



► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O < + Ø ► O



1

There is a better way

# SKOOL

- Les bases de l'Ops J1



## ICE BREAKER

There is a better way



Prénom, tribu, mission



3 choses à propos de  
toi



Ce que tu attends de  
la formation



# AGENDA

- DevOps

- Bases « système »
  - Commandes de base
  - Hands On

- Environnements
- Machine virtuelle & Containers
- Cloud

- Outils clés en mission
  - SSH
  - HandsOn
  - Docker
  - HandsOn



# Présentation de la tribu Ops



## Pourquoi on se lève le matin ?

- Casser des murs
- Jouer aux docteurs
- Nous amuser avec des technos hyper cool
- Faire du travail de qualité



## Notre métier ?

- Transformer les organisations
- Évoluer vers le Cloud
- Automatiser
- Accompagner les deliveries Octo
- Conseil / Audit



## Où nous trouver ?

- Au cinquième côté Opera
- #Opstinés #OpsCure
- WTF with Devops / l'école de l'Ops



## Nos missions en ce moment ?

- InVivo
- Nexsis
- CNAM
- Total Digital 🚢
- DGAC
- Scaleway
- RelevanC
- ERPC
- PassCulture
- Randstad
- CAGIP



01

# DevOps



# QU'EST-CE QU'UN DEV ?





# QU'EST-CE QU'UN DEV ?



## VU DES DEV

C'est une personne qui :

- Fabrique du rêve
- Délivre rapidement un produit de qualité pour son client
- Incarne l'innovation

## VU DES OPS

C'est une personne qui :

- Fabrique des bugs pour la production
- Ne respecte pas les standards
- Ne documente pas ce qu'il fait
- Yolo est son motto



## QU'EST-CE QU'UN OPS ?



# QU'EST-CE QU'UN OPS ?



## VU DES OPS

C'est une personne qui :

- Sauve la prod tous les jours
- Est l'expert des infrastructures
- Est le roi du Datacenter
- Se réveille la nuit à cause du travail bâclé des Dev

## VU DES DEV

C'est une personne qui :

- Est psychorigide
- Réticente au changement
- Planquée
- Lente



# CONSTATS : DES DOULEURS RÉCURRENTES



## Douleur #1

Délai très important pour le provisionnement des environnements



## Douleur #2

Collaboration douloureuse entre les DEV et les OPS



## Douleur #3

Problème de qualité à l'origine d'anomalies en productions



## Douleur #4

Activité de déploiement chronophage

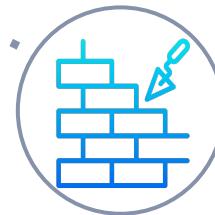


# DES POINTS DE VUE DIFFÉRENTS ...

There is a better way

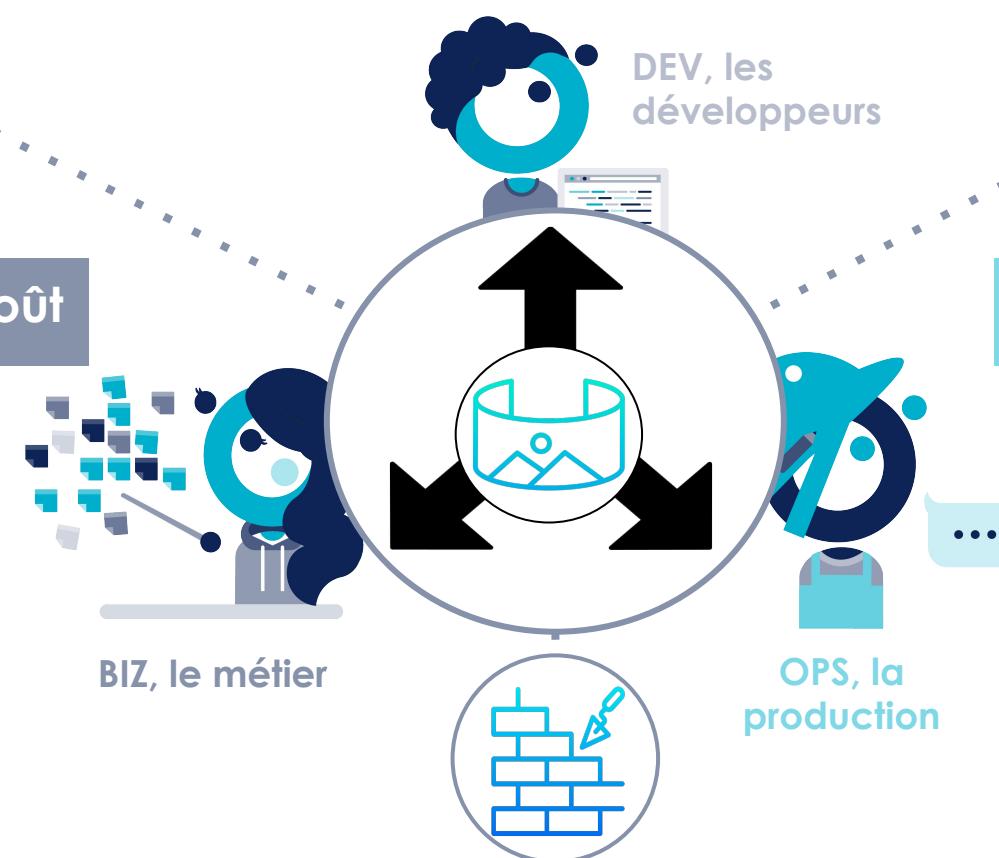
## Rapidité

- Livrer plus **rapidement** des **nouvelles fonctionnalités**
- Chercher à **innover**
- Culture du **changement**



## Rapidité Qualité Stabilité Coût

- Fournir le plus **rapidement** possible des services de **qualité** au client final
- Obtenir une **qualité** de service sans failles
- Contrôler les **budgets IT**

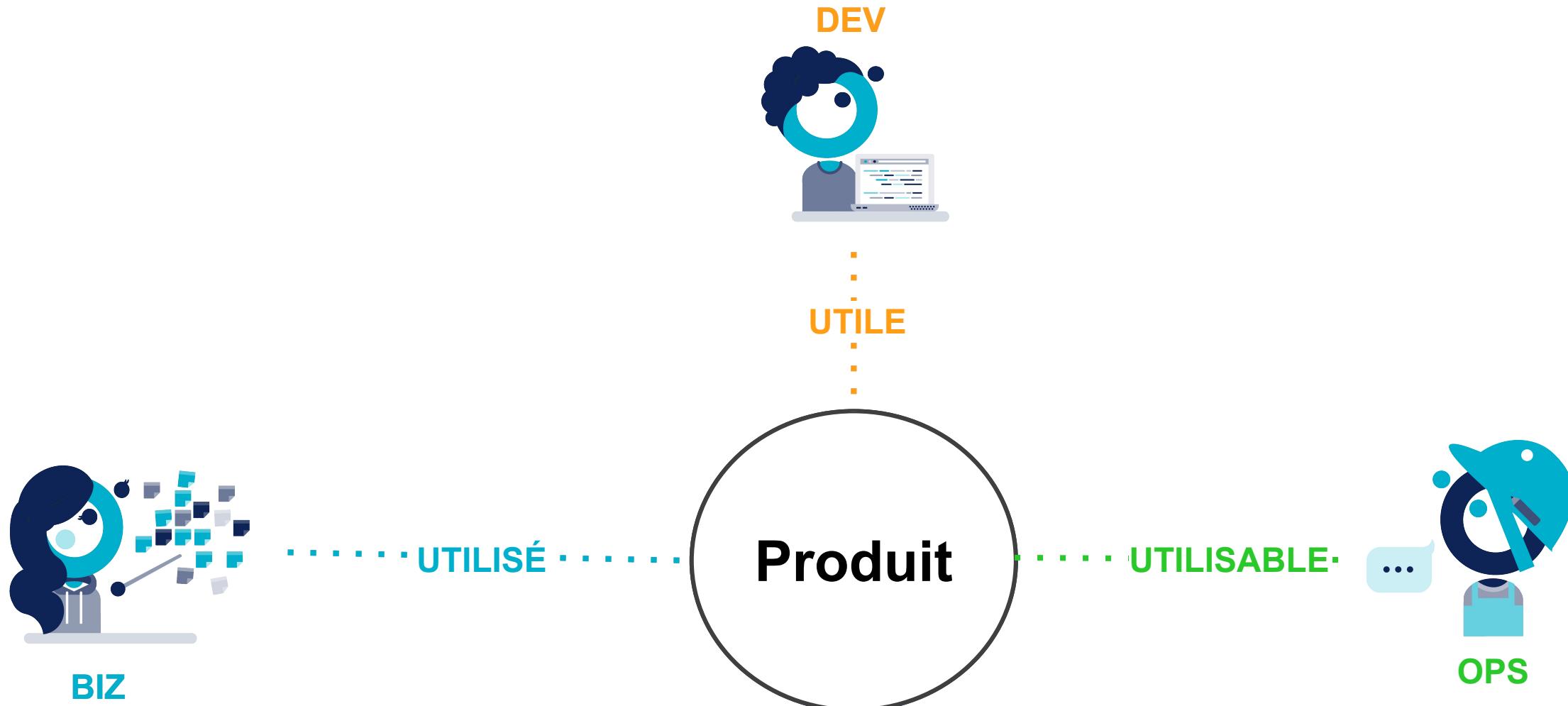


## Stabilité Qualité Coûts

- Garantir la **stabilité**
- Contrôler la **qualité** des changements
- Chercher à **rationaliser**
- Culture du **service** (SLA)

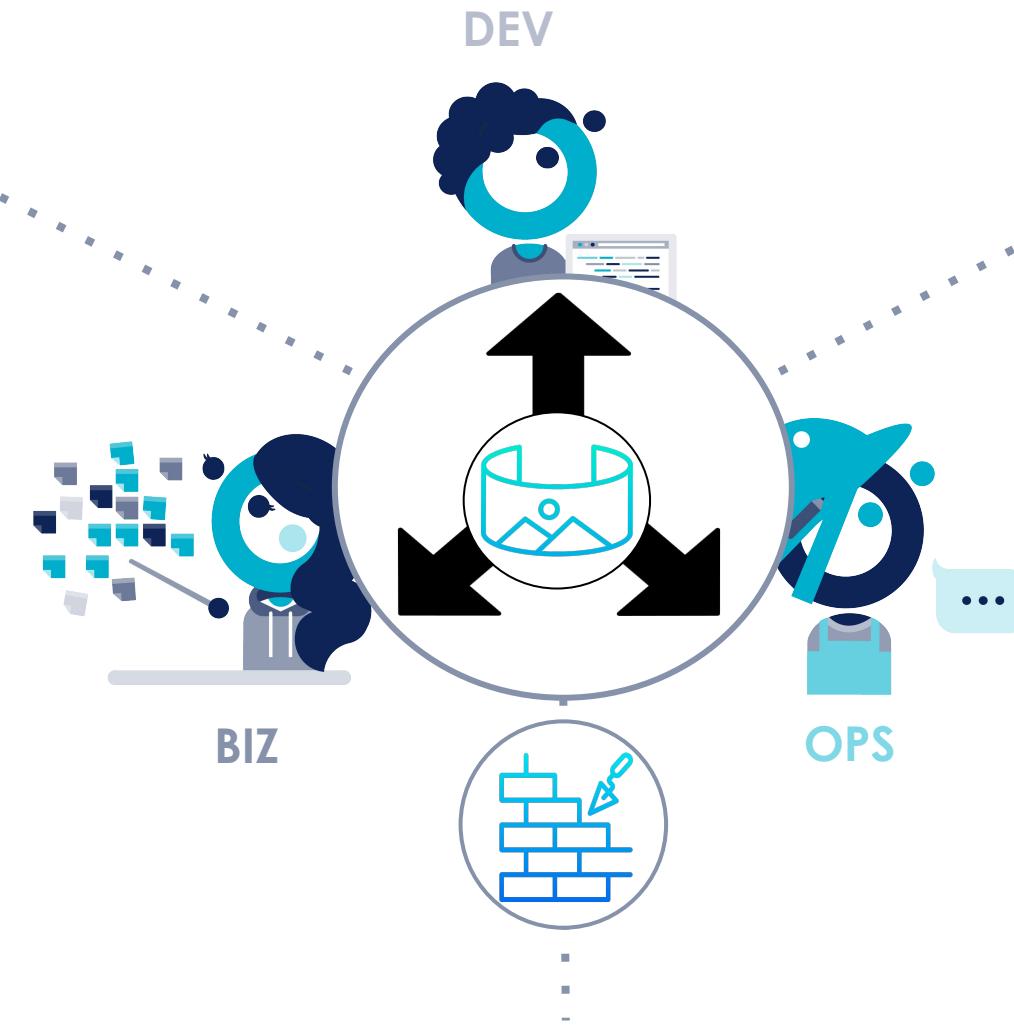


# UN ENJEU COMMUN



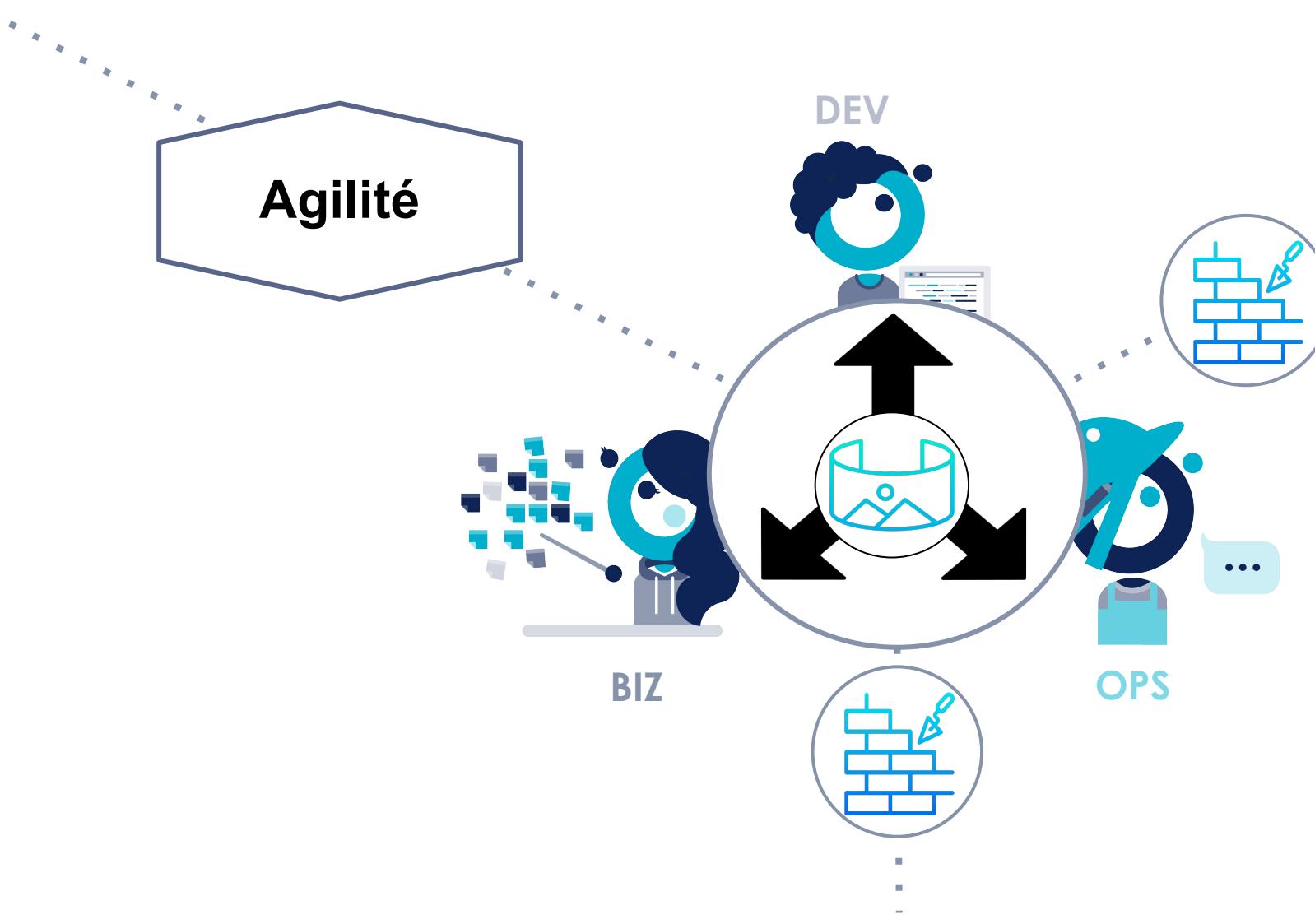
# LES PRATIQUES POUR ROMPRE LES MURS...

There is a better way



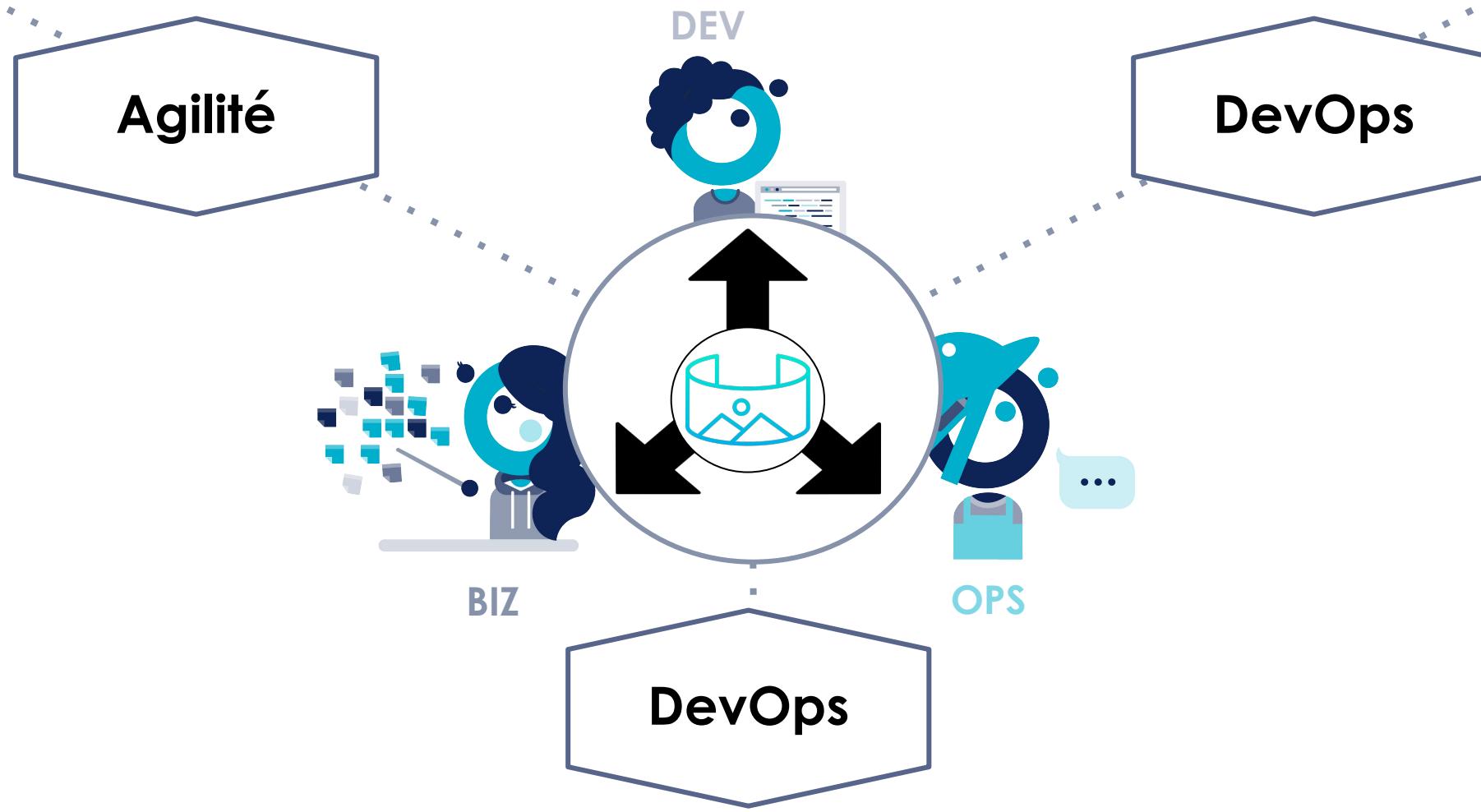
# LES PRATIQUES POUR ROMPRE LES MURS...

There is a better way



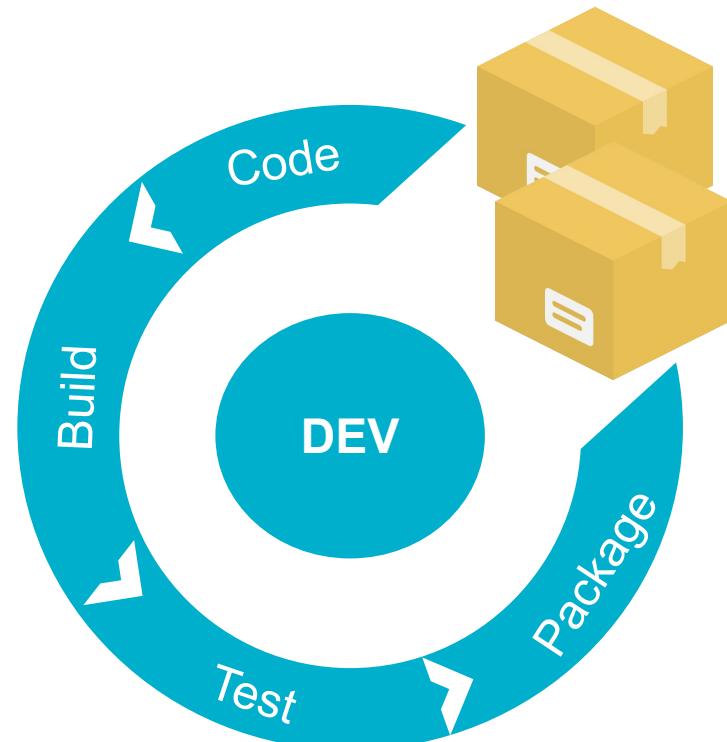
# LES PRATIQUES POUR ROMPRE LES MURS...

There is a better way



# DES ORGANISATIONS DIFFÉRENTES

Développement Agile, équipe autonome



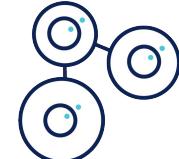
Durée : 2 - 3 semaines

Mode optimisation de flux

Écosystème en silo



Équipe réseau



Équipe système



Équipe base de donnée



...

Durée : 2 - 3 mois

Mode optimisation de l'occupation du temps

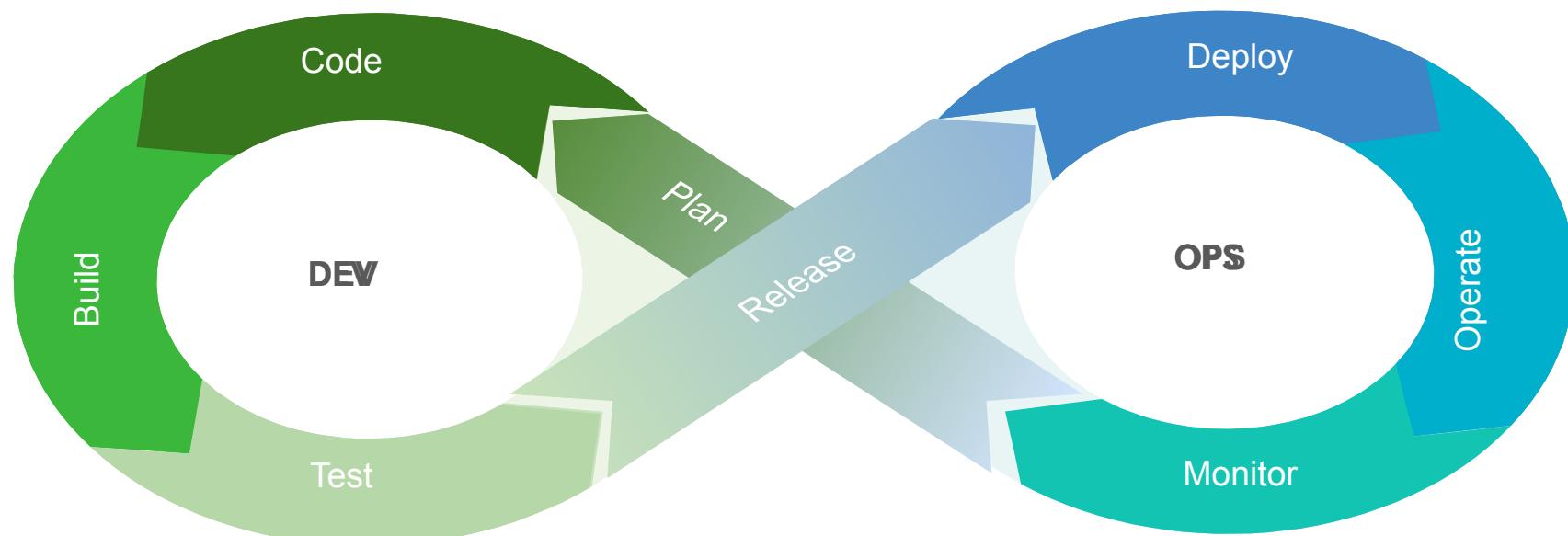


# ÉTENDRE L'AGILITÉ À LA PRODUCTION

Agilité des Développements

Agilité des Opérations

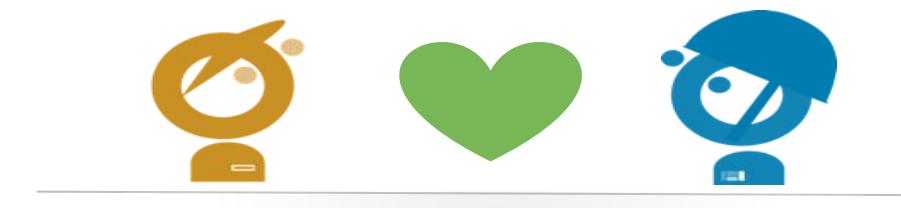
CONVERGENCE DES PROCESSUS : DEVOPS



Durée : 2 - 3 semaines  
Durée : 2 - 3 semaines

## LA PHILOSOPHIE DEVOPS

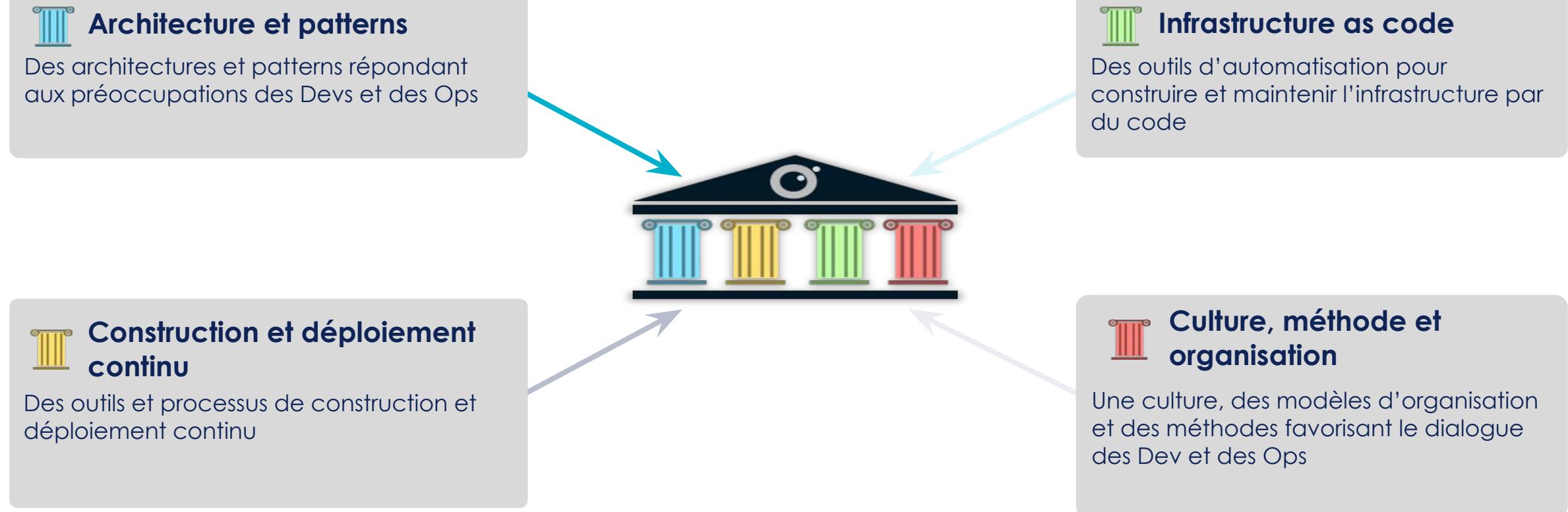
« **DevOps est un ensemble de pratiques qui visent à réduire le Time to Market et améliorer la Qualité en optimisant la coopération entre les Développeurs et la Production** »



→ **Un mouvement friand de technologies s'adressant à la fois aux dev et aux ops**

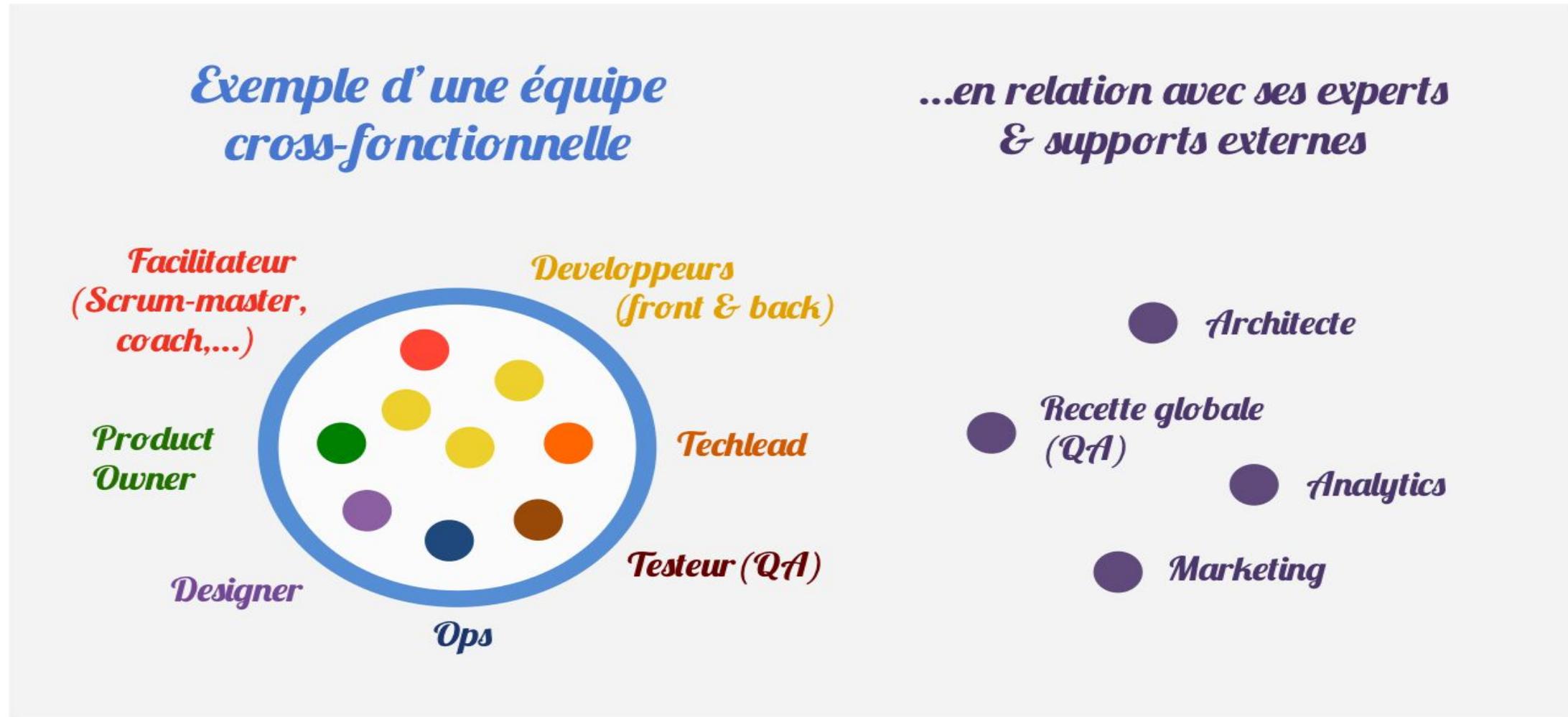
# LES 4 PILIERS DE DEVOPS

There is a better way



# FEATURE TEAM

- ▷ L'union des compétences pour atteindre l'objectif est un élément clef de DevOps



# CULTURE, MÉTHODE ET ORGANISATION

## Des modèles d'organisation favorisant l'autonomie et la responsabilisation

- Décloisonnement des organisations (destruction des silos techniques)
- « You build it, you run it »
- Autonomisation des équipes

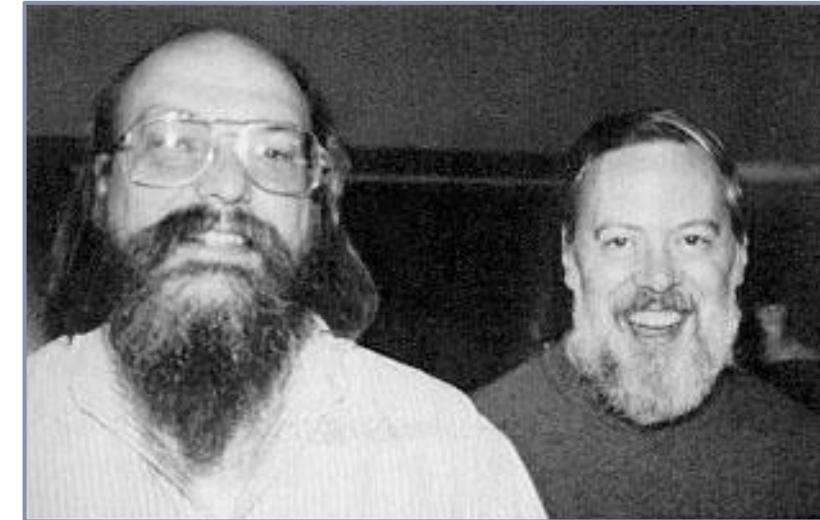


02

# Bases Linux

# WHAT IS LINUX?

- Historique :
  - 1969 : Unix écrit par Ken Thompson and Dennis Ritchie  
=> Non libre, appartient à AT&T Bell Labs
  - 1977 : BSD (Berkeley Software Distribution), Unix-like  
(1992 : AT&T porte plainte contre BSD pour utilisation de code propriétaire)
  - 1987 : MINIX, Unix-like à destination de l'éducation  
Open-source, mais à modification restreinte
  - 1992 : Linus Torvalds publie, à 21 ans, le code de linux



Hello everybody out there using minix -

*I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).*

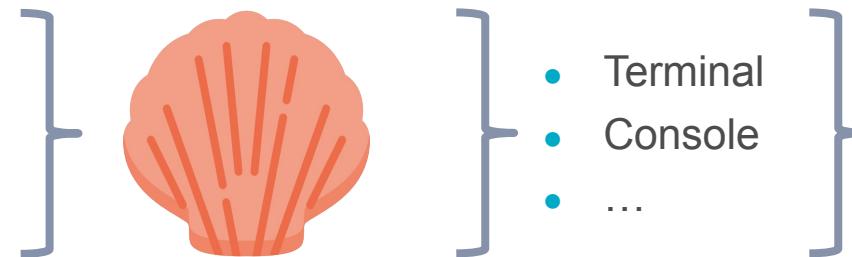


# WHY LINUX?

- Linux est **LE** système d'exploitation des serveurs d'application
  - Les chiffres différent, mais tout le monde s'accorde à dire qu'il est majoritaire
- Les applications tourneront dessus
- Vous devez donc...
  - L'utiliser
  - Et donc savoir l'utiliser
- Les avantages
  - Made by engineers for engineers
  - OpenSource : on sait ce que l'on exécute et possède une large communauté
  - Plus simple à automatiser
  - ...

# WHAT IS LINUX?

- Vocabulaire
  - Shell
    - Sh : Bourne Shell
    - Bash : Bourne Again Shell
    - Zsh : Z Shell
  - Kernel
    - Noyau, cœur du système autour duquel gravitent les programmes
    - Gère la communication avec le matériel
  - FileSystem
    - Système de fichiers
    - Définit l'organisation des données sur un support de stockage, comment sont gérés et organisés les fichiers par l'OS
  - Démons ou services : process qui tournent en tâche de fond



Interface qui vous permet d'exécuter des programmes avec des commandes



# UTILISER LA LIGNE DE COMMANDE

## 1. Explorer le système de fichiers

Où suis-je ? -----

```
$ pwd
```

Qu'y a-t-il dans ce dossier ? -----

```
$ ls
```

Changer de dossier -----

```
$ cd <dossier>
```

Que contient ce fichier ? -----

```
$ cat <fichier>
```



À tout moment, notre shell  
“se trouve” dans un dossier

## 2. Modifier le système de fichiers

Créer un dossier -----

```
$ mkdir <dossier>
```

Supprimer un dossier vide -----

```
$ rmdir <dossier>
```

Créer un fichier vide -----

```
$ touch <fichier>
```

Supprimer un fichier -----

```
$ rm <fichier>
```

Copier un fichier -----

```
$ cp <fichier> <dest>
```

Copier un dossier -----

```
$ cp -r <dossier> <dest>
```

Renommer/déplacer -----

```
$ mv <source> <dest>
```



# UTILISER LA LIGNE DE COMMANDE

## 3. Filtrer les données

Afficher le début d'un fichier ----- `$ head -n <nb de lignes> <fichier>`

Afficher la fin d'un fichier ----- `$ tail -n <nb de lignes> <fichier>`

Chercher dans un fichier ----- `$ grep <string à chercher> <fichier>`

Lire un fichier progressivement ----- `$ less <fichier>`

## 4. RTFM (Read The Fantastic Manual)

Aide succincte ----- `$ <commande> -h/--help`

Aide complète (Manuel) ----- `$ man <commande>`

Aide succincte avec exemple ----- `$ tldr <commande>`

(non natif)



# LE MAN, ÉTUDE DE CAS

```
$ grep --help
Usage: grep [OPTION]... PATTERN [FILE]...
Search for PATTERN in each FILE or standard input.
PATTERN is, by default, a basic regular expression (BRE).
Example: grep -i 'hello world' menu.h main.c

Regexp selection and interpretation:
-E, --extended-regexp      PATTERN is an extended regular expression (ERE)
-F, --fixed-strings         PATTERN is a set of newline-separated strings
-G, --basic-regexp          PATTERN is a basic regular expression (BRE)
-P, --perl-regexp           PATTERN is a Perl regular expression
-e, --regexp=PATTERN        use PATTERN for matching
-f, --file=FILE              obtain PATTERN from FILE
-i, --ignore-case            ignore case distinctions
-w, --word-regexp            force PATTERN to match only whole words
-x, --line-regexp             force PATTERN to match only whole lines
-z, --null-data               a data line ends in 0 byte, not newline

Miscellaneous:
-s, --no-messages            suppress error messages
-v, --invert-match           select non-matching lines
-V, --version                 display version information and exit
--help                       display this help text and exit

Output control:
-m, --max-count=NUM          stop after NUM matches
-b, --byte-offset              print the byte offset with output lines
-n, --line-number              print line number with output lines
--line-buffered                flush output on every line
-H, --with-filename            print the file name for each match
-h, --no-filename              suppress the file name prefix on output
.
.
```

\$ man grep  
GREP(1) General Commands Manual GREP(1)

**NAME**  
grep, egrep, fgrep, rgrep - print lines matching a pattern

**SYNOPSIS**  
grep [OPTIONS] PATTERN [FILE...]  
grep [OPTIONS] [-e PATTERN]... [-f FILE]... [FILE...]

**DESCRIPTION**  
grep searches the named input FILES for lines containing a match to the given PATTERN. If no files are specified, or if the file “-” is given, grep searches standard input. By default, grep prints the matching lines.  
In addition, the variant programs egrep, fgrep and rgrep are the same as grep -E, grep -F, and grep -r, respectively. These variants are deprecated, but are provided for backward compatibility.

**OPTIONS**  
Generic Program Information  
--help Output a usage message and exit.  
-V, --version Output the version number of grep and exit.

Matcher Selection  
-E, --extended-regexp Interpret PATTERN as an extended regular expression (ERE, see below).  
-F, --fixed-strings Interpret PATTERN as a list of fixed strings (instead of regular expressions), separated by newlines, any of which is to be matched.  
. . .

# QUELQUES DE NOTIONS DE SYSTÈME, LES PROCESSUS

## 1. Les processus

Lister les processus -----

```
$ ps
```

- ax pour tous les afficher
- u pour avoir leurs propriétaires
- f pour afficher l'arborescence (pas sur Mac)

Pour quelque chose de plus interactif -----

```
$ top
```

(appuyer sur Q pour quitter)

Envoyer des signaux à un process -----

```
$ kill
```

- -15, demande l'arrêt du process (poliment)
- -9, arrêt sans délai du process (violent)
- Et bien d'autres...

Dans un shell :

- CTRL+C arrête le processus en cours (eq. kill -2)
- CTRL+Z met le processus en cours en pause et rend la main au shell
- la commande **bg** relance un processus en pause à l'arrière-plan
- la commande **fg** relance un processus en pause ou re-attache un processus présent en arrière plan

Pour lancer une commande directement à l'arrière plan, utiliser "&", ex :

```
$ sleep 10 &
```

# QUELQUES NOTIONS DE SYSTÈME, LES PERMISSIONS

## 2. Les permissions et les fichiers cachés

Certaines options du programme **ls** donne plus d'informations :

- **-a** permet d'afficher les fichiers et dossiers cachés
- **-l** permet d'afficher les métadonnées

On voit notamment apparaître :

- Des fichiers et dossiers dont le nom commence par “.”

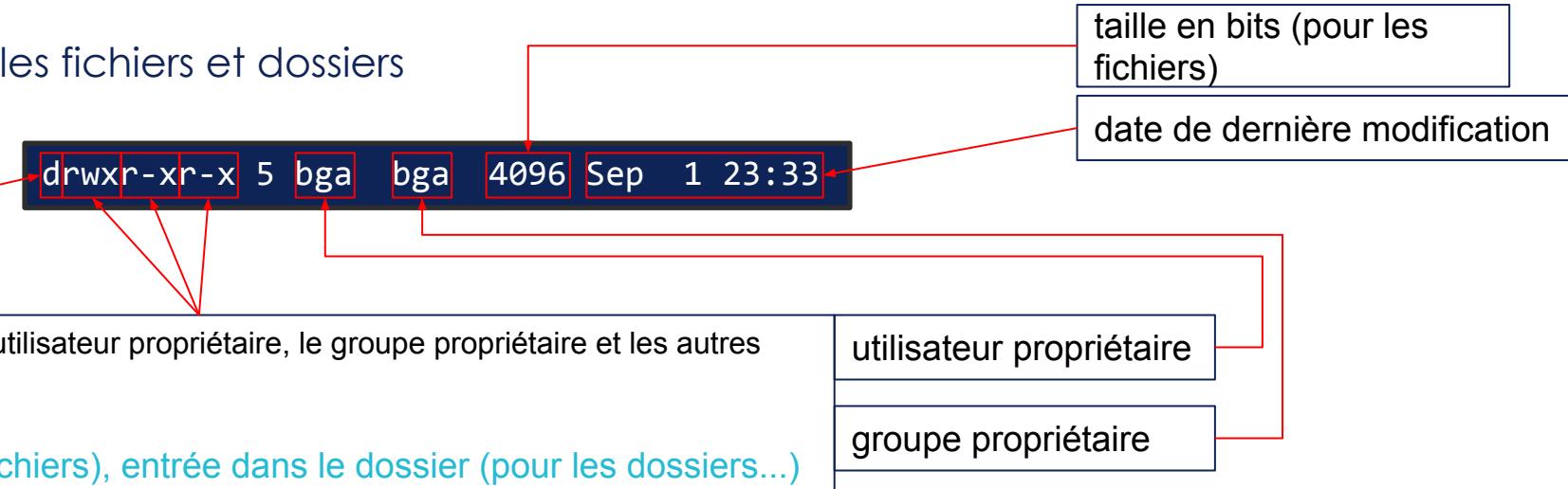
Ce sont les fichiers et dossiers cachés.

Deux dossiers cachés sont toujours présents :

- > “.” le dossier qu'on est en train de lister
- > “..” le dossier parent de ce dossier

- plein d'informations sur les fichiers et dossiers

```
$ ls -la
total 36
drwxr-xr-x 5 bga bga 4096 Sep  1 23:33 .
drwxr-xr-x 3 root root 4096 Aug 30 01:13 ..
drwx----- 3 bga bga 4096 Sep  1 23:33 .ansible
-rw----- 1 bga bga 417 Sep 23 00:46
.bash_history
-rw-r--r-- 1 bga bga 220 Aug 31 2015
.bash_logout
-rw-r--r-- 1 bga bga 3771 Aug 31 2015 .bashrc
drwx----- 2 bga bga 4096 Aug 30 01:15 .cache
```





# QUELQUES NOTIONS DE SYSTÈME, LES PERMISSIONS

## 3. Modifier les permissions

Modifier les propriétaires d'un fichier -----

```
$ chown <user>:<group> <cible>
```

Modifier les droits sur un fichier -----

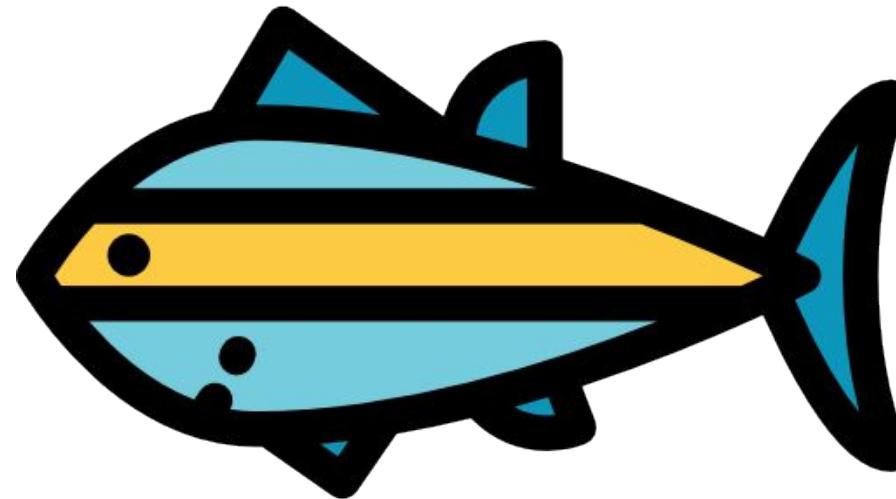
```
$ chmod <permissions> <cible>
```

Les permissions à donner peuvent s'écrire de deux façons :

- notation absolue : 3 chiffres entre 0 et 7 (octal: read = 4, write = 2, execute = 1)  
1er chiffre : droits de l'utilisateur propriétaire  
2ème chiffre : droits du group propriétaire  
3ème chiffre : droits pour les autres  
ex : 644 => rw-r--r--
- notation relative, une liste de modifications séparées par les virgules, chacune de la forme :  
QUI (u = utilisateur, g = groupe, o = autres)  
OP, opération à effectuer ("+" = ajouter, "-" = retirer, "=" = mettre comme la valeur qui suit (u,g), si rien pas de droits)  
PERM, permissions (r, w, x)  
ex : ug+x,o= => ajoute les droits d'exécution à l'utilisateur et au groupe et retire toute permission aux autres

## COMMANDES DE BASE : RÉSEAU

- wget google.com
- curl
  - > curl octo.com
  - > curl -I octo.com
- ping
- telnet octo.com <port>
- netstat -tunapl
  - > « tuna please »
- ip a





# VARIABLES D'ENVIRONNEMENT ET INTERPOLATION

Le Shell fournit également un système de variable d'environnements. Certaines sont définies par défaut et sont utilisées en interne par celui-ci. En Shell, toutes les variables sont des chaînes de caractères.

Pour les afficher --- `$ env`

Pour les définir ---- `$ export <variable>=<valeur>`

Ces variables peuvent être interpolées (avec « \$ ») n'importe où dans une commande shell, par exemple :

Pour afficher le contenu du fichier dont le chemin est dans la variable NOM\_FICHIER --- `$ cat $NOM_FICHIER`

L'interpolation fonctionne également pour le résultat d'une commande (son stdout) avec

l'opérateur « \$() » : --- `$ CONTENU_TOTO=$(cat fichier_avec_du_contenu.txt)`



# COMMANDES DE BASE : SERVICES

Sur les systèmes récents

- systemctl list-units
  - > Liste les services disponibles
- Services :
  - > systemctl <ACTION> <NAME>
- systemctl status mysql
- systemctl stop mysql
  - > ne marche pas
- sudo systemctl stop mysql
- journalctl -xeu <NAME>



# QUELQUES DE NOTIONS DE SYSTÈME, LE FHS

There is a better way

- File Hierarchy Standard

- > ls -l /

- > Configuration dans /etc

- > Utilisateurs dans /home

- > Binaires dans /bin, /usr/bin, /usr/sbin, ...

- > Logs dans /var/log

Fichiers « qui varient avec le temps, qui ont diverses utilisés » : /var

-> Logs, db, mail...

- Le PATH

- > Une commande, c'est un programme

- > which ls

- > /usr/bin/ls /bin

- > echo \$PATH

```
[hello@nodejs /]$ ll
total 62
lrwxrwxrwx.  1 root    root      7 Jul 29 2016 bin -> usr/bin
dr-xr-xr-x.  5 root    root    1024 Mar 27 14:38 boot
drwxr-xr-x. 18 root    root   2920 Mar 27 14:38 dev
drwxr-xr-x. 79 root    root  4096 Mar 27 16:24 etc
drwxr-xr-x.  5 root    root  4096 Mar 27 14:44 home
lrwxrwxrwx.  1 root    root      7 Jul 29 2016 lib -> usr/lib
lrwxrwxrwx.  1 root    root     9 Jul 29 2016 lib64 -> usr/lib64
drwx-----  2 root    root 16384 Jul 29 2016 lost+found
drwxr-xr-x.  2 root    root  4096 Aug 12 2015 media
drwxr-xr-x.  2 root    root  4096 Aug 12 2015 mnt
drwxr-xr-x.  3 root    root  4096 Mar 27 14:39 opt
dr-xr-xr-x. 94 root    root     0 Mar 27 14:38 proc
dr-xr-x---.  4 root    root  4096 Mar 27 15:34 root
drwxr-xr-x. 25 root    root    820 Mar 27 16:24 run
lrwxrwxrwx.  1 root    root      8 Jul 29 2016 sbin -> usr/sbin
drwxr-xr-x.  2 root    root  4096 Aug 12 2015 srv
dr-xr-xr-x. 13 root    root     0 Mar 27 14:38 sys
drwxrwxrwt. 16 root    root  4096 Mar 27 17:52 tmp
drwxr-xr-x. 13 root    root  4096 Jul 29 2016 usr
drwxr-xr-x.  4 vagrant vagrant 4096 Mar 27 14:36 vagrant
drwxr-xr-x. 19 root    root  4096 Mar 27 14:38 var
```

A close-up photograph of a wet otter. The otter is positioned diagonally, facing towards the bottom right. Its dark brown, wet fur is visible, along with its long, bushy tail. It is holding a small fish in its front paws and appears to be eating it. The background consists of dark, rippling water. In the top right corner, the words "Hands-On" are written in a large, white, sans-serif font.

Hands-On



## HANDS ON



<https://gitlab.com/octo-technology/octo-ops/les-bases-ops-formation>

**01 Hands On Linux.md  
dans le dossier :  
HandsOn\_01**



**Hands On!**

03

# Environnements

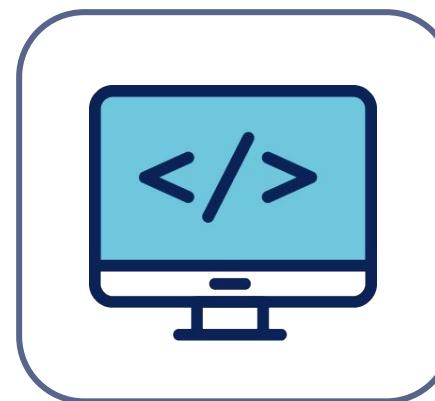


# QU'EST-CE QU'UN ENVIRONNEMENT ?

# QU'EST-CE QU'UN ENVIRONNEMENT ?



**Environnement** : L'ensemble des *matériels* et des *logiciels système*, dont le *système d'exploitation*, sur lesquels sont exécutés les *programmes* de l'application.



Local

Un environnement



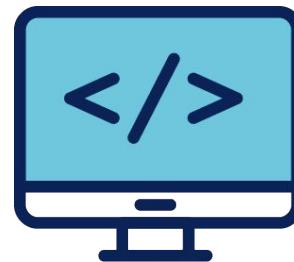
Production

Un autre environnement

# QU'EST-CE QU'UN ENVIRONNEMENT ?

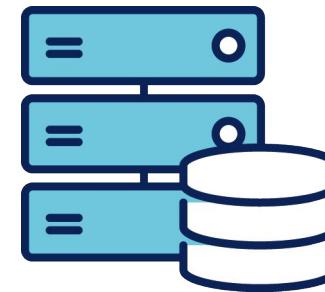


**Environnement** : L'ensemble des *matériels* et des *logiciels système*, dont le *système d'exploitation*, sur lesquels sont exécutés les *programmes* de l'application.



Local

```
$ php --version  
PHP 5.6.30 (cli) (built:  
Mar 11 2017 09:56:27)
```

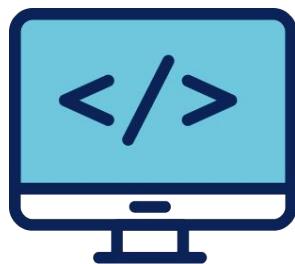


Production

```
$ php --version  
PHP 5.4.16 (cli) (built:  
Nov 6 2016 00:29:02)
```

# QU'EST-CE QU'UN ENVIRONNEMENT ?

On a besoin de plusieurs environnement, avec des responsabilités différentes.



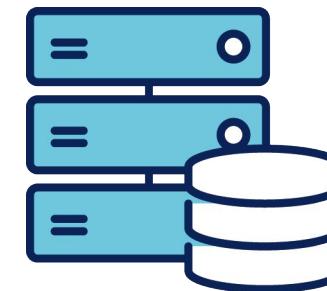
Local

MacOS X  
PHP 5.6.30



Test

CentOS  
PHP 5.4.16



Pré-production

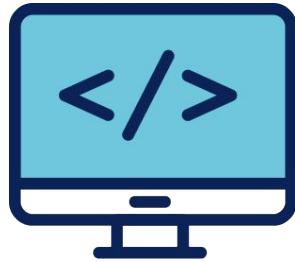
RHEL  
PHP 5.4.16



Production

RHEL  
PHP 5.4.16

# QU'EST-CE QU'UN ENVIRONNEMENT ?



Local



Test



Pré-production



Production



Mais dans la réalité, les environnements diffèrent souvent, car :

- « ils vivent »
- problématique de coûts
- etc...

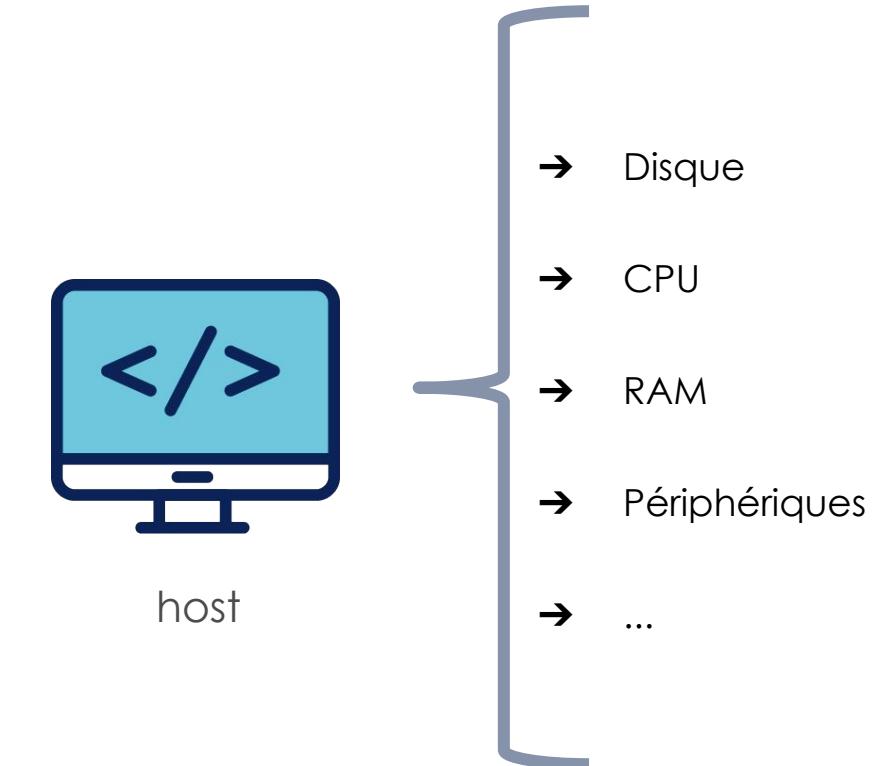
04

# Machines virtuelles & Conteneurs



# QU'EST-CE QU'UNE MACHINE VIRTUELLE ?

# QU'EST-CE QU'UNE MACHINE VIRTUELLE ?

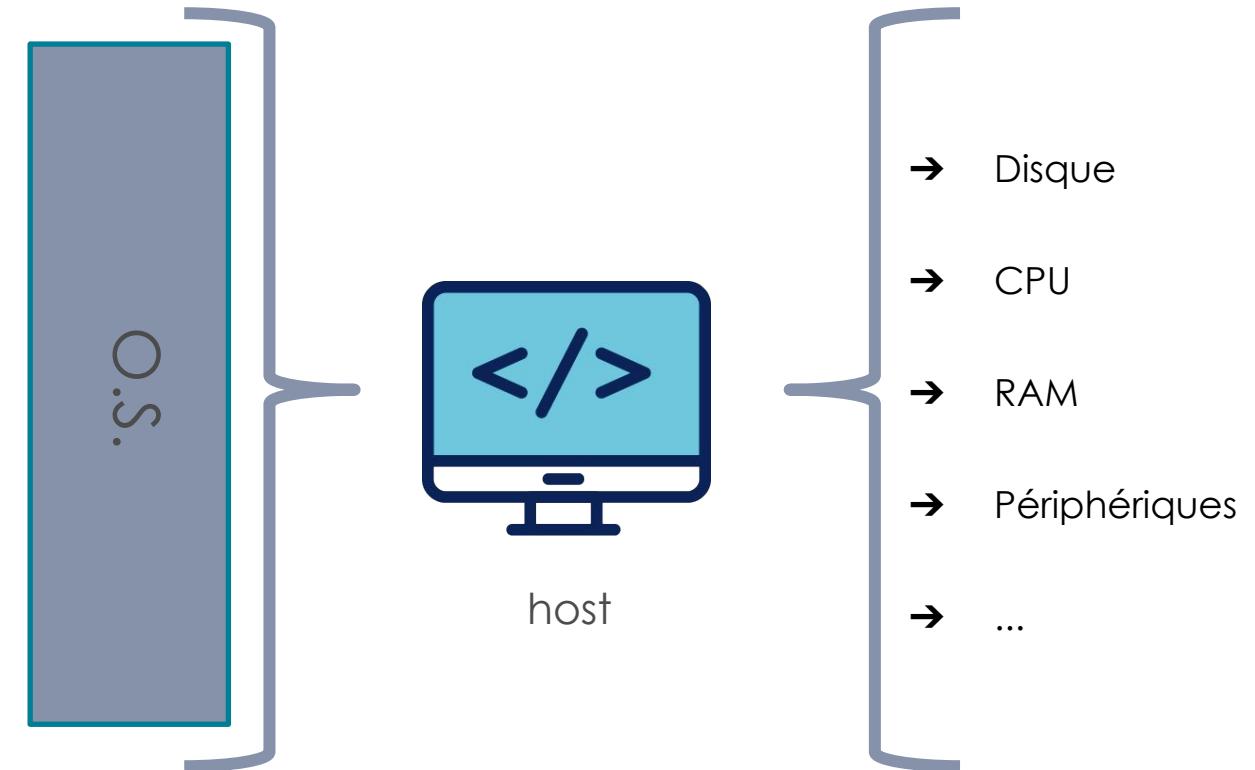


# QU'EST-CE QU'UNE MACHINE VIRTUELLE ?

There is a better way



L'OS (*Operating System*, pour « Système d'Exploitation », est la couche d'abstraction logiciel des ressources de la machine (Disque, CPU, ...)

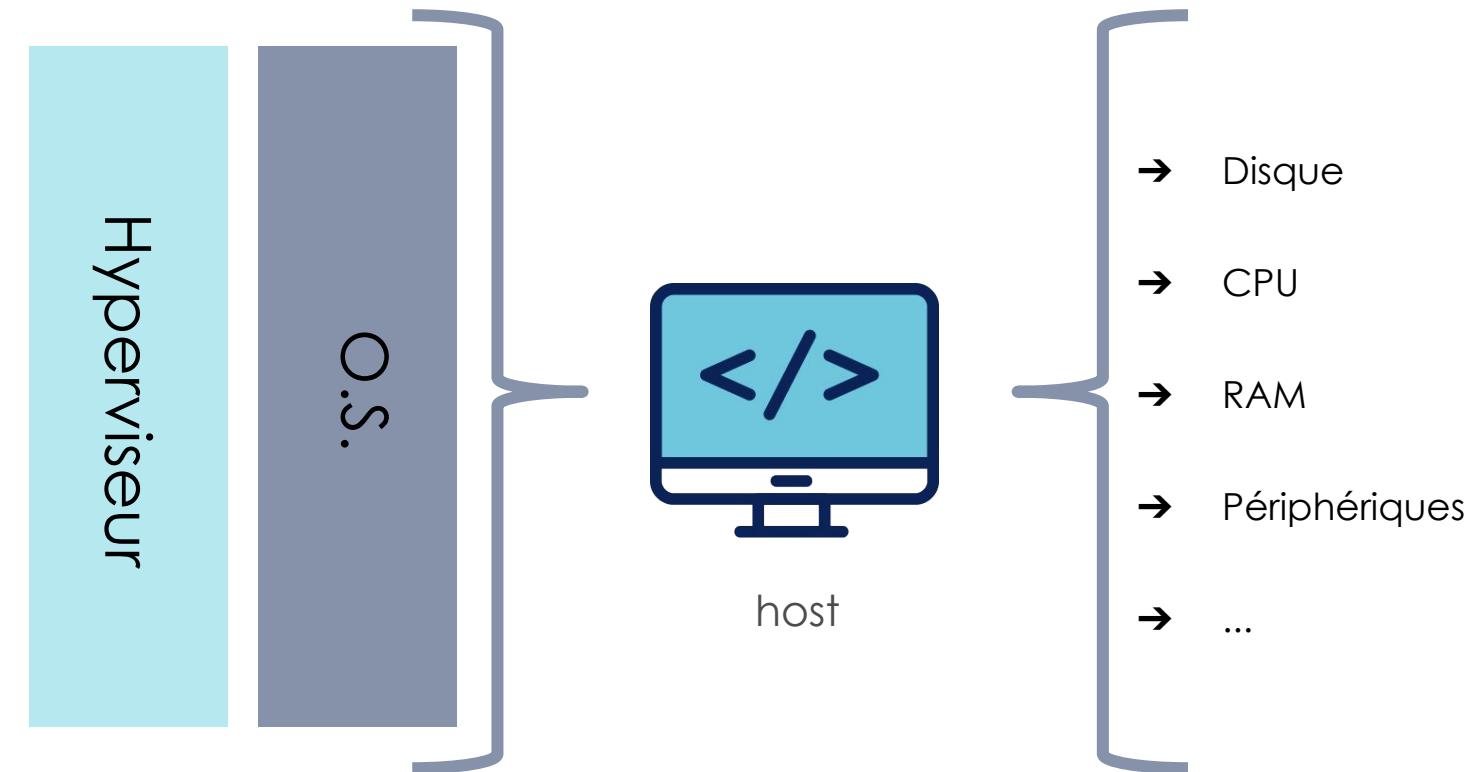


# QU'EST-CE QU'UNE MACHINE VIRTUELLE ?

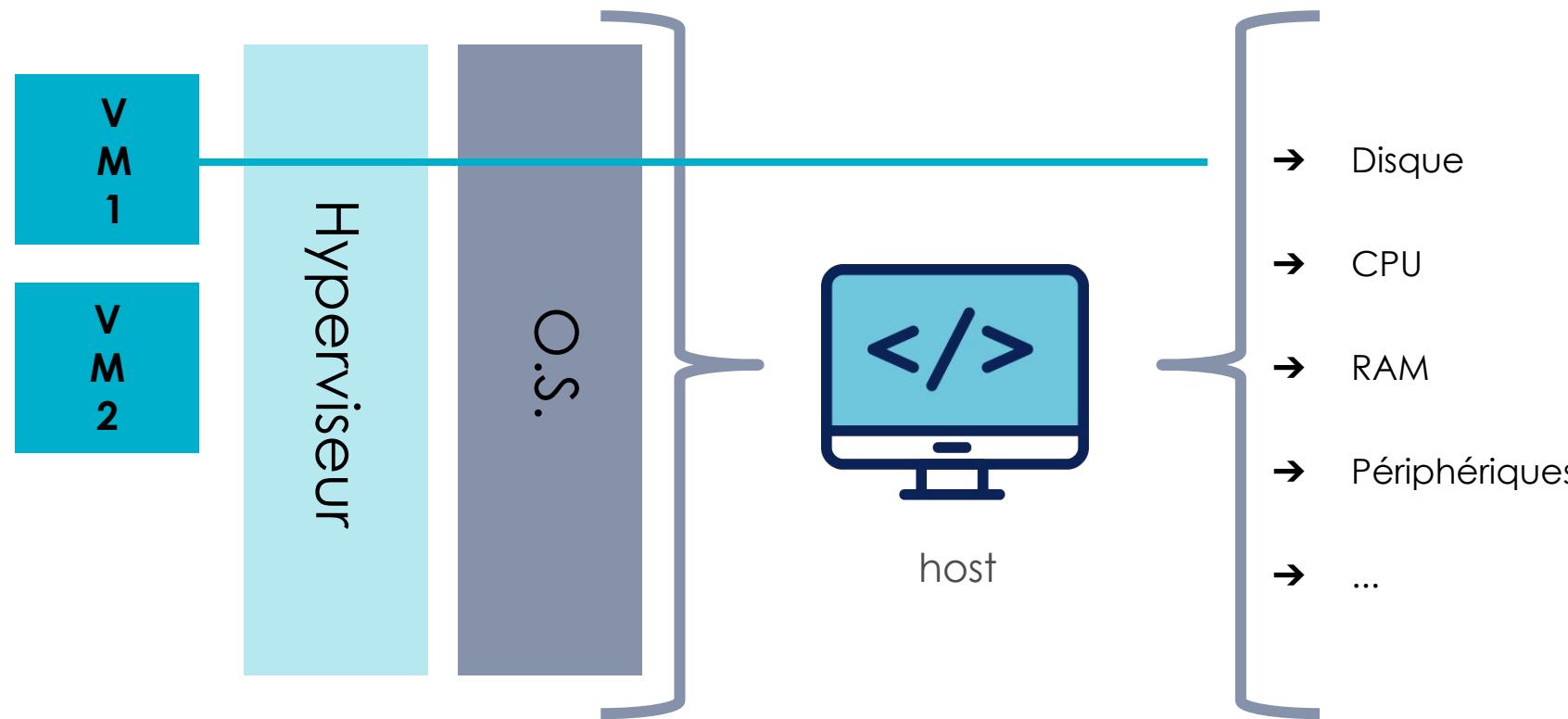
There is a better way



Un Hyperviseur permet un partage et une isolation des ressources mises à disposition par l'OS.



# QU'EST-CE QU'UNE MACHINE VIRTUELLE ?

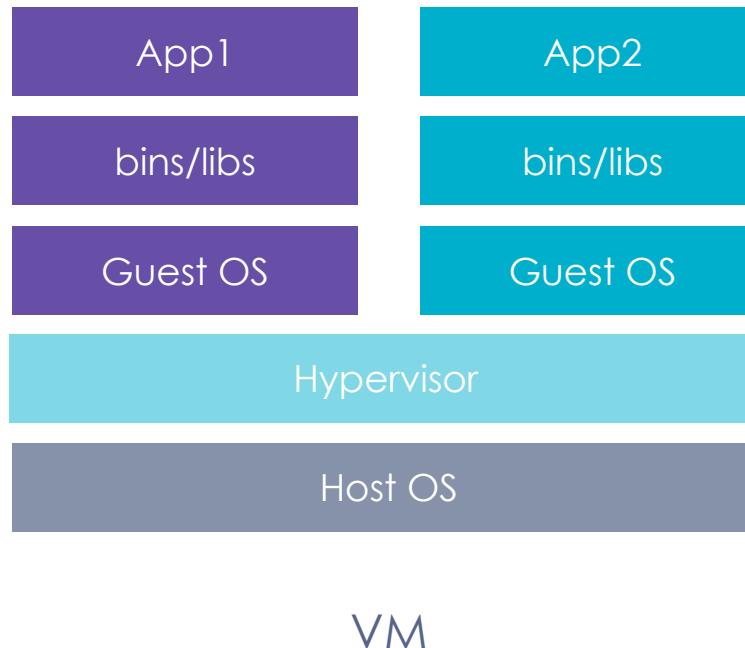


- Plusieurs machines virtuelles peuvent alors démarrer, émulant chacune leur propre système d'exploitation.
- Cela permet de démarrer plusieurs OS différents, chacun ayant accès à des ressources définies.
- L'environnement est désormais souple et jetable.

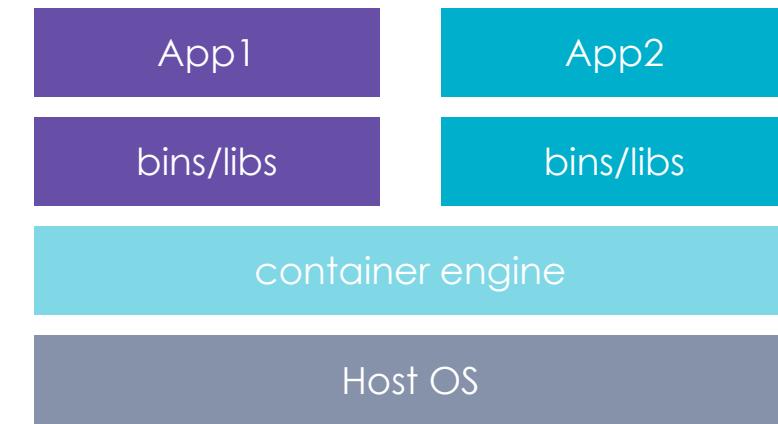


# QU'EST-CE QU'UN CONTENEUR ?

There is a better way



Vs.



05

# Cloud



Google Cloud Platform



Microsoft Azure



HEROKU

*Scalingo*



OVHcloud



# LES DIFFÉRENTS NIVEAUX DE CLOUD

There is a better way

## Traditional

On Premise

Business Logic

Application

Runtime

Operation System

Virtualization

Server

Storage

Network

## IAAS

Infrastructure as a Service

Business Logic

Application

Runtime

Operation System

Virtualization

Server

Storage

Network

## CAAS

Container as a Service

Business Logic

Application

Runtime

Operation System

Virtualization

Server

Storage

Network

## PAAS

Platform as a Service

Business Logic

Application

Runtime

Operation System

Virtualization

Server

Storage

Network

## FAAS

Function as a Service

Business Logic

Application

Runtime

Operation System

Virtualization

Server

Storage

Network

## SAAS

Software as a Service

Business Logic

Application

Runtime

Operation System

Virtualization

Server

Storage

Network

You manage

Vendors manage

# PAAS (PLATFORM AS A SERVICE)

- Le fournisseur tiers héberge les composants d'**infrastructure**
- Il fournit également les briques logiciels nécessaire à l'exécution de votre application.
  - Base de données
  - Exécuteur de code (PHP, NodeJS, Ruby, ...)
  - ...
- Le client n'apporte que le code de son application, qu'il « dépose », et ne se soucie pas (peu) de tout ce qui l'entoure



**Scalingo**



# IAAS (INFRASTRUCTURE AS A SERVICE)

- Le fournisseur tiers héberge les composants d'**infrastructure**
  - > Serveurs
  - > Ressources de stockage
  - > Réseau
  - > ...
- Il fournit au client une machine « jusqu'à l'OS »
  - > « Une VM avec Debian 8.7 installé dessus »
  - > == vagrant init / vagrant up



OVHcloud



Google Cloud Platform



Microsoft Azure



# SAAS (SOFTWARE AS A SERVICE)

- Toute l'application est fournie
- Gmail
- Dropbox
- Jupyter
- ...
- Exemples de IaaS / PaaS / SaaS ?



06

# Outils clés en mission

# LE PROTOCOLE

- Le **protocole SSH** permet d'établir une **connexion sécurisée** entre **deux machines**
- SSH se base sur TCP (écoute sur le port 22 par défaut)
- SSH prévoit un mode d'authentification du serveur :
  - Clé asymétrique serveur
- SSH prévoit plusieurs modes d'authentification du client :
  - Clé asymétrique utilisateur (mode recommandé)
  - Login / mot de passe, interactif...
  - Par certificats ssh
- La confusion entre clé privée et clé publique est souvent la cause de pas mal de problèmes.



On dit **chiffrer** / on parle de **chiffrement** (crypter = à bannir de votre vocabulaire "technique")  
**Authentification** (Savoir qui est qui) n'est pas pareil qu'**autorisation** (Savoir qui à le droit de faire quoi)

# Petit point de vocabulaire

## Cryptologie

### Cryptographie

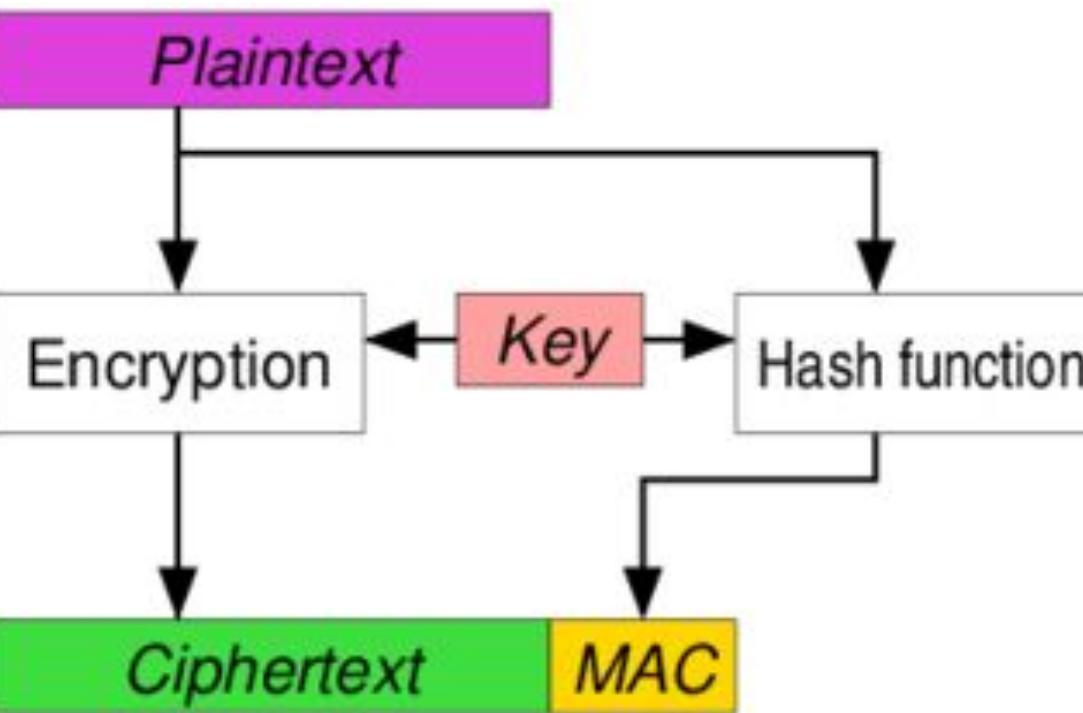
- **Chiffrer** : Texte en clair vers texte chiffré
- **Déchiffrer** : Texte chiffré vers texte en clair

### Cryptanalyse

- **Décrypter** : Texte chiffré vers texte en clair (de manière illégitime)

~~Crypter / Cryptage~~

# Le chiffrement authentifié (approche retenue par SSH)



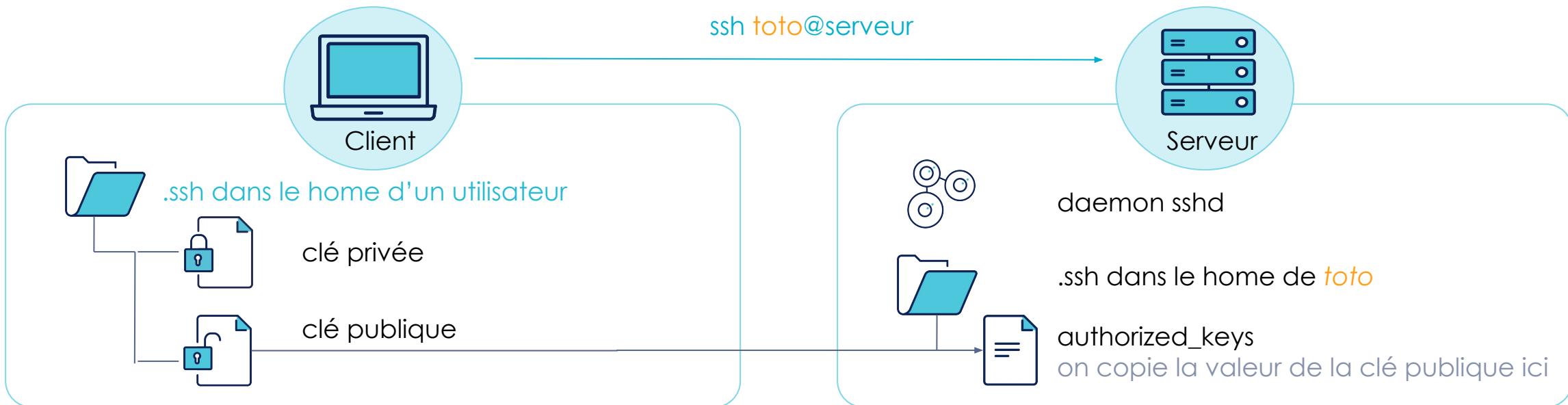
- Le texte est chiffré
- Le MAC est calculé sur le texte en clair

Approche retenue par  
SSH



# SSH

Ce qu'il faut pour établir une connexion



D'autres fichiers peuvent être trouvés dans le .ssh, à quoi correspondent-ils ?



config



known\_hosts



## KNOWN\_HOSTS

À la première connexion entre un client et un serveur, le client ne (re)connait pas la clé serveur car il ne l'a jamais vue !

```
The authenticity of host 'srv.bogops.io (193.55.37.7)' can't be established.  
ECDSA key fingerprint is SHA256:u2NnrW4yCw7eUV7iMleMoM41K9YrfoY8HkPfb8JL06c.  
Are you sure you want to continue connecting (yes/no)?
```

Il faut donc manuellement lui demander d'accepter l'empreinte (SHA256) du serveur. Il va la stocker dans `~/.ssh/known_hosts`. Cette empreinte sera associée à deux noms :

- le FQDN de la machine (`srv.bogops.io`)
- l'adresse IP de la machine (`193.55.37.7`)



## KNOWN\_HOSTS

Si la clé du serveur change (réinstallation d'une machine), la clé serveur est régénérée, et là, c'est le drame :

```
@@@@@WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
@ IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:u2NnrW4yCw7eUV7iMleMoM41K9YrfoY8HkPfb8JL06c.
Please contact your system administrator.
Add correct host key in /home/arno/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/arno/.ssh/known_hosts:1363
ECDSA host key for w.x.y.z has changed and you have requested strict checking.
Host key verification failed.
```

La commande `ssh-keygen -R w.x.y.z` permet d'« oublier » l'ancienne clé.  
(Modifier le fichier `~/.ssh/known_hosts` à la main produit le même résultat)

## LE FICHIER `~/.SSH/CONFIG`

- Le fichier `~/.ssh/config` permet de rentrer des paramètres pour éviter de taper des lignes de commande SSH de 3 km de long
- Il se compose d'un ensemble de règles qui, si elles matchent, s'appliquent pour la connexion
  - C'est un gain de temps considérable
- Quand on commence un projet, on fait proprement un `ssh_config` et on se le refile

```
Host mon_bastion
  HostName ec2-63-33-141-12.eu-west-1.compute.amazonaws.com
  User admin
  UserKnownHostsFile /dev/null

Host *.mon_projet
  UserKnownHostsFile /dev/null
  User admin
  ProxyJump mon_bastion
```

## AUTHENTIFICATION CLIENT PAR CLÉ, VUE DU SERVEUR

- Le serveur SSH est très psychorigide sur les droits des fichiers et des répertoires qui contiennent les fichiers authorized\_keys
- Si les droits sont trop ouverts (au groupe, aux autres utilisateurs), la clé publique ne sera pas utilisée parce qu'on ne plaisante pas avec la sécurité ici.
- Côté client, on va juste voir que l'authentification par clé échoue, et constater un fall-back sur l'authentification par login / mot de passe



“ssh -vvv” permet d'avoir plein de debug (parfois) utile pour comprendre ce qui se passe pendant la connexion entre le client et le serveur

# AUTHENTIFICATION CLIENT PAR CLÉ, VUE DU CLIENT

## Authentification par clé privée simple

On précise explicitement une clé (privée) à utiliser :

```
$ ssh -i ~/.ssh/id_rsa-srv1 bogops@srv1.machine.org
```

On laisse ssh chercher tout seul dans les clés par défaut :

- `~/.ssh/identity` (protocole version 1, danger)
- `~/.ssh/id_dsa` (bof, viteuf)
- `~/.ssh/id_ecdsa`
- `~/.ssh/id_ed25519`
- `~/.ssh/id_rsa`



## LES TUNNELS

Monter des tunnels TCP au travers de SSH est sans doute l'une des fonctions annexes les plus populaires pour dépasser le simple remote shell

Il y a plusieurs variantes de tunnels :

- Les tunnels en écoute locale (dits tunnels «L») 97%
- Les tunnels en écoute distante (dits tunnels «R») 1%
- Les tunnels dynamiques (dits tunnels «D») 2%

## TUNNEL EN ÉCOUTE LOCALE

```
machine-a$ ssh -L 8080:localhost:8088 machine-b
```

```
machine-a$ curl http://localhost:8080
```



C'est le cas classique régulièrement utilisé pour accéder à un service masqué

## TUNNEL EN ÉCOUTE LOCALE

```
machine-a$ ssh -L 8080:machine-c:8088 machine-b
```

```
machine-a$ curl http://localhost:8080
```



**Le tunnel va jusqu'à une autre machine que le serveur SSH.  
C'est machine-b qui fait la résolution du nom de machine-c**

## TUNNEL EN ÉCOUTE DISTANTE

```
machine-a$ ssh -R 8080:localhost:8088 machine-b
```

```
machine-b$ sudo systemctl tomcat stop && curl http://localhost:8080
```



**Le cas d'utilisation standard est lors du déboggage local d'une application censée tourner sur machine-b**



# Hands-On



## HANDS ON



<https://gitlab.com/octo-technology/octo-ops/les-bases-ops-formation>

**02 Hands On SSH.md  
dans le dossier :  
HandsOn\_01**



**Hands On!**

# Docker

G  
V  
T  
U

## BACK TO 2013 : CONTEXTE D'ARRIVÉE DE DOCKER

- De nombreuses problématiques liées aux applications
  - la portabilité des applications
  - la distribution des applications
  - le besoin de décorrélérer applications et infrastructure
  - la rationalisation des infrastructures
- La montée en puissance
  - des solutions de PaaS
  - de la philosophie DevOps

## UNE DÉFINITION DE DOCKER

Technologie permettant de standardiser le packaging et l'opération des applications par des conteneurs



# LES CONCEPTS DE DOCKER (1/2)



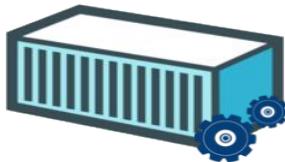
## L'image

Une arborescence de fichiers contenant tous les éléments requis pour faire tourner une application



## Le montage de répertoires

Un espace de stockage indépendant du conteneur utilisable pour les données persistantes



## Le conteneur

Une instantiation d'une image en cours d'exécution sur un système hôte



## Le Dockerfile

Un fichier contenant les instructions permettant de construire une image

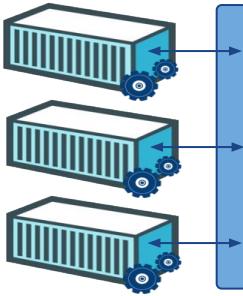


## Le registre

Un service centralisé de stockage et distribution d'images

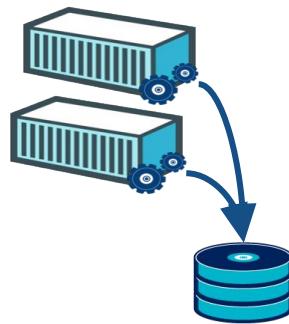


## LES CONCEPTS DE DOCKER (2/2)



### Les networks (docker network)

Permet la communication de conteneurs dans un ou plusieurs réseaux sur une ou plusieurs machines hôtes



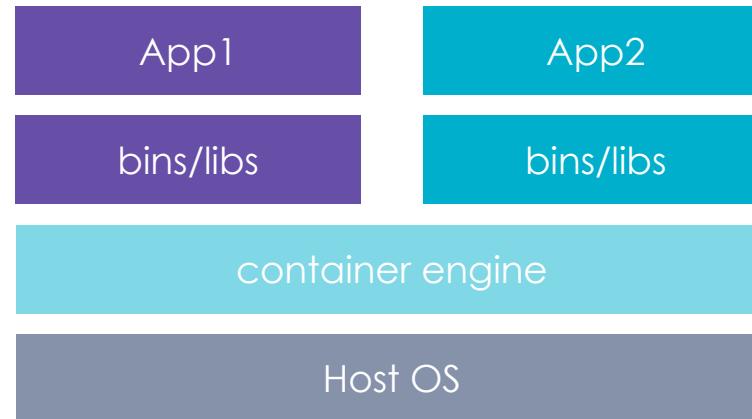
### Les volumes (docker volume)

Un espace de stockage indépendant de l'image utilisable pour les données persistantes. Ils possèdent leur propre cycle de vie, sont créés à côté des conteneurs sur lesquels ils peuvent être attachés (et détachés).



# QU'EST-CE QU'UN CONTENEUR ?

There is a better way

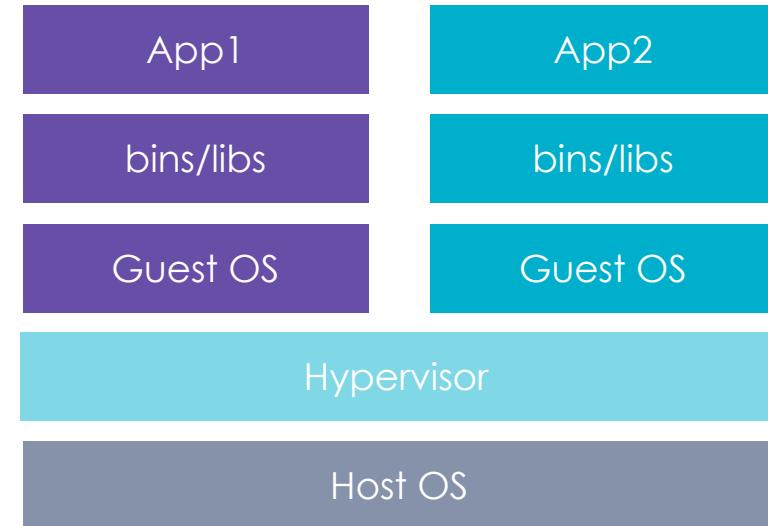


Container

Vs.



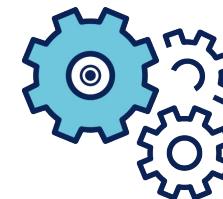
Ce n'est pas une VM light



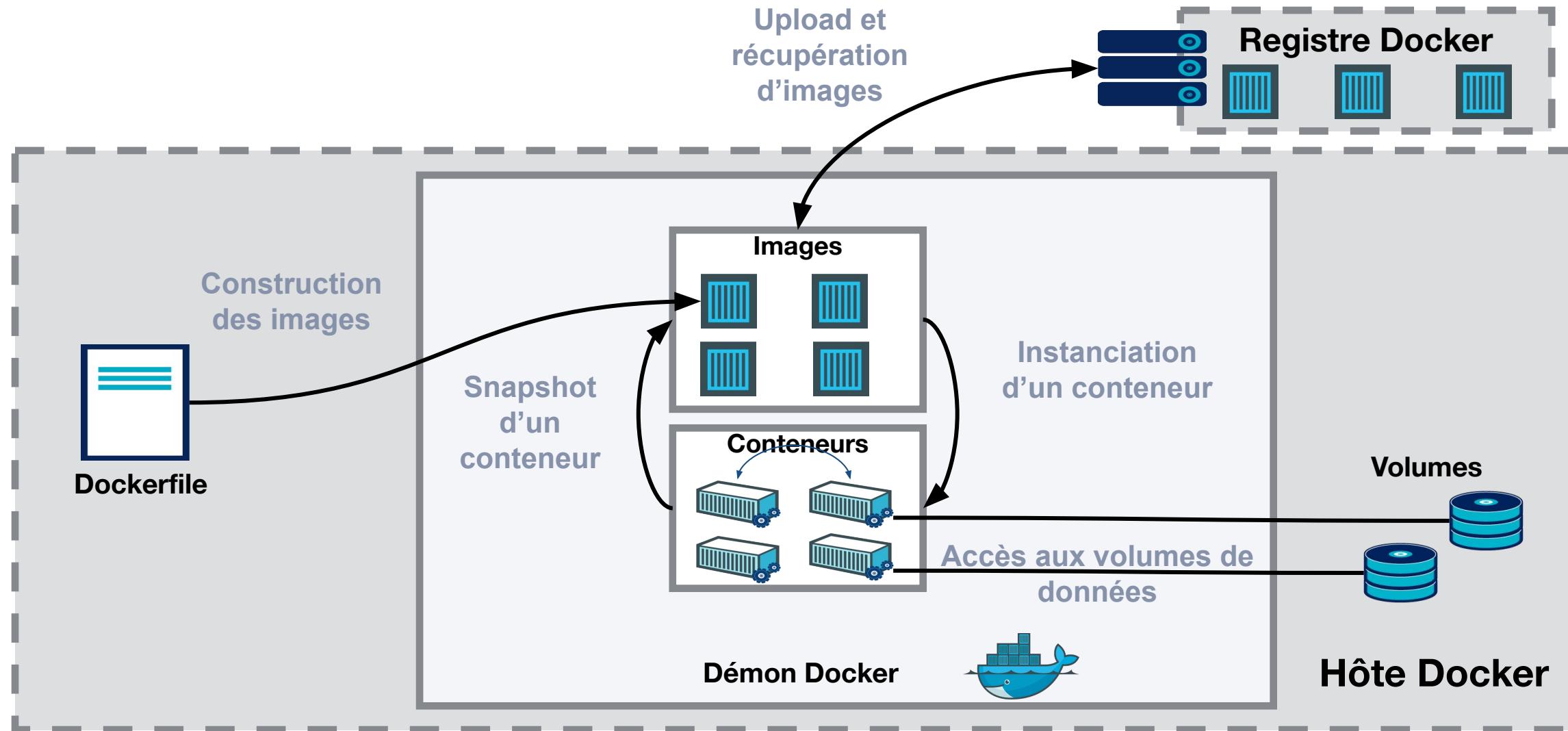
VM



1 container = 1 process



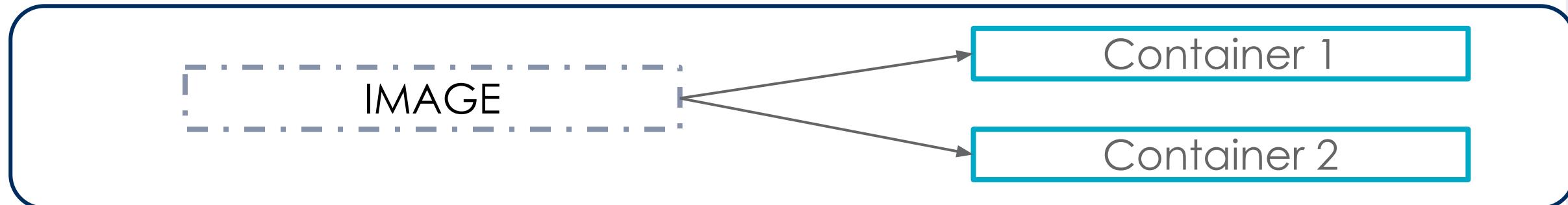
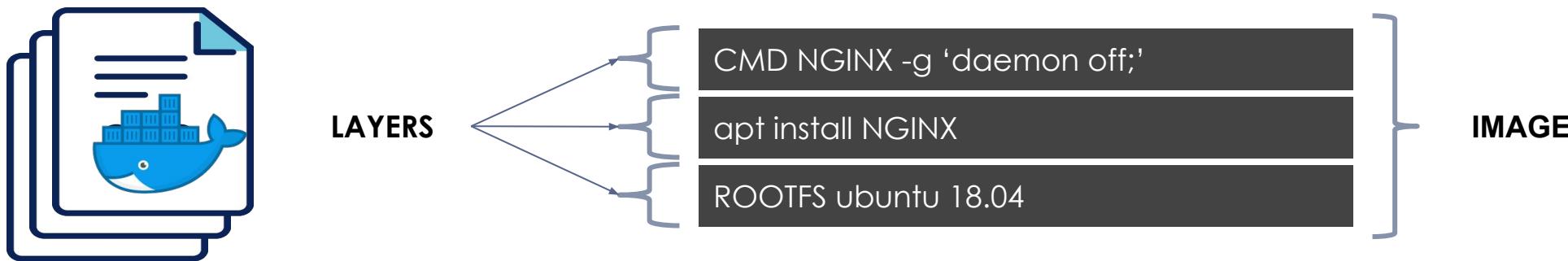
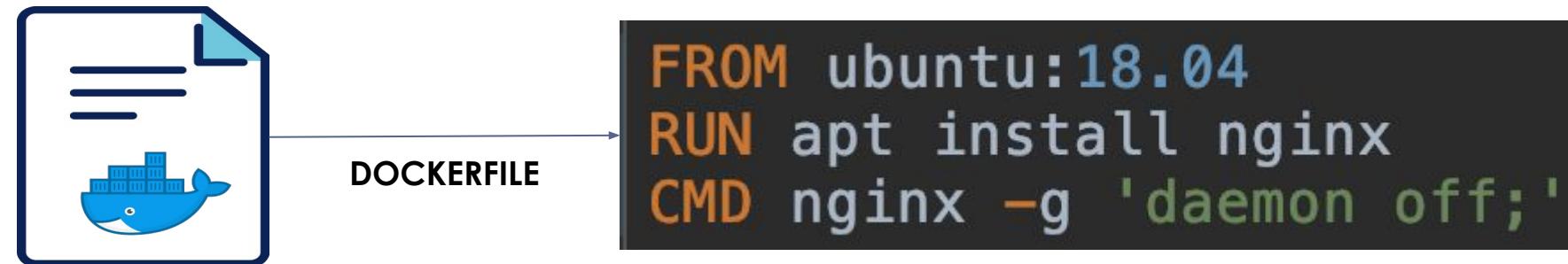
# LES CONCEPTS DE DOCKER





# DOCKER, IMAGES ET CONTENEURS

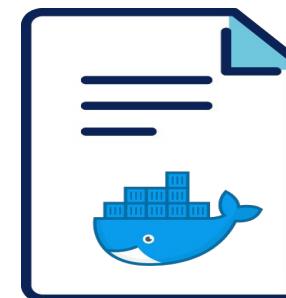
There is a better way



# LES IMAGES DOCKER

Une **image Docker** est une « photo » de l'état souhaité de notre serveur (code, dépendances).  
Toute image a :

Un système de fichiers auto-suffisant  
contenant *a minima* librairies et  
binaires de base (libc, libresolv, bash...)



Un identifiant unique assigné  
à l'image à sa création

Des métadonnées pour préciser la façon  
d'instancier l'image :

- le processus à exécuter à l'instanciation de l'image
- les variables d'environnement à positionner
- l'utilisateur qui va lancer l'application
- la configuration réseau (ports exposés, réseau...)
- les volumes de données à connecter



# L'ESSENTIEL DES COMMANDES : *docker image*

There is a better way

## **docker image pull**

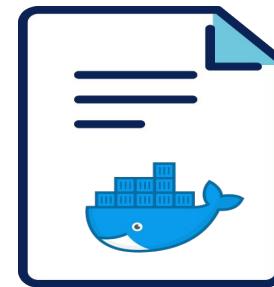
Télécharger une image à partir du Registre

## **docker image tag**

Labelliser une image

## **docker image ls**

Lister les images locales



## **docker image rm**

Supprimer une image

## **docker image history**

Lister les couches d'une image

## **docker image build**

Construire une image à l'aide d'un Dockerfile

# COMMANDÉ DOCKER : *docker image pull*

**docker image pull** : demande à Docker de récupérer localement une image de conteneurs sur un registre distant

## Récupération de la dernière image d'ubuntu

:

```
$ docker image pull ubuntu
```



Syntaxe d'un nom d'image : [**<repository>/**]image[:<tag>]

- **repository**: url du dépôt d'images (si non précisé, va sur le dépôt par défaut : hub.docker.com)
- **image** : nom de l'image
- **tag** : identifie l'image de manière unique dans le dépôt (vaut *latest* par défaut)

# COMMANDE DOCKER : *docker image ls*

**docker image ls** : Interroge le registre local à l'hôte Docker et affiche la liste des images présentes localement

## Exemple de sortie de la commande image

```
$ docker image ls
```

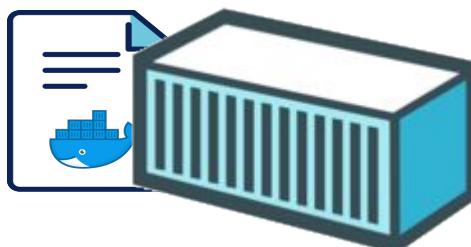
REPOSITORY	TAG	IMAGE ID	CREATED	VIRTUAL SIZE
<none>	<none>	5b7faa37374	2 hours ago	1.207 GB
ubuntu	latest	9aafc45696a	6 hours ago	546 MB
ubuntu	willy	9aafc45696a	6 hours ago	546 MB
tutum	mysql	9bb40134b3d	8 hours ago	1.041 GB

- Une image locale peut n'être associée à aucun repository ou tag
- **VIRTUAL SIZE** représente la somme de toutes les couches de l'image

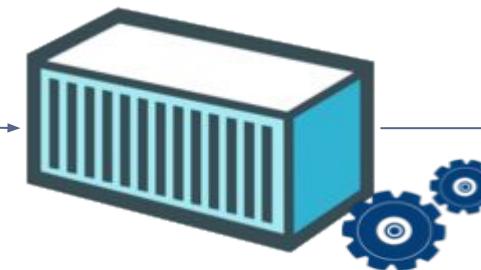
# LE CONTENEUR DOCKER

Une conteneur Docker est instancié à partir d'une image. C'est une "boîte" où tourne un processus, que l'on utilise puis que l'on détruit.

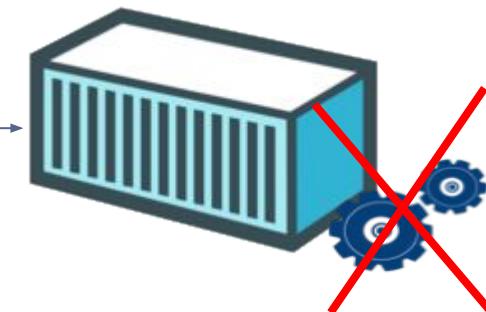
On instancie un conteneur à partir d'une image



Le conteneur ne vit que tant que son processus tourne



Si le processus s'arrête, le conteneur est stoppé



- Le contenu modifié subsiste tant que le conteneur n'est pas détruit
- **Une couche en écriture est créée à linstanciation du conteneur et supprimée à sa destruction**



# L'ESSENTIEL DES COMMANDES

**docker container run**

Créer un conteneur et lancer une commande

**docker container start**

Démarrer un conteneur

**docker container rm**

Supprimer un conteneur

**docker container ls**

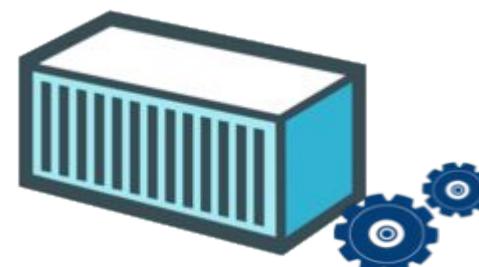
Lister les conteneurs

**docker container stop**

Stopper un conteneur

**docker container restart**

Redémarrer un conteneur



# COMMANDE DOCKER : *docker container run*

**docker container run** : lance un conteneur à partir d'une image

## Exemple de lancement d'un conteneur ubuntu

```
$ docker container run -i -t ubuntu /bin/bash
```

### Anatomie de la ligne de commande

- **docker container run** : lance un conteneur
- **-i** : laisse l'entrée standard ouverte
- **-t** : assigne un terminal (tty) sur le conteneur et permet d'avoir un shell interactif
- **ubuntu** : utiliser l'image “ubuntu:latest”
- **/bin/bash** : lancer le shell bash à l'intérieur du conteneur

# DOCKER LOGS ET LA GESTION DES LOGS

- **Une application conteneurisée doit envoyer ses logs sur les sorties standard / erreur**  
Plus de logging direct dans un fichier ou vers un solution de gestion de logs externe !
- **Récupération automatique des sorties par Docker** et sauvegarde dans un fichier local par défaut

## Exemple de sortie de docker logs pour un conteneur wordpress

```
$ docker container logs wordpress
```

```
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.4. Set the  
'ServerName' directive globally to suppress this message  
PHP Warning: file_put_contents(/var/www/html/wp-content/themes//lib/css/theme.css) [192.168.99.1 - - [19/May/2016:07:13:36 +0000] "POST /wp-admin/admin-ajax.php HTTP/1.1" 200 491 [...]  
Chrome/50.0.2661.102 Safari/537.36"  
[Wed Mar 26 06:26:19 2014] [error] [client 127.0.0.1] KILLED QUERY: SELECT DISTINCT `user`.`ID` AS user_id, t.* FROM ...
```



# LES COMMANDES AVANCÉES

**docker container create**

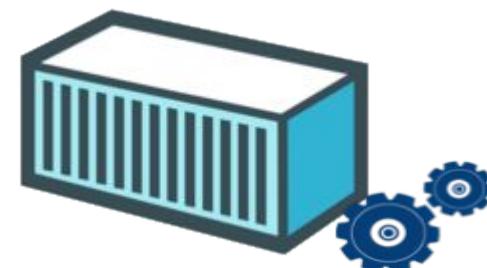
Créer un conteneur

**docker container top**

Lister les processus

**docker container diff**

Affiche les différences dans le système de fichiers du conteneur



**docker container exec**

Exécuter une commande dans le conteneur

**docker container kill**

Tue un conteneur

**docker container inspect**

Affiche des informations sur un conteneur

# COMMANDE DOCKER : *docker container exec*

**docker container exec** : Exécute une commande à l'intérieur d'un conteneur en cours d'exécution

**Exemple d'exécution de la commande “touch /tmp/testfile”**

```
$ docker container exec my_nginx touch /tmp/testfile
```



- Permet de **faciliter le debugging** dans un conteneur
- À condition d'avoir au moins un shell dans le conteneur (bash, sh, ash)

# COMMANDE DOCKER : *docker container inspect*

**docker container inspect** : Affiche des informations de bas niveau sur un conteneur

**Exemple de commande inspect pour obtenir le fichier de logs du conteneur**

```
$ docker container inspect --format='{{.LogPath}}' my_container
```

- Sortie de l'intégralité des informations **au format JSON**
- Possibilité de sélectionner les informations (--format)
- Quelques informations utiles à récupérer
  - Code de sortie du conteneur: '{{.State.ExitCode}}'
  - Adresse IP de l'instance: '{{.NetworkSettings.IPAddress}}'
  - Ports exposés: '{{.Config.ExposedPorts}}'
  - Variables d'environnement du conteneur: '{{.Config.Env}}'

# GESTION DES DONNÉES AVEC DOCKER

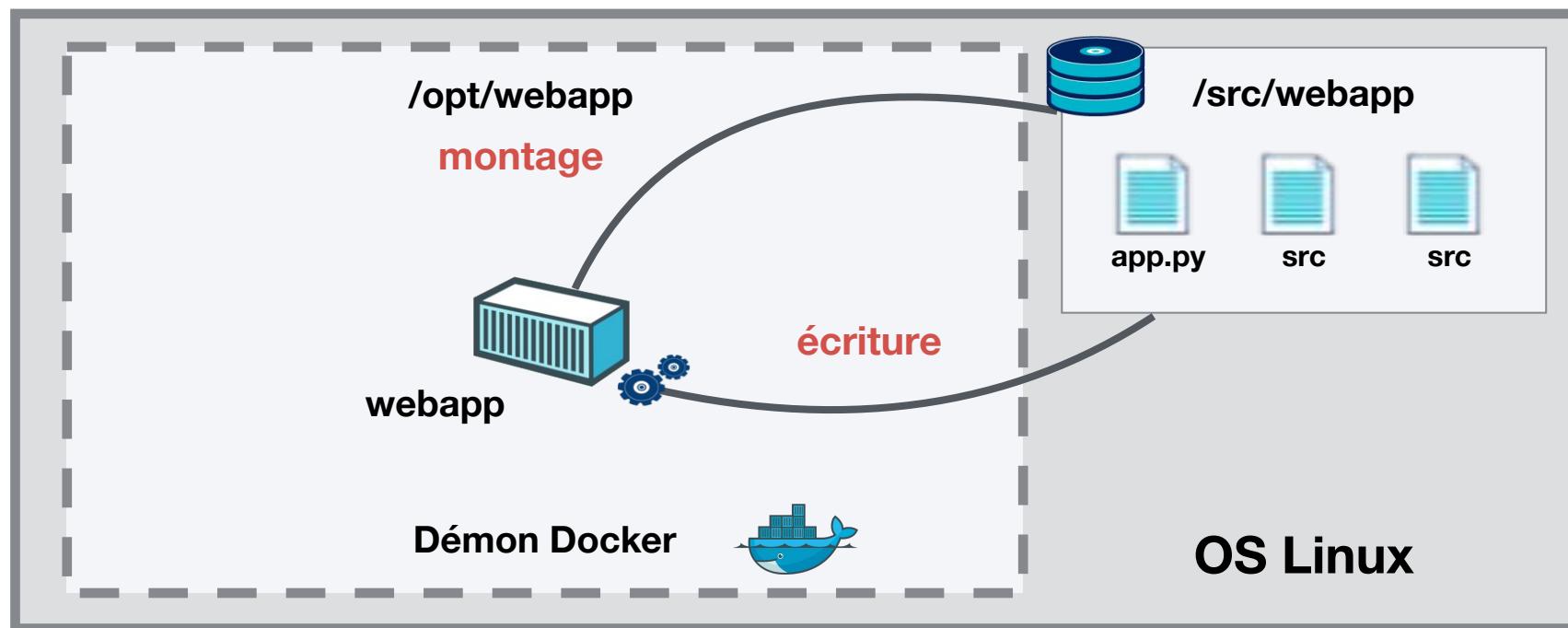
- Les conteneurs sont **légers et éphémères** : ils ne sont pas faits pour enregistrer des données de manières persistantes
- Les données doivent être stockées en dehors de l'image du conteneur dans des **volumes de données dédiés** à cet usage
- **2 techniques historiques pour accéder à des volumes** de données
  - l'accès au système de fichiers de l'hôte,
  - le volume lié au conteneur

# L'ACCÈS AU SYSTÈME DE FICHIERS DE L'HÔTE

Un répertoire ou un fichier de l'hôte est rendu accessible dans le conteneur

**Exemple de création d'un conteneur avec un volume associé monté sur /opt/webapp**

```
$ sudo docker container run -v /src/webapp:/opt/webapp webapp python app.py
```



# LES VOLUMES DOCKER

Objectifs :

- **Gérer des applications stateful** : pour les bases de données et les applications à architecture traditionnelle
- **Conserver l'indépendance avec les hôtes** : les données ne doivent pas être liées à un hôte et doivent suivre le déplacement des conteneurs qui y sont associés
- **Pouvoir conserver des données indépendamment de l'application** : pour gérer le cycle de vie de la donnée
- **Gérer les niveaux de services pour le stockage de la donnée** : SSD, IOPS garanties...
- **Faciliter les tâches opérationnelles** : snapshot, sauvegarde, restauration, copie...

A red panda is climbing a tree trunk in a dense forest. The panda's body is angled downwards, with its front paws gripping a horizontal branch and its back legs clinging to the trunk. Its thick, reddish-brown fur is visible, along with its white face, ears, and paws. The background is filled with out-of-focus green foliage and other tree trunks.

Hands-On



# HANDS ON



<https://gitlab.com/octo-technology/octo-ops/les-bases-ops-formation>

Part 01.md  
dans le dossier :  
**HandsOn\_02**



**Hands On!**

# LE DOCKERFILE

- Fichier contenant des suites d'instructions Docker et de commandes à exécuter pour construire un conteneur
- Permet par exemple d'installer tous les paquets requis par une application (Apache, Java, nginx...)

## Exemple de fichiers Dockerfile

```
FROM ubuntu

# Update the repository and install nginx
RUN apt-get update && apt-get install -y nginx

# Copy a configuration file from the current directory
# ADD nginx.conf /etc/nginx/

EXPOSE 80

CMD ["nginx"]
```

## DOCKERFILE : INSTRUCTIONS DE BASE

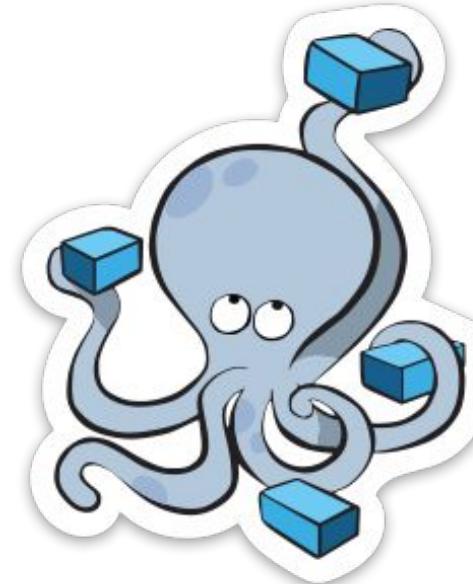
- **RUN** : exécute une commande pour construire l'image et ajoute une couche à l'image finale
- **ENTRYPOINT** : si présent, script qui sera exécuté au démarrage du conteneur.
- **CMD** : commande par défaut qui sera exécutée. Si l'entrypoint est défini, le contenu de CMD est passé en argument de l'entrypoint
- **WORKDIR** : configure le répertoire courant, toutes les prochaines commandes seront exécutées depuis ce répertoire
- **VOLUME** : déclare un volume au sein du conteneur pour sauvegarder des données
- **ENV** : configure une variable d'environnement.
- **USER** : après cette instruction, toutes les commandes sont exécutées par l'utilisateur spécifié.
- **EXPOSE** : documente le fait que l'application écoute sur un port. Pour pouvoir l'exposer, il faut le préciser au lancement du container.
- **COPY** : copie un fichier ou un dossier de l'Hôte dans le conteneur au chemin spécifié.
- **ADD** : ajoute un fichier ou un dossier dans le conteneur au chemin spécifié. Décomprime également les archives.

## DOCKERFILE : UN MOT SUR LES IMAGES DE BASE

- Toutes les distributions historiques fournissent des images de base : Debian, CentOS, Ubuntu...
- Ces images peuvent être considérées comme trop riches et complexes dans le cadre de la conteneurisation
- Des images encore plus minimalistes sont apparues

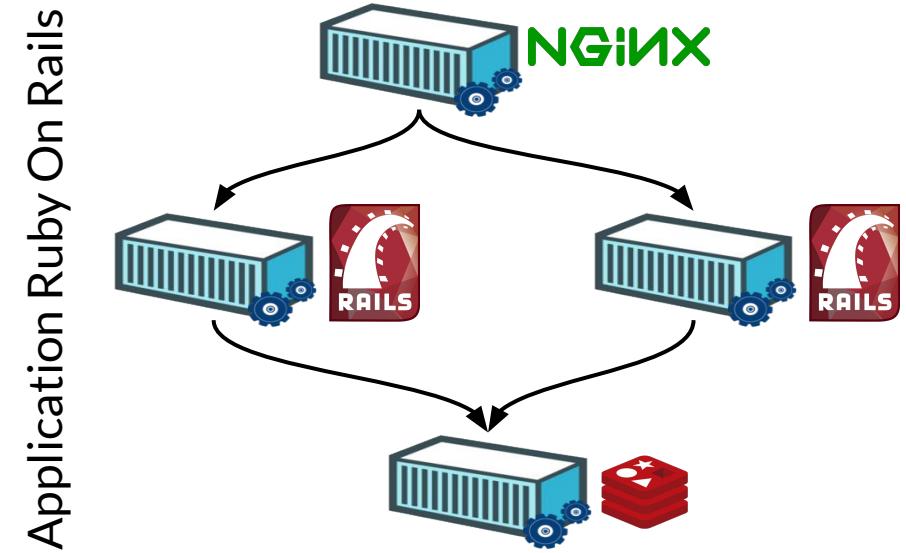


# Le déploiement de topologies avec Docker Compose



# POURQUOI DOCKER COMPOSE ?

- Une application moderne est généralement composée de **multiples conteneurs** avec de **nombreux liens** entre eux, qui utilisent potentiellement des **volumes**...
- **De nombreuses étapes à scripter** pour déployer une telle application nativement ... pour chaque environnement !



## EXEMPLE POUR MONTER UN ENVIRONNEMENT COMPLET

```
$ cd ruby_on_rails_app
$ docker build --tag ruby_on_rails_app .
$ docker network create --driver overlay mynet
$ docker volume create --driver=volplugin --name=myvolume1 --opt size=40G
$ docker volume create --driver=volplugin --name=myvolume2 --opt size=40G
$ docker container run --name my_redis --network mynet --detach redis
$ docker container run --name ruby_on_rails_app_1 --network mynet --detach --volume-driver=volplugin
--volume myvolume1:/var/lib/data ruby_on_rails_app
$ docker container run --name ruby_on_rails_app_2 --network mynet --detach --volume-driver=volplugin
--volume myvolume2:/var/lib/data ruby_on_rails_app
...
...
```

La complexité du lancement d'un environnement complet peut vite devenir imposante et propice à des erreurs...

# QU'EST-CE QUE DOCKER COMPOSE ?

- Un outil Python en ligne de commande qui apporte
  - **un format de description** d'une application multi-conteneurs
  - **le déploiement d'applications multi-conteneurs** en local ou sur un cluster
- Initialement développé en dehors de Docker sous le nom de **Fig**, acquis par Docker Inc en 2014
- Permet notamment
  - le lancement de **multiples versions d'un environnement**
  - **le scaling (manuel)** des conteneurs stateless
- Actuellement en version 3

# LE FORMAT DE DESCRIPTION

There is a better way

Déclaration de chaque conteneur : créé un alias DNS

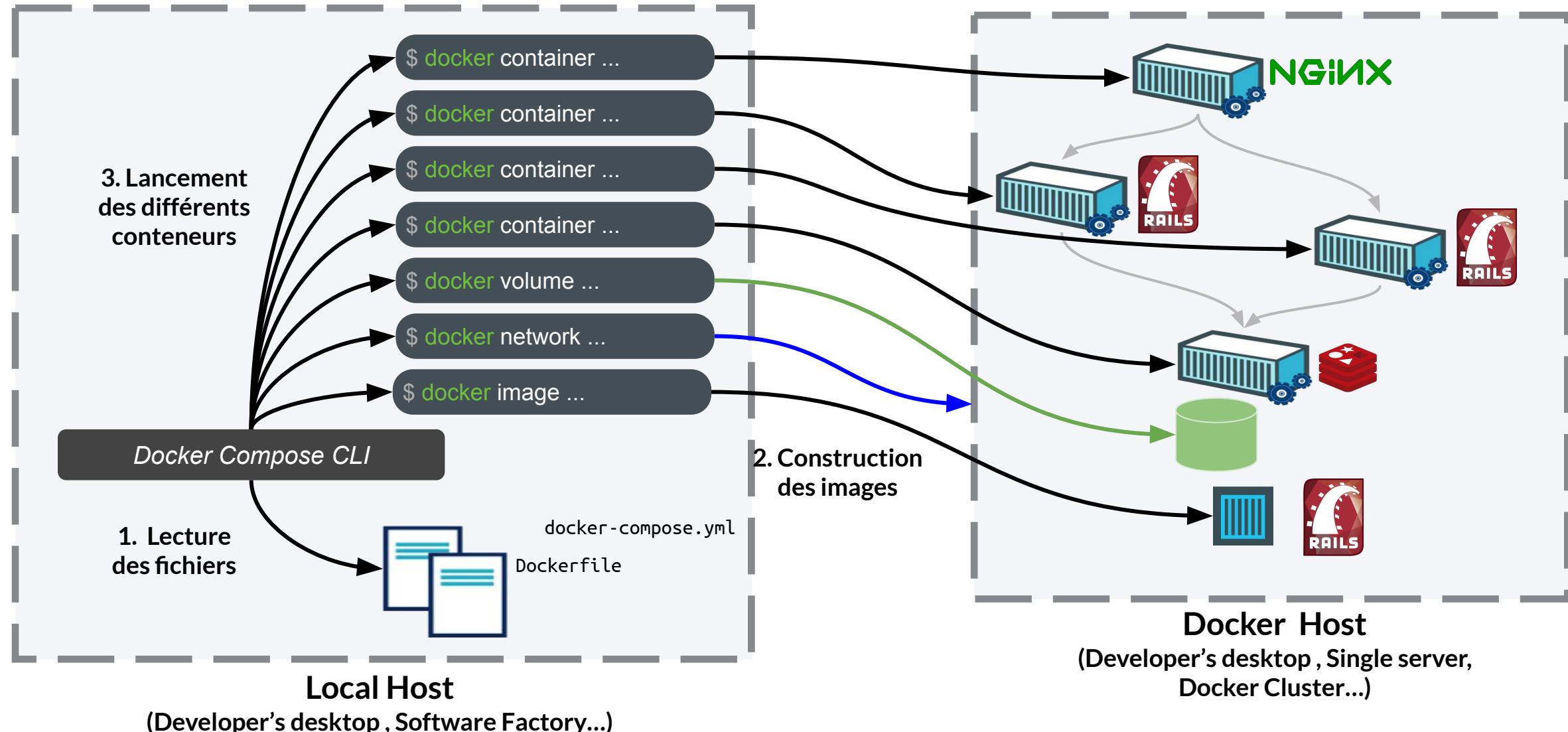
`<> docker-compose.yml`

```
version: "3"
services:
  web:
    build: .
    command: python app.py
    ports:
      - "5000:5000"
    volumes:
      - ./code
    depends_on:
      - redis
    networks:
      - mynet
  redis:
    image: orchardup/redis
    volumes:
      - redis-data:/var/lib/redis
    networks:
      - mynet
    mynet:
      driver: bridge
    volumes:
      redis-data:
        driver: local
```

Instructions pour construire ou récupérer l'image



# FONCTIONNEMENT DE DOCKER COMPOSE





# L'ESSENTIEL DES COMMANDES

**docker-compose up [-d]**

Créer un environnement de développement  
avec un **docker-compose.yml**

**docker-compose logs**

Accéder aux logs des conteneurs

**docker-compose rm**

Supprimer les conteneurs stoppés

**docker-compose ps**

Lister les conteneurs lancés

**docker-compose up --scale**

Ajouter des instances d'un conteneur

**docker-compose stop | start**

Stoppe/Démarre les services

**docker-compose push**

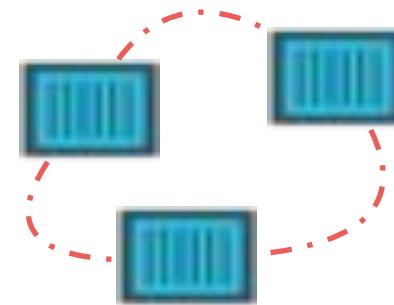
Pousse les images buildées localement

**docker-compose restart**

Redémarrer les services

**docker-compose build**

Construire les images



# LIMITATIONS DE DOCKER COMPOSE

On vous conseille de n'utiliser **Docker Compose que sur votre poste de dev en local**

:



- Docker Compose est incompatible avec un orchestrateur
- Tous les conteneurs sont sur la même machine (pas de haute-dispo)
- Rien pour relancer le Docker Compose s'il plante

A close-up photograph of a young lion cub lying in tall, green grass. The cub is captured in the middle of a wide yawn, with its mouth wide open showing its tongue and lower teeth. Its eyes are closed, and its ears are perked up. The background is a soft-focus blend of green and brown grass.

Hands-On



## HANDS ON



<https://gitlab.com/octo-technology/octo-ops/les-bases-ops-formation>

Part 02.md  
dans le dossier :  
**HandsOn\_02**



**Hands On!**

05

# Take away