

```
    setOnLongClickListener(this)  
}
```

# DevOps

CPE - 2023

```
override fun onLongClick(view: View?): Boolean {  
    if (download != null && downloadName != null)  
        AlertDialog.Builder(context).showQuestion(  
            Toast.makeText(context, R.string.download_name_error, Toast.LENGTH_SHORT))  
    return true  
}
```



# Planning

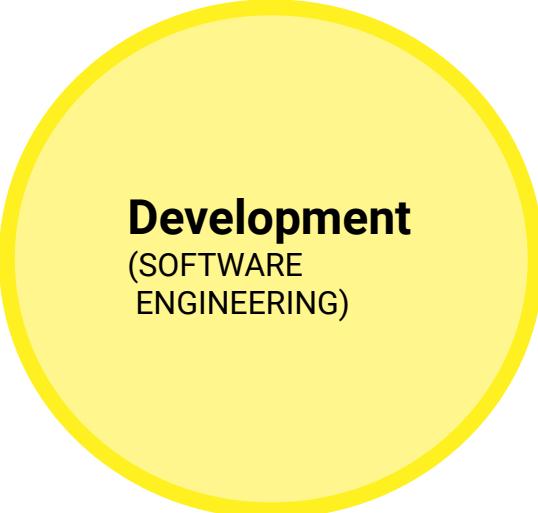
	Lundi	Mardi	Mercredi	Jeudi
AM	Cours 	Cours	Cours	
PM		Guide TD		

Lundi 20/02 15h45-17h45
Examen écrit

# DevOps

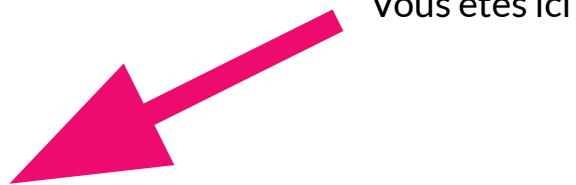
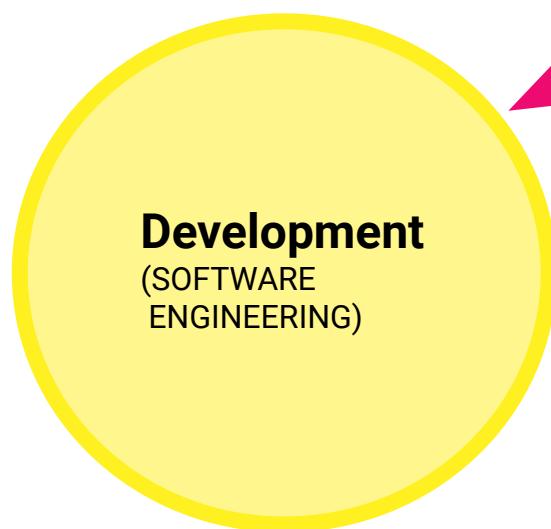


# Dev vs DevOps



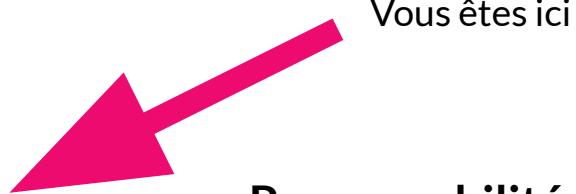
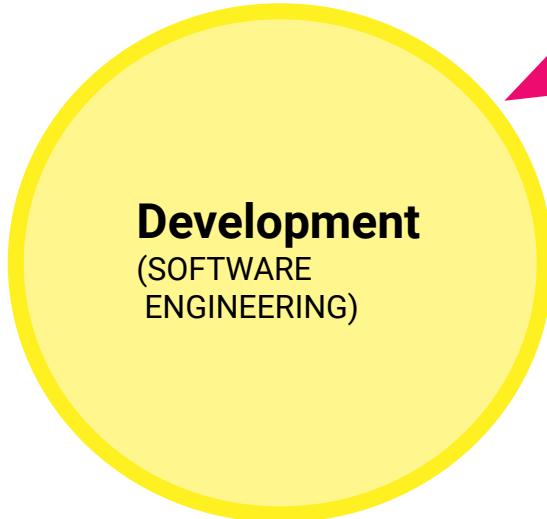
**Development**  
(SOFTWARE  
ENGINEERING)

# Dev vs DevOps



Vous êtes ici

# Dev vs DevOps

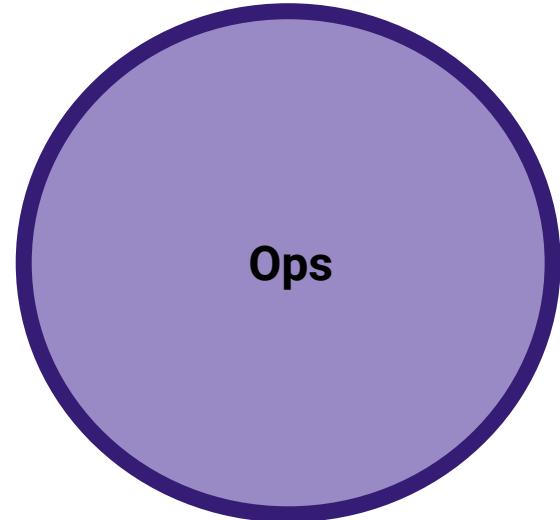


## Responsabilité principale

Innover, créer des fonctionnalités

Le plus vite possible, le moins cher possible

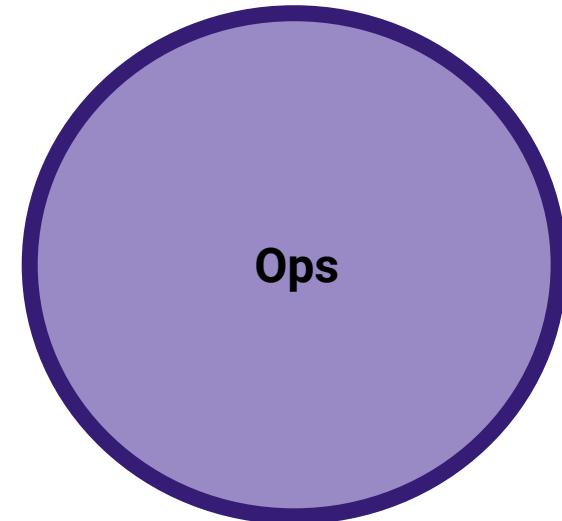
# Dev vs DevOps



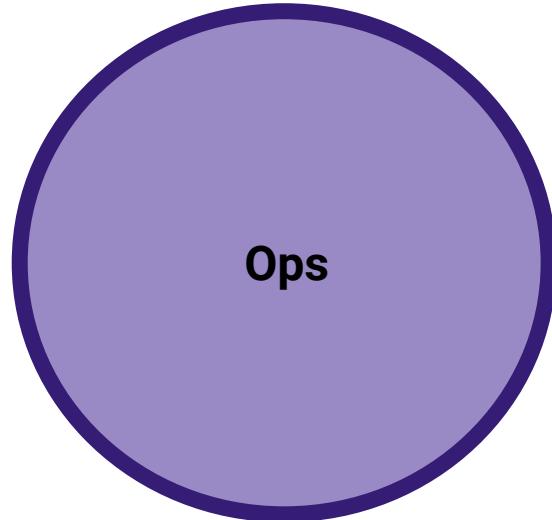
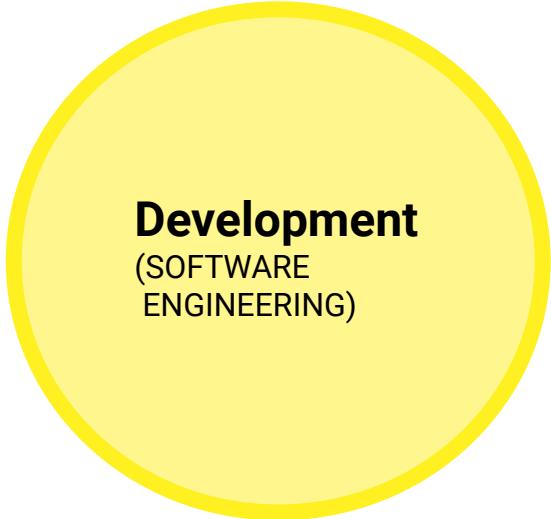
# Dev vs DevOps

**Responsabilité principale**

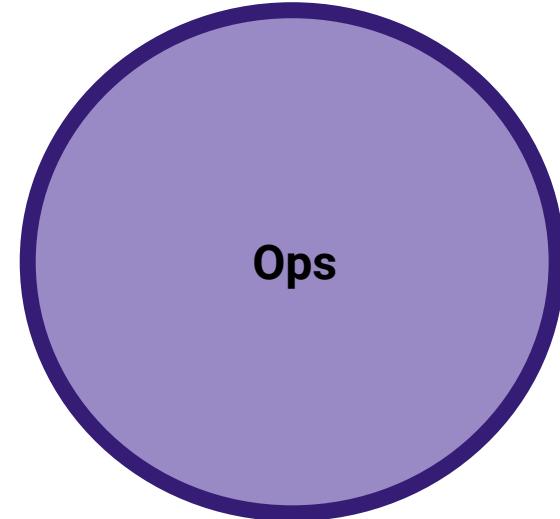
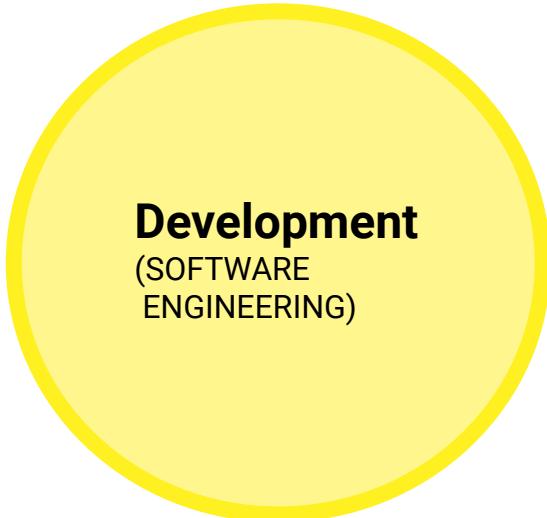
Faire en sorte que ça marche



# Dev vs DevOps



# Dev vs DevOps



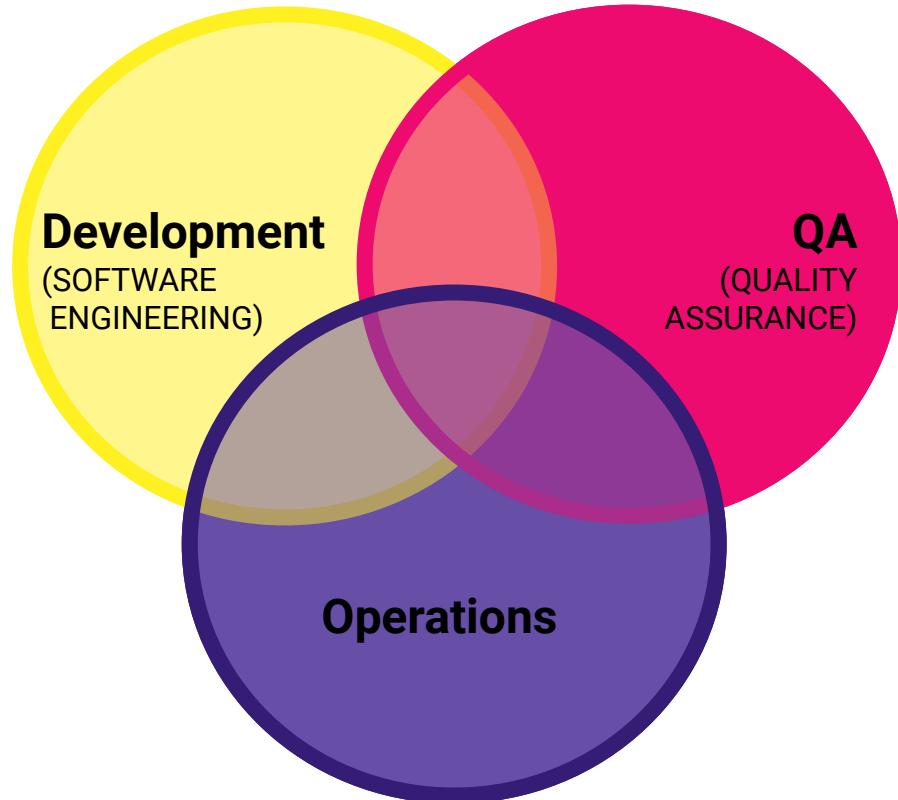
# Dev vs DevOps



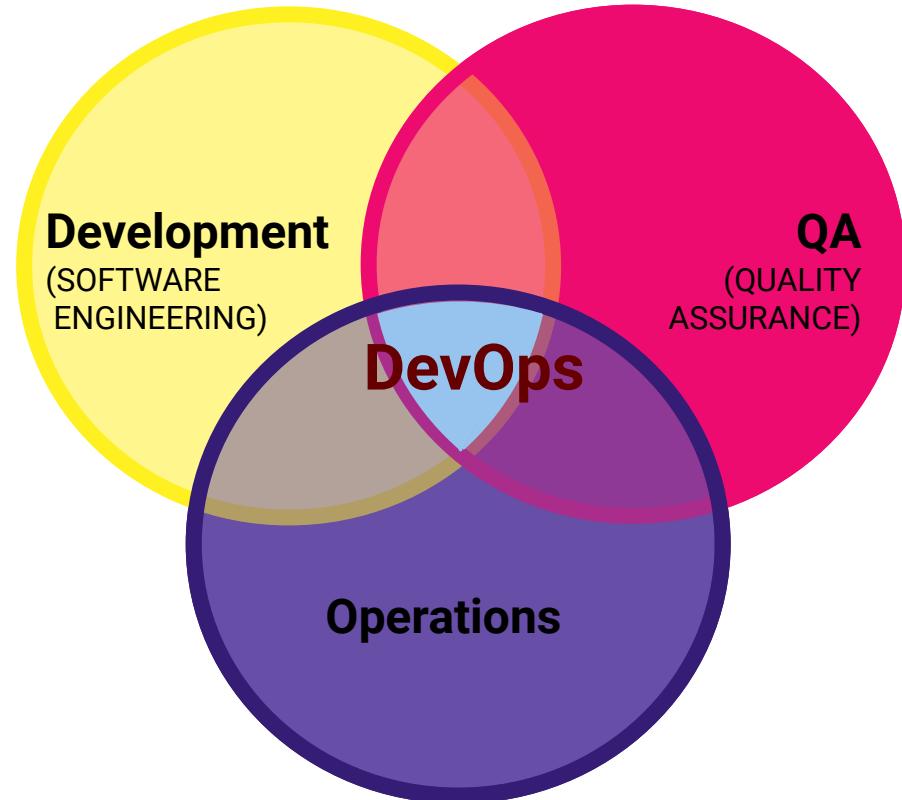
# Dev? Ops? DevOps

Mais l'innovation accélère...

On ne peut plus attendre 6 mois



# Dev? Ops? DevOps

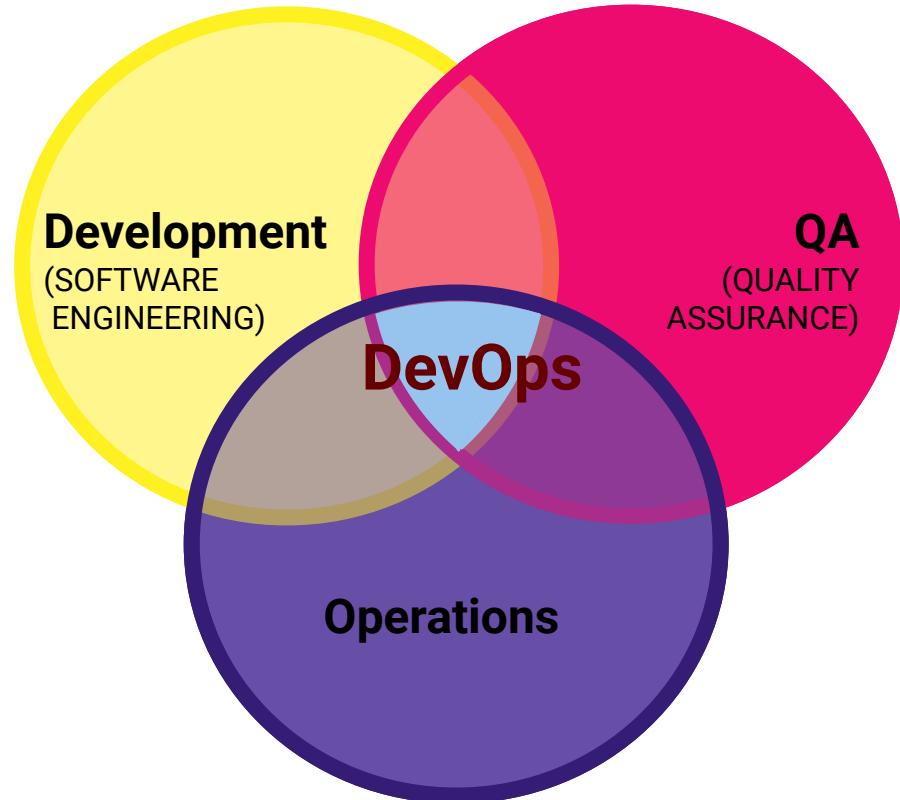


# Dev? Ops? DevOps

DevOps = une philosophie

Favoriser l'innovation continue

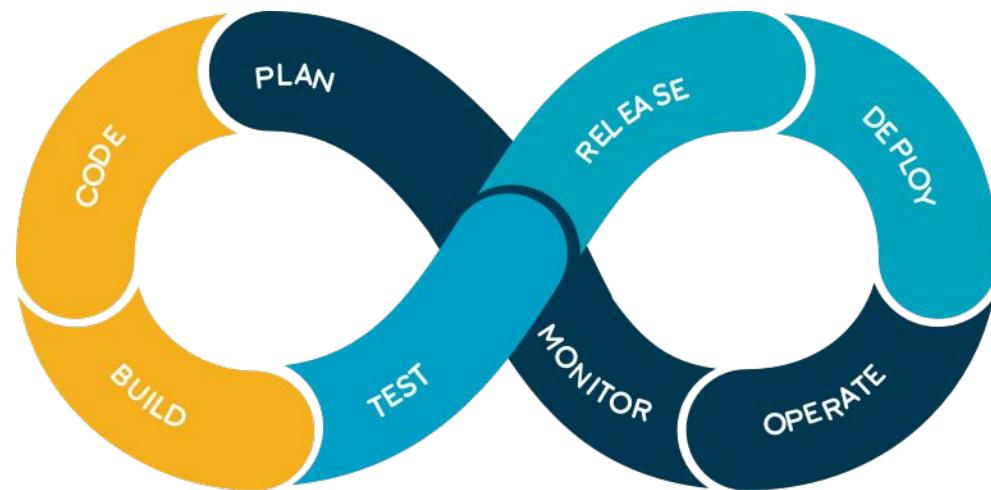
- Réorganisation des rôles
- Automatisant les process



# Dev? Ops? DevOps

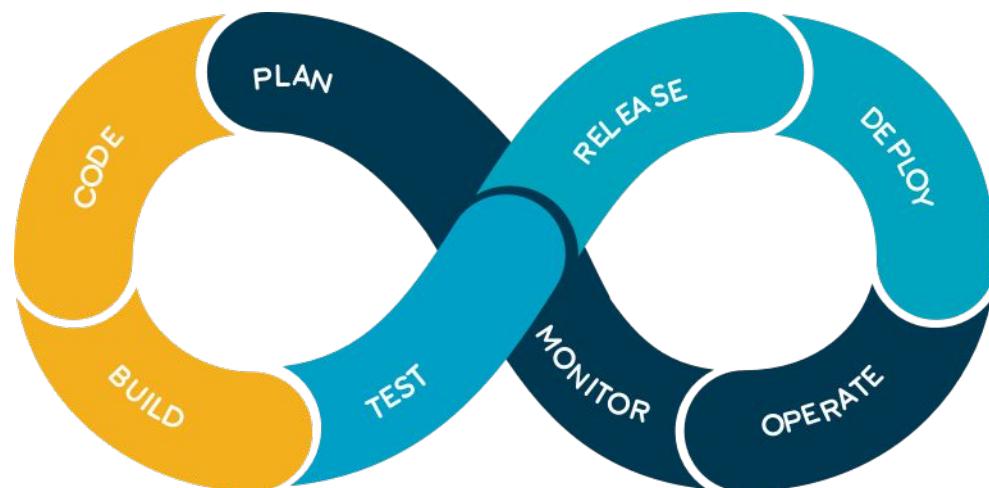
- 3 grands principes : Continuous...

- Improvements
- Feedback
- Learning & Experiment



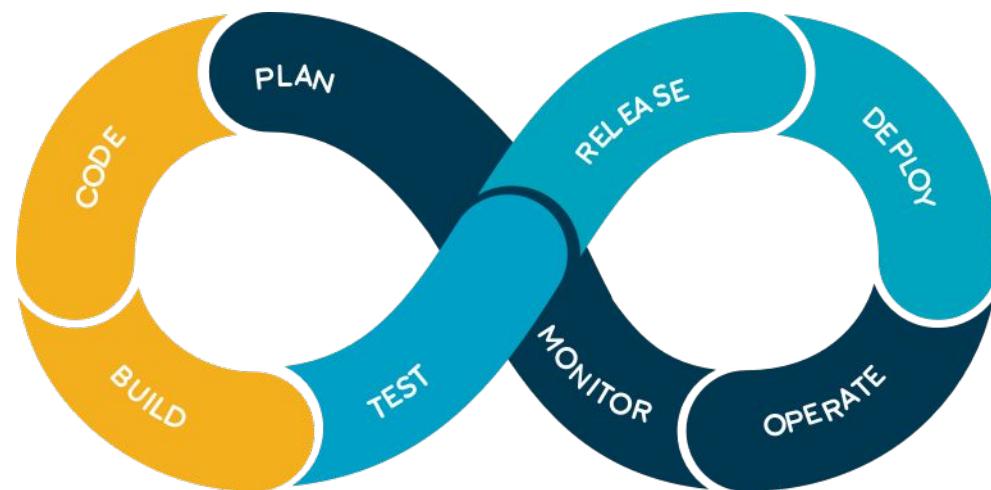
# Dev? Ops? DevOps

- 3 grands principes : Continuous...
  - Improvements
  - Feedback
  - Learning & Experiment
- Automatiser grâce à
  - Des outils de versioning de code
  - Une CI/CD
  - Des outils de monitoring / alerting



# Dev? Ops? DevOps

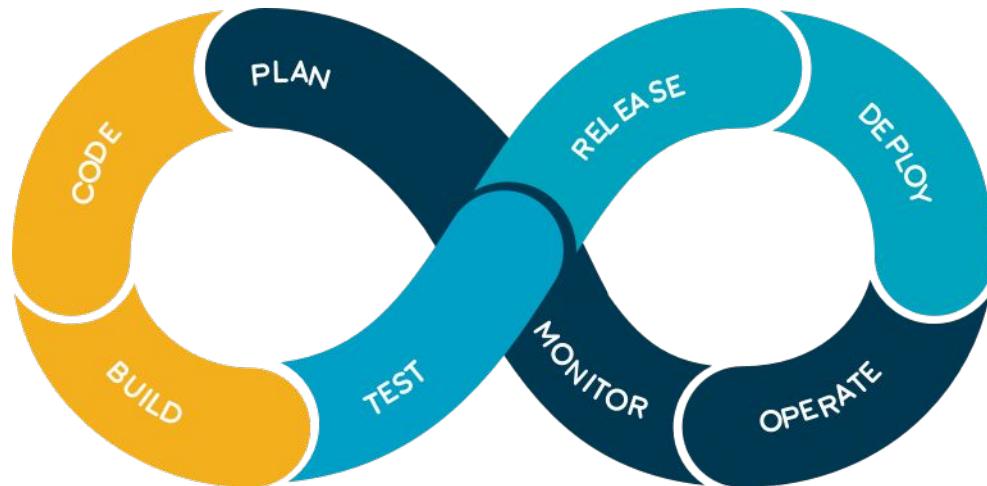
- 3 grands principes : Continuous...
  - Improvements
  - Feedback
  - Learning & Experiment
- Automatiser grâce à
  - Des outils de versioning de code
  - Une CI/CD
  - Des outils de monitoring / alerting



**“Everything as code”**

# Everything as code ?

- Pipeline & deployment as code
- Environments as code
- Orchestration as code
- Infrastructure as code
- Kitten as code



# Un univers très dense



APMs, Monitoring, Alerting



APPDYNAMICS



New  
Relic.



DATADOG

Provisioning, Environments,  
Infrastructure & Orchestration



CHEF



ANSIBLE



OPENSIFT



CI/CD, Test & Quality Gates,  
Pipeline tools



# À retenir

- Le DevOps est une philosophie qui permet :
  - d'améliorer la collaboration entre les équipes
  - de développer des applications de production robuste
- Basé sur :
  - du **code**
  - de l'automatisation
  - les bonnes pratiques de craftsmanship

⇒ C'est un univers complet

# Un univers très dense



APMs, Monitoring, Alerting



APPDYNAMICS



New  
Relic.



DATADOG



Provisioning, Environments,  
Infrastructure & Orchestration



ANSIBLE



docker



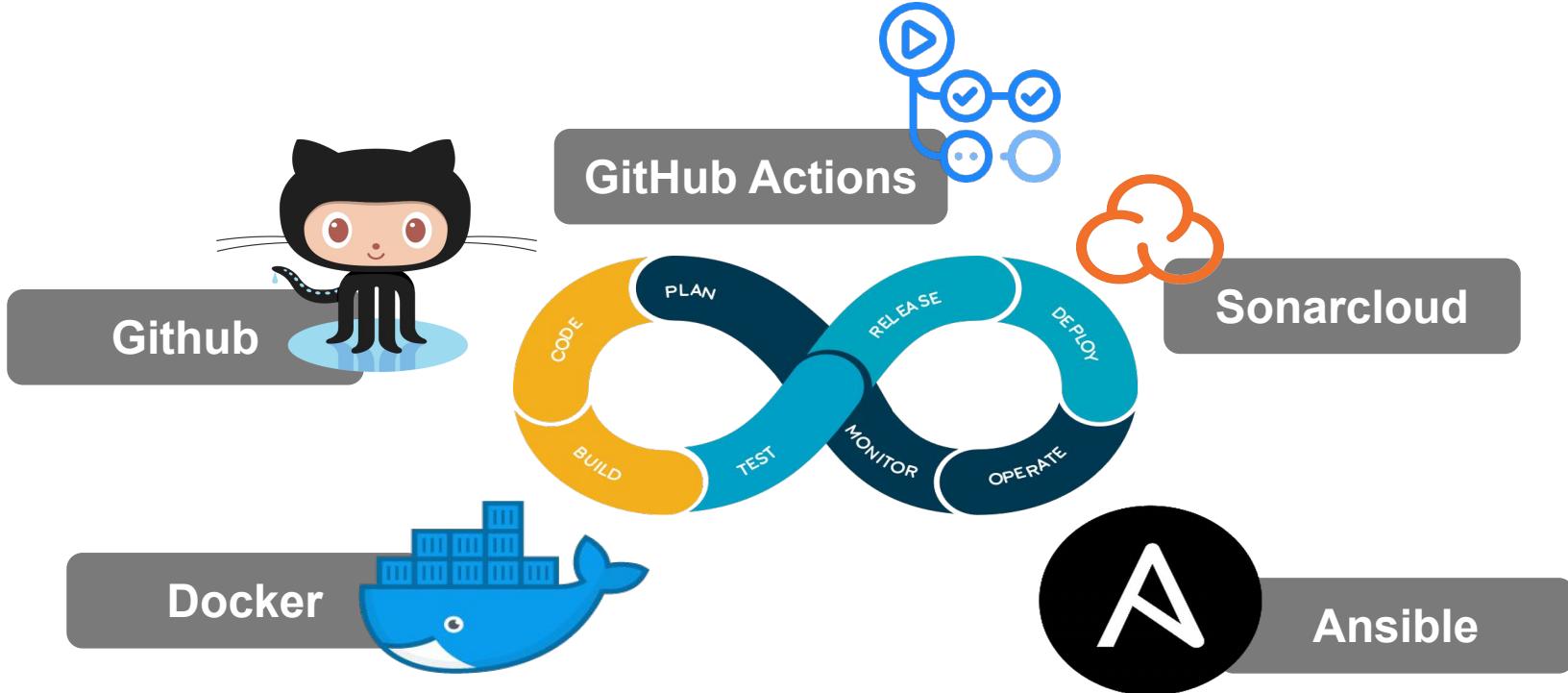
OPENSIFT



CI/CD, Test & Quality Gates,  
Pipeline tools



# Alors, comment on organise ça?



# Mettre une application dans un container

avec Docker

# **Compiler, Tester, et Déployer automatiquement une appli**

Continuous Integration & Continuous Delivery

# Provisionner un serveur

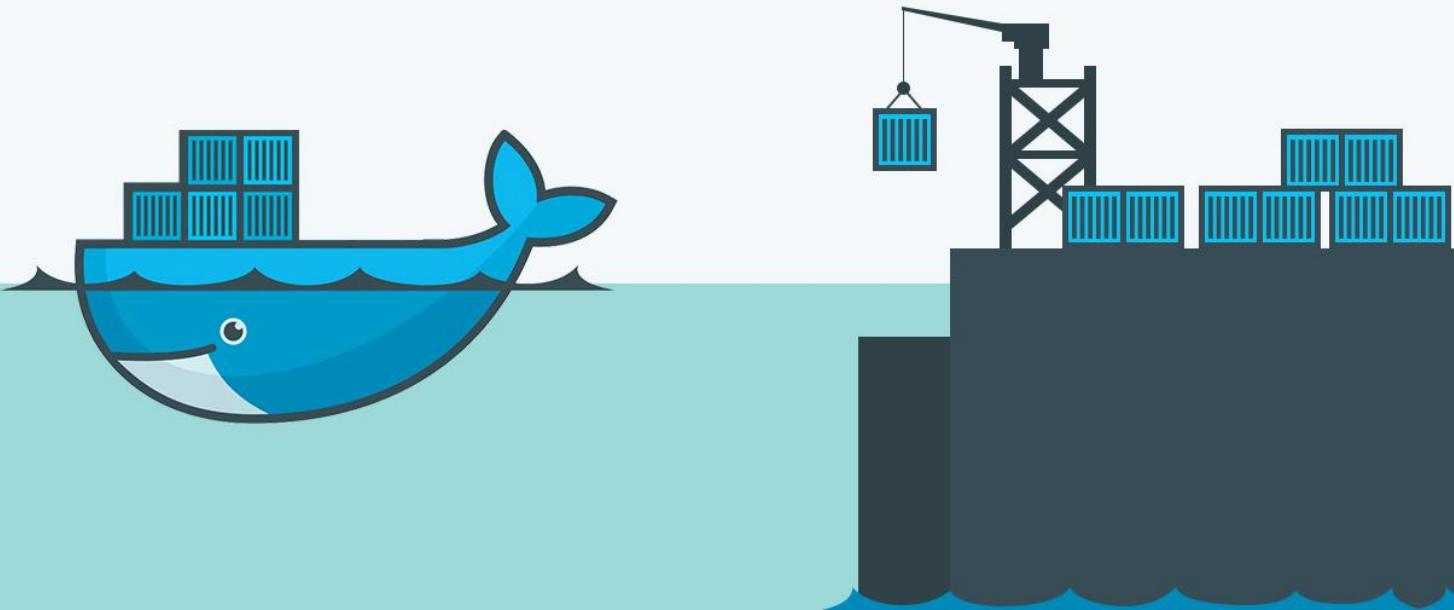
avec Ansible



Ready ?

# Docker

Put your application into containers





Docker

# Mise en situation

Une application 3-tiers basique :



Un serveur HTTP pour votre site internet



Une application Java 11 pour votre API



Une base de donnée Postgres pour la persistance

# Il était une fois...

Et vous allez voir votre équipe Ops

Qui va s'occuper de runner tout ça.

Que doit-elle installer ?

Entreprise



# Il était une fois...

Elle installe les runtimes (HTTPD, JRE 11, PostgreSQL)

Elle configure les runtimes

Elle donne les bons fichiers de configuration

Elle ouvre les ports correspondant / gère la sécurité

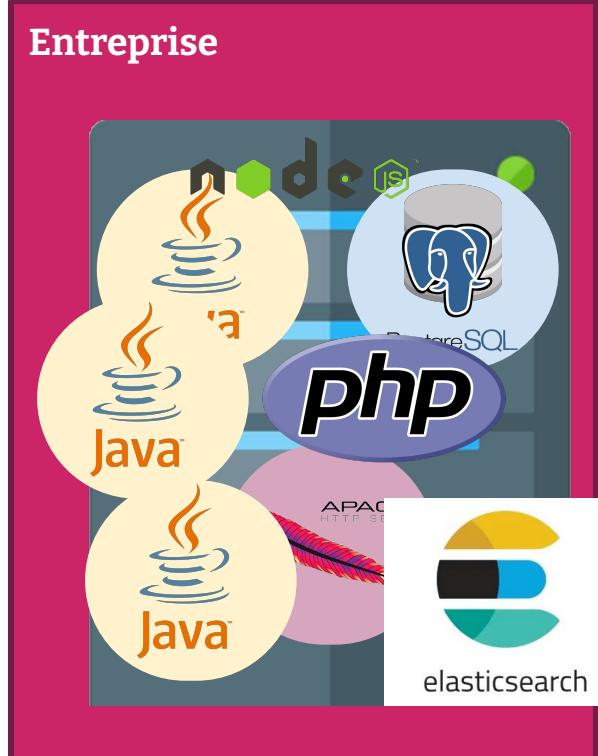


# Il était une fois...

Et plus le temps avance... plus on en a...

- ✗ avec différentes stacks
  - qui ont des langages différents
  - qui ont des versions différentes
- ✗ avec beaucoup de config
  - serveurs http, SSL certs, firewalls, LDAP, ...
- ✗ toutes sur la même machine
  - mauvais pour la sécurité & la résilience\*

## Entreprise



# Il était une fois...

Ca devient le bordel...

Qu'est-ce qu'on fait ?

Entreprise



# Il était une fois...

Ca devient le bordel...

Qu'est-ce qu'on fait ?

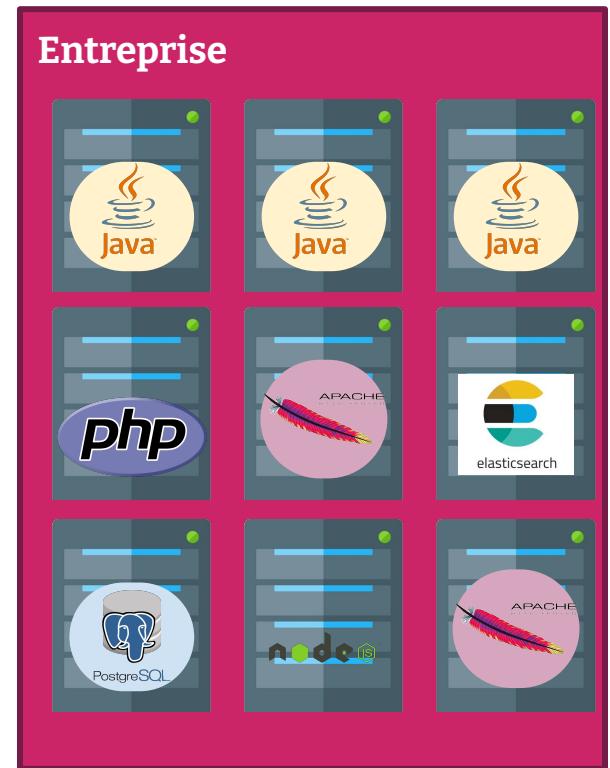
⇒ On met un server / App

Entreprise



# Il était une fois...

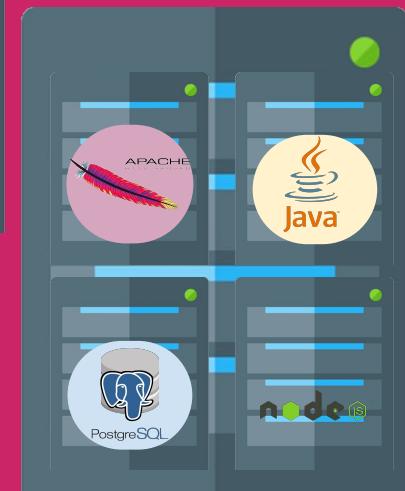
- Un server / App
  - ✓ Parfait pour l'isolation  
(IP / FS / ressources / OS...)
  - ✓ Best performances
  - ✗ Coûteux
  - ✗ Plus dur à déployer
  - ✗ Non scalable
  - ✗ Gestion des pannes complexe



# Il était une fois...

- Une VM / App
  - ✓ TB pour l'isolation  
(IP / FS / ressources / OS...)
  - ✓ Bonnes performances
  - ✓ Bonne déployabilité
  - ✓ Moins coûteux
  - ✓ Scalable
  - ✓ Plus facile pour gérer les pannes

## Entreprise



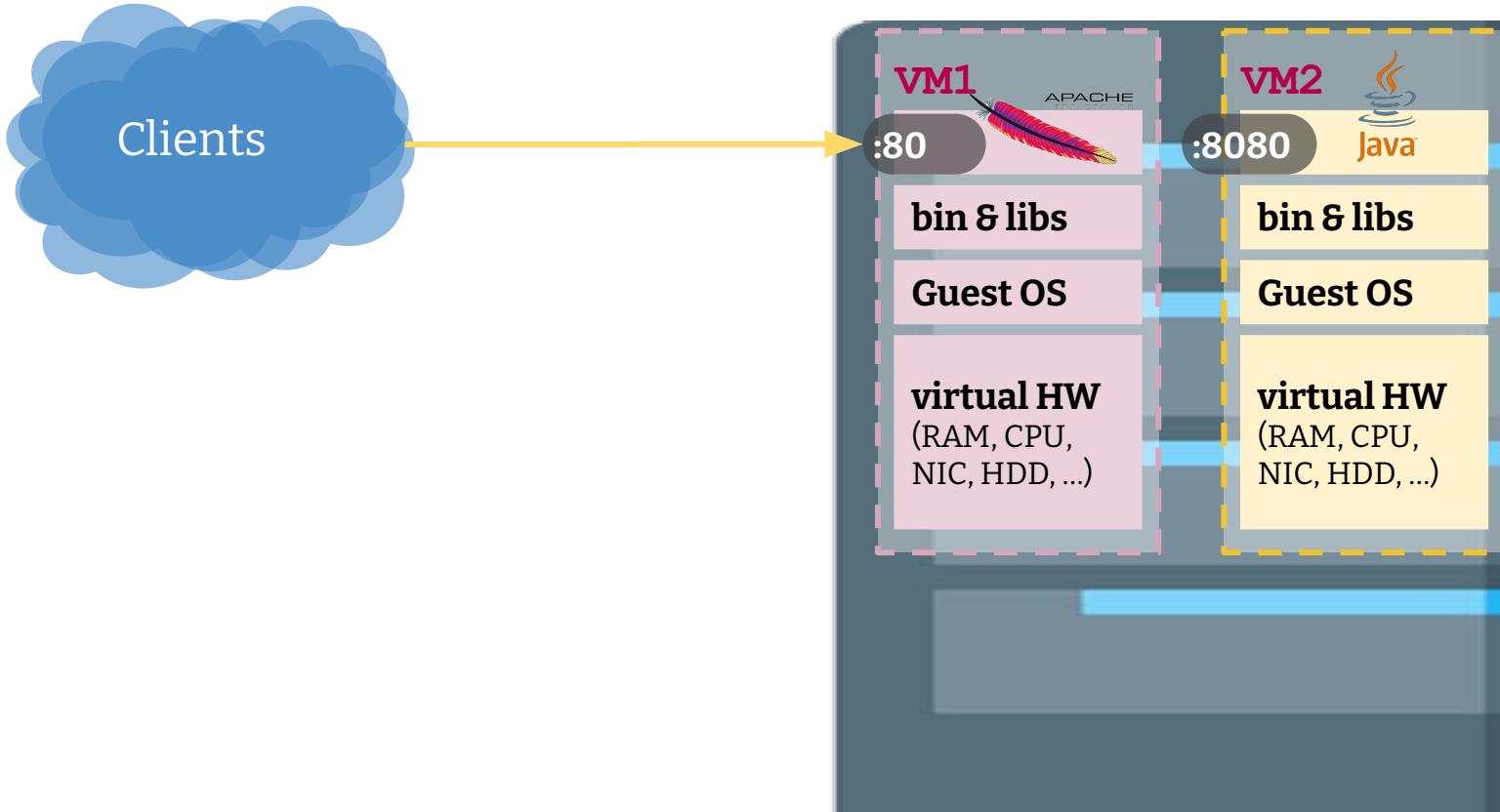
# Containers

## Pourquoi diable ?



# Pourquoi diable les containers

## Le modèle VM



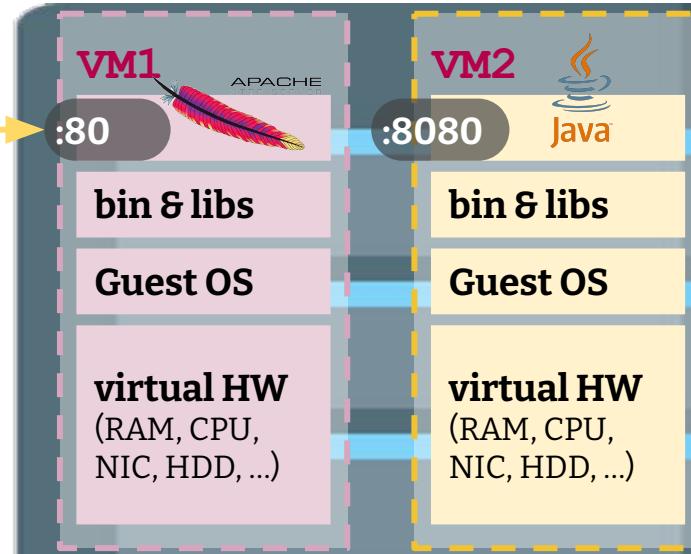
Pourquoi diable les containers

## Le modèle VM



→

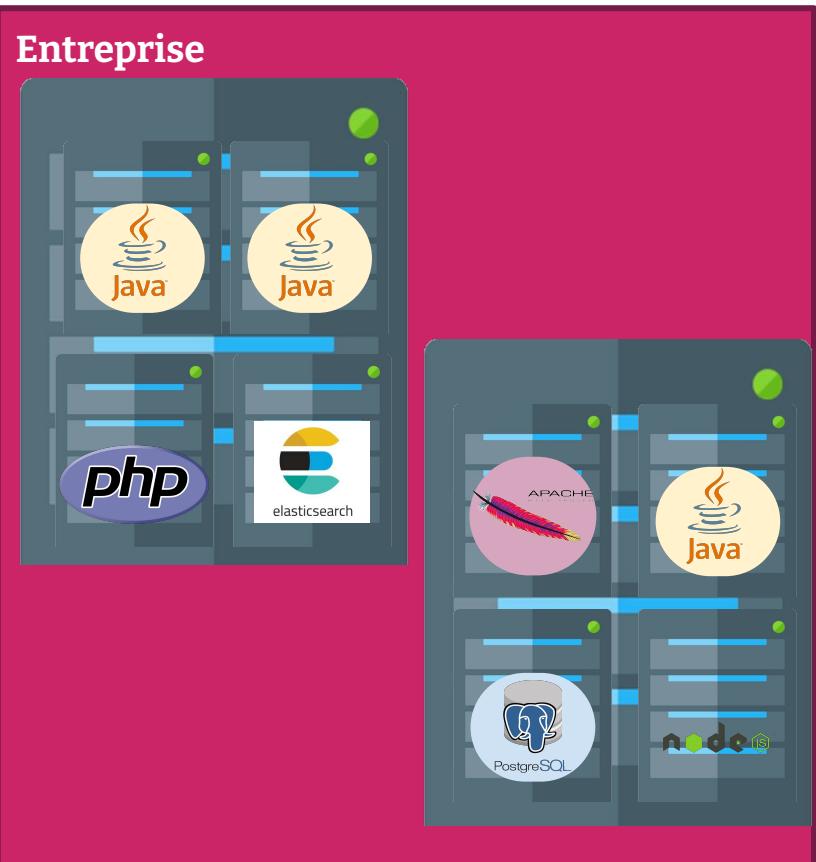
Client VMs



- ✗ hardware virtuel moins performant
- ✗ overhead du guest OS
- ✗ démarrage lent du guest OS
- ✗ image de VM ⇔ un disk snapshot  
⇒ dur à distribuer

# Il était une fois...

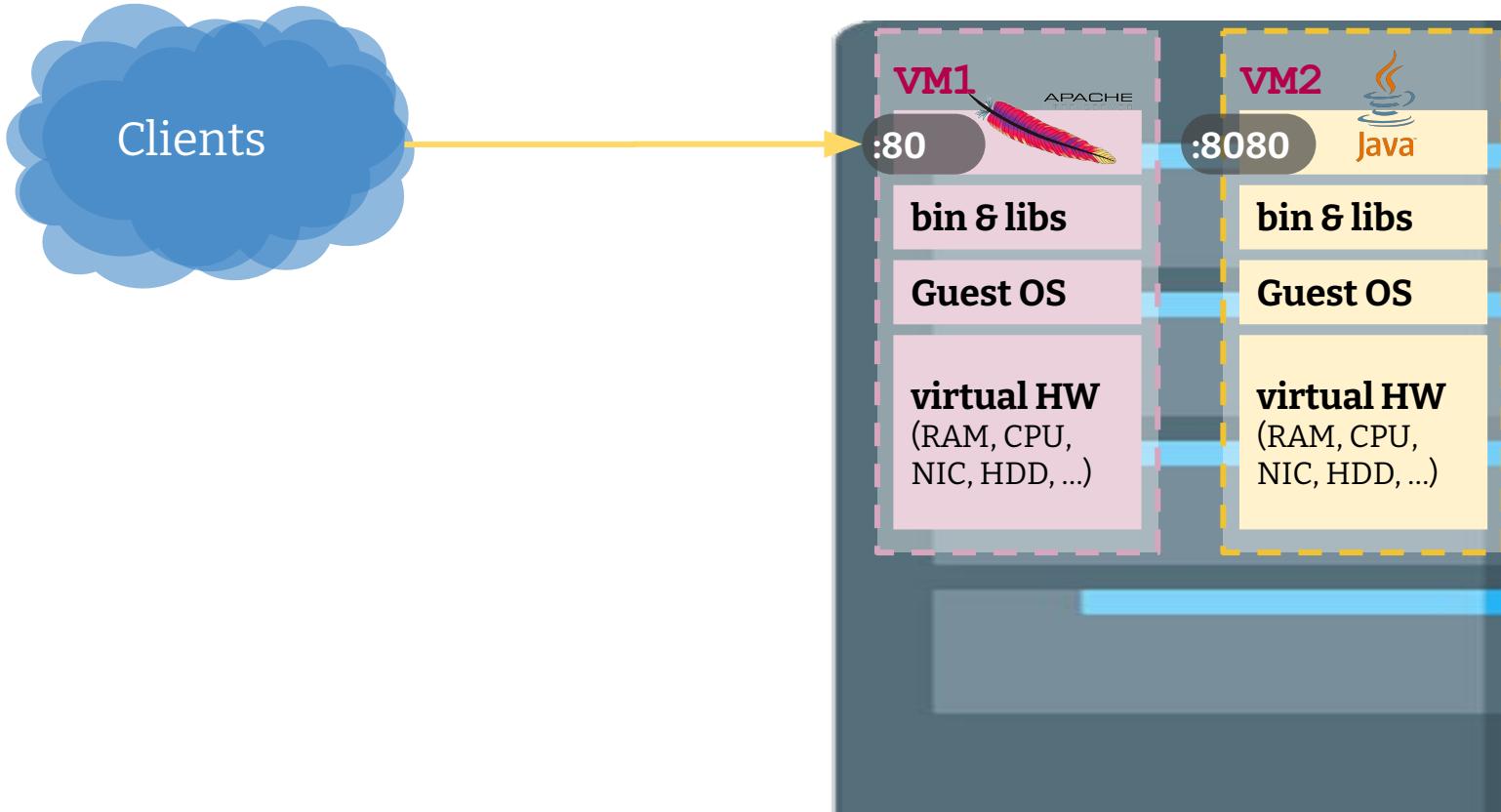
- Une VM / App
  - ✓ TB pour l'isolation  
(IP / FS / ressources / OS...)
  - ✓ Bonnes performances
  - ✗ Déploiement “volumineux”
  - ✗ Overhead OS ⇒ Coûteux
  - ✗ Maintenabilité (avant Vagrant)





# Pourquoi diable les containers

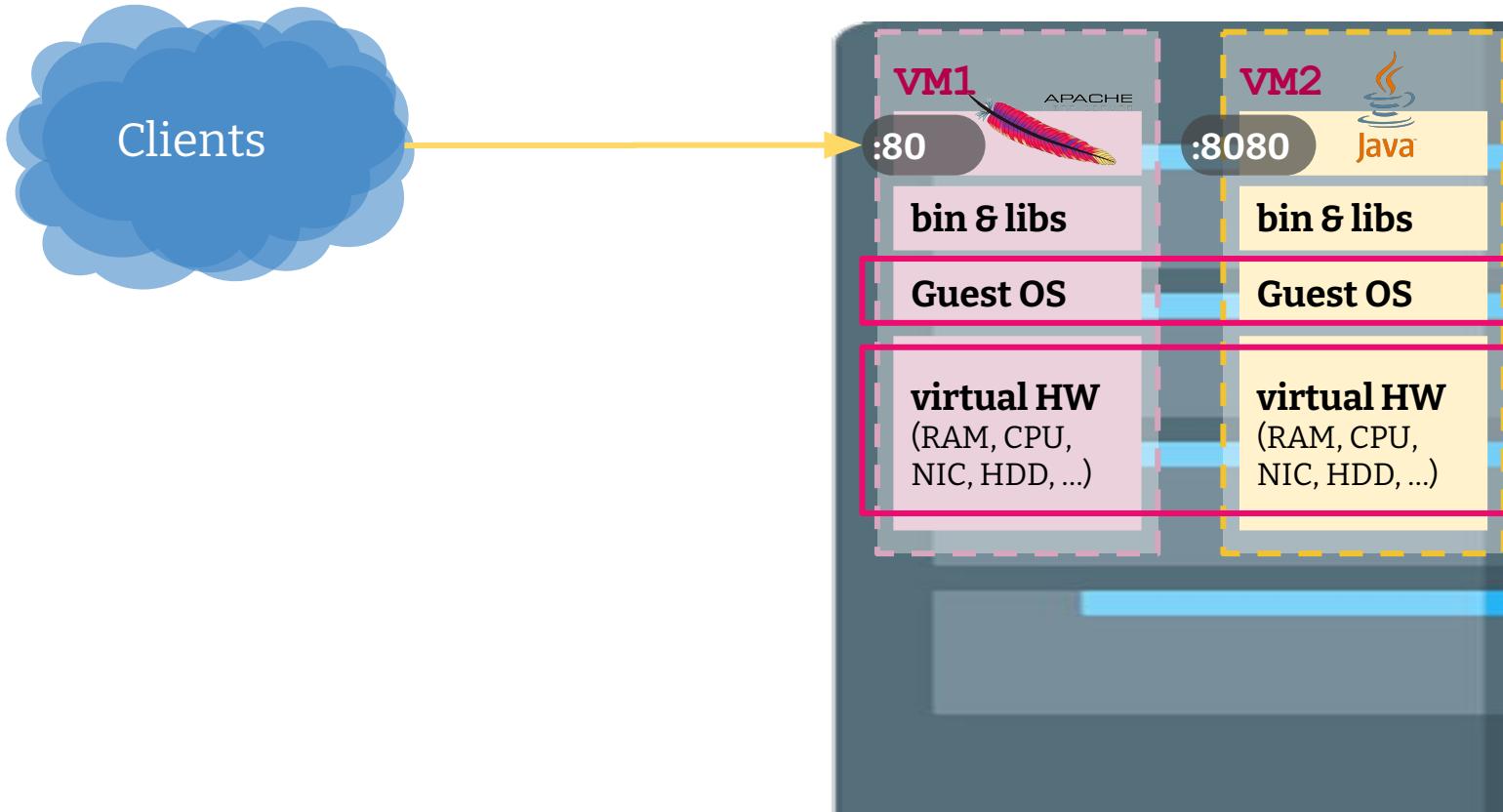
## Le modèle VM





# Pourquoi diable les containers

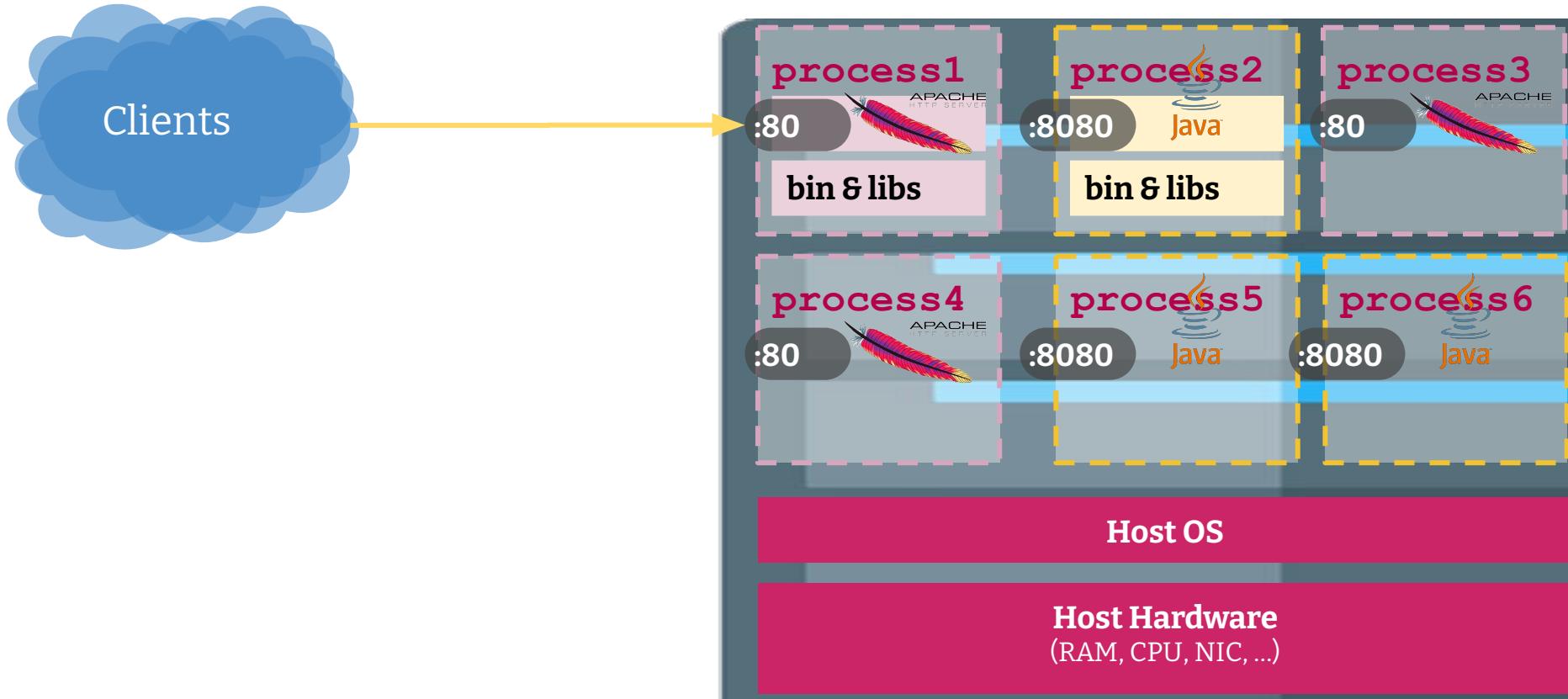
## Le modèle VM





Pourquoi diable les containers ?

## Le “rêve”



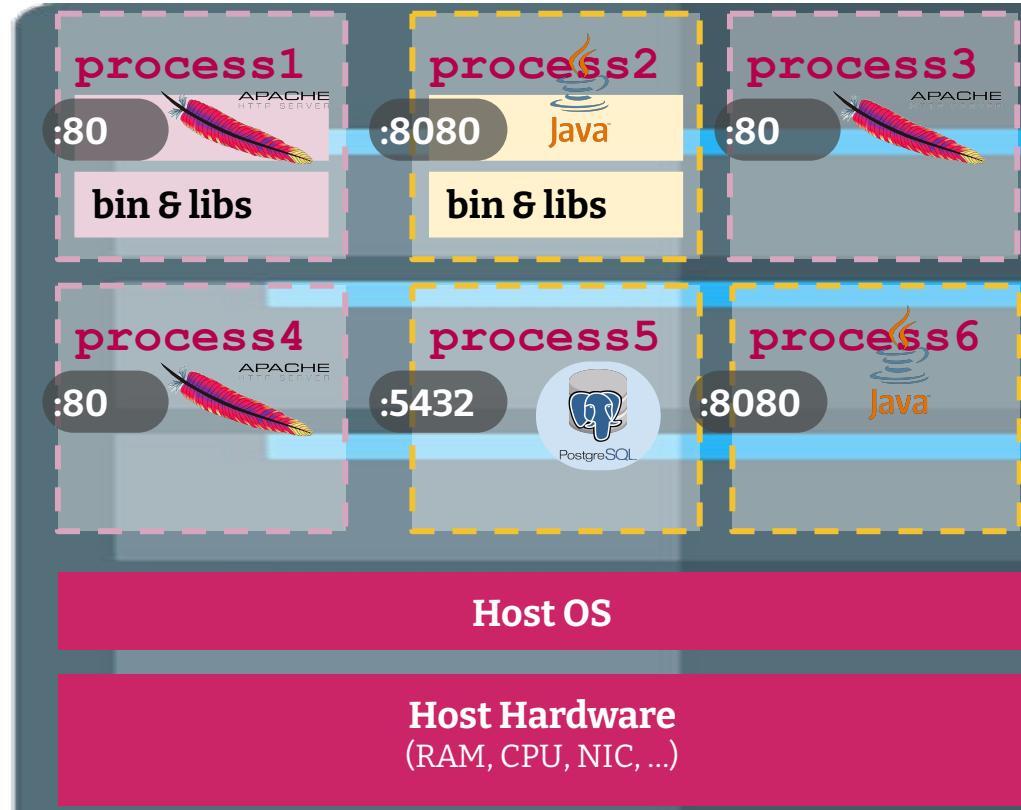


# Pourquoi diable les containers ?

## Le “rêve”

Clients

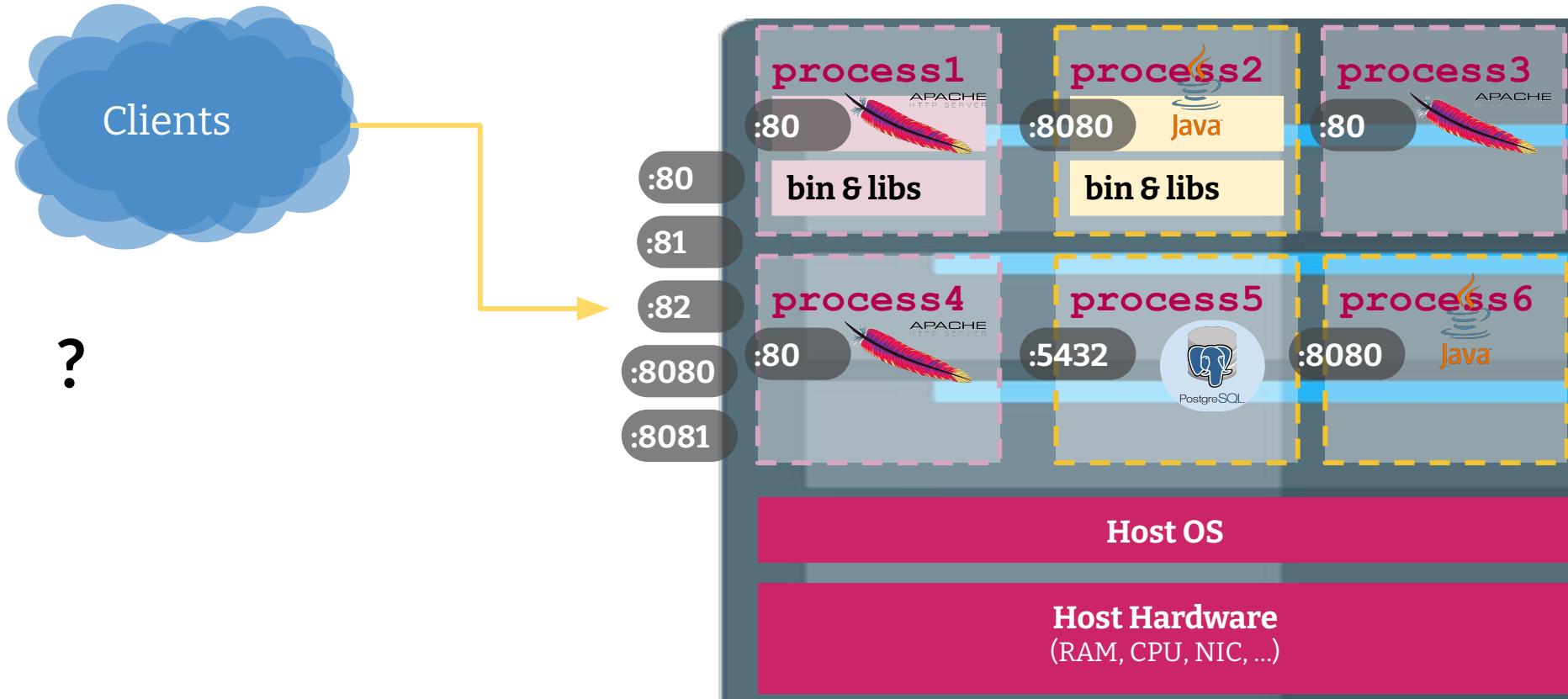
LXC - 2009





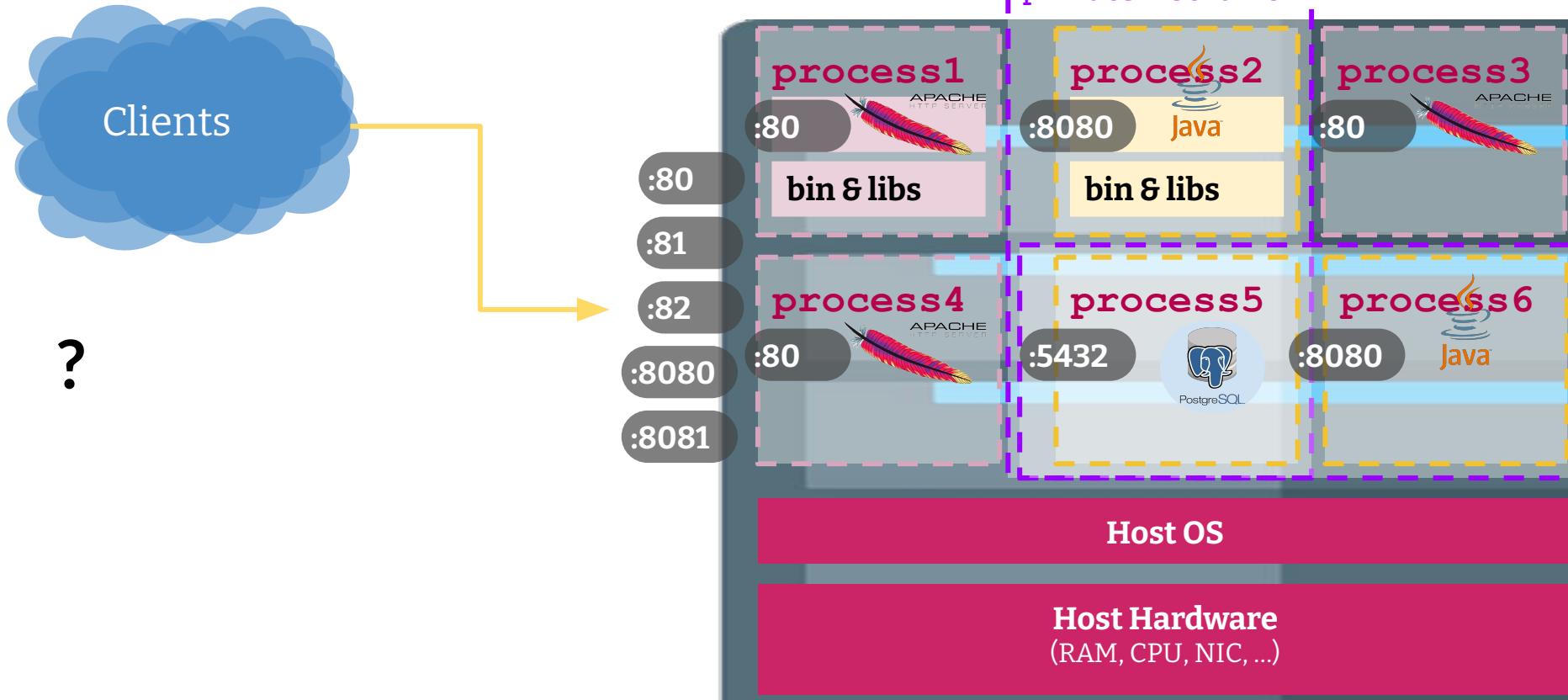
Pourquoi diable les containers ?

## Le “rêve”



Pourquoi diable les containers ?

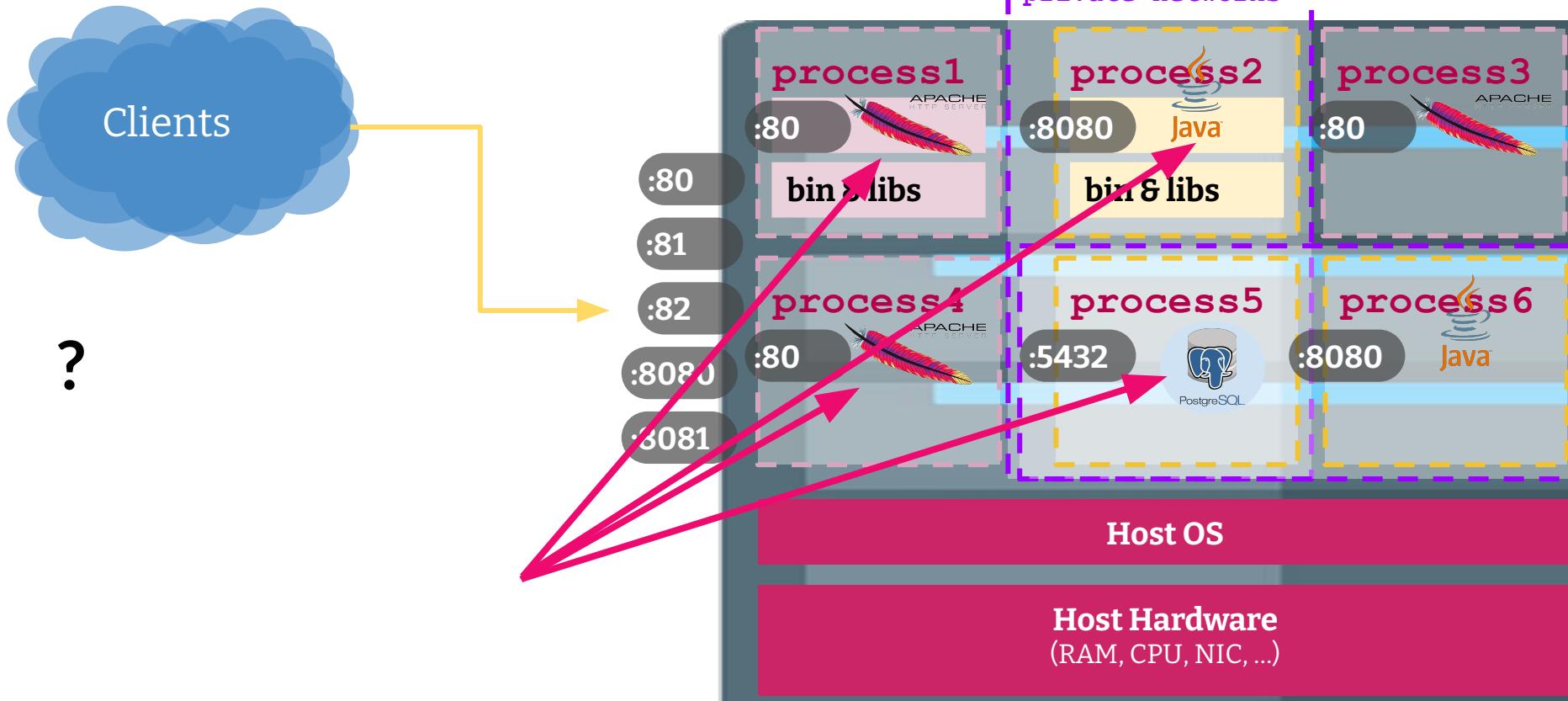
## Le “rêve”





Pourquoi diable les containers ?

## Le “rêve”



# Docker

containers, containers everywhere !



# La feuille de route

- Faciliter la déployabilité d'une application
  - Une image, le plus léger possible
  - Contenir tout ce qu'il faut pour exécuter l'application  $\Leftrightarrow$  le runtime
- Avoir un overhead le plus léger possible
  - LXC + cgroups  $\Rightarrow$  on isole un "process"
- Gérer efficacement la couche d'isolation réseau
  - IPtables
  - Firewalld
  - Interfaces virtuelles

# La définition

“Outil permettant de mettre en oeuvre des containers  
afin de faciliter le déploiement d’applications distribuées”

— Docker != VM  $\Leftrightarrow$  Docker  $\approx$  lightweight VM

- Contient tout ce qui est nécessaire pour lancer l’appli
- Processus isolé

— Build once, ship everywhere

- Une image docker pour tous les environnements

— Version Control

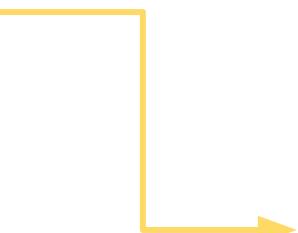
- Images versionnées

# Les piliers

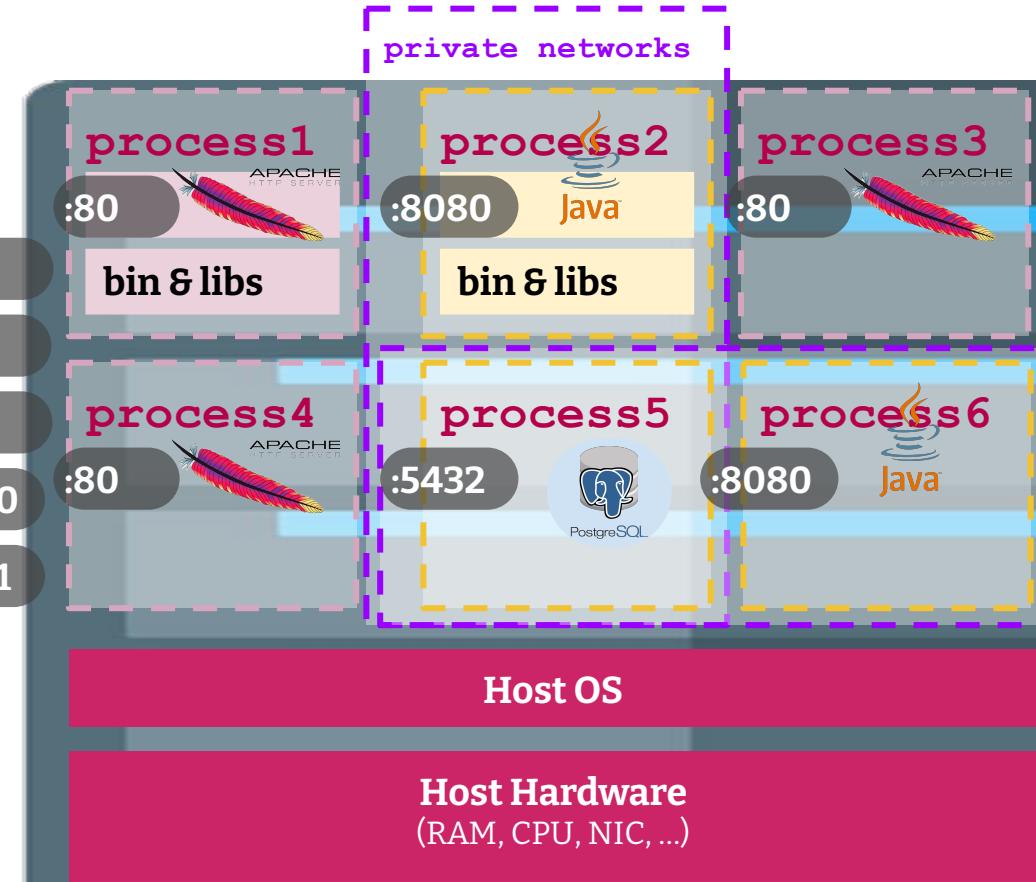
- Opensource
- 1 process = 1 container
  - on ne met pas “toute une appli” (DB + webapp + website + reverse-proxy) dans un container
- Build once, ship everywhere
  - Une image “environnement-agnostic” (recette, pré-prod, prod)
- Version Control
  - Images versionnées

Pourquoi diable les containers ?

## Le “rêve”



- ✓ utilise le hardware hôte (rapide)
- ✓ couche de virtualisation réduite
- ✓ pas d'overhead guest OS (léger)
- ✓ pas de guest OS à booter (rapide)
- ✓ images légères ⇒ easy to deploy

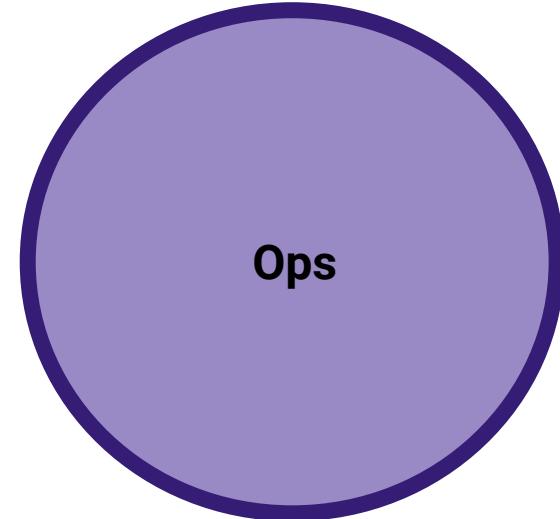
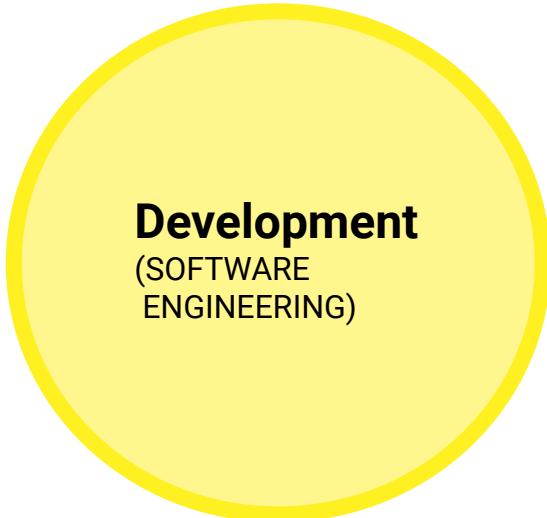


# À retenir

Docker plutôt que VMs

- Exécute les processus directement sur l'OS
  - très peu d'overhead
  - rapide à lancer
  - démarrer, arrêter et recréer des containers en quelques secondes
- Processus “isolés” ( ! sécurité)
- Images Docker
  - légères à décrire
  - faciles à déployer
- One “artifact”, deploy everywhere

# Dev vs DevOps



# Avant

Conversation 1:

D : “On a une nouvelle app à runner sur Java 8”

O : “Tout notre écosystème est sur Java 6. Sorry”

D : “Pourquoi tant de haine ?”

O : “Parce que c'est galère à installer de partout”

D : “Mais lisez la doc, y'a plein de nouveautés”

O : “Désolé, pas mon problème, je suis pas dev”

Conversation 2:

D : “On aimeraient ajouter cette lib en python 2”

O : “Euh, nous on est sur du python 3 de partout”

D : “Mais tu connais, y'a virtualenv qui peut...”

O : “Démerdez-vous, je casse pas mes serveurs pour une pauvre lib (soupir)”

## Conversation 1:

D : "Voici le nom de l'image à déployer"

O : "Tu as besoin de quoi en conf ? Quel port ?"

D : "Voici la conf. Ca n'expose qu'un port"

O : "Ok"

## Conversation 2:

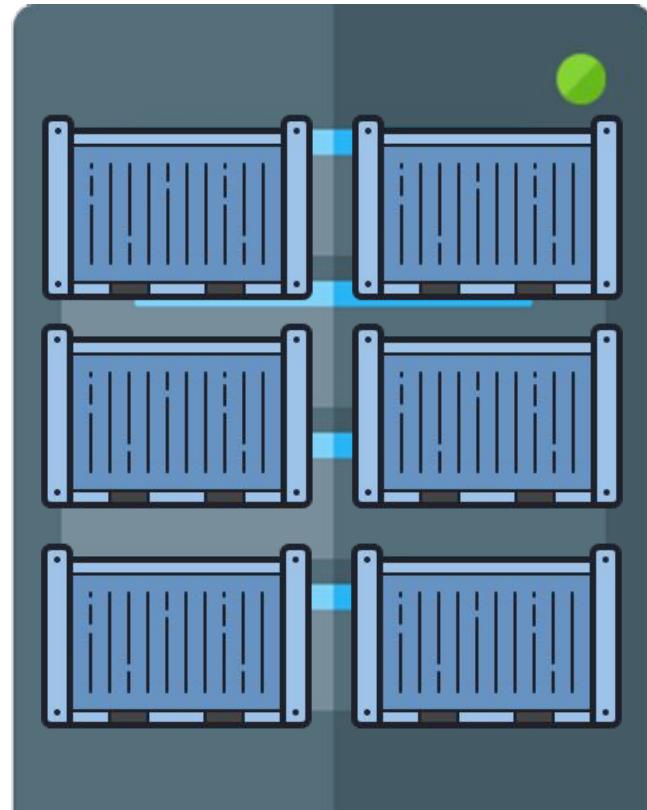
D : "Voici le nom de l'image à déployer"

O : "Tu as besoin de quoi en conf ? Quel port ?"

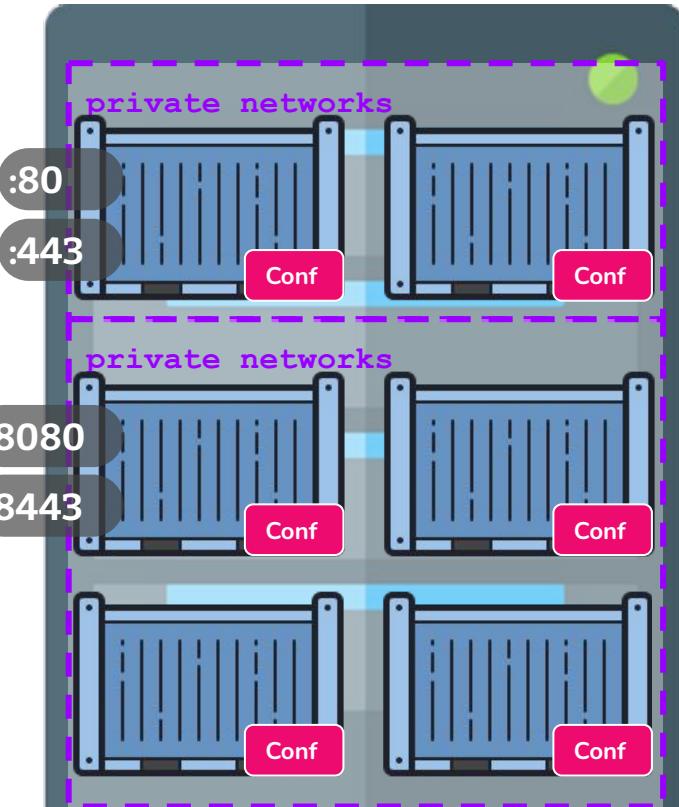
D : "Rien, juste besoin que ce soit exposé au container X"

O : "Ok"

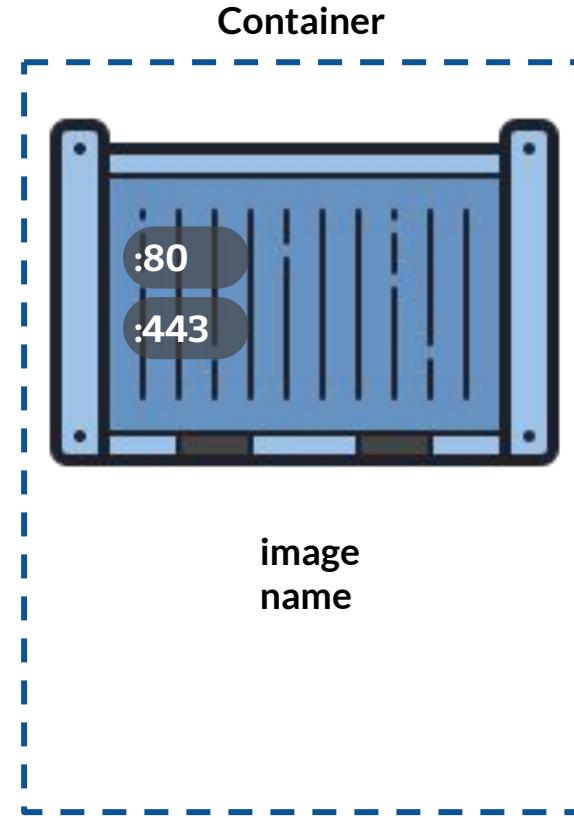
# Et donc...



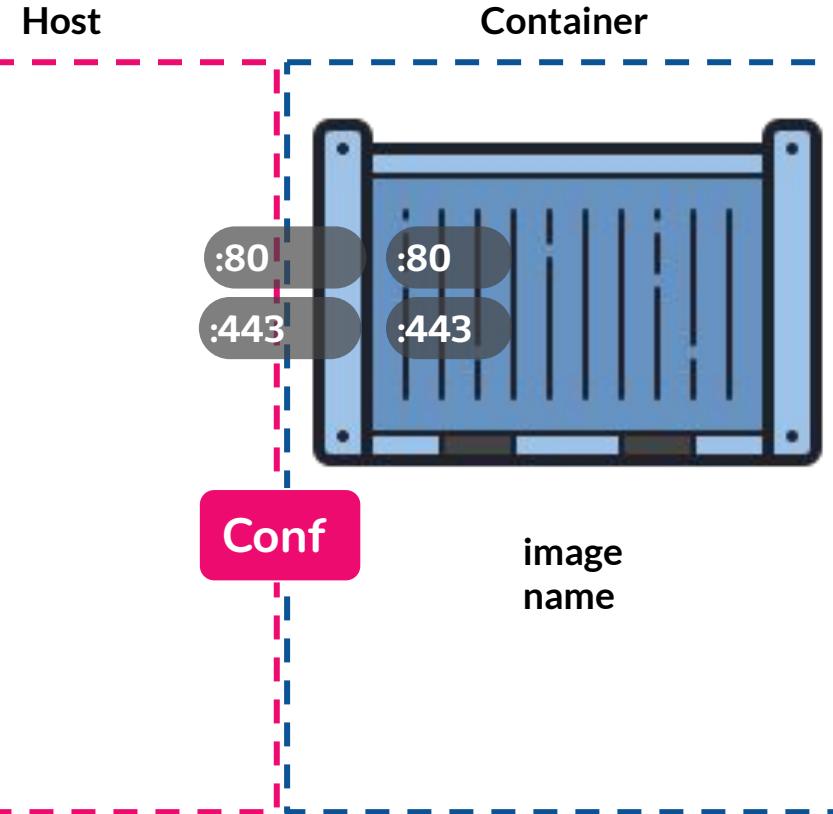
# Et donc...



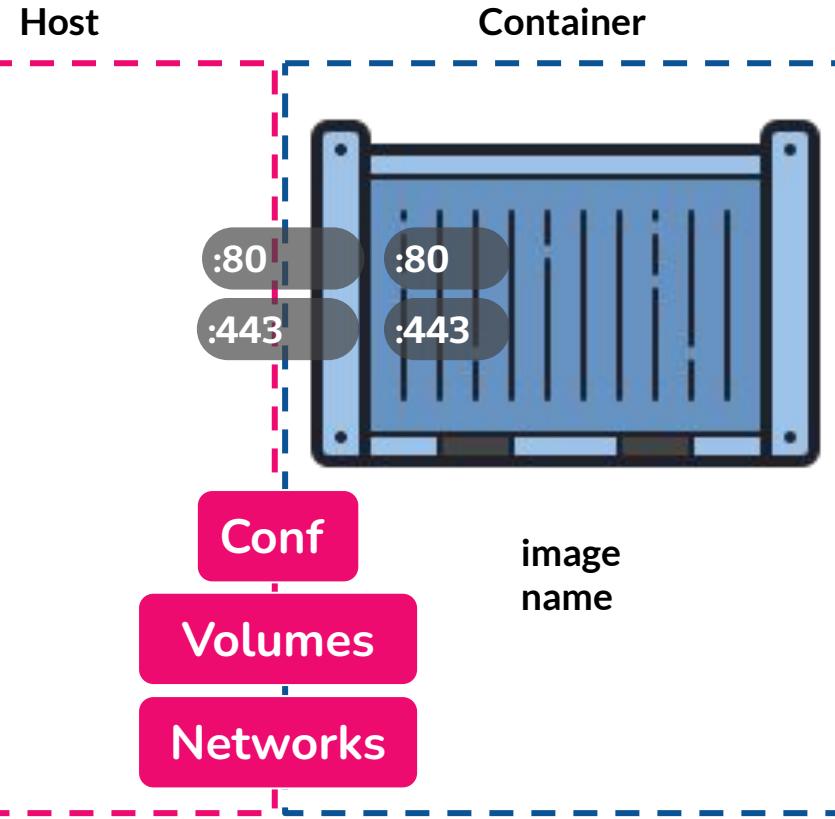
# Et donc...



# Et donc...



# Et donc...



# Dev & Ops en philosophie DevOps

## Dev

- Je gère maintenant mon runtime
- Je m'assure que tout est testé
- Je suis responsable que le code marche

## Ops

- Je fais en sorte que l'infra marche
- Je mets en place et maintient le socle qui permet de tout orchestrer



# Docker stuff

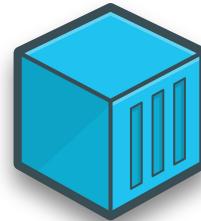
From images to running containers



# Image vs Container



**mariadb**

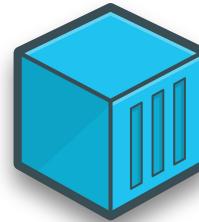


**myproject-db**

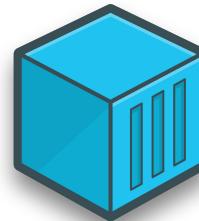
# Image vs Container



mariadb



another-project-db

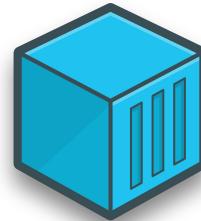


myproject-db

# Image vs Container



La classe (l'usine pour fabriquer des objets)



L'instance (un objet)

# Image vs Container

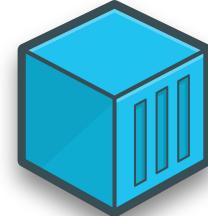


Le “snapshot versionné prêt à l'emploi”



L'application utilisée

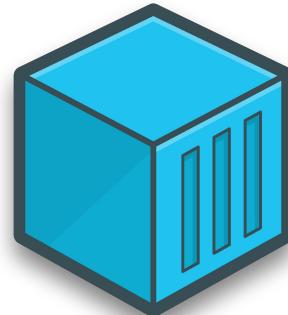
# Comment lancer un container ?



```
# run a container from the mysql image (latest version)
docker run -p 3306:3306 mysql
```

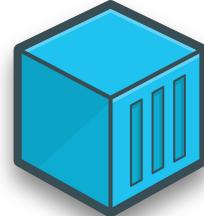


docker image



docker container

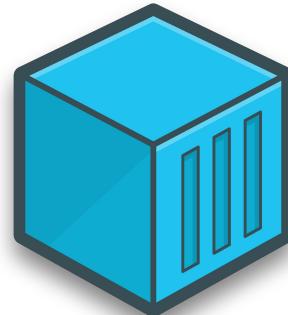
# Comment lancer un container ?



```
# run a container from the mysql image (latest version)
# map the container port 3306 to the host port 3306
docker run -p 3306:3306 mysql
```

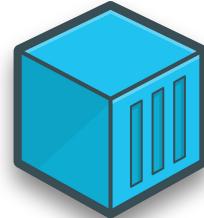


docker image



docker container

# Comment lancer un container ?



```
# run a container from the mysql image (latest version)
# map the container port 3306 to the host port 3306
docker run -p 3306:3306 mysql
```

```
# run a container from the postgresql image (latest version)
# map the container port 5432 to the host port 5432
docker run -p 5432:5432 postgresql
```

```
# run a container from the mariadb image (latest version)
# in a container named my-db
# map the container port 3306 to the host port 3306
# set the password to 'toor'
docker run --name my-db -p 3306:3306 -e MYSQL_ROOT_PASSWORD=toor mariadb
```

# Cheatsheet

```
# Run a nginx container with name my-nginx
docker container run nginx

# Run a nginx container of version 1.15.8 with name my-nginx
docker container run nginx:1.15.8 --name my-nginx -it

# Run a container in detached mode (not linked to a terminal)
docker container run nginx:1.15.8 --name my-nginx -it -d

# Stop a running container
docker container stop my-nginx

# Kill a running container (sends a SIGKILL)
docker container kill my-nginx

# Lists all running containers (-a to add those that died)
docker container ps
```

# Comment créer une image ?



## Écrire un Dockerfile :

- on prend une image déjà dispo
- on en crée un dérivé en effectuant des opérations spécifiques (provisioning)
- on décrit quelle commande (<=> process) va être lancée

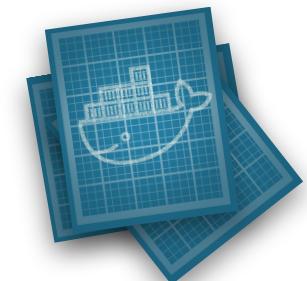
```
# Base image
FROM ubuntu:18.10

# Run a command in the container
# (eg: install a dependency)
RUN apt-get install sl

# Default program to run when
# container starts
ENTRYPOINT ["sl"]

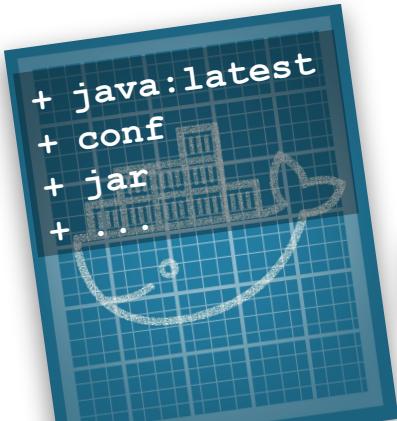
# Default command/args to pass to
# command above
CMD [".", "-Fe"]
```

# Comment créer une image ?

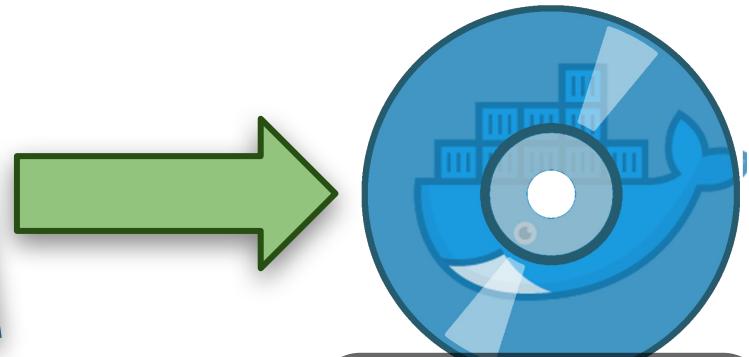


## Transformer le Dockerfile en image

```
# build docker image  
docker build . -t my-image
```



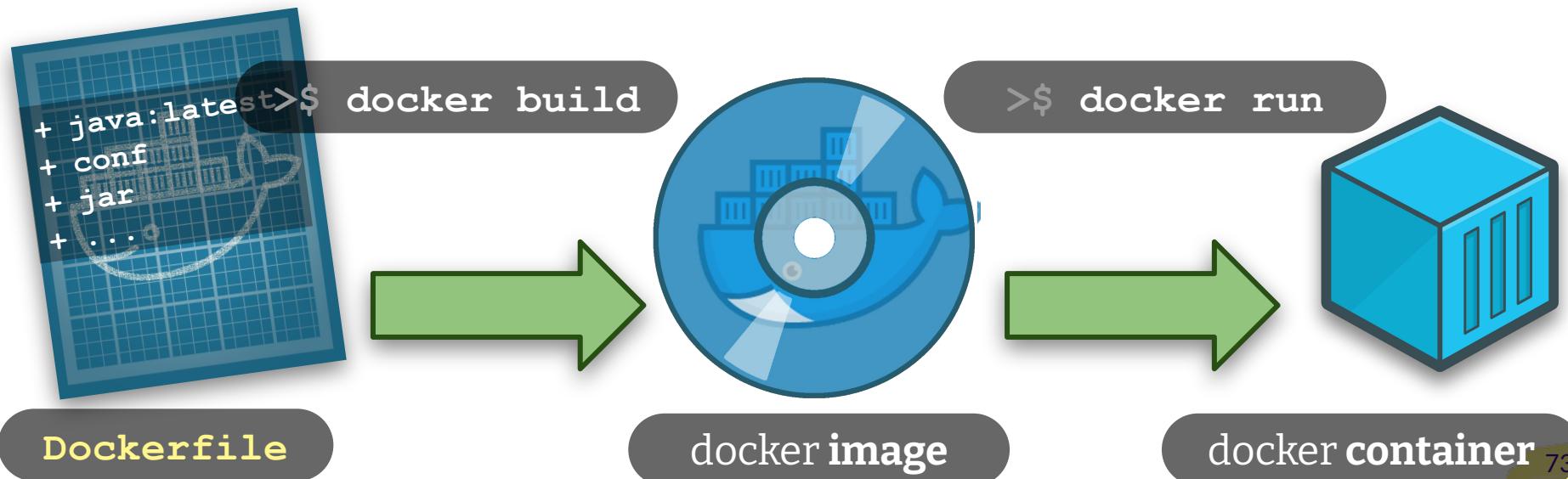
Dockerfile



docker image

# De l'image au container

En bref :



# Demo Livecoding

## appli LAMP avec docker

## À retenir

- Commencez avec une image de base **appropriée** (java, maven, nginx, ...)
- Utilisez des images officielles
- Visez la plus petite taille d'image possible
- Fixez les versions (évitez *latest*)

### Best practices :

<https://docs.docker.com/develop/dev-best-practices/#how-to-keep-your-images-small>

[https://docs.docker.com/engine/userguide/eng-image/dockerfile\\_best-practices](https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices)

# Docker universe

Docker-compose, Docker HUB & co



# Docker compose

- Décrire un **ensemble** de containers  
(sur le même host)
- Décrire leurs dépendances
- Décrire leur exposition
- Syntaxe YAML déclarative
  - Services (<=> containers)
  - Networks, Ports, Volumes, Variables, etc...

```
# docker-compose.yml
version: '3'
services:
  myapp-web:
    image: "httpd"
    ports:
    - "80:80"
  myapp-api:
    image: "java-jre"
  myapp-db:
    image: "postgres"
```

# Docker Hub

- Un repository d'**images** Docker  
↔ le Maven Central / le dépôt de paquets de Docker
- Héberge aussi vos propres images privées

Vous trouverez vos images “de base” dessus

[hub.docker.com](https://hub.docker.com)

# Évaluation

## 50% TP

Le TP se divise en 3 parties (plus partie bonus).

Le TP est à réaliser **individuellement ou à 2**.

Le rendu du TP sera votre repo Github et la documentation fournie: **README.md**.

Pour vous aider à identifier les points principaux se référer aux balises dans l'énoncé:

- ! Checkpoint ! Nous appeler pour valider l'étape
- ? Une information à documenter dans le README
- i Un point d'attention qui pourrait être utile

## 50% DS

Le DS est un examen écrit sur papier qui reprend des points vus en cours, en TD et en TP.

Le DS est composé : d'une partie QCM, de questions courtes et de questions un peu plus longues.

Enough speaking.  
Take me to the code !

<http://school.pages.takima.io/devops-resources/>