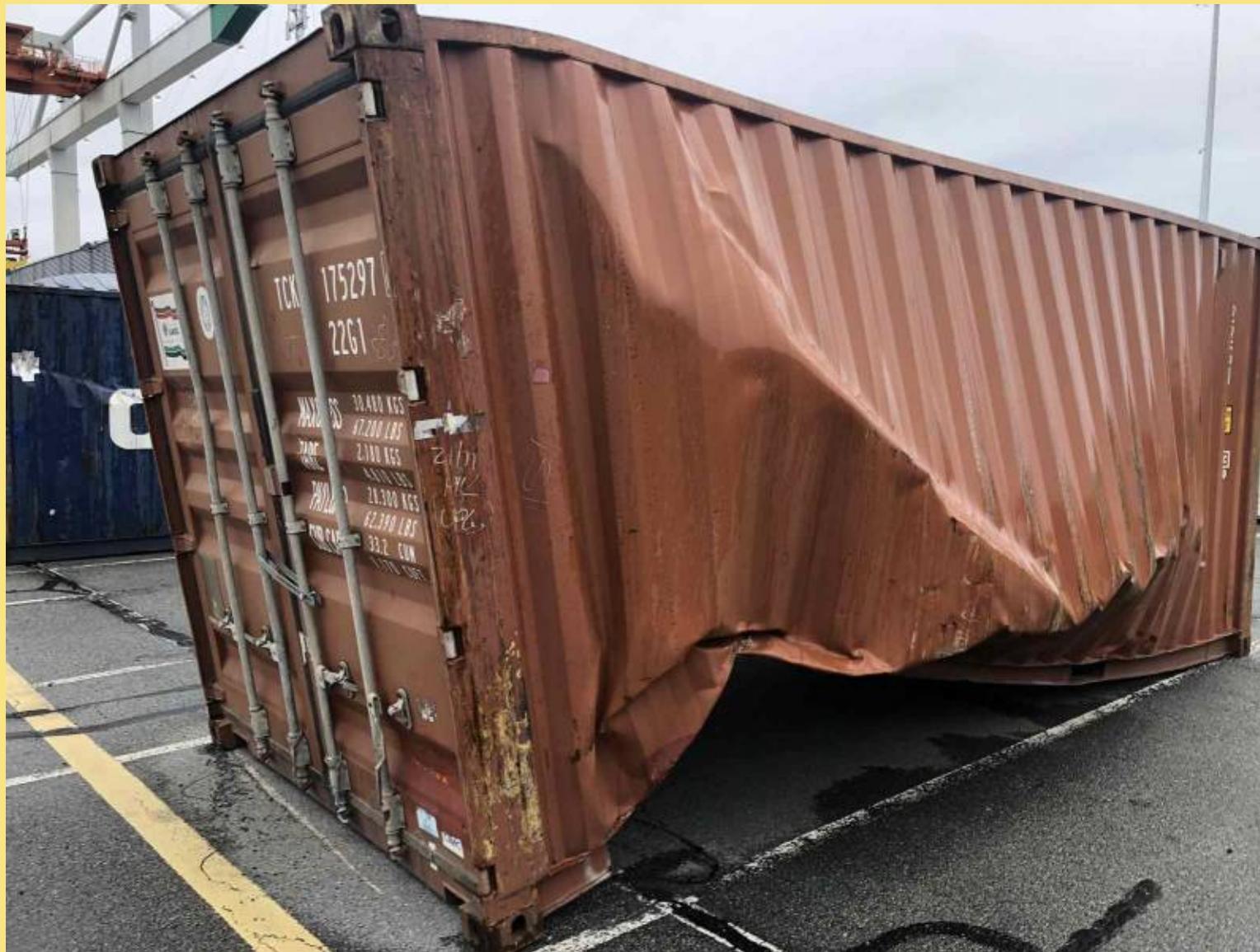
🤔 Nous voilà bien...

*C'est quoi un container?*

Vous voulez la version enfant de 5 ans? Je ne crois pas...

# Docker est vieux

10 ans déjà...



# Docker est vieux

Mais moi aussi...



## Docker Pirates ARMed with explosive stuff

Roaming the seven seas in search for golden container plunder.

[HOME](#)  
[GETTING STARTED](#)  
[DOWNLOADS](#)  
[FAQ](#)  
[COMMUNITY](#)  
[ABOUT US](#)  
[CREW](#)  
[DONATE](#)

© 2020 Hypriot

[Legal Notice](#)

[Edit this blog on GitHub](#)

## Docker on Raspberry Pi Workshop in Brussels

Thu, Mar 17, 2016

A couple of weeks ago we have been invited by the fine folks over at the Docker User Group in Brussels to help conduct a workshop. And of course this workshop was about Docker. Still it was not your ordinary Docker workshop.

It was special because instead of being a workshop about Docker on big servers it was about Docker on really small ARM devices. The very same devices that power the upcoming IoT revolution.

Turns out it is really amazing what you can do with Docker on those tiny machines.



# Docker on arm

Aucune raison que ça soit réservé aux puissants! La révolution vaincra!

The screenshot shows a news article on the LinuxGizmos.com website. The header features a green banner with a penguin icon and the text "LinuxGizmos.com". Below the banner is a navigation bar with links for All News, Boards, Chips, Devices, Software, LinuxDevices.com Archive, About, Contact, and Subscribe. To the right of the navigation bar is a "Follow LinuxGizmos:" section with social media icons for Twitter, Facebook, Pinterest, and RSS, and a link to "get email updates". A search bar is also present. The main content area features a news article titled "Open source ResinOS adds Docker to ARM/Linux boards" by Eric Brown, published on Oct 17, 2016, with 4,449 views. The article includes social sharing icons for Twitter, Facebook, LinkedIn, Reddit, Pinterest, and Email. It discusses Resin.io's ResinOS 2.0 distro for running Docker containers on Linux IoT devices. A diagram illustrates the ResinOS 2.0 architecture, showing a "resinOS device (development mode image)" connected to a "developer machine" via a "Local network". The developer machine contains a "User Device Model", "ResinOS language interface", and "ResinOS API". The resinOS device contains "User Container 1", "User Container 2", "User Storage", "ResinOS", "ResinOS", "ResinOS", and "Linux kernel". A caption below the diagram says "(click image to enlarge)". At the bottom of the article, a note states: "When enterprise-level CIOs are asked to integrate embedded devices into their networks, their first question is usually 'Can they run Docker?' The answer is probably not... especially if they run on ARM processors." The right sidebar contains sections for "Search LinuxGizmos:", "Search LinuxGizmos.com + LinuxDevices Archive:", "LinuxGizmos Sponsor ads:", and "Top 10 trending posts...". The "Top 10 trending posts..." section lists various news items related to low-cost SBCs, mini PCs, and AIoT kits.

**Open source ResinOS adds Docker to ARM/Linux boards**

Oct 17, 2016 — by Eric Brown 4,449 views

[Twitter](#) [Facebook](#) [LinkedIn](#) [Reddit](#) [Pinterest](#) [Email](#)

Resin.io has spun off the Yocto-based OS behind its Resin.io IoT framework as a ResinOS 2.0 distro for running Docker containers on Linux IoT devices.

Resin.io, the company behind the Linux/Javascript-based Resin.io IoT framework for deploying applications as Docker containers, began spinning off the Linux OS behind the framework as an open source project over a year ago. The open source ResinOS is now publicly available on its own in a stable 2.0.0-beta.1 version, letting other developers create their own Docker-based IoT networks. ResinOS can run on 20 different mostly ARM-based embedded Linux platforms including the Raspberry Pi, BeagleBone, and Odroid-C1, enabling secure rollouts of updated applications over a heterogeneous network.

**ResinOS 2.0 architecture**  
(click image to enlarge)

When enterprise-level CIOs are asked to integrate embedded devices into their networks, their first question is usually "Can they run Docker?" The answer is probably not... especially if they run on ARM processors.

**Follow LinuxGizmos:**

[Twitter](#) [Facebook](#) [Pinterest](#) [RSS](#)

\* get email updates \*

**Search LinuxGizmos:**

**Search LinuxGizmos.com + LinuxDevices Archive:**

ENHANCED BY Google

**LinuxGizmos Sponsor ads:**

(advertise here)

**Top 10 trending posts...**

- » Libre Computer showcases low-cost SBC with PoE support
- » ASRock Industrial's present 4x4 BOX 7040 Series mini PCs
- » (Updated) The ZimaBlade is an upcoming low-cost x86 personal server
- » GOWIN & Andes Technologies collaborate and reveal 22nm SoC FPGA
- » Espressif Systems presents ESP32-S3-BOX-3 AIoT kit
- » Milk-V Duo is a \$9.00 RISC-V tiny embedded computer
- » SATA HATs support up to four drives on Raspberry Pi 4 or Rock Pi 4
- » Sipeed to launch RISC-V based Lichee Cluster 4A
- » New SBC powered by Allwinner T507-H processor
- » Orange Pi introduces a new Rockchip based Computer Module

**LinuxGizmos Sponsor ads:**

(advertise here)

**Follow LinuxGizmos or subscribe to our posts:**

[Twitter](#) [Facebook](#) [Pinterest](#) [RSS](#)



# On n'avait pas parlé d'un problème?



Intégrateur

Développeur

Testeuse

**DEV**

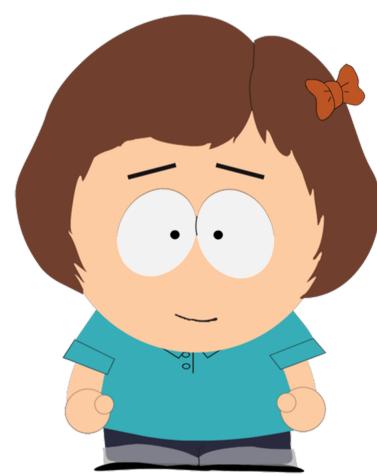
# On n'avait pas parlé d'un problème?



Intégrateur



Développeur



Testeuse



DBAs



Sys Admins



Ingé réseau

**DEV**

**OPS**

# On n'avait pas parlé d'un problème?

On veut du neuf ! Des trucs  
hypés !



Intégrateur



Développeur



Testeuse



DBAs



Sys Admins



Ingé réseau

**DEV**

**OPS**

# On n'avait pas parlé d'un problème?

On veut du neuf ! Des trucs  
hypés !



Intégrateur



Développeur



Testeuse

On veut un truc stable !



DBAs



Sys Admins

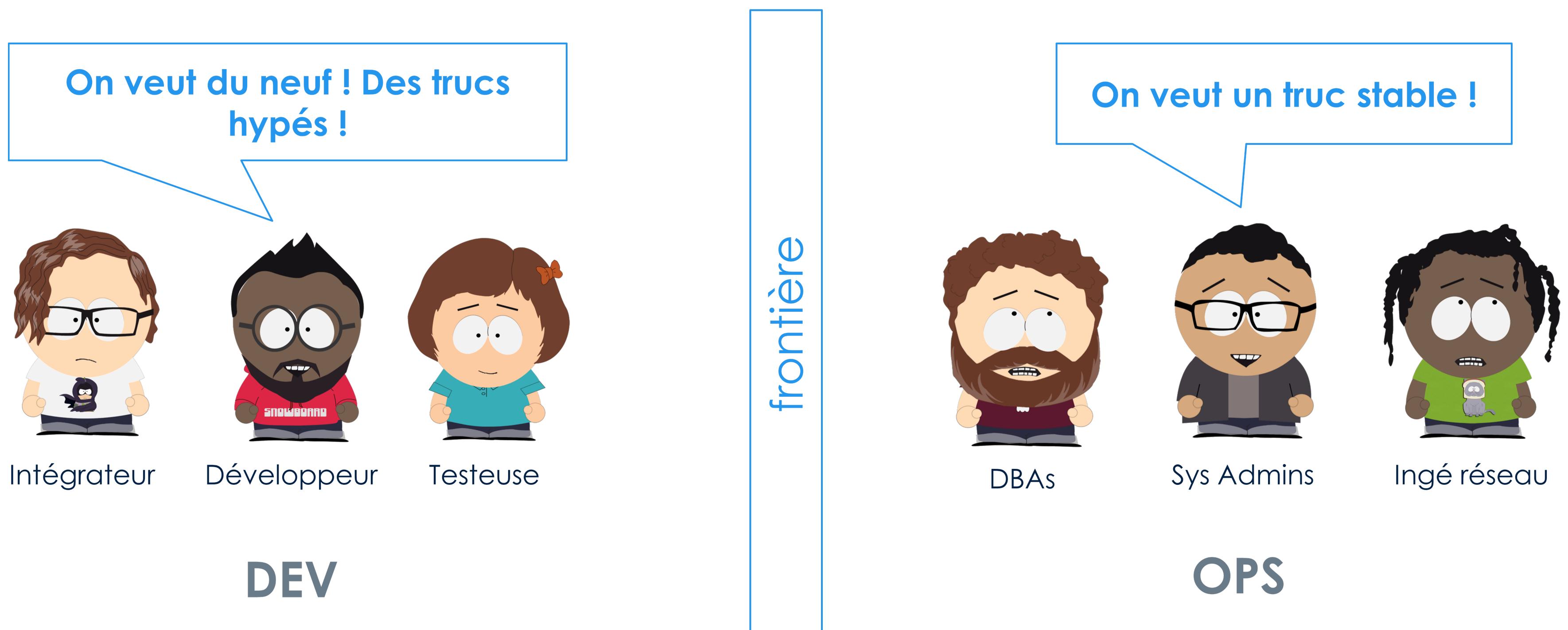


Ingé réseau

**DEV**

**OPS**

# On n'avait pas parlé d'un problème?



# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

*cc*

# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

*cc*

Tu peux mettre à jour les packages système steup  
? j'ai un truc à tester.

# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

*cc*

Tu peux mettre à jour les packages système steup  
? j'ai un truc à tester.

*nan*

# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

*cc*

Tu peux mettre à jour les packages système steup  
? j'ai un truc à tester.

*nan*

???

# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

*cc*

Tu peux mettre à jour les packages système steup  
? j'ai un truc à tester.

*nan*

???

*pas standard dsl*

# On n'avait pas parlé d'un problème?

Histoire vraie.

Hey salut !

*cc*

Tu peux mettre à jour les packages système steup  
? j'ai un truc à tester.

*nan*

???

*pas standard dsl*



# On n'avait pas parlé d'un problème?

	?	?	?	?	?	?	?
Static Website	?	?	?	?	?	?	?
Web Frontend	?	?	?	?	?	?	?
Background Workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?
	Development VM	QA Server	Single Prod Server	Production Cluster	Public cloud	Developer's Laptop	Customer Servers

Source: <https://blog.docker.com/2013/08/paas-present-and-future/>

Problème de temps **exponentiel**

# Déjà vu ?

L'IT n'est pas la seule industrie à résoudre des problèmes...

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?

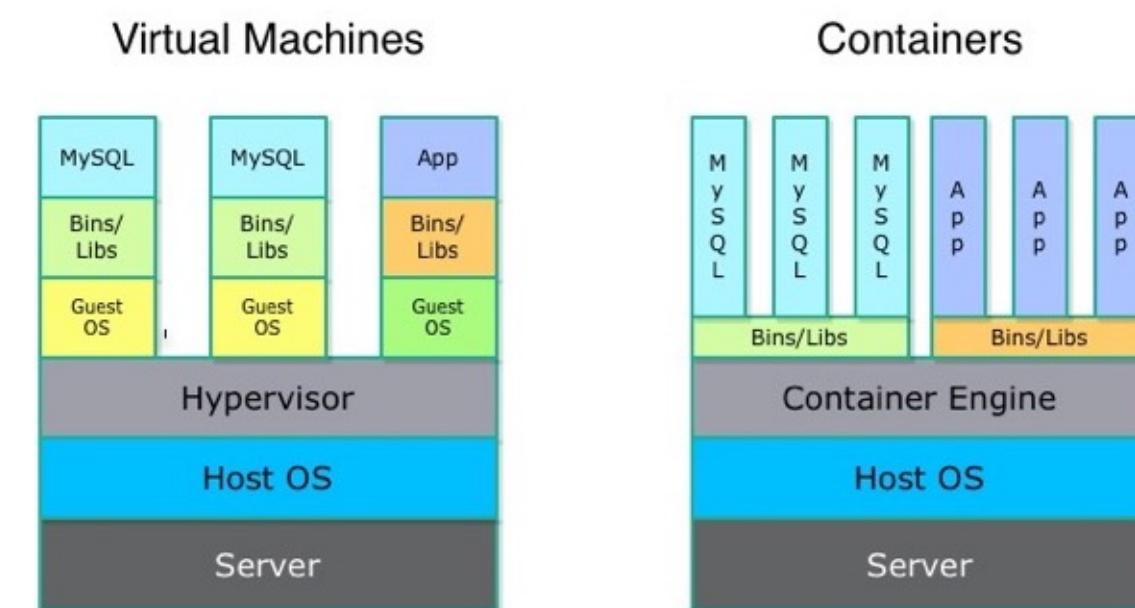
# Solution: Le conteneur intermodal

"Separation of Concerns"



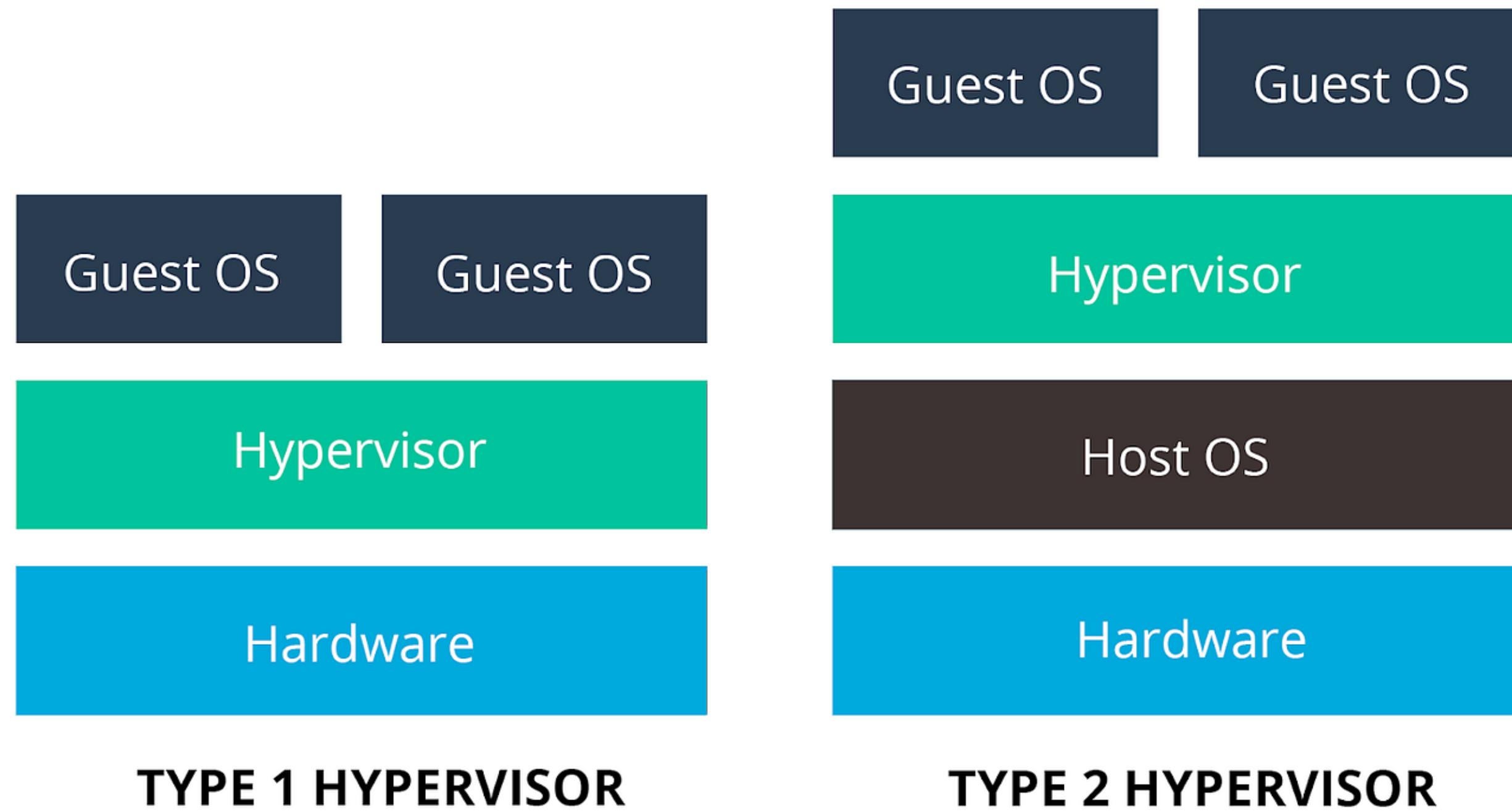
# Comment ça marche ?

"Virtualisation Légère"



# Comment ça marche ?

Virtualisation



# Comment ça marche ?

Virtualisation

## Type 1 Hypervisors

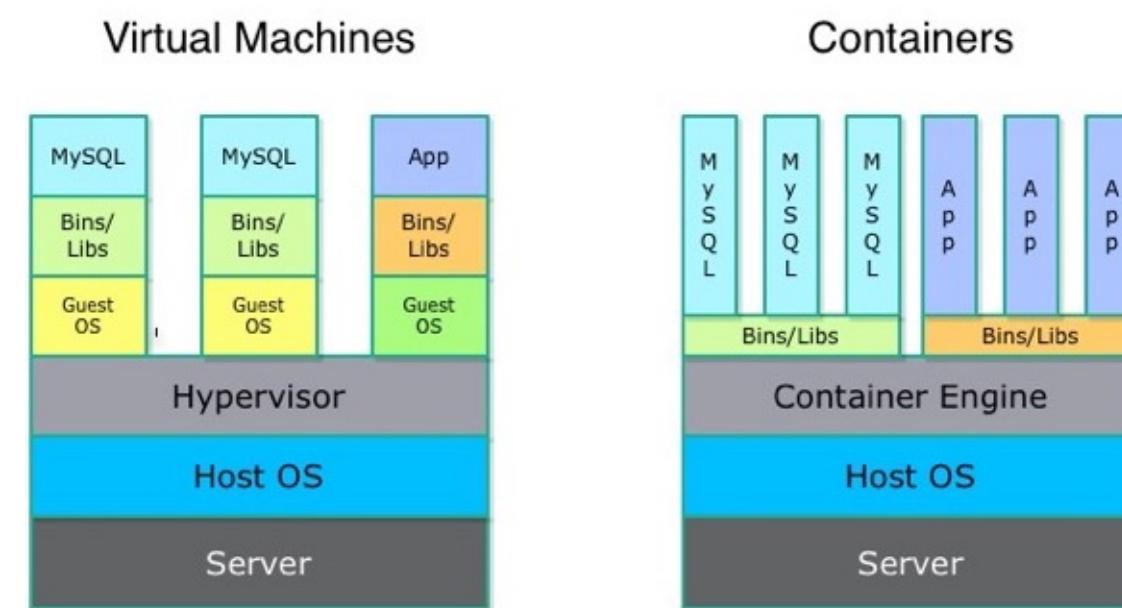


## Type 2 Hypervisors



# Comment ça marche ?

"Virtualisation Légère"



- Légère, vraiment?

## Challenge: We Have a Winner!



VICTOR COISNE

Oct 20 2015

At the [end of DockerCon 2015](#), [Dieter Reuter from Hypriot](#) presented a demo running 500 Docker containers on a [Raspberry Pi 2](#) device but he knew that number could be at least doubled.

[TL;DR Dieter was right.](#)

# Légère, vraiment!

<http://web.archive.org/web/20200810061020/https://www.docker.com/blog/raspberry-pi-dockercon-challenge-winner/>

A big congratulations to [Damien Duportal](#), [Nicolas de Loof](#) and [Yoann Dubreuil](#) on running 2500 web servers in Docker containers on a single Raspberry Pi 2!

Damien, Nicolas and Yoann each win a complimentary pass to [DockerCon EU 2015](#) and speaking slot during the conference to demo how they accomplished this.

*#RpiDocker 2740 web servers running on a #Rpi, could have more. But using a patched docker daemon with a hack that isn't a valuable fix. —*

*Nicolas De loof (@ndeloof) October 13, 2015*

### Post Tags

- dockercon
- DockerCon Europe
- raspberry pi

### Categories

- All
- Products
- Community
- Engineering
- Company



Aside from the above issues (and there are more caveats in the documentation), my installation is a pretty faithful reproduction of ESXi on a server or home lab. However, ESXi itself uses about 1.3GB of RAM, leaving only around 6.5GB usable. This could support perhaps 4-5 small VMs, but would be much more useful for running containerised applications and eliminating the individual VM overhead.

Légère, vraiment!

## Proof of Concept

Remember that Flings are just proofs of concept and not always aimed at final production. VMware suggests a range of hardware options for ARM-based workloads and this is more likely where we will see the initial development of ESXi on ARM. The challenge for end users here is to determine whether the price/performance/power/cooling ratio works better on ARM than x86. Of course applications also need to be available, but it's not hard to find ARM-compiled versions of most open source software and operating systems.



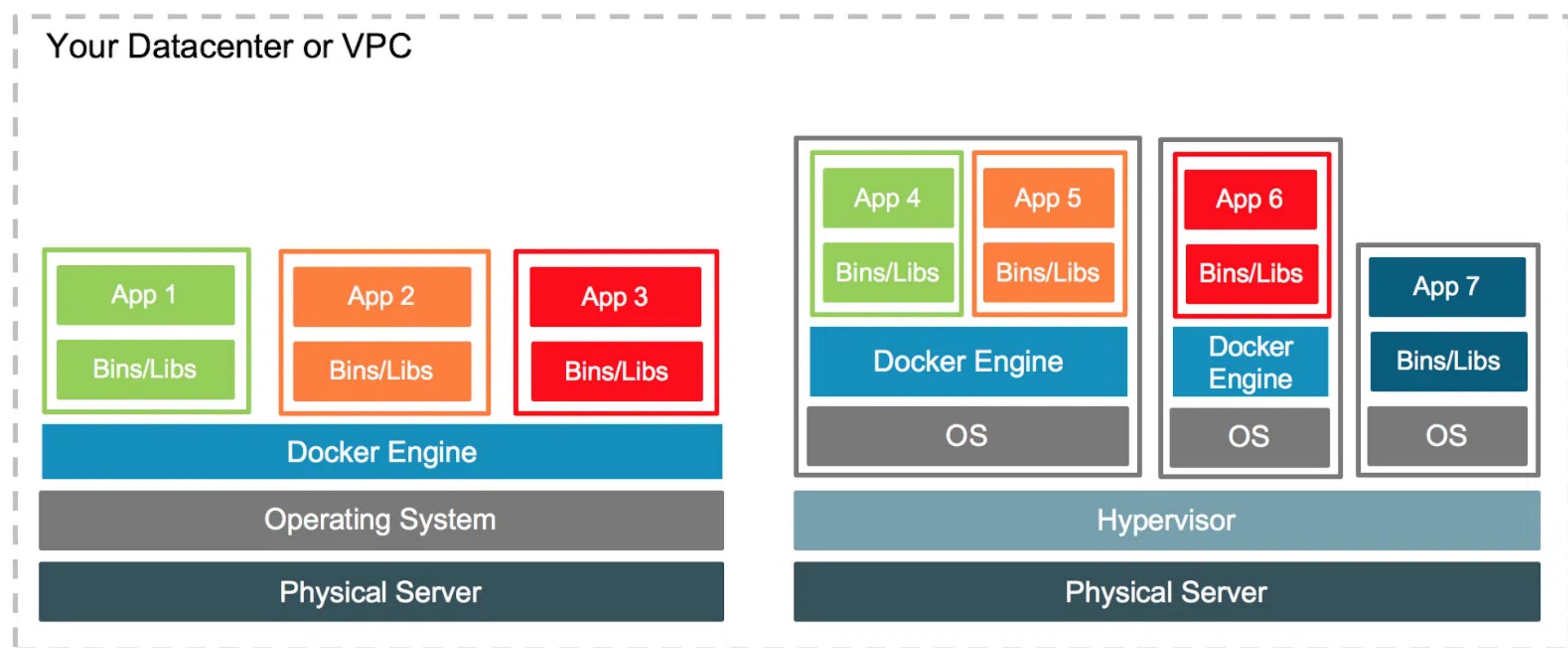
# Conteneur != VM

Machine Physique

Conteneur != VM

# VMs & Conteneurs

Non exclusifs mutuellement



# Cas d'usage

Le bac à sable



# Cas d'usage

Le bac à sable



- Un environnement tout propre tout neuf !

# Cas d'usage

Le bac à sable



- Un environnement tout propre tout neuf !
- Le poste de travail n'est pas impacté

# Cas d'usage

Le bac à sable



- Un environnement tout propre tout neuf !
- Le poste de travail n'est pas impacté
- La configuration reste également isolée sans effet sur le poste de travail

# Cas d'usage

La machine de dév



# Cas d'usage

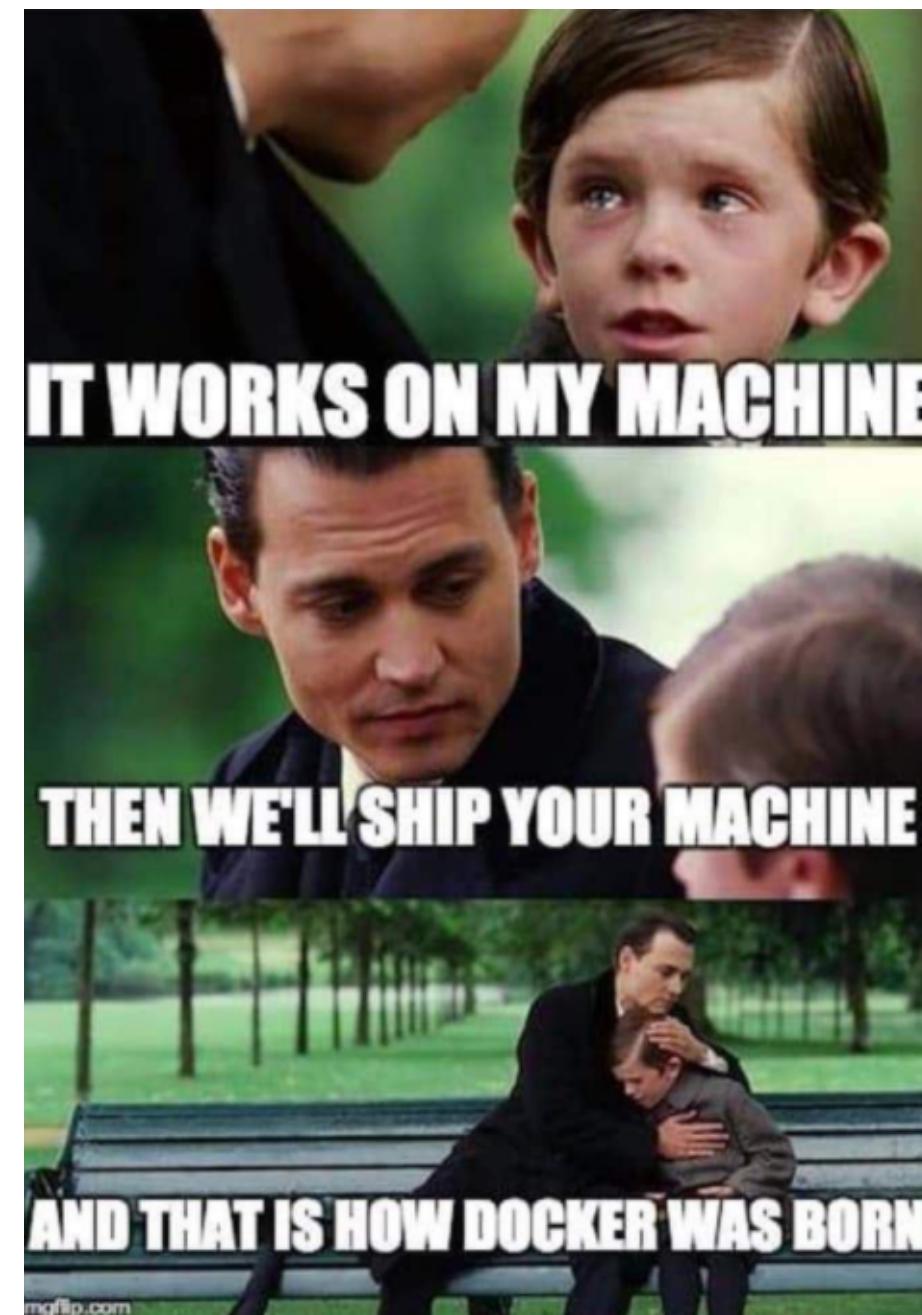
La machine de dév



- Avoir un environnement reproductible pour rendre homogène le développement et les tests.

# Cas d'usage

La machine de dév



# Cas d'usage

La machine de dév

- It works on my computer
- Yes, but we are not going to give your computer to the client



# Cas d'usage

Déploiement en production



# Cas d'usage

Déploiement en production



- Avec un container, si ça fonctionne en local, ça fonctionne en prod !

# Cas d'usage

Outilage jetable



# Cas d'usage

Outilage jetable



- On instancie des applications sans les installer

# Cas d'usage

Outilage jetable



- On instancie des applications sans les installer
- On crée le container, on l'utilise puis on le jette

Beau boulot! 😊



On a la base, remplissons là maintenant de containers!

La suite: vers la droite ➡

# Préparer votre environnement de travail

# Outils Nécessaires



- Un navigateur web récent (et décent)
- Un compte sur  GitHub

# GitPod

GitPod.io : Environnement de développement dans le ☁ "nuage"

- **But:** reproductible
- Puissance de calcul sur un serveur distant
- Éditeur de code VSCode dans le navigateur
- Gratuit pour 50h par mois (⚠)

# Démarrer avec GitPod



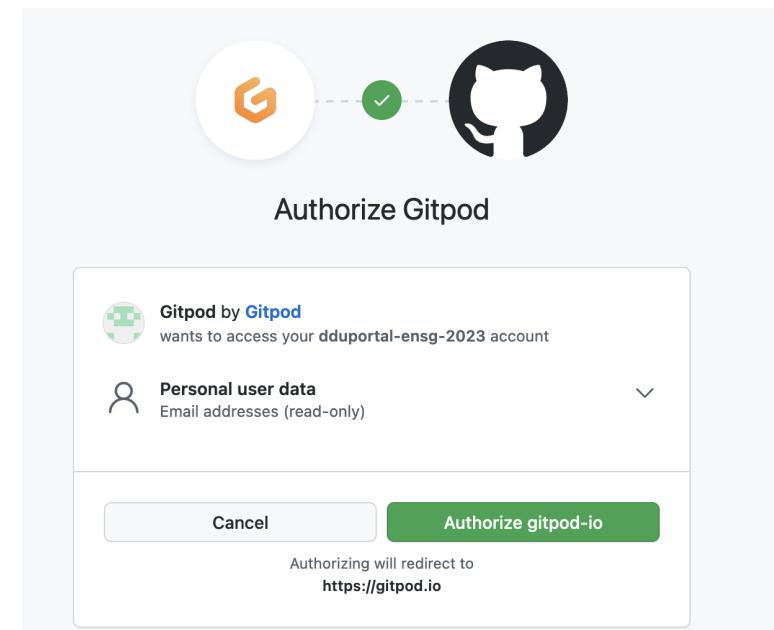
- Rendez vous sur <https://gitpod.io>
- Authentifiez vous en utilisant votre compte GitHub:
  - Bouton "Login" en haut à droite
  - Puis choisissez le lien "\_CONTINUE WITH GITHUB\_"

▲ Pour les "autorisations", passez sur la slide suivante

# Autorisations demandées par GitPod



Lors de votre première connexion, GitPod va vous demander l'accès (à accepter) à votre email public configuré dans GitHub :



⚠ Passez à la slide suivante avant d'aller plus loin

# Validation du Compte GitPod



GitPod vous demande votre numéro de téléphone mobile afin d'éviter les abus (service gratuit). Saisissez un numéro de téléphone valide pour recevoir par SMS un code de déblocage :

User Validation Required

⚠️ To use Gitpod you'll need to validate your account with your phone number. This is required to discourage and reduce abuse on Gitpod infrastructure.

Enter a mobile phone number you would like to use to verify your account. Having trouble? [Contact support](#)

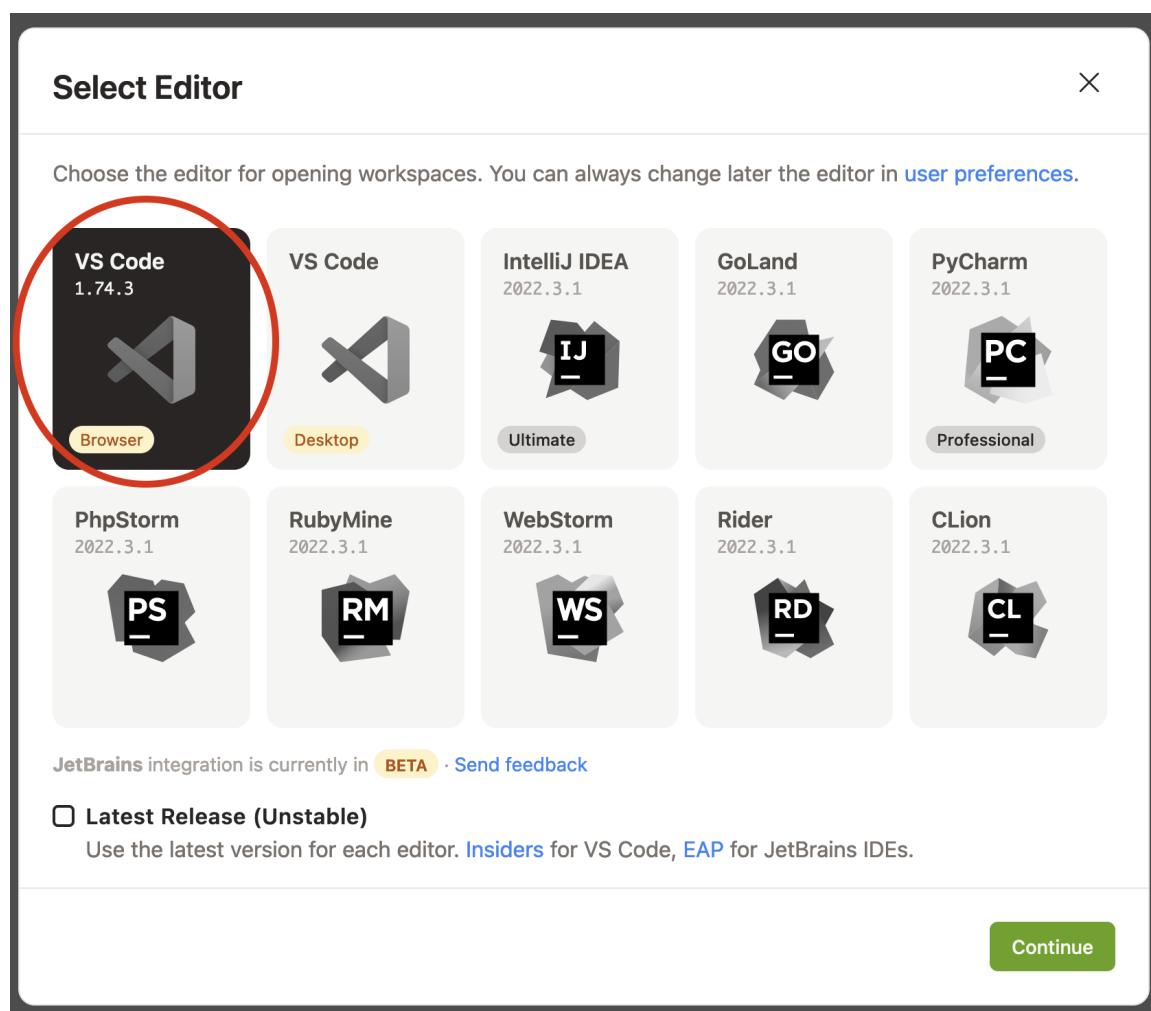
Mobile Phone Number

Send Code via SMS

⚠️ Passez à la slide suivante avant d'aller plus loin

# Choix de l'Éditeur de Code

Choisissez l'éditeur "VSCode Browser" (la première tuile) :



⚠️ Passez à la slide suivante avant d'aller plus loin

# Workspaces GitPod



- Vous arrivez sur la page listant les "workspaces" GitPod :
- Un workspace est une instance d'un environnement de travail virtuel (C'est un ordinateur distant)
- ↗ Faites attention à réutiliser le même workspace tout au long de ce cours ↗

The screenshot shows the GitPod Workspaces interface. At the top, there's a navigation bar with a logo, a 'Workspaces' button, and a 'New Team' button. On the right, there are 'Feedback' and a user icon. Below the navigation is a search bar labeled 'Filter Workspaces' and a limit selector 'Limit: 50'. A 'New Workspace' button is also present. The main area is titled 'Workspaces' and contains two workspace entries:

Workspace Name	Repository	Branch	Last Commit	More Options
cicdlectures-gitpod-jcu32jcv4q2	cicd-lectures/gitpod	main	No Changes 3 minutes ago	⋮
cicdlectures-gitpod-vv8g7mywidp	cicd-lectures/gitpod	main	No Changes 4 minutes ago	⋮

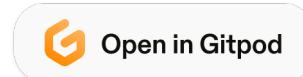
# Permissions GitPod <→ GitHub



- Pour les besoins de ce cours, vous devez autoriser GitPod à pouvoir effectuer certaines modifications dans vos dépôts GitHub
- Rendez-vous sur [la page des intégrations avec GitPod](#)
- Éditez les permissions de la ligne "GitHub" (les 3 petits points à droite) et sélectionnez uniquement :
  - user:email
  - public\_repo
  - workflow

# Démarrer l'environnement GitPod

Cliquez sur le bouton ci-dessous pour démarrer un environnement GitPod personnalisé:



Après quelques secondes (minutes?), vous avez accès à l'environnement:

- Gauche: navigateur de fichiers ("Workspace")
- Haut: éditeur de texte ("Get Started")
- Bas: Terminal interactif
- À droite en bas: plein de popups à ignorer (ou pas?)

Source disponible dans : <https://github.com/gounthar/cours-devops-docker>

# Checkpoint

- Vous devriez pouvoir taper la commande `whoami` dans le terminal de GitPod:
  - Retour attendu: `gitpod`
- Vous devriez pouvoir fermer le fichier "Get Started" ...
  - ... et ouvrir n'importe quel autre fichier ...

Bien, on peut maintenant fermer ce workspace, il ne s'agirait pas de gaspiller vos 50 heures.

On peut commencer !

# Containers



# Exercice : Votre premier conteneur

C'est à vous (ouf) !

- Allez dans Gitpod
- Dans un terminal, tapez la commande suivante :

```
docker container run hello-world  
# Équivalent de l'ancienne commande 'docker run'
```

Copy



## Anatomie

- Un service "Docker Engine" tourne en tâche de fond et publie une API REST
- La commande `docker container run ...` a lancé un client docker qui a envoyé une requête POST au service docker
- Le service a téléchargé une **Image** Docker depuis le registre **DockerHub**,
- Puis a exécuté un **conteneur** basé sur cette image



## Anatomie

\$

\$



# Anatomie



# Anatomie



# Anatomie



# Anatomie



# Anatomie



# Anatomie



# Anatomie



# Anatomie



Anatomie



## Exercice : Où est mon conteneur ?

C'est à vous !

```
docker container ls --help
# ...
docker container ls
# ...
docker container ls --all
```

Copy

⇒ 🤔 comment comprenez vous les résultats des 2 dernières commandes ?

## ✓ Solution : Où est mon conteneur ?

Le conteneur est toujours présent dans le "Docker Engine" même en étant arrêté

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	Copy
109a9cdd3ec8	hello-world	"/hello"	33 seconds ago	Exited (0) 17 seconds ago		festive_faraday	

- Un conteneur == une commande "conteneurisée"
  - cf. colonne "**COMMAND**"
- Quand la commande s'arrête : le conteneur s'arrête
  - cf. code de sortie dans la colonne "**STATUS**"

 Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

Copy

 Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

Copy

- Le moteur Docker crée un container à partir de l'image "busybox".



# Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

Copy

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.



## Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

Copy

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.
- La commande "echo hello world" est exécutée à l'intérieur du container.



# Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

Copy

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.
- La commande "echo hello world" est exécutée à l'intérieur du container.
- On obtient le résultat dans la sortie standard de la machine hôte.



## Rappels d'anatomie

```
$ docker container run busybox echo hello world
```

Copy

- Le moteur Docker crée un container à partir de l'image "busybox".
- Le moteur Docker démarre le container créé.
- La commande "echo hello world" est exécutée à l'intérieur du container.
- On obtient le résultat dans la sortie standard de la machine hôte.
- Le container est stoppé.



## Cas d'usage

Outilage jetable



- Tester une version de Maven, de JDK, de NPM, ...



## Exercice : Cycle de vie d'un conteneur simple

- Lancez un nouveau conteneur nommé `bonjour`
  - 💡 `docker container run --help` ou Documentation en ligne
- Affichez les "logs" du conteneur (==traces d'exécution écrites sur le stdout + stderr de la commande conteneurisée)
  - 💡 `docker container logs --help` ou Documentation en ligne
- Lancez le conteneur avec la commande `docker container start`
  - Regardez le résultat dans les logs
- Supprimez le container avec la commande `docker container rm`

## ✓ Solution : Cycle de vie d'un conteneur simple

```
docker container run --name=bonjour hello-world
# Affiche le texte habituel

docker container logs bonjour
# Affiche le même texte : pratique si on a fermé le terminal

docker container start bonjour
# N'affiche pas le texte mais l'identifiant unique du conteneur 'bonjour'

docker container logs bonjour
# Le texte est affiché 2 fois !

docker container ls --all
# Le conteneur est présent
docker container rm bonjour
docker container ls --all
# Le conteneur n'est plus là : il a été supprimé ainsi que ses logs

docker container logs bonjour
# Error: No such container: bonjour
```

Copy



## Que contient "hello-world" ?

- C'est une "image" de conteneur, c'est à dire un modèle (template) représentant une application auto-suffisante.
  - On peut voir ça comme un "paquetage" autonome
- C'est un système de fichier complet:
  - Il y a au moins une racine /
  - Ne contient que ce qui est censé être nécessaire (dépendances, librairies, binaires, etc.)

# Docker Hub

- <https://hub.docker.com/> : C'est le registre d'images "par défaut"
  - Exemple : Image officielle de conteneur "Ubuntu"
- 🎓 Cherchez l'image hello-world pour en voir la page de documentation
  - 💡 pas besoin de créer de compte pour ça
- Il existe d'autre "registres" en fonction des besoins (GitHub GHCR, Google GCR, etc.)



## Lancer un container interactif

```
docker container run -it alpine
```

Copy



## Lancer un container interactif

```
docker container run -it alpine  
/ #
```

Copy



## Lancer un container interactif

```
docker container run -it alpine  
/ #
```

Copy

- On lance un container à partir de l'image "alpine"



## Lancer un container interactif

```
docker container run -it alpine  
/ #
```

Copy

- On lance un container à partir de l'image "alpine"
- On lance un `sh` dans ce container



## Lancer un container interactif

```
docker container run -it alpine  
/ #
```

Copy

- On lance un container à partir de l'image "alpine"
- On lance un `sh` dans ce container
- On redirige l'entrée standard avec `-i`



## Lancer un container interactif

```
docker container run -it alpine  
/ #
```

Copy

- On lance un container à partir de l'image "alpine"
- On lance un `sh` dans ce container
- On redirige l'entrée standard avec `-i`
- On déclare un pseudo-terminal avec `-t`



## Exercice : conteneur interactif

- Quel distribution Linux est utilisée dans le terminal Gitpod ?
  - 💡 Regardez le fichier `/etc/os-release`
- Exécutez un conteneur interactif basé sur `alpine:3.18.3` (une distribution Linux ultra-légère) et regardez le contenu du fichier au même emplacement
  - 💡 `docker container run --help`
  - 💡 Demandez un `tty` à Docker
  - 💡 Activez le mode interactif
- Exécutez la même commande dans un conteneur basé sur la même image mais en **NON** interactif
  - 💡 Comment surcharger la commande par défaut ?

## ✓ Solution : conteneur interactif

```
$ cat /etc/os-release
# ... Ubuntu ...

$ docker container run --tty --interactive alpine:3.18.3
/ # cat /etc/os-release
# ... Alpine ...
# Notez que le "prompt" du terminal est différent DANS le conteneur
/ # exit
$ docker container ls --all

$ docker container run alpine:3.18.3 cat /etc/os-release
# ... Alpine ...
```

Copy



## Utiliser un container interactif

Revenons dans notre container interactif de tout à l'heure...

```
/ # curl google.fr
```

Copy



## Utiliser un container interactif

```
/ # curl google.fr  
/bin/sh: curl: not found
```

Copy



## Utiliser un container interactif

```
/ # curl google.fr  
/bin/sh: curl: not found
```

Copy

CURL n'est pas disponible par défaut sur Alpine. Il faut l'installer au préalable



## Utiliser un container interactif

```
/ # curl google.fr  
/bin/sh: curl: not found
```

[Copy](#)

CURL n'est pas disponible par défaut sur Alpine. Il faut l'installer au préalable

```
/ # apk update && apk add curl
```

[Copy](#)



## Utiliser un container interactif

```
/ # curl google.fr  
/bin/sh: curl: not found
```

Copy

CURL n'est pas disponible par défaut sur Alpine. Il faut l'installer au préalable

```
/ # apk update && apk add curl  
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/main/x86_64/APKINDEX.tar.gz  
fetch https://dl-cdn.alpinelinux.org/alpine/v3.18/community/x86_64/APKINDEX.tar.gz  
v3.18.3-169-gd5adf7d7d28 [https://dl-cdn.alpinelinux.org/alpine/v3.18/main]  
v3.18.3-171-g503977088b4 [https://dl-cdn.alpinelinux.org/alpine/v3.18/community]  
OK: 20063 distinct packages available  
(1/7) Installing ca-certificates (20230506-r0)  
(2/7) Installing brotli-libs (1.0.9-r14)  
(3/7) Installing libunistring (1.1-r1)  
(4/7) Installing libidn2 (2.3.4-r1)  
(5/7) Installing nghttp2-libs (1.55.1-r0)  
(6/7) Installing libcurl (8.2.1-r0)  
(7/7) Installing curl (8.2.1-r0)  
Executing busybox-1.36.1-r2.trigger  
Executing ca-certificates-20230506-r0.trigger  
OK: 12 MiB in 22 packages
```

Copy



## Utiliser un container interactif

```
/ # curl google.fr
<HTML><HEAD><meta http-equiv="content-type" content="text/html; charset=utf-8">
<TITLE>301 Moved</TITLE></HEAD><BODY>
<H1>301 Moved</H1>
The document has moved
<A HREF="http://www.google.fr/">here</A>.
</BODY></HTML>
```

Copy

C'est bon, on a cURL





## Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ # exit
```

Copy



## Utiliser un container interactif

On peut quitter `sh` et revenir à la machine hôte !

```
/ # exit
```

Copy

Si on veut réutiliser `CURL` sur Alpine, c'est simple, on relance le shell, non? 🤔



## Utiliser un container interactif

On peut quitter sh et revenir à la machine hôte !

```
/ # exit
```

Copy

Si on veut réutiliser cURL sur Alpine, c'est simple, on relance le shell, non? 🤔

```
docker container run -it alpine
```

Copy



## Utiliser un container interactif

On peut quitter sh et revenir à la machine hôte !

```
/ # exit
```

Copy

Si on veut réutiliser cURL sur Alpine, c'est simple, on relance le shell, non? 🤔

```
docker container run -it alpine
```

Copy

On relance cURL:

```
/ # curl google.fr
```

Copy



# Utiliser un container interactif

On peut quitter sh et revenir à la machine hôte !

```
/ # exit
```

Copy

Si on veut réutiliser cURL sur Alpine, c'est simple, on relance le shell, non? 😅

```
docker container run -it alpine
```

Copy

On relance cURL:

```
/ # curl google.fr  
/bin/sh: curl: not found
```

Copy





## Utiliser un container interactif

En fait, c'est logique !

```
docker container run
```

Copy

- Cette commande instancie un "nouveau container à chaque fois" !



## Utiliser un container interactif

En fait, c'est logique !

```
docker container run
```

Copy

- Cette commande instancie un "nouveau container à chaque fois" !
- Chaque container est différent.



# Utiliser un container interactif

En fait, c'est logique !

```
docker container run
```

Copy

- Cette commande instancie un "nouveau container à chaque fois" !
- Chaque container est différent.
- Aucun partage entre les containers à part le contenu de base de l'image.



## Utiliser un container interactif



*"On m'a vendu un truc qui permet de lancer des tonnes de microservices... mais là, on télécharge nimp et s'amuse à le perdre..."*



## Conteneur en tâche de fond

Lançons un container bien particulier...

```
docker container run -it jpetazzo/clock
```

Copy



## Conteneur en tâche de fond

Lançons un container bien particulier...

```
docker container run -it jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
...
```

Copy



## Conteneur en tâche de fond

Lançons un container bien particulier...

```
docker container run -it jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
...
```

Copy

Ce container va tourner indéfiniment sauf si on le stoppe avec ...



# Conteneur en tâche de fond

Lançons un container bien particulier...

```
docker container run -it jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
```

Copy

Ce container va tourner indéfiniment sauf si on le stoppe avec Ctrl + C ...

... mais ça va stopper le container !





## Conteneur en tâche de fond

Lançons un container bien particulier...

```
docker container run -it jpetazzo/clock
Mon Sep 25 10:33:51 UTC 2023
Mon Sep 25 10:33:52 UTC 2023
Mon Sep 25 10:33:53 UTC 2023
Mon Sep 25 10:33:54 UTC 2023
Mon Sep 25 10:33:55 UTC 2023
Mon Sep 25 10:33:56 UTC 2023
...
...
```

Copy

Ce container va tourner indéfiniment sauf si on le stoppe avec ...

... mais ça va stopper le container !

Oui car quand on stoppe le processus principal d'un container, ce dernier n'a plus de raison d'exister et s'arrête naturellement.



## Conteneur en tâche de fond

La solution : le flag `-d`

```
docker container run -d jpetazzo/clock
```

Copy



## Conteneur en tâche de fond

La solution : le flag `-d`

```
docker container run -d jpetazzo/clock  
399f3b23bc0585991afa80dfee854cf0a953d782b99153b4e2cbc74ab6b07770
```

Copy

Le retour de cette commande correspond à l'identifiant unique du conteneur.

Cette fois-ci, le container tourne, mais en arrière plan !



## Conteneur en tâche de fond



*"Okay, maintenant il n'écrit la date nulle part... mais toutes les secondes... à l'aide ! "*



## Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard... du container !

Comment retrouver le contenu de la sortie standard du container ?



## Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard... du container !

Comment retrouver le contenu de la sortie standard du container ?

```
docker logs 399f3
```

Copy



## Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard... du container !

Comment retrouver le contenu de la sortie standard du container ?

```
docker logs 399f3
Tue Sep 12 12:37:01 UTC 2023
Tue Sep 12 12:37:02 UTC 2023
Tue Sep 12 12:37:03 UTC 2023
Tue Sep 12 12:37:04 UTC 2023
Tue Sep 12 12:37:05 UTC 2023
Tue Sep 12 12:37:06 UTC 2023
Tue Sep 12 12:37:07 UTC 2023
Tue Sep 12 12:37:08 UTC 2023
Tue Sep 12 12:37:09 UTC 2023
Tue Sep 12 12:37:10 UTC 2023
Tue Sep 12 12:37:11 UTC 2023
```

Copy



## Conteneur en tâche de fond

Le processus principal de ce container écrit dans la sortie standard... du container !

Comment retrouver le contenu de la sortie standard du container ?

```
docker logs 399f3
Tue Sep 12 12:37:01 UTC 2023
Tue Sep 12 12:37:02 UTC 2023
Tue Sep 12 12:37:03 UTC 2023
Tue Sep 12 12:37:04 UTC 2023
Tue Sep 12 12:37:05 UTC 2023
Tue Sep 12 12:37:06 UTC 2023
Tue Sep 12 12:37:07 UTC 2023
Tue Sep 12 12:37:08 UTC 2023
Tue Sep 12 12:37:09 UTC 2023
Tue Sep 12 12:37:10 UTC 2023
Tue Sep 12 12:37:11 UTC 2023
```

Copy

Ouf ! On n'est pas obligé de saisir l'identifiant complet ! Il suffit de fournir le nombre suffisant de caractères pour que ce soit discriminant.



## Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?



Commande vue un peu plus tôt...



## Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

```
docker container ls
```

Copy



# Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

```
docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
02cbf2fb3721	cours-devops-docker-serve	"/sbin/tini -g gulp ..."	3 hours ago	Up 3 hours	0.0.0.0:8000->8000/tcp	cours-devops-docker-serve-1
ebfbe695b2ec	moby/buildkit:buildx-stable-1	"buildkitd"	2 months ago	Up 3 hours		buildx_buildkit_exciting_williams0

Copy



# Lister les containers

Comment savoir si j'ai des containers en cours d'exécution ?

```
docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
02cbf2fb3721	cours-devops-docker-serve	"/sbin/tini -g gulp ..."	3 hours ago	Up 3 hours	0.0.0.0:8000->8000/tcp	cours-devops-docker-serve-1
ebfbe695b2ec	moby/buildkit:buildx-stable-1	"buildkitd"	2 months ago	Up 3 hours		buildx_buildkit_exciting_williams0

Copy

On obtient un tableau de tous les containers en cours d'exécution.



## Stop / Start

Il est possible de stopper un container.

```
docker container stop 399f3
```

Copy

## Stop / Start

Il est possible de stopper un container.

```
docker container stop 399f3
```

 Copy

Pour redémarrer un container :

```
docker container start 399f3
```

 Copy



## Lister tous les containers

Même les .



## Lister tous les containers

Comment savoir si j'ai des containers stoppés ?

```
docker container ls --all
```

Copy



# Lister tous les containers

Comment savoir si j'ai des containers stoppés ?

docker container ls --all						Copy
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
90725f661d4e	hello-world	"/hello"	13 seconds ago	Exited (0) 12 seconds ago		hardcore_wescoff
9d0a6586b9e1	busybox	"echo hello world"	22 seconds ago	Exited (0) 21 seconds ago		gracious_moser
368ed08a35e3	jpetazzo/clock	"/bin/sh -c 'while d..."	6 minutes ago	Up 5 minutes		sweet_clarke
c036e57bbf05	jpetazzo/clock	"/bin/sh -c 'while d..."	17 minutes ago	Exited (130) 17 minutes ago		cool_euclid
02cbf2fb3721	cours-devops-docker-serve	"/sbin/tini -g gulp ..."	3 hours ago	Up 3 hours	0.0.0.0:8000->8000/tcp	cours-devops-docker-serve-1
ebfbe695b2ec	moby/buildkit:buildx-stable-1	"buildkitd"	2 months ago	Up 4 hours		buildx_buildkit_exciting_williams0



# Lister tous les containers

Comment savoir si j'ai des containers stoppés ?

docker container ls --all						Copy
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	POR	NAMES
90725f661d4e	hello-world	"/hello"	13 seconds ago	Exited (0) 12 seconds ago		hardcore_wescoff
9d0a6586b9e1	busybox	"echo hello world"	22 seconds ago	Exited (0) 21 seconds ago		gracious_moser
368ed08a35e3	jpetazzo/clock	"/bin/sh -c 'while d..."	6 minutes ago	Up 5 minutes		sweet_clarke
c036e57bbf05	jpetazzo/clock	"/bin/sh -c 'while d..."	17 minutes ago	Exited (130) 17 minutes ago		cool_euclid
02cbf2fb3721	cours-devops-docker-serve	"/sbin/tini -g gulp ..."	3 hours ago	Up 3 hours	0.0.0.0:8000->8000/tcp	cours-devops-docker-serve-1
ebfbe695b2ec	moby/buildkit:buildx-stable-1	"buildkitd"	2 months ago	Up 4 hours		buildx_buildkit_exciting_williams0

Avec le flag `-a` , on obtient un tableau de tous les containers quel que soit leur état.



## Nettoyage

Tout container stoppé peut être supprimé.



## Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e
```

Copy



## Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e  
90725f661d4e
```

Copy



## Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e  
90725f661d4e
```

Copy

Container "auto-nettoyant"



## Nettoyage

Tout container stoppé peut être supprimé.

```
docker container rm 90725f661d4e  
90725f661d4e
```

Copy

Container "auto-nettoyant"

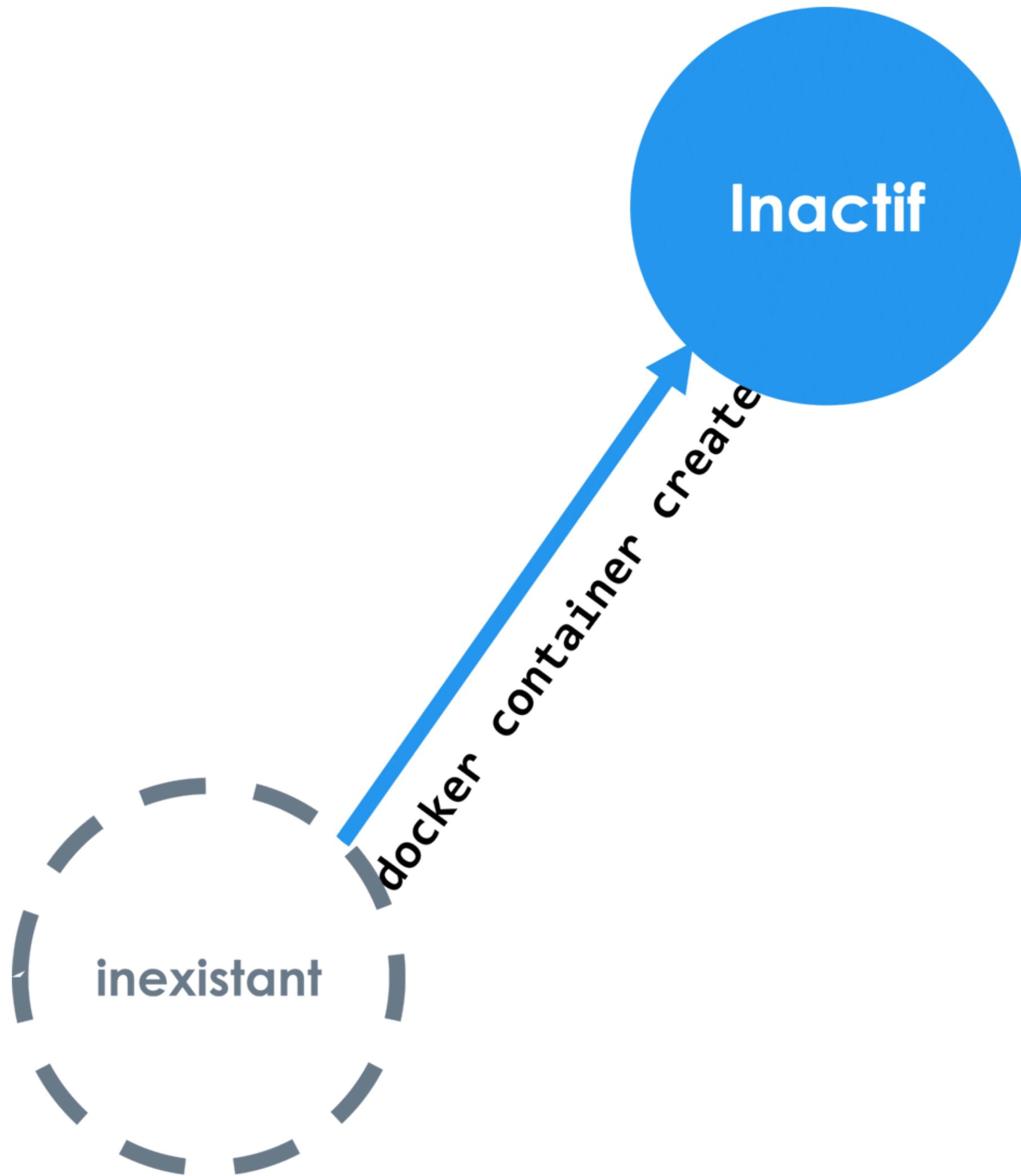
```
docker container run -it --rm jpetazzo/clock
```

Copy

Aussitôt stoppé, aussitôt supprimé !



Rappel: cycle de vie d'un container





Rappel: cycle de vie d'un container



Rappel: cycle de vie d'un container



Rappel: cycle de vie d'un container



Rappel: cycle de vie d'un container





## Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.



## Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.



## Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container exec <containerID> echo "hello"
```

Copy



## Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container ls
```

Copy



# Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

docker container ls							Copy
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES	
368ed08a35e3	jpetazzo/clock	"/bin/sh -c 'while d..."	4 hours ago	Up 4 hours		sweet_clarke	
02cbf2fb3721	cours-devops-docker-serve	"/sbin/tini -g gulp ..."	7 hours ago	Up 7 hours	0.0.0.0:8000->8000/tcp	cours-devops-docker-serve-1	
ebfbe695b2ec	moby/buildkit:buildx-stable-1	"buildkitd"	2 months ago	Up 7 hours		buildx_buildkit_exciting_williams0	



# Reprendre le contrôle

Sur un container en arrière-plan

Il est possible d'interagir avec un container en arrière-plan en cours d'exécution.

La commande suivante permet de lancer une commande à l'intérieur d'un container.

```
docker container ls
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS          NAMES
368ed08a35e3   jpetazzo/clock   "/bin/sh -c 'while d..."  4 hours ago   Up 4 hours
02cbf2fb3721   cours-devops-docker-serve   "/sbin/tini -g gulp ..."  7 hours ago   Up 7 hours    0.0.0.0:8000->8000/tcp   cours-devops-docker-serve-1
ebfbe695b2ec   moby/buildkit:buildx-stable-1 "buildkitd"
                                                               2 months ago  Up 7 hours

docker container exec 368ed08a35e3 echo hello
hello
```

Copy



## Reprendre le contrôle

Sur un container en arrière-plan



```
docker container exec -it <containerID> bash
```

Copy

Ca fonctionne aussi en interactif !



# Exercice : conteneur en tâche de fond

## Étapes

- Lancer un container "daemon" `jpetazzo/clock`
- Utiliser l'équivalent de `tail -f` pour lire la sortie standard du container
- Lancer un second container "daemon"
- Stocker l'identifiant de ce container dans une variable du shell, en une seule commande et en jouant avec `docker container ls`
- Stopper le container avec cet identifiant
- Afficher les containers lancés
- Afficher les containers arrêtés

## ✓ Solution : conteneur en tâche de fond

```
# Lancer un container "daemon" `jpetazzo/clock`  
docker container run --detach --name first-clock jpetazzo/clock  
  
# Utiliser l'équivalent de `tail -f` pour lire la sortie standard du container💡  
docker container logs -f first-clock  
  
# * Lancer un second container "daemon" 🎭  
docker container run --detach --name second-clock jpetazzo/clock  
  
# Stocker l'identifiant de ce container dans une variable du shell, en une seule commande et en jouant avec `docker container ls`  
# --filter "name=second-clock" filters the list of containers to include only those with the name "second-clock."  
container_id=$(docker container ls -q --filter "name=second-clock")  
  
# Stopper le container avec cet identifiant  
docker container stop "$container_id"  
  
# Afficher les containers lancés🏃‍♂️📦  
docker container ls  
  
# Afficher les containers arrêtés🛑📦  
docker container ls --filter "status=exited"
```



## Exercice : conteneur en tâche de fond

- Relancer un des containers arrêtés.
- Exécuter un `ps -ef `dans ce container
- Quel est le PID du process principal ?
- Vérifier que le container "tourne" toujours
- Supprimer l'image (tip : docker rmi)
- Supprimer les containers
- Supprimer l'image (pour de vrai cette fois)

# ✓ Solution : conteneur en tâche de fond

```
# Relancer un des containers arrêtés.  
docker container start second-clock  
  
# Exécuter un `ps -ef` dans ce container  
docker container exec second-clock ps -ef  
PID   USER      TIME    COMMAND  
  1  root      0:00 /bin/sh -c while date; do sleep 1; done  
 56  root      0:00 sleep 1  
 57  root      0:00 ps -ef  
  
# Quel est le PID du process principal ?  
# 1  
  
# Vérifier que le container "tourne" toujours  
docker container ls  
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES  
7d7085809ad2        cours-devops-docker-serve "/sbin/tini -g gulp ..." 6 minutes ago     Up 5 minutes          0.0.0.0:8000->8000/tcp   cours-devops-docker-serve-1  
edea0c46c6d8        jpetazzo/clock      "/bin/sh -c 'while d..." 9 minutes ago     Up About a minute  
ebfbe695b2ec        moby/buildkit:buildx-stable-1 "buildkitd"          2 months ago       Up 11 hours  
  
# Supprimer l'image (tip : `docker rmi`)  
docker rmi jpetazzo/clock  
Error response from daemon: conflict: unable to remove repository reference "jpetazzo/clock" (must force) - container edea0c46c6d8 is using its referenced image 7a8965d6553e  
  
# Supprimer les containers  
docker stop second-clock  
second-clock  
  
docker rm second-clock  
second-clock
```

Copy



## Exercice : conteneur en tâche de fond

- Exécutez un conteneur, basé sur l'image nginx en tâche de fond ("Background"), nommé webserver-1
  - 💡 On parle de processus "détaché" (ou bien "démonisé") 🦇
  - ⚠️ Pensez bien à docker container ls
- Regardez le contenu du fichier /etc/os-release dans ce conteneur
  - 💡 docker container exec
- Essayez d'arrêter, démarrer puis redémarrer le conteneur
  - ⚠️ Pensez bien à docker container ls à chaque fois
  - 💡 stop, start, restart

## ✓ Solution : conteneur en tâche de fond

```
docker container run --detach --name=webserver-1 nginx
# <ID du conteneur>

docker container ls
docker container ls --all

docker container exec webserver-1 cat /etc/os-release
# ... Debian ...

docker container stop webserver-1
docker container ls
docker container ls --all

docker container start webserver-1
docker container ls
docker container ls --all

docker container start webserver-1
docker container ls
```

Copy

# Checkpoint



Vous savez désormais:

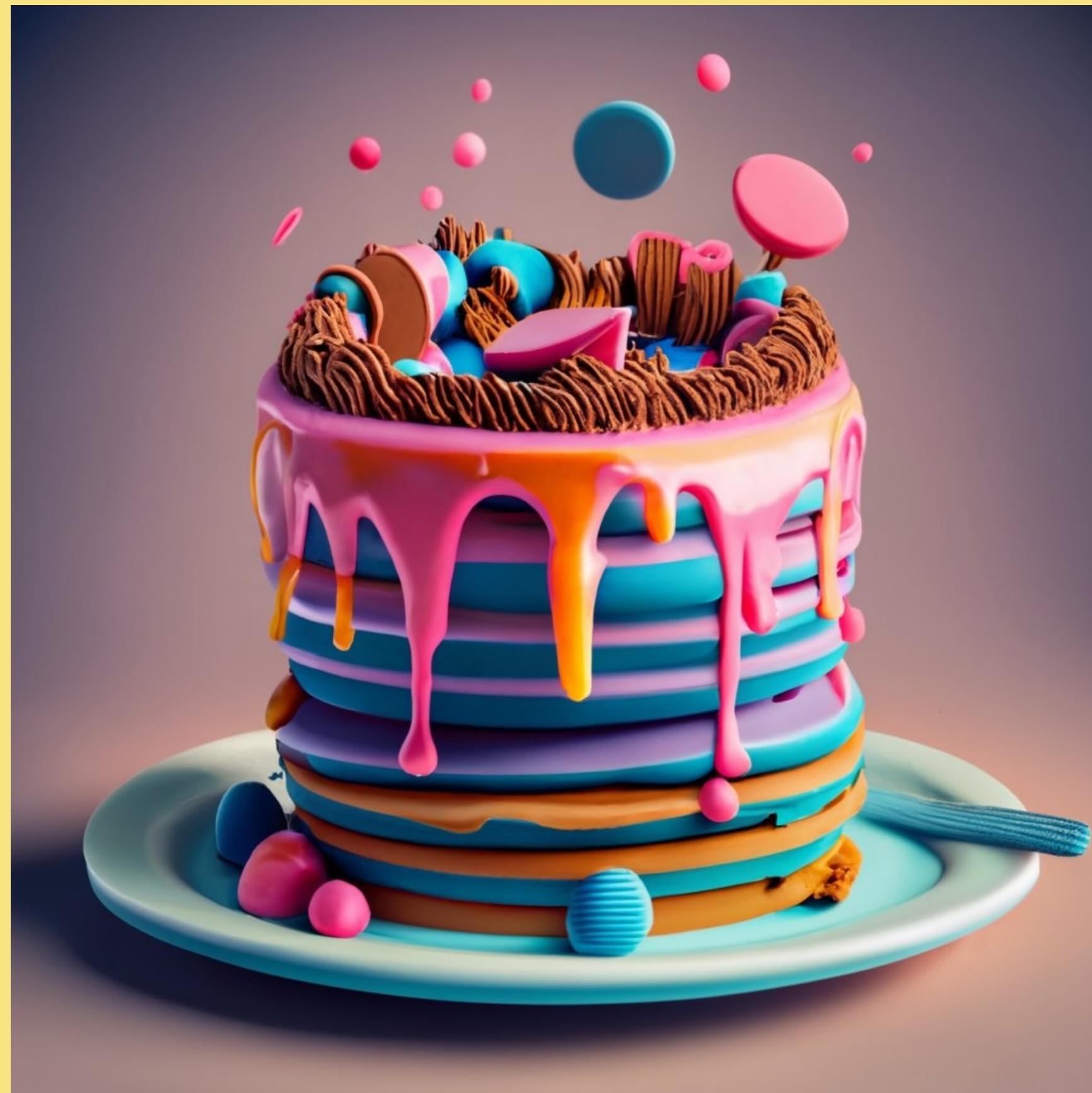
- Maîtriser le cycle de vie des containers
- Interagir avec les containers existants



# Checkpoint

- Docker essaye de résoudre le problème de l'empaquetage le plus "portable" possible
    - On n'en a pas encore vu les effets, ça arrive !
  - Vous avez vu qu'un conteneur permet d'exécuter une commande dans un environnement "préparé"
    - Catalogue d'images Docker par défaut : Le Docker Hub
  - Vous avez vu qu'on peut exécuter des conteneurs selon 3 modes :
    - "One shot"
    - Interactif
    - En tâche de fond
- ⇒  Mais comment ces images sont-elles fabriquées ? Quelle confiance leur accorder ?

# Docker Images





## Pourquoi des images ?

- Un **conteneur** est toujours exécuté depuis une **image**.
- Une **image de conteneur** (ou "Image Docker") est un modèle ("template") d'application auto-suffisant.  
⇒ Permet de fournir un livrable portable (ou presque).



## C'est quoi une image ?

C'est une collection de fichiers et de metadonnées.



## C'est quoi une image ?

C'est une collection de fichiers et de metadonnées.

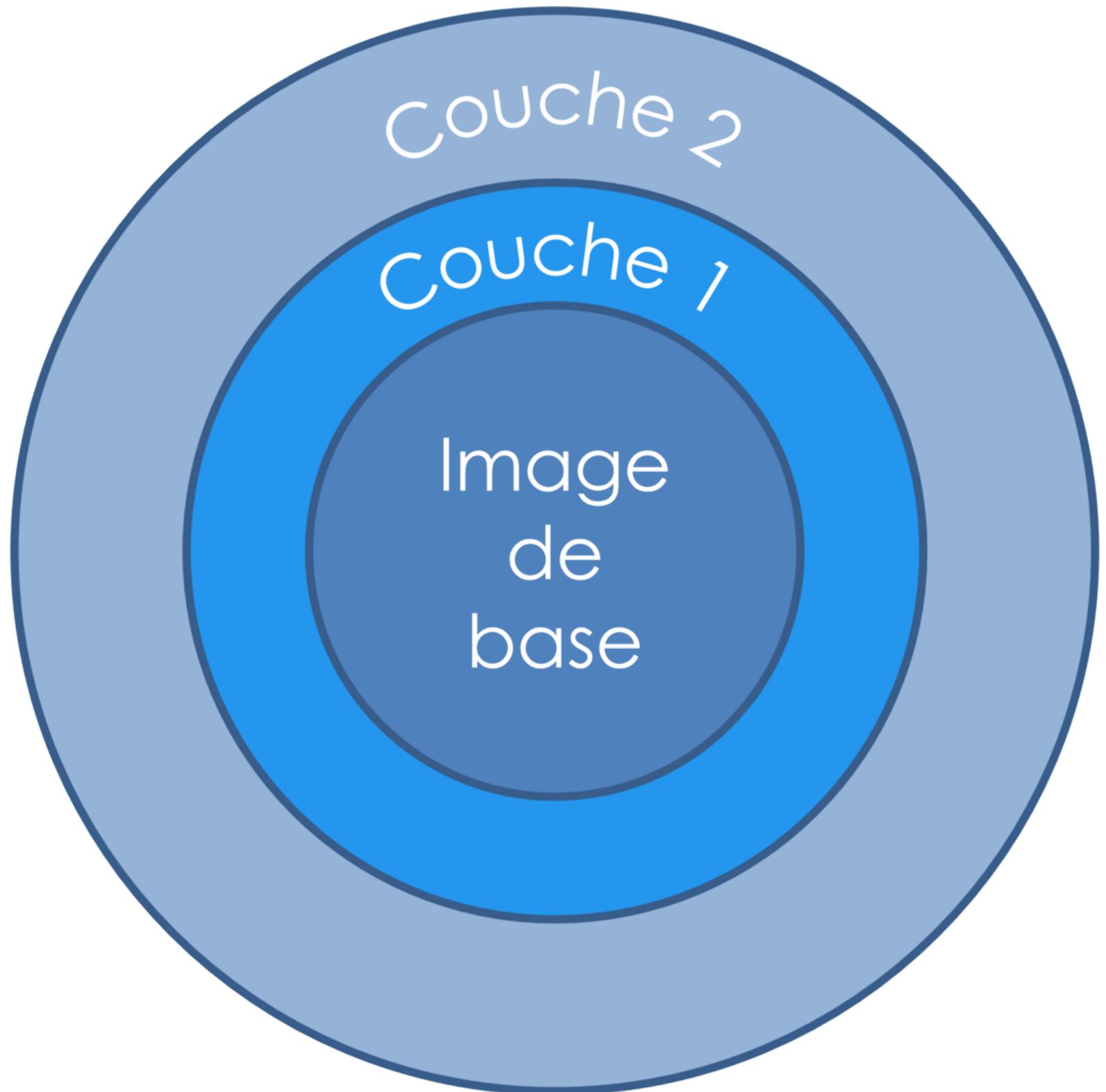
C'est une suite de couches superposées.



## C'est quoi une image ?

C'est une collection de fichiers et de metadonnées.

C'est une suite de couches superposées.

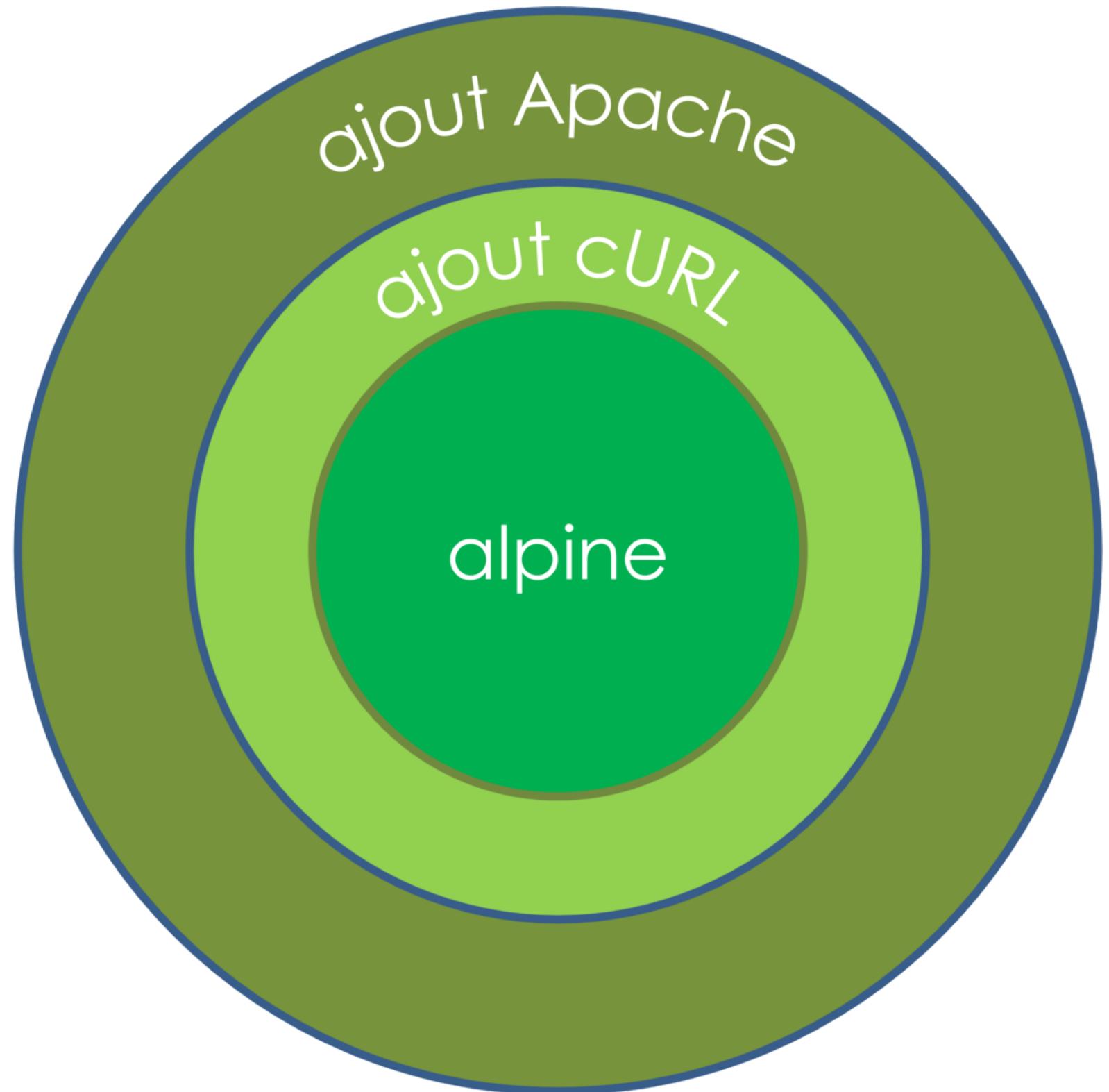
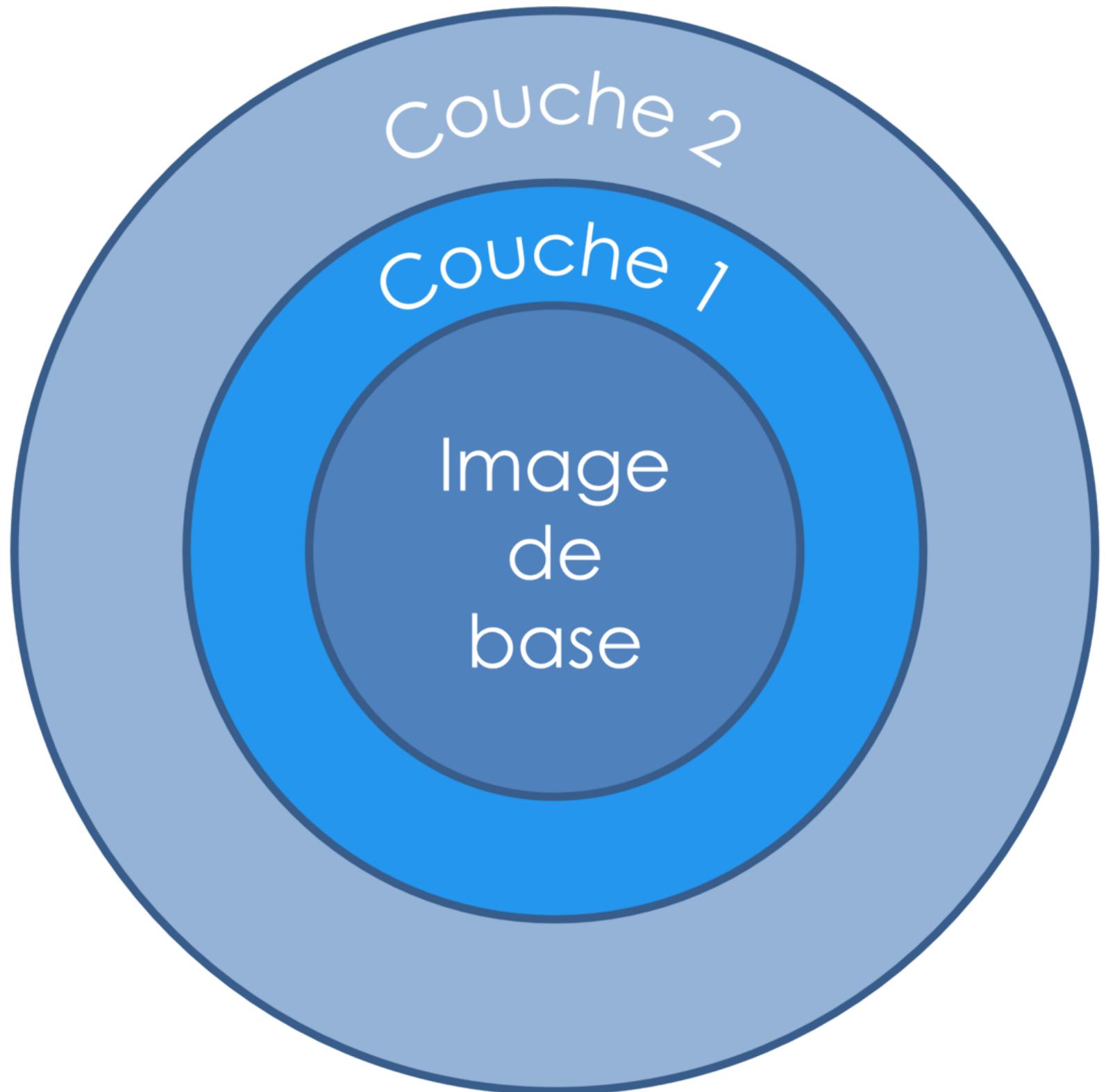




## C'est quoi une image ?

C'est une collection de fichiers et de metadonnées.

C'est une suite de couches superposées.







## Détail des couches

Exemple d'une image Apache HTTPd "custom"



## Détail des couches

Exemple d'une image Apache HTTPd "custom"



## Détail des couches

Exemple d'une image Apache HTTPd "custom"



## Détail des couches

Exemple d'une image Apache HTTPd "custom"



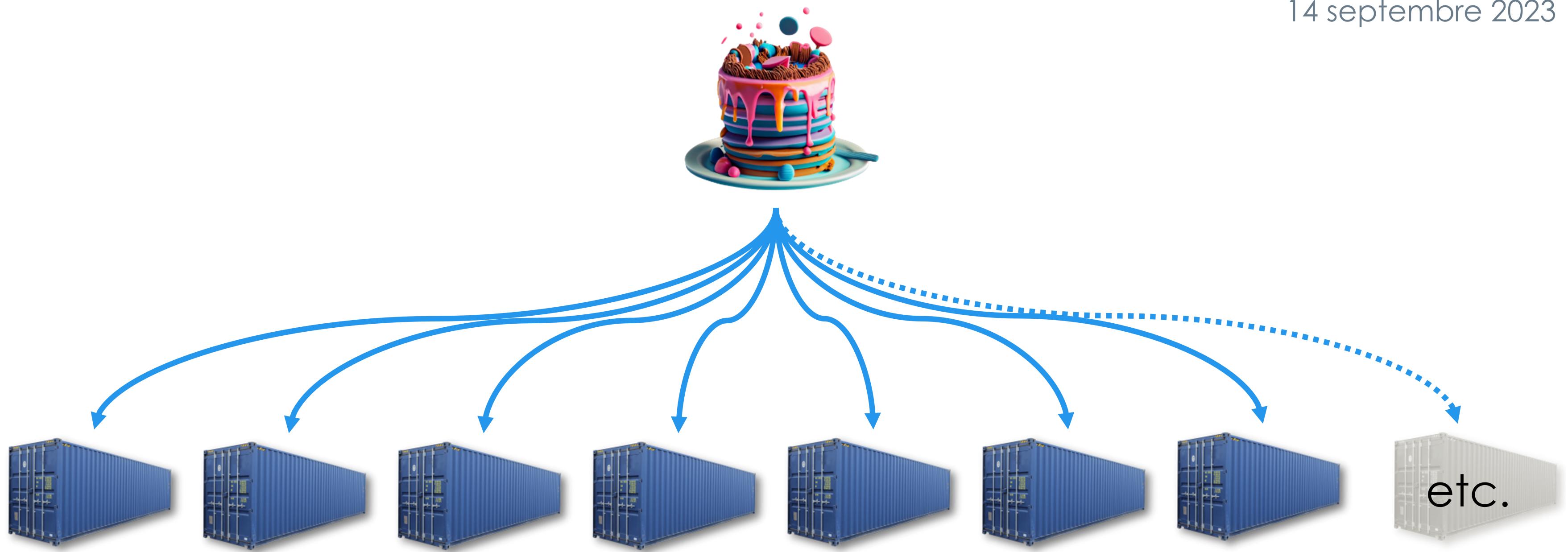
## Détail des couches

Exemple d'une image Apache HTTPd "custom"

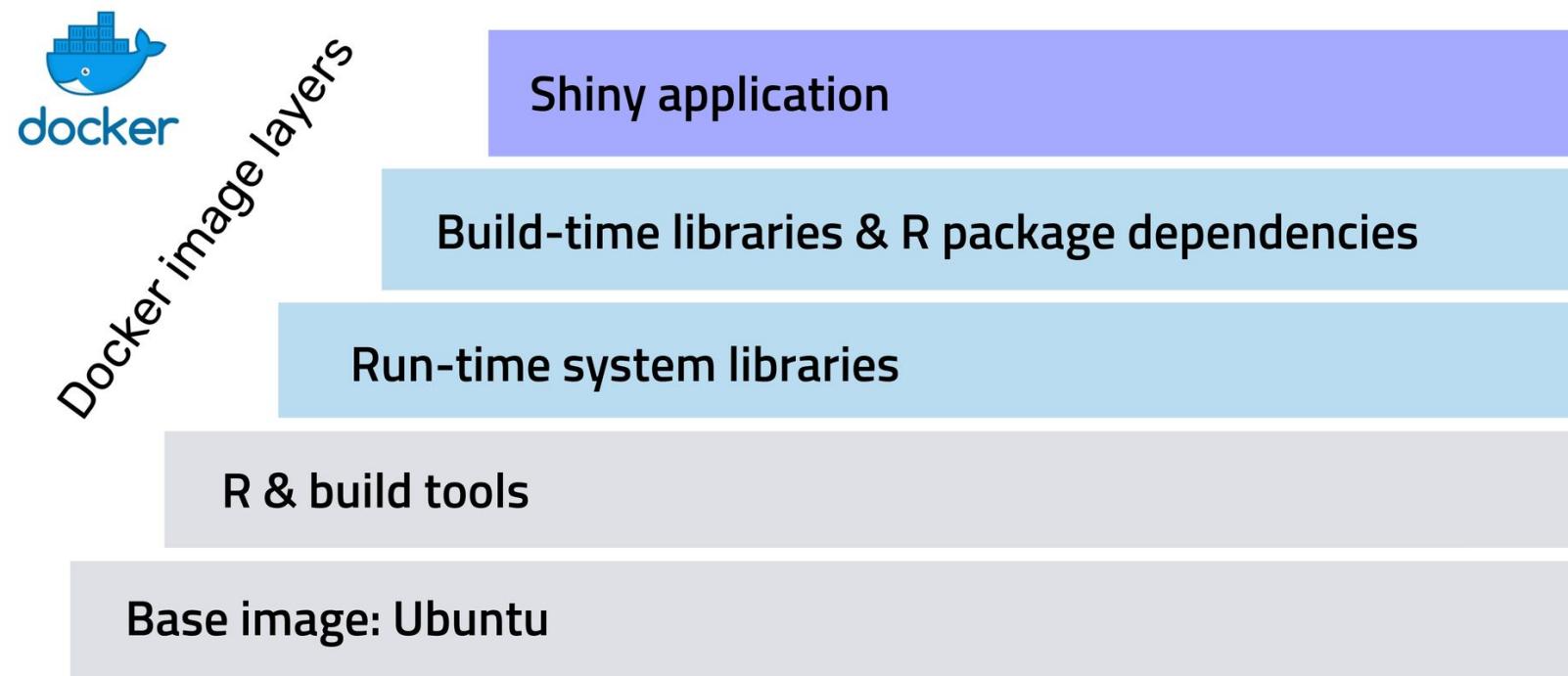
# Dicton du jour

*"L'image est à la classe ce que le container est à l'objet"*

Amaury W.  
14 septembre 2023

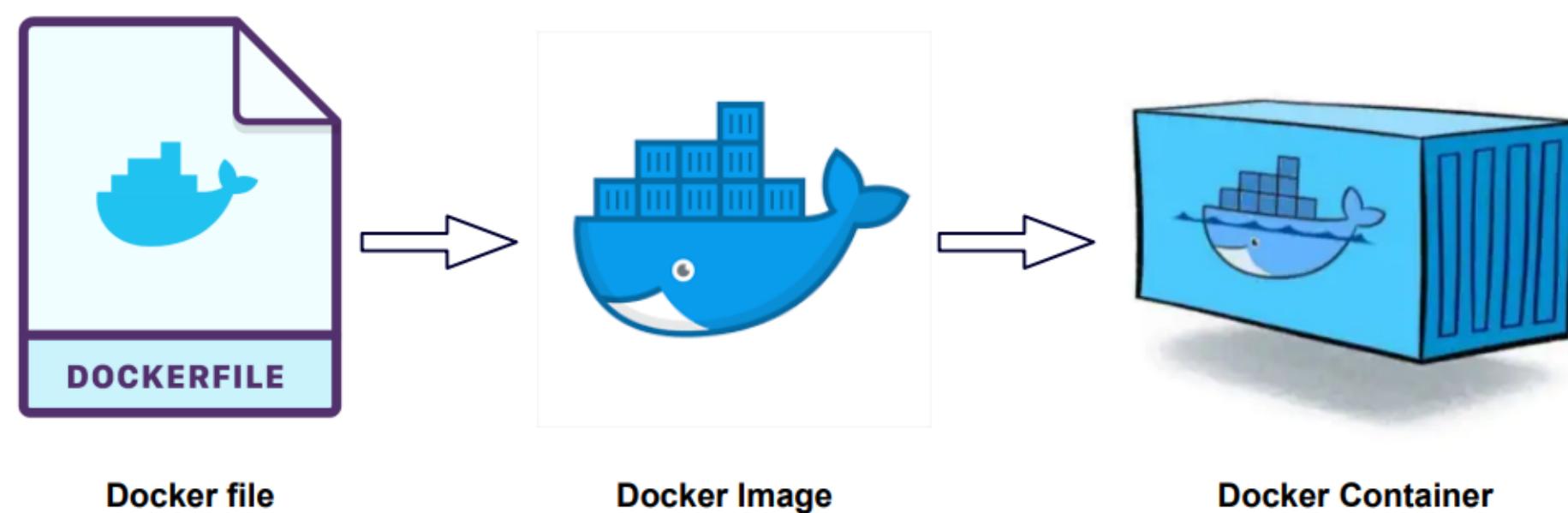


🤔 Application Auto-Suffisante ?



© Analythium

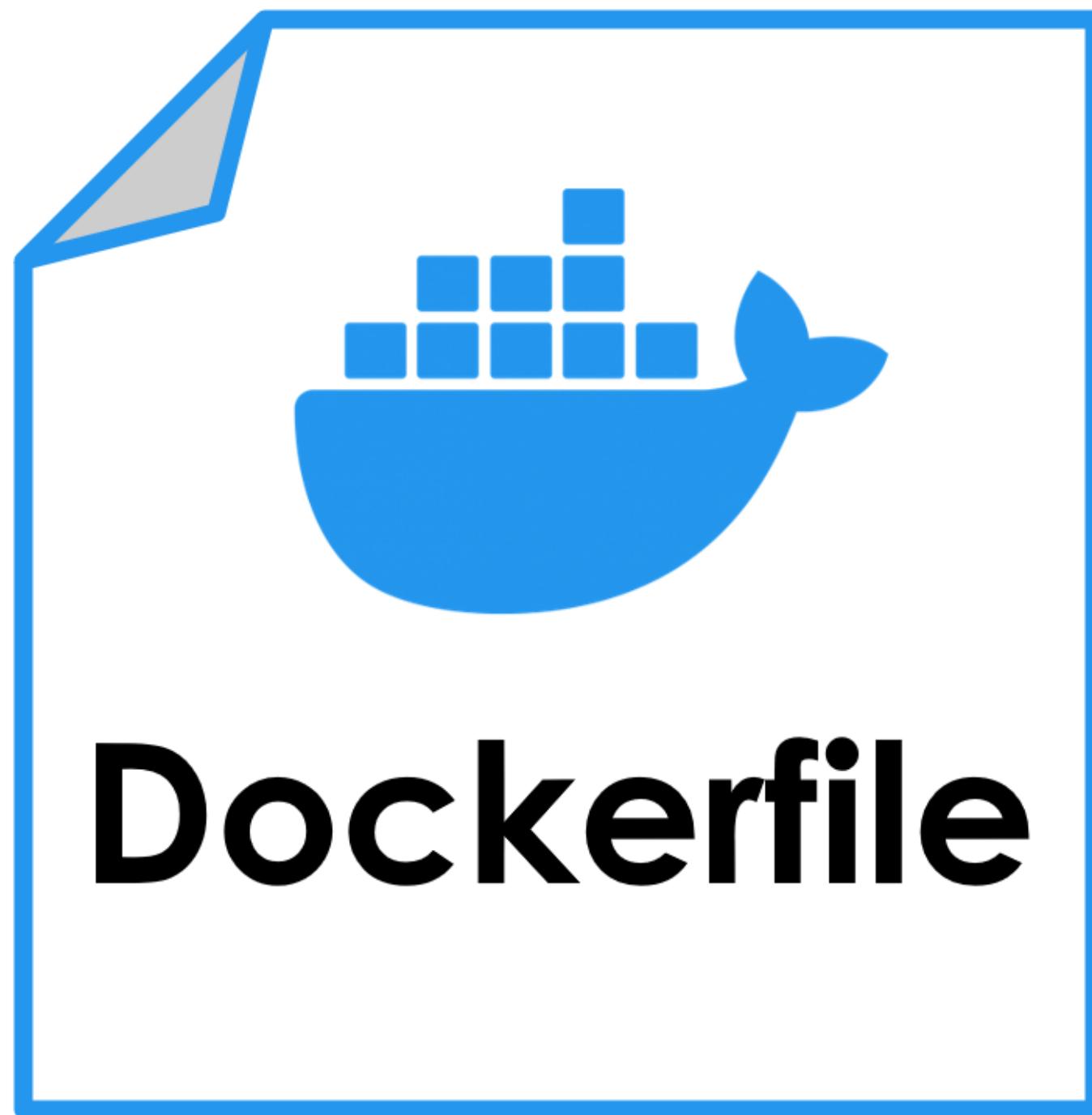
# C'est quoi le principe ?





Le livre de recettes







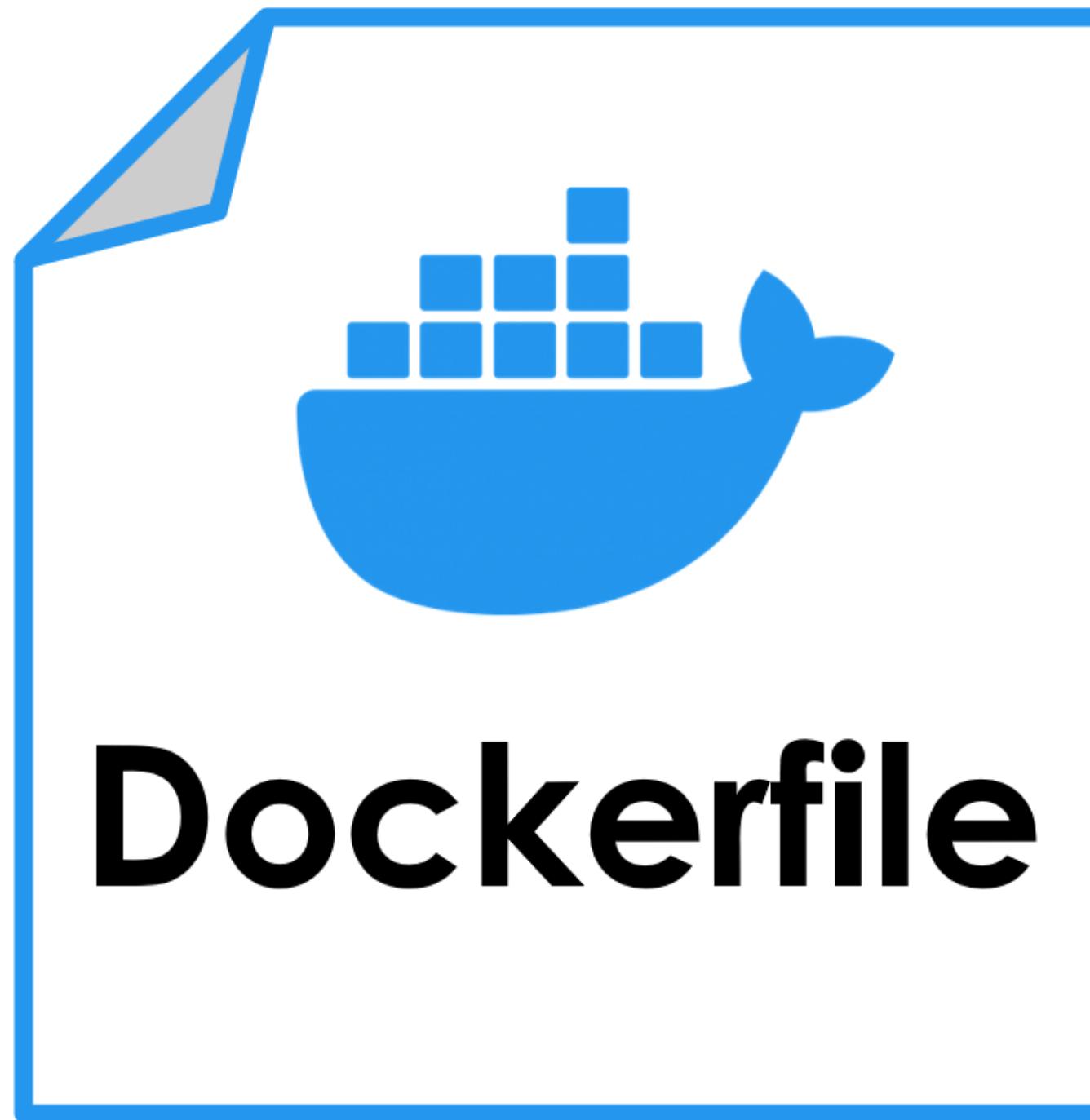
Le livre de recettes



Un simple fichier nommé "Dockerfile" (majuscule sur le D et pas d'extension).



## Le livre de recettes

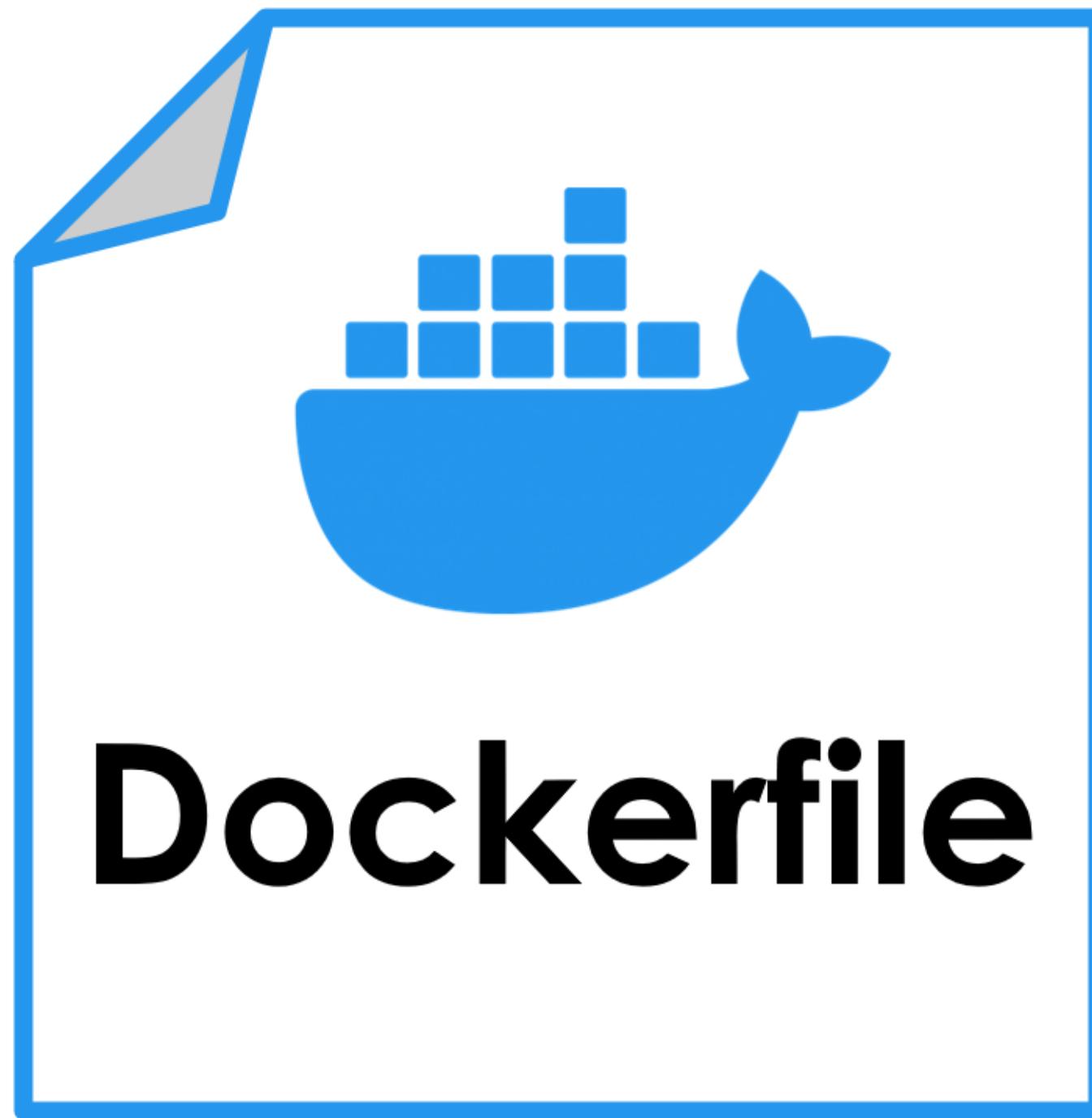


Un simple fichier nommé "Dockerfile" (majuscule sur le D et pas d'extension).

C'est du texte, très pratique à stocker dans Git.



## Le livre de recettes



Un simple fichier nommé "Dockerfile" (majuscule sur le D et pas d'extension).

C'est du texte, très pratique à stocker dans Git.

Une suite de clé-valeur.



# Pourquoi fabriquer sa propre image ?

! Problème :

```
cat /etc/os-release
# ...
git --version
# ...

# Même version de Linux que dans GitPod
docker container run --rm ubuntu:20.04 git --version
# docker: Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "git": executable file not found in $PATH

# En interactif ?
docker container run --rm --tty --interactive ubuntu:20.04 git --version
```

Copy



## Fabriquer sa première image

- **But :** fabriquer une image Docker qui contient git
- Dans votre workspace Gitpod, créez un dossier nommé docker-git/
- Dans ce dossier, créer un fichier Dockerfile avec le contenu ci-dessous :

```
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
```

Copy

- Fabriquez votre image avec la commande docker image build --tag=docker-git <chemin/vers/docker-git/
- Testez l'image fraîchement fabriquée
  - 💡 docker image ls

## ✓ Fabriquer sa première image

```
cat <<EOF >Dockerfile
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
EOF

docker image build --tag=docker-git ./
docker image ls | grep docker-git

# Doit fonctionner
docker container run --rm docker-git:latest git --version
```

Copy

# Conventions de nommage des images

[REGISTRY/] [NAMESPACE/] NAME [:TAG|@DIGEST]

Copy

- Pas de Registre ? Défaut: `registry.docker.com`
- Pas de Namespace ? Défaut: `library`
- Pas de tag ? Valeur par défaut: `latest`
  - $\triangle$  Friends don't let friends use `latest`
- Digest: signature unique basée sur le contenu

# Conventions de nommage : Exemples

- ubuntu:20.04 ⇒ registry.docker.com/library/ubuntu:20.04
- dduportal/docker-asciidoc ⇒ registry.docker.com/dduportal/docker-asciidoc:latest
- ghcr.io/dduportal/docker-asciidoc:1.3.2@sha256:xxxx

 Utilisons les tags

- Rappel : ⚠ Friends don't let friends use latest
- Il est temps de "taguer" votre première image !

```
docker image tag docker-git:latest docker-git:1.0.0
```

Copy

  - Testez le fonctionnement avec le nouveau tag
  - Comparez les 2 images dans la sortie de docker image ls

## ✓ Utilisons les tags

```
docker image tag docker-git:latest docker-git:1.0.0
# 2 lignes
docker image ls | grep docker-git
# 1 ligne
docker image ls | grep docker-git | grep latest
# 1 ligne
docker image ls | grep docker-git | grep '1.0.0'

# Doit fonctionner
docker container run --rm docker-git:1.0.0 git --version
```

Copy



## Mettre à jour votre image (1.1.0)

- Mettez à jour votre image en version 1.1.0 avec les changements suivants :
  - Ajoutez un `LABEL` dont la clef est `description` (et la valeur de votre choix)
  - Configurez `git` pour utiliser une branche `main` par défaut au lieu de `master` (commande `git config --global init.defaultBranch main`)
- Indices :
  - 💡 Commande `docker image inspect <image name>`
  - 💡 Commande `git config --get init.defaultBranch` (dans le conteneur)
  - 💡 Ajoutez des lignes **à la fin** du Dockerfile
  - 💡 Documentation de référence des Dockerfile

## ✓ Mettre à jour votre image (1.1.0)

```
cat ./Dockerfile
FROM ubuntu:20.04
RUN apt-get update && apt-get install --yes --no-install-recommends git
LABEL description="Une image contenant git préconfiguré"
RUN git config --global init.defaultBranch main

docker image build -t docker-git:1.1.0 ./docker-git/
# Sending build context to Docker daemon 2.048kB
# Step 1/4 : FROM ubuntu:20.04
#  --> e40cf56b4be3
# Step 2/4 : RUN apt-get update && apt-get install --yes --no-install-recommends git
#  --> Using cache
#  --> 926b8d87f128
# Step 3/4 : LABEL description="Une image contenant git préconfiguré"
#  --> Running in 0695fc62ecc8
# Removing intermediate container 0695fc62ecc8
#  --> 68c7d4fb8c88
# Step 4/4 : RUN git config --global init.defaultBranch main
#  --> Running in 7fb54ecf4070
# Removing intermediate container 7fb54ecf4070
#  --> 2858ff394edb
Successfully built 2858ff394edb
Successfully tagged docker-git:1.1.0

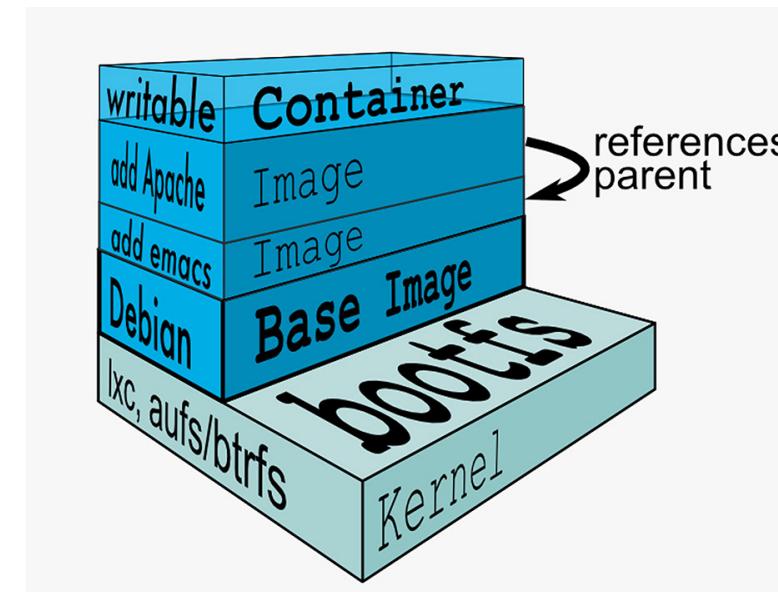
docker container run --rm docker-git:1.0.0 git config --get init.defaultBranch
docker container run --rm docker-git:1.1.0 git config --get init.defaultBranch
# main
```

# Cache d'images & Layers

```
Step 2/4 : RUN apt-get update && apt-get install --yes --no-install-recommends git  
---> Using cache
```

Copy

🤔 En fait, Docker n'a PAS exécuté cette commande la seconde fois ⇒ ça va beaucoup plus vite !



🎓 Essayez de voir les layers avec (dans Gitpod) `dive <image>:<tag>`



## Cache d'images & Layers

- **But :** manipuler le cache d'images
- Commencez par vérifier que le cache est utilisé : relancez la dernière commande `docker image build` (plusieurs fois s'il le faut)
- Invalidez le cache en ajoutant le paquet `APT make` à installer en même temps que `git`
  - $\triangle$  Tag 1.2.0
- Vérifiez que le cache est bien présent de nouveau

## ✓ Cache d'images & Layers

```
# Build one time
docker image build -t docker-git:1.1.0 ./docker-git/
# Second time is fully cached
docker image build -t docker-git:1.1.0 ./docker-git/

cat Dockerfile
# FROM ubuntu:20.04
# RUN apt-get update && apt-get install --yes --no-install-recommends git make
# LABEL description="Une image contenant git préconfiguré"
# RUN git config --global init.defaultBranch main

# Build one time
docker image build -t docker-git:1.2.0 ./docker-git/
# Second time is fully cached
docker image build -t docker-git:1.2.0 ./docker-git/

## Vérification
# Renvoie une erreur
docker run --rm docker-git:1.1.0 make --version
# Doit fonctionner
docker run --rm docker-git:1.2.0 make --version
```

Copy

# Checkpoint

- Une image Docker fournit un environnement de système de fichier auto-suffisant (application, dépendances, binaries, etc.) comme modèle de base d'un conteneur
- Les images Docker ont une convention de nommage permettant d'identifier les images très précisément
- On peut spécifier une recette de fabrication d'image à l'aide d'un `Dockerfile` et de la commande `docker image build`

⇒ 🤔 et si on utilisait Docker pour nous aider dans l'intégration continue ?

Fichiers, nommage, inspect...

# Volumes

# Réseaux

# Docker compose

# Bibliographie

# Ligne de commande

- <https://tldp.org>
- <https://en.wikipedia.org/wiki/POSIX>
- [https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print\\_loop](https://en.wikipedia.org/wiki/Read%E2%80%93eval%E2%80%93print_loop)
- <https://linuxhandbook.com/linux-directory-structure/>

# Git / VCS

- <https://docs.github.com>
- <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control>
- <http://martinfowler.com/bliki/VersionControlTools.html>
- <http://martinfowler.com/bliki/FeatureBranch.html>
- <https://about.gitlab.com/2014/09/29/gitlab-flow/>
- <https://www.atlassian.com/git/tutorials/comparing-workflows>
- <http://nvie.com/posts/a-successful-git-branching-model/>

# Intégration Continue

- <http://martinfowler.com/articles/continuousIntegration.html>
- <http://martinfowler.com/bliki/ContinuousDelivery.html>
- <https://jaxenter.com/implementing-continuous-delivery-117916.html>
- <https://technologyconversations.com/2014/04/29/continuous-delivery-introduction-to-concepts-and-tools/>
- <http://blog.arungupta.me/continuous-integration-delivery-deployment-maturity-model>
- <http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment>

# Docker

- <https://gounthar.github.io/cours/cnam-docker-2018>
- <https://kodekloud.com/blog/docker-for-beginners/>
- <https://www.slideshare.net/dotCloud/why-docker>
- <https://docs.docker.com/engine/reference/builder/>
- <https://www.r-bloggers.com/2021/05/best-practices-for-r-with-docker/>
- <https://github.com/wagoodman/dive>
- <https://docs.docker.com/engine/tutorials/networkingcontainers/>
- <https://towardsdatascience.com/docker-networking-919461b7f498>

# Merci !

✉ gounthar@gmail.com

Slides: <https://gounthar.github.io/gounthar/cours-devops-docker/updated-dependencies>



Source on : <https://github.com/gounthar/gounthar/cours-devops-docker>