

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
 - ▷ Qu'est ce que Docker
 - ▷ Architecture et concepts Docker
- TP#1

Docker en pratique

- ▷ Les images Docker
 - ▷ Utilisation de Docker
 - ▷ Les volumes
 - ▷ Création d'images et registres
 - ▷ Docker Compose
- TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
 - ▷ Utilisation du client kubectl
- TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps
 - ▷ Liveness et Readiness
 - ▷ Routes HTTP
 - ▷ Maîtrise des capacités
 - ▷ Monitoring applicatif
 - ▷ Log Management
- TP#4
TP#5
TP#6
TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
 - ▷ Les statefulsets
 - ▷ CRD et opérateurs
- TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

« **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 adressant à la fois les dev et les ops



Comment assurer le déploiement homogène d'une application sur tous ses environnements ?



	Environnement de développement	Assurance Qualité	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Jobs en Arrière Plan	?	?	?	?	?	?	?
Base de données	?	?	?	?	?	?	?
Analytics	?	?	?	?	?	?	?
Files de messages	?	?	?	?	?	?	?



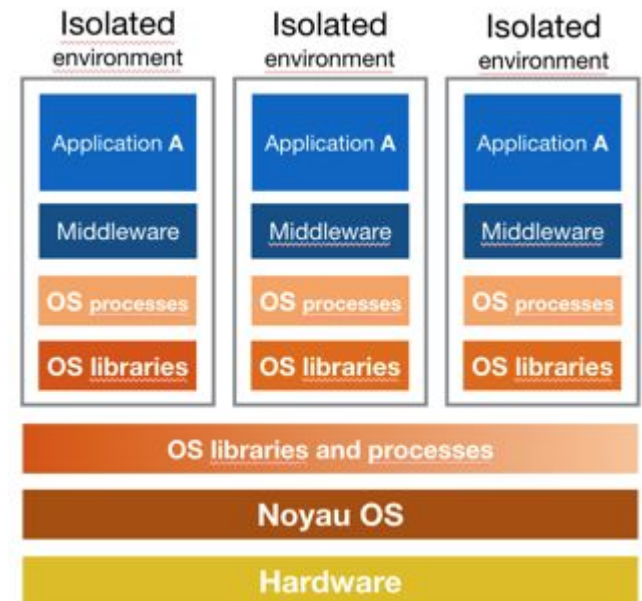
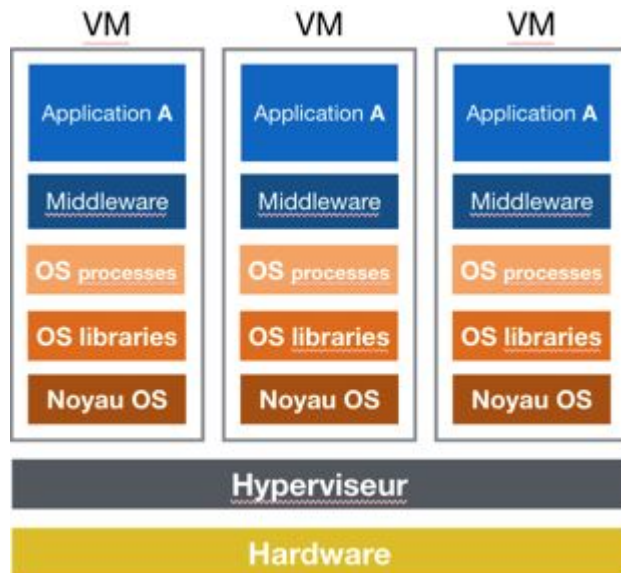
Comment optimiser l'utilisation des ressources ? Comment décorrélérer application et infrastructure ?

Les 2 technologies de virtualisation des systèmes

Virtualisation

vs

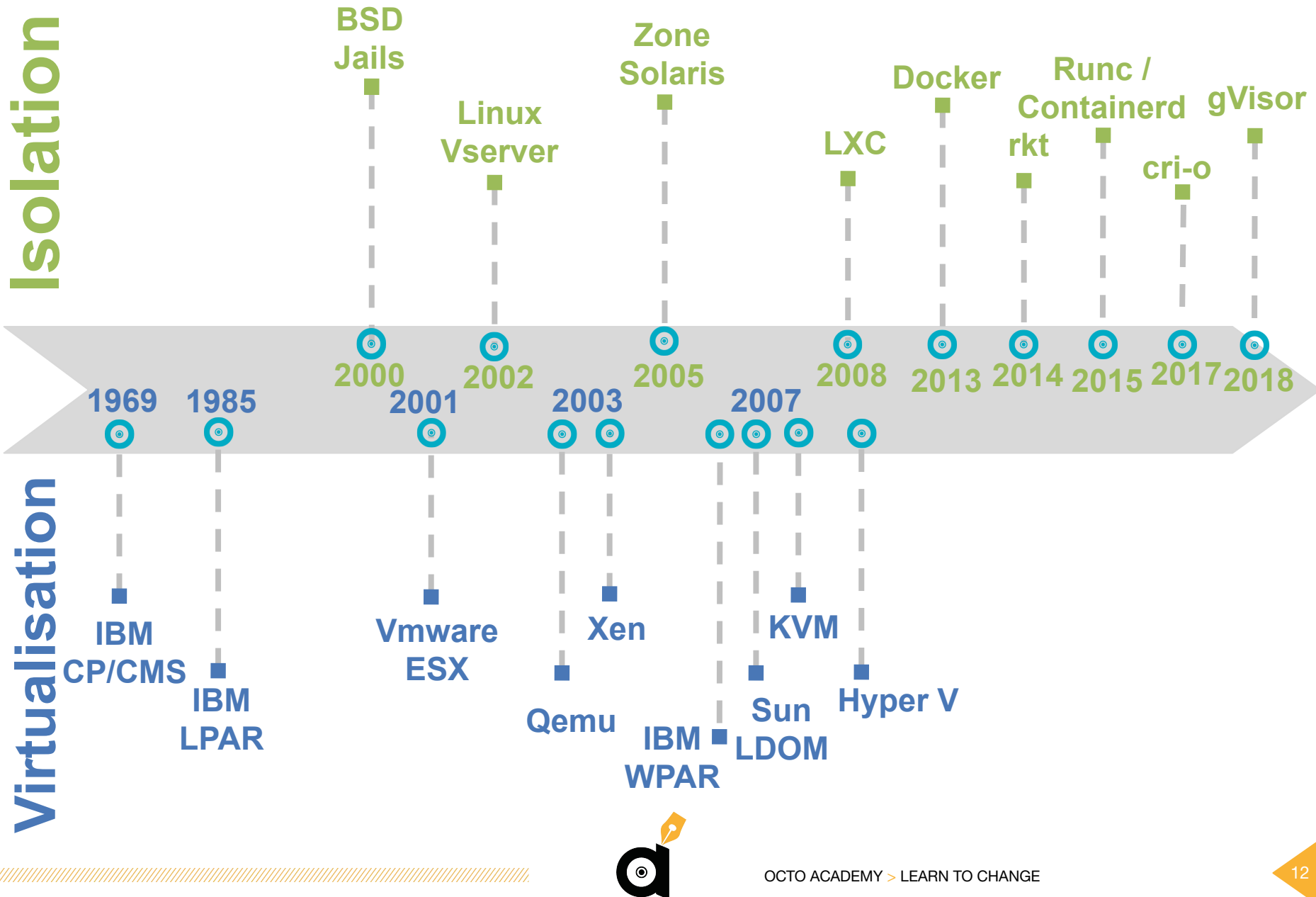
Isolation



Back to 2013 : Des technologies qui ne datent pas d'hier

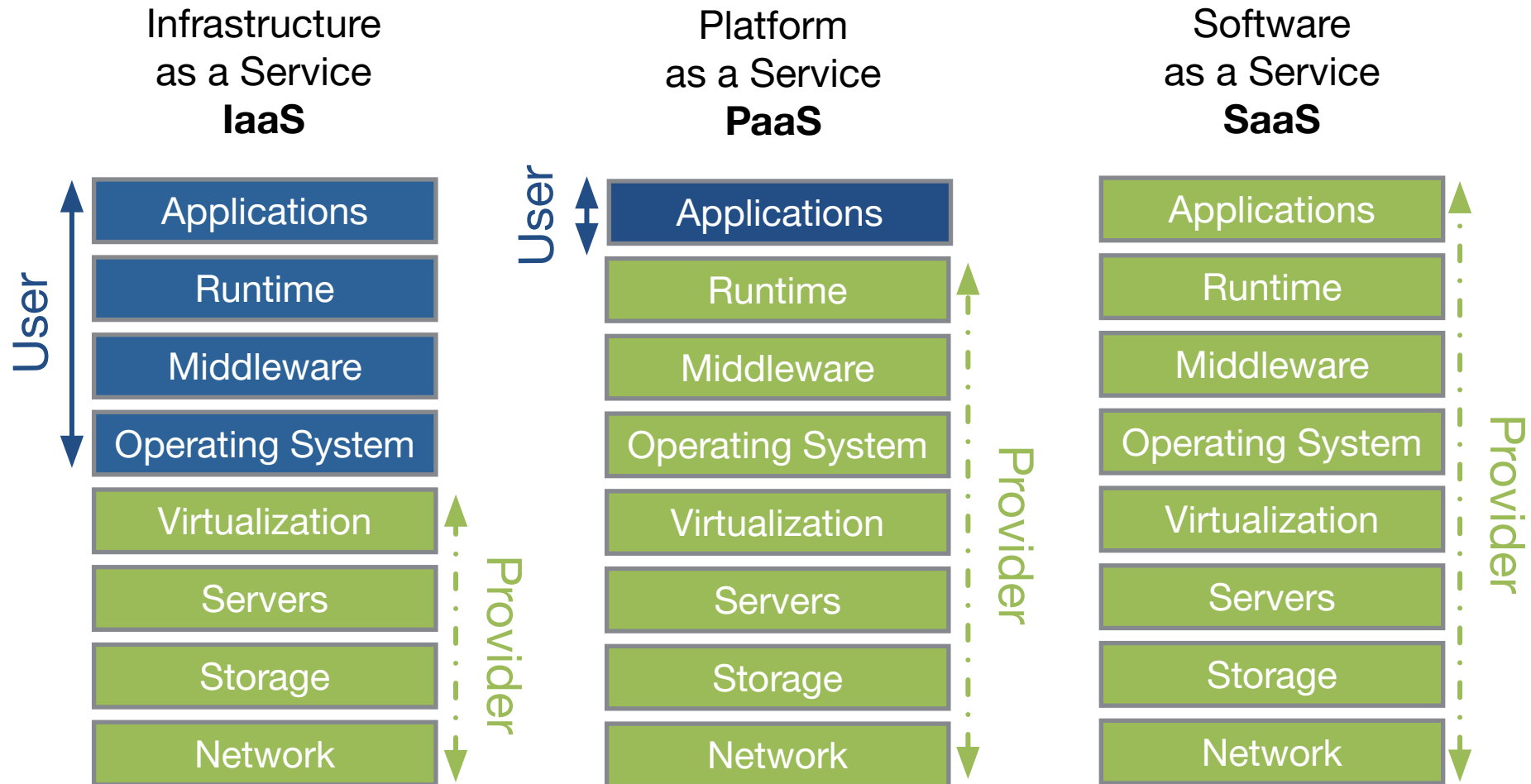
Isolation

Virtualisation



Back to 2013 : Qu'est ce qu'un PaaS ?

Schéma des différents niveaux de services Cloud



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ **Qu'est ce que Docker**
- ▷ Architecture et concepts Docker

TP#1

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose

TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl

TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

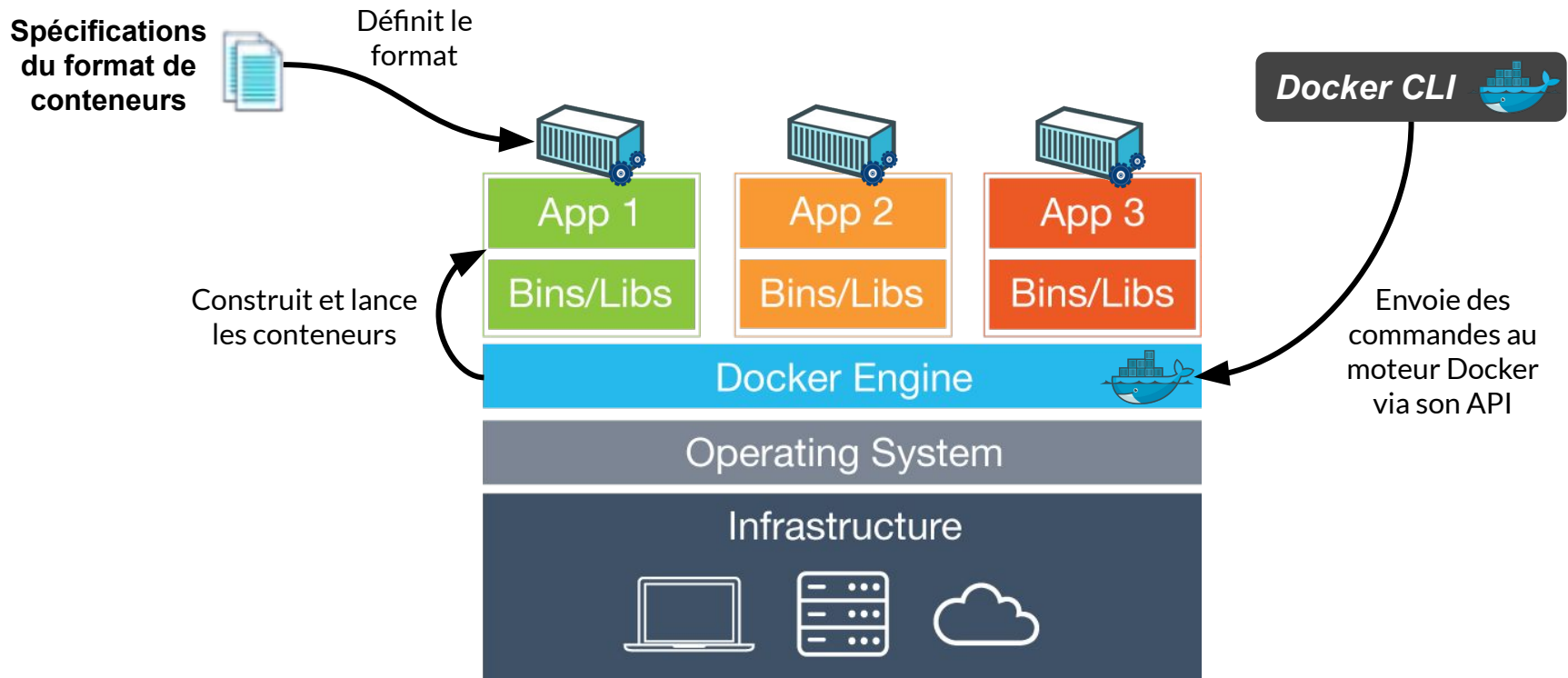
Une définition de Docker

« Une **technologie** permettant de **standardiser** le **packaging** et l'**opération** des **applications** »



Docker Engine, Conteneur Docker et Docker CLI

- L'ensemble de technologies **initialement appelé Docker**



- Des technologies **standardisées** au sein de l'Open Container Initiative (OCI)



Comprendre Docker par ses caractéristiques

Des caractéristiques uniques

PO

PORTABLE



DI

DISPOSABLE



LI

LIVE



SO

SOCIAL



Portable




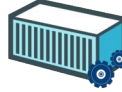



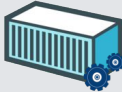

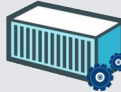







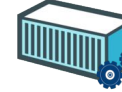
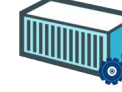
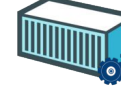
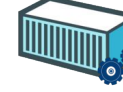
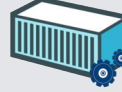
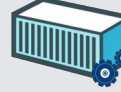





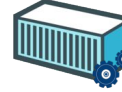

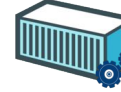
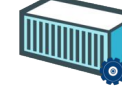
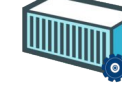
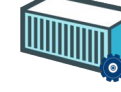
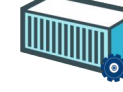


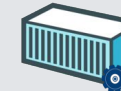
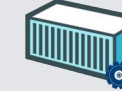





	Environnement de développement	Assurance Qualité	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Jobs en Arrière Plan	?	?	?	?	?	?	?
Base de données	?	?	?	?	?	?	?
Analytics	?	?	?	?	?	?	?
Files de messages	?	?	?	?	?	?	?



Portable



	Environnement de développement	Assurance Qualité	Serveur de Production	Cluster de machines	Cloud Public	Ordinateur Personnel	Serveur du Client
Site web statique							
Web frontend							
Jobs en Arrière Plan							
Base de données							
Analytics							
Files de messages							



Disposable



- ▷ Des images de conteneurs en lecture seule : la modification entraîne la création d'une nouvelle image
- ▷ Les modifications dans les conteneurs sont locales et temporaires
- ▷ La vie d'un conteneur est liée à l'application





- ▷ Les images de conteneurs sont versionnées et incrémentales
 - > La configuration (port, processus, ...)
 - > Le système de fichiers

- ▷ Un système de versionning similaire à Git
 - > Gestion des diffs
 - > Gestion des versions en arbre
 - > Gestion des Tags





- ▷ Système de dépôt d'images de conteneurs (registre)
 - > Accessible depuis internet ou en interne
 - > Recherche facile

- ▷ Des outils communautaires
 - > Ouverts gratuitement
 - > Système de vote sur les images de conteneurs
 - > “Trusted images” et images de conteneurs “officielles”



TP#1

AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
 - ▷ Qu'est ce que Docker
 - ▷ Architecture et concepts Docker
- TP#1

Docker en pratique

- ▷ Les images Docker
 - ▷ Utilisation de Docker
 - ▷ Les volumes
 - ▷ Création d'images et registres
 - ▷ Docker Compose
- TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
 - ▷ Utilisation du client kubectl
- TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps
 - ▷ Liveness et Readiness
 - ▷ Routes HTTP
 - ▷ Maîtrise des capacités
 - ▷ Monitoring applicatif
 - ▷ Log Management
- TP#4
TP#5
TP#6
TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
 - ▷ Les statefulsets
 - ▷ CRD et opérateurs
- TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

Les concepts de Docker



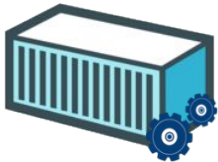
L'image

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



Le montage de répertoires

Un espace de stockage indépendant de l'image 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

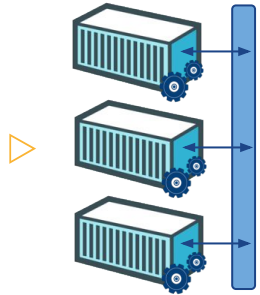


Le registre

Un service centralisé de stockage et distribution d'images

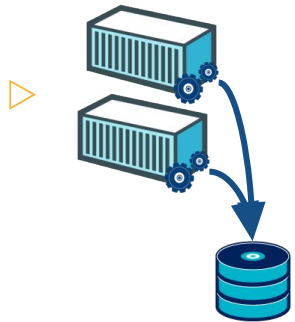


Les concepts de Docker (que vous ne verrez pas dans Kubernetes)



les networks (docker network)

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

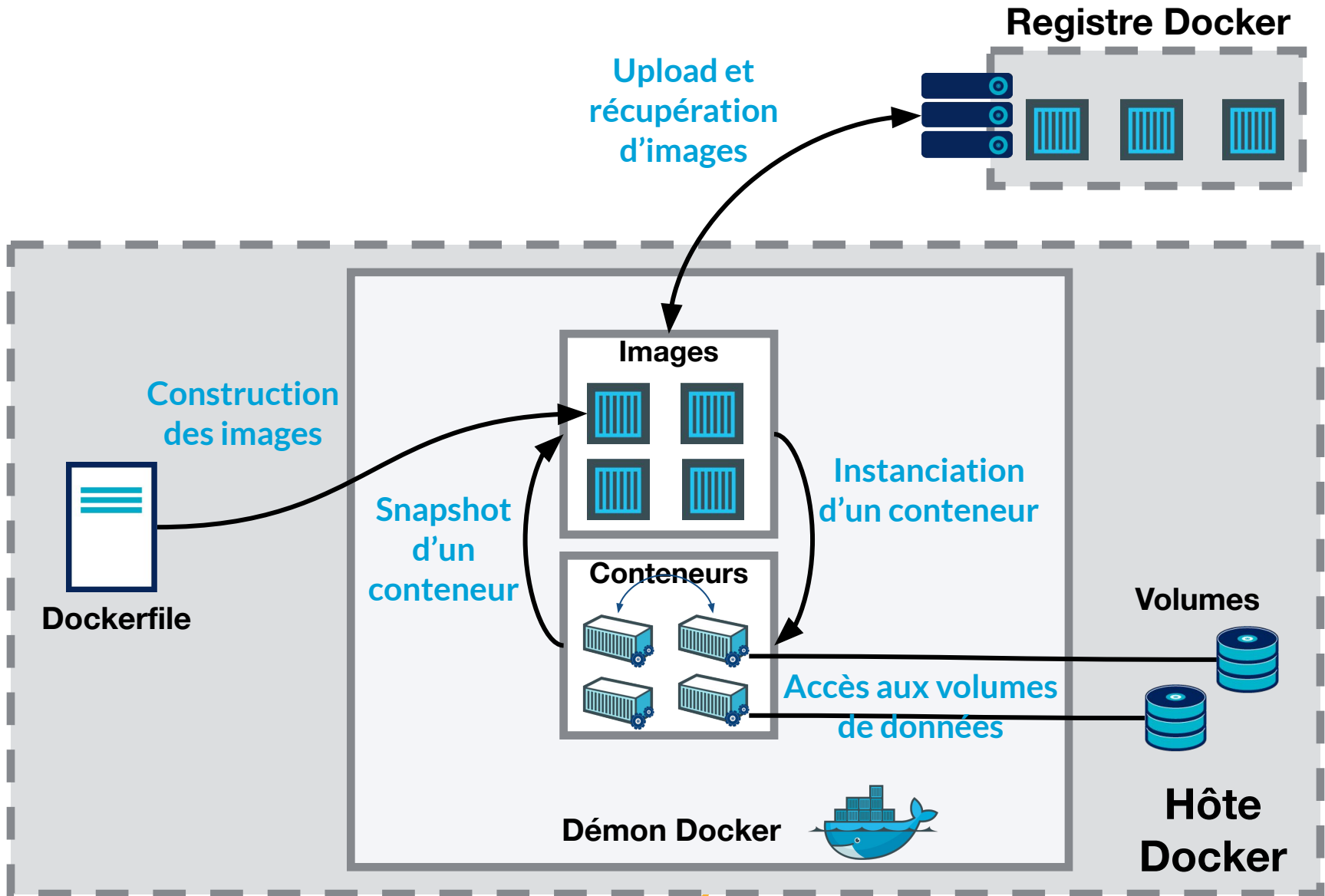


Les volumes (docker volume)

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).

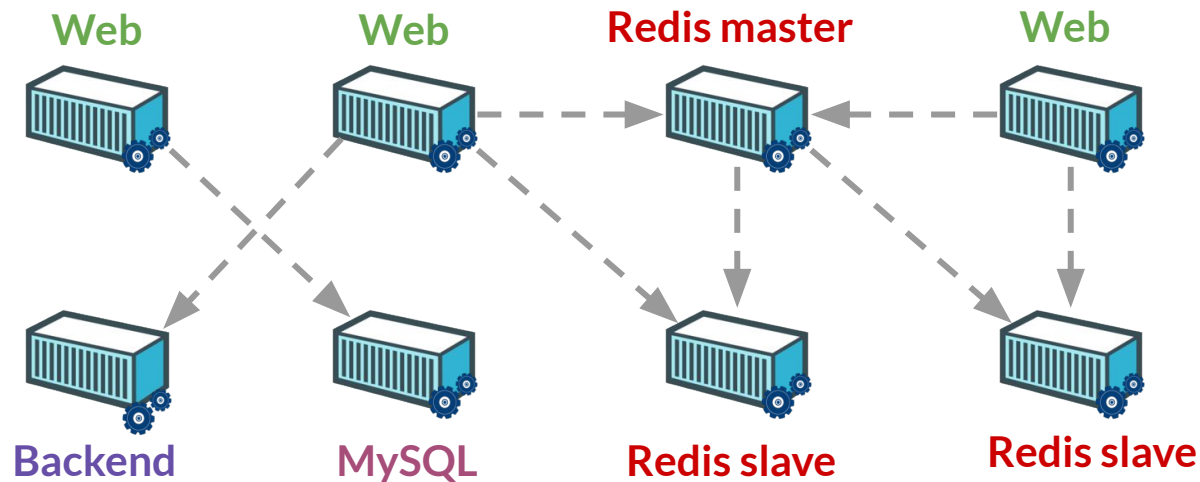


Les concepts de Docker



La philosophie Docker des conteneurs

- ▷ Un principe fort : **Un conteneur = 1 processus**
- ▷ Une brique de base à composer dans une architecture rapidement multi-conteneurs



- Le conteneur Docker **n'est pas une VM light**
- Il recentre l'infrastructure autour de l'application



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

Les images Docker

Une image Docker c'est

- ▷ **un système de fichiers autosuffisant** contenant *a minima* les 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,
 - > ...

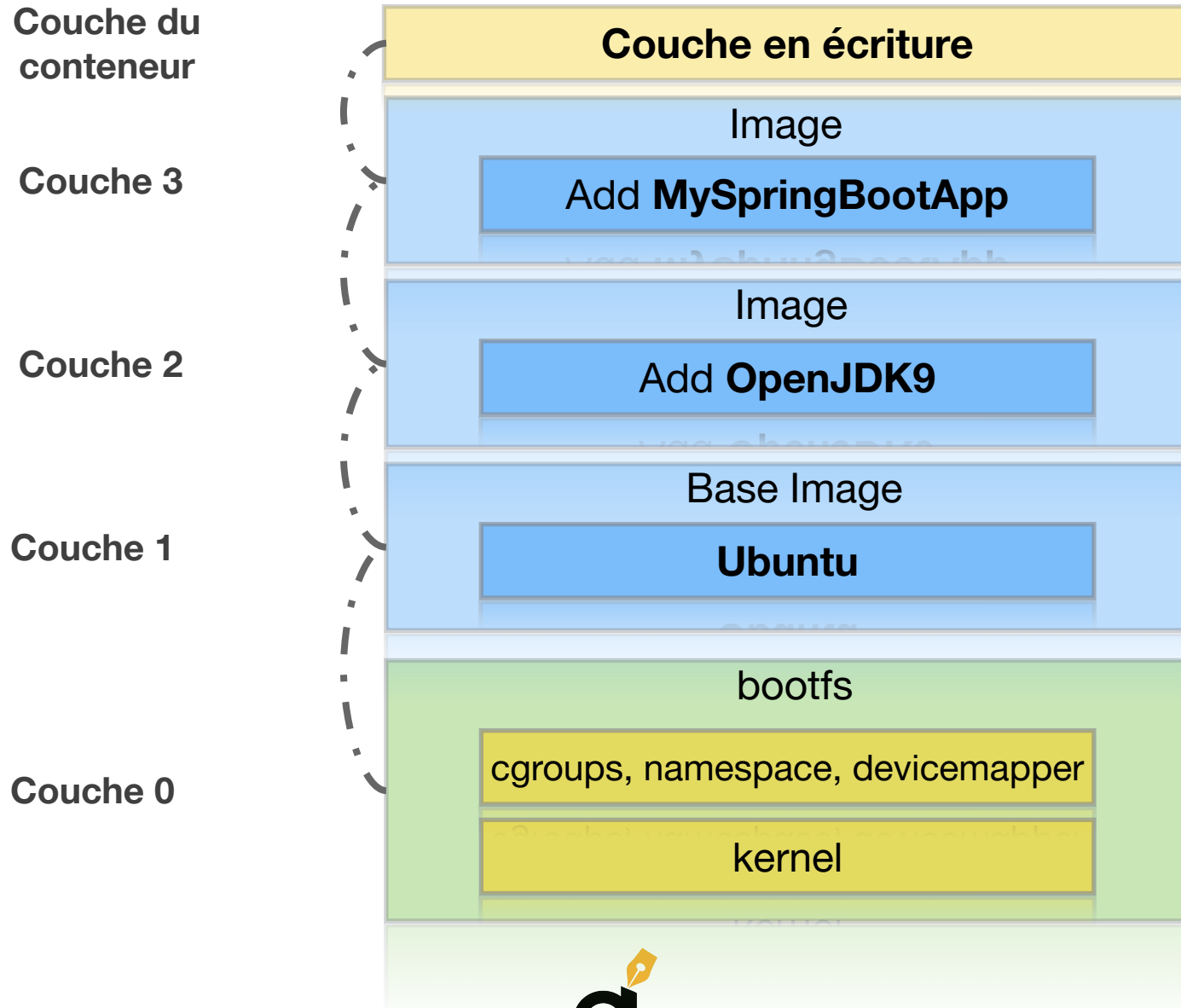


Le mille-feuille des images Docker

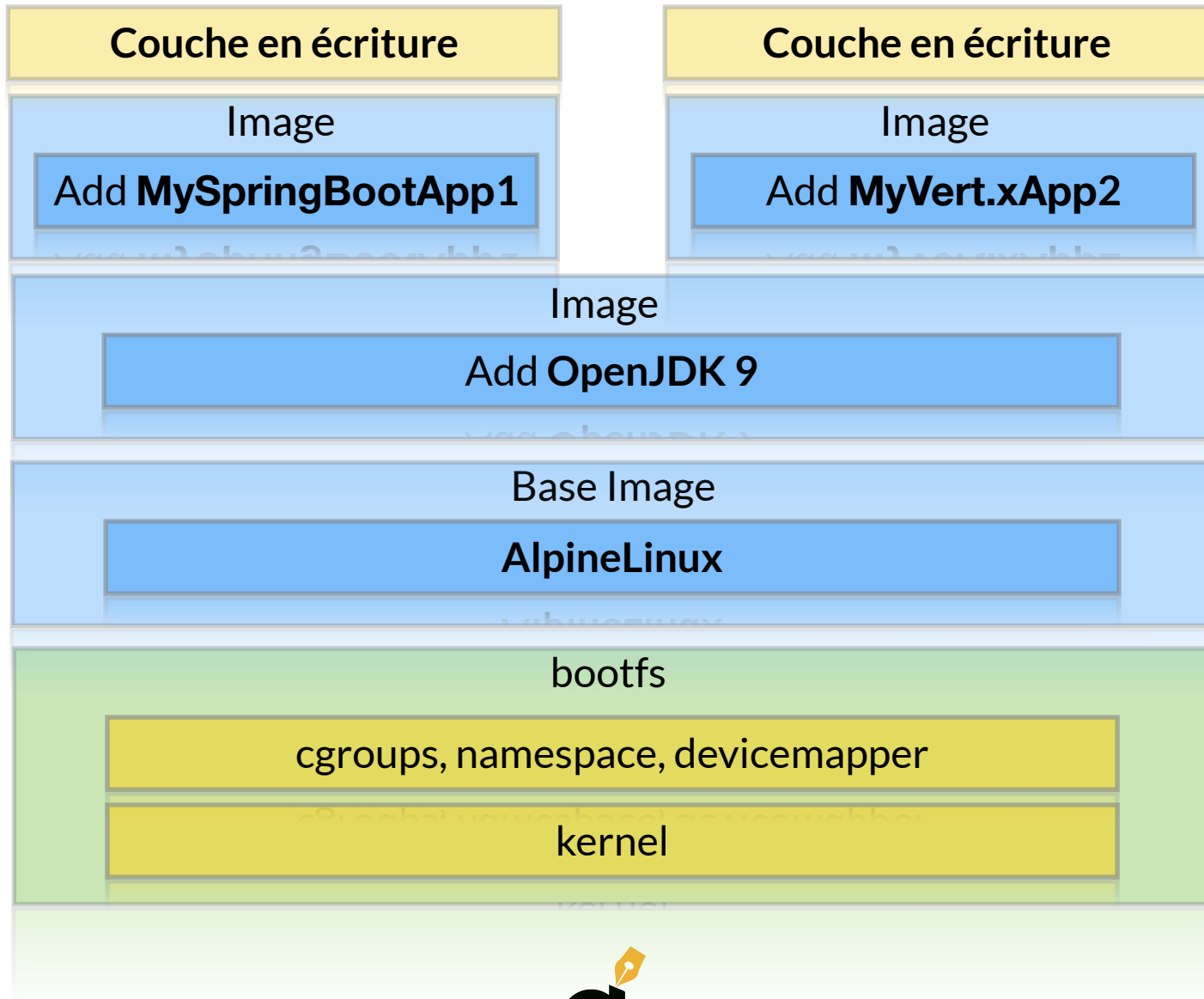
- ▷ Docker utilise un systèmes de fichiers avec un **système de couches** pour les images de conteneurs
- ▷ **Principe**
 - > Un ensemble de couches partagées en lecture seule
 - > Unifiées par le système pour simuler un unique système de fichiers à plat pour le conteneur
- ▷ **Avantages**
 - > **Évite la perte d'espace** avec $n \times 500\text{Mo}$ par OS Ubuntu dans des VMs
 - > Permet la récupération et le démarrage rapide des conteneurs



Le mille-feuille des images Docker



Le mille-feuille des images Docker



L'essentiel des commandes

Lister les images
locales

docker image ls

Télécharger une
image à partir du
Registre

docker image pull



docker image rm

Supprimer une
image

docker image tag

Labelliser une
image

docker image history

Lister les couches
d'une image

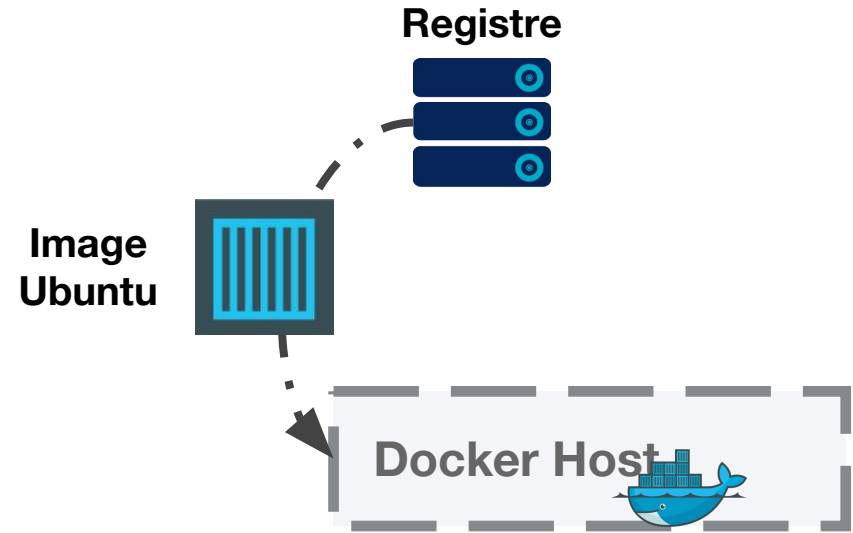


Commande Docker : *docker image pull*

Demande à Docker de récupérer localement une image de conteneurs sur un registre distant

Exemple de récupération de la dernière image d'ubuntu

```
$ docker image pull ubuntu
```



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

- *registry* indique l'url du dépôt d'images
si non précisé, va sur le dépôt par défaut (hub.docker.com)
- *repository* sert à classer les images par utilisateur/organisation
- *image* indique le nom de l'image
- et *tag* identifie l'image de manière unique dans le dépôt et vaut *latest* par défaut



Commande Docker : *docker image ls*

Interroge le registre local à l'hôte Docker et affiche la liste des images présentes localement

```
$ docker image ls
```

Exemple de sortie de la commande image

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



Commande Docker : *docker image history*

Affiche toutes les couches d'une image
et les commandes utilisées pour la créer

```
$ docker image history <image>
```

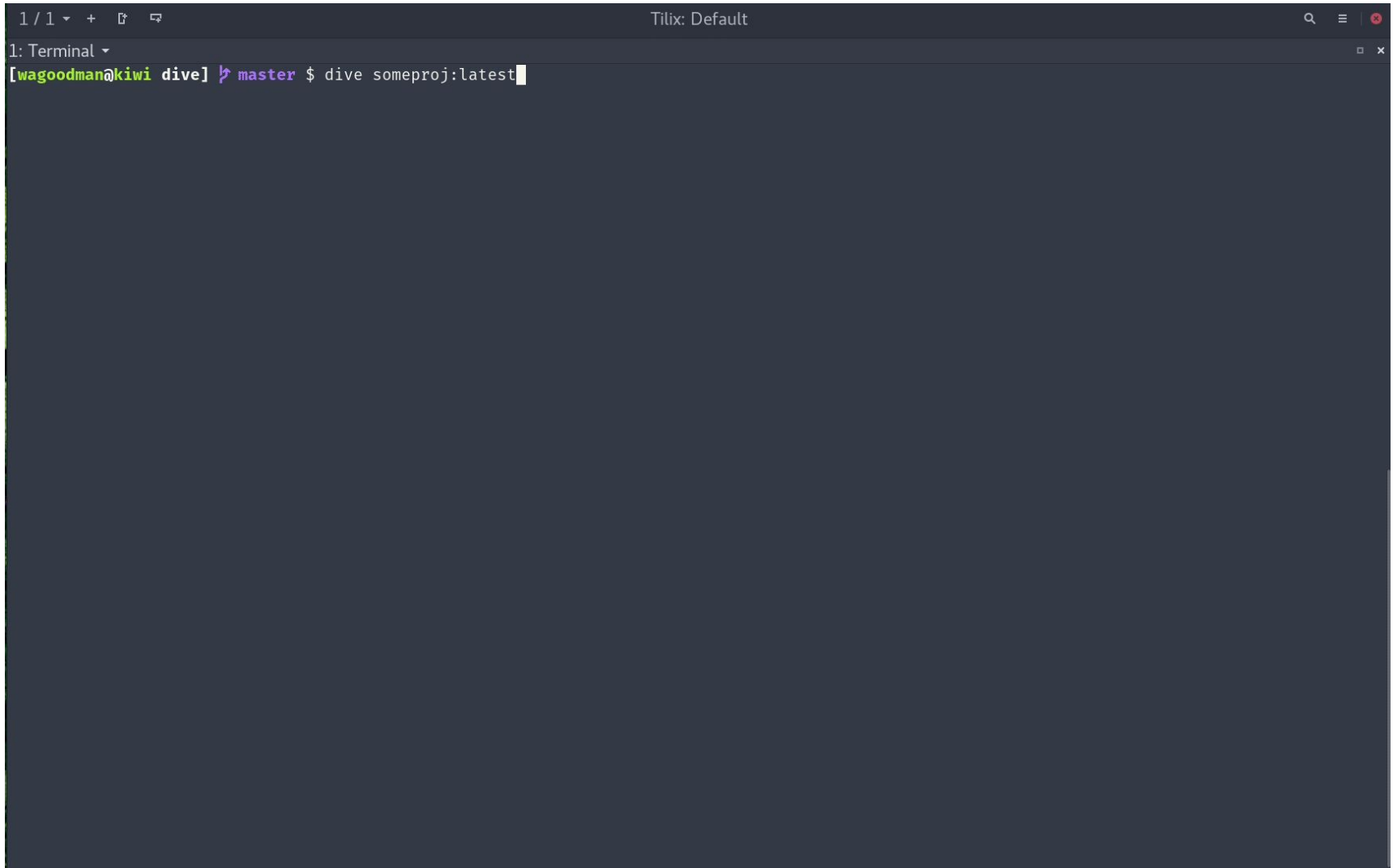
Exemple de sortie de la commande history

IMAGE	CREATED	CREATED BY	SIZE
3e23a5875458	8 days ago	/bin/sh -c #(nop) ENV LC_ALL=C.UTF-8	0 B
8578938dd170	8 days ago	/bin/sh -c dpkg-reconfigure locales && loc	1.245 MB
be51b77efb42	8 days ago	/bin/sh -c apt-get update && apt-get install	338.3 MB
4b137612be55	6 weeks ago	/bin/sh -c #(nop) ADD jessie.tar.xz in /	121 MB
750d58736b4b	6 weeks ago	/bin/sh -c #(nop) LABEL Jean Veuplut	0 B

- Chaque commande donne lieu à la création d'une couche supplémentaire associée à un identifiant
- La commande de création et la taille réelle de chaque couche peut être inspectée facilement



Un outils à connaître : “Dive”



The screenshot shows a terminal window titled "Tilix: Default". The prompt is "[wagoodman@kiwi dive]". The user has entered the command "dive someproj:latest". The terminal is currently empty, showing only the command and the cursor.

```
1 / 1 + [ ] [ ]
Tilix: Default
1: Terminal
[wagoodman@kiwi dive] ↵ master $ dive someproj:latest
```

- ▶ Pour le télécharger : <https://github.com/wagoodman/dive>



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

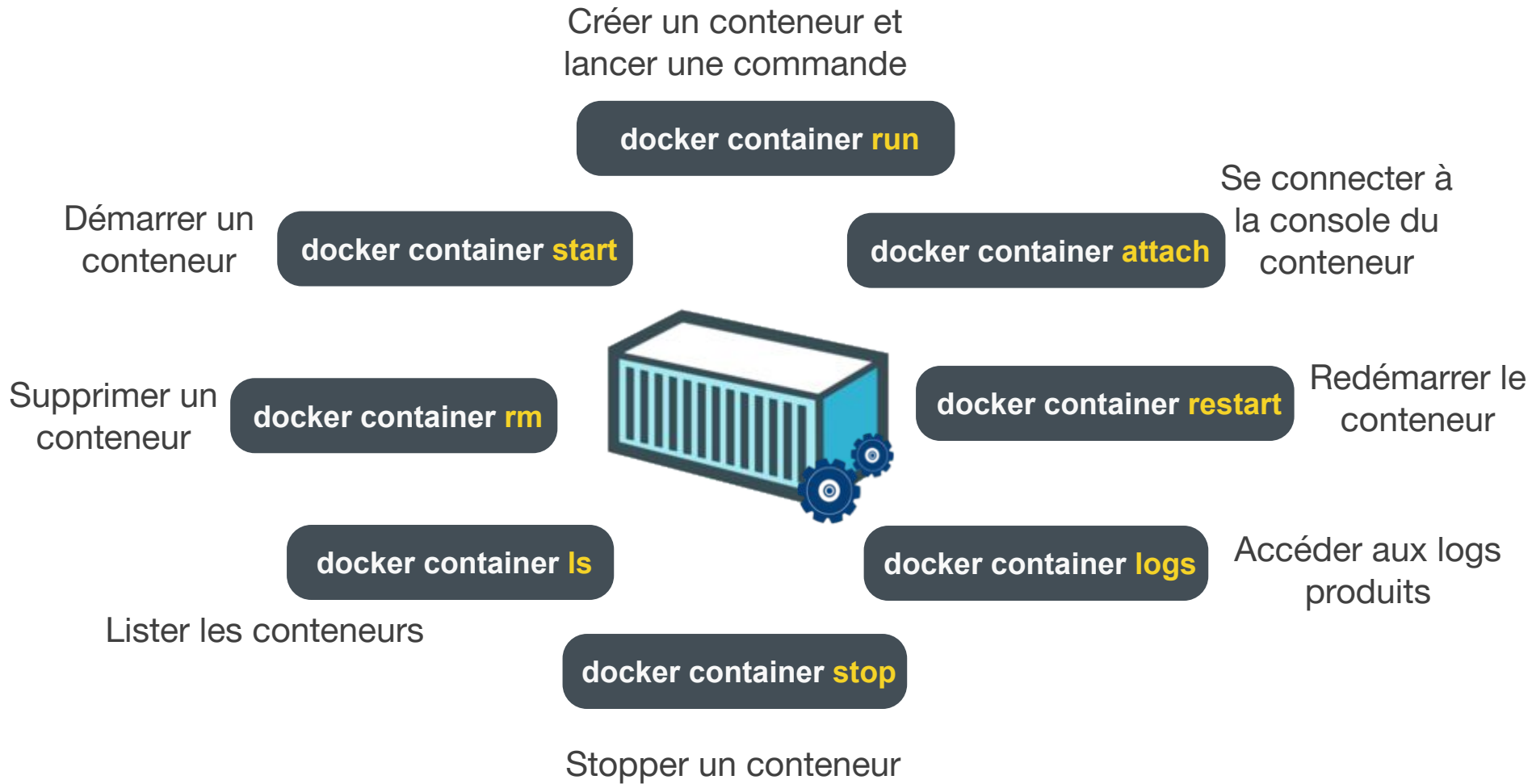
Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

L'essentiel des commandes

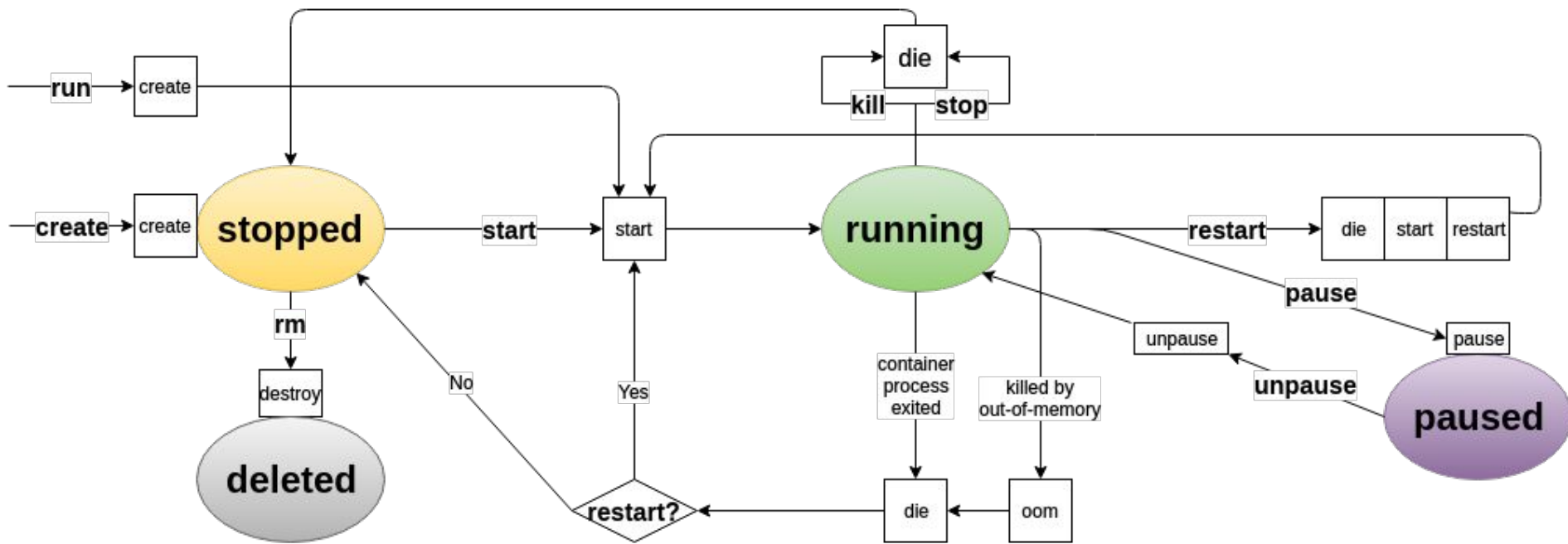


Le conteneur Docker

- ▷ Le conteneur est **la brique de base de Docker**
- ▷ Il est toujours **instancié à partir d'une image**
- ▷ Un conteneur **ne vit que pour le(s) processus** qu'il contient
- ▷ **Si ce(s) processus s'arrête(nt)**
 - > Le conteneur est stoppé
 - > Le contenu modifié subsiste tant que le conteneur n'est pas détruit
- ▷ **Une couche en écriture est créée à l'instanciation** du conteneur et supprimée à sa destruction



Cycle de vie d'un conteneur



Commande Docker : *docker container run*

Lancer 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` : lancer un conteneur
- ◆ `-i` : laisse l'entrée standard ouverte
- ◆ `-t` : assigne un terminal (tty) sur le conteneur
permet d'avoir un shell interactif
- ◆ `ubuntu` : utiliser l'image "ubuntu:latest"
- ◆ `/bin/bash` : lancer le shell bash à l'intérieur du conteneur



docker container *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 une 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) [<a
href='function.file-put-contents'>function.file-put-contents</a>]: failed to open
stream: Permission denied [...]
```

```
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 ...
```



Docker et la limitation des ressources

Un système natif de limitation des ressources

- ▷ **Configurable au lancement du conteneur** via `docker container run`
 - > à chaud pour CPU et Mémoire via `docker container update`
- ▷ **Support des limites des ressources** CPU, Mémoire et I/O Disques
- ▷ **Basé sur la technologie cgroups** (*control groups*) du noyau Linux



Docker et la limitation des ressources

Des ressources finement configurables

▷ CPU

- > quelle « part » (share) de CPU attribuer au conteneur
- > assigner un conteneur à un ou des CPUs en particulier

▷ Mémoire

- > selon le type de mémoire (applicative, noyau, swap, partagée)
- > en mode limitation *hard* ou *soft* (limitation stricte ou en cas de contention)



Docker et la limitation des ressources

▷ I/O disques

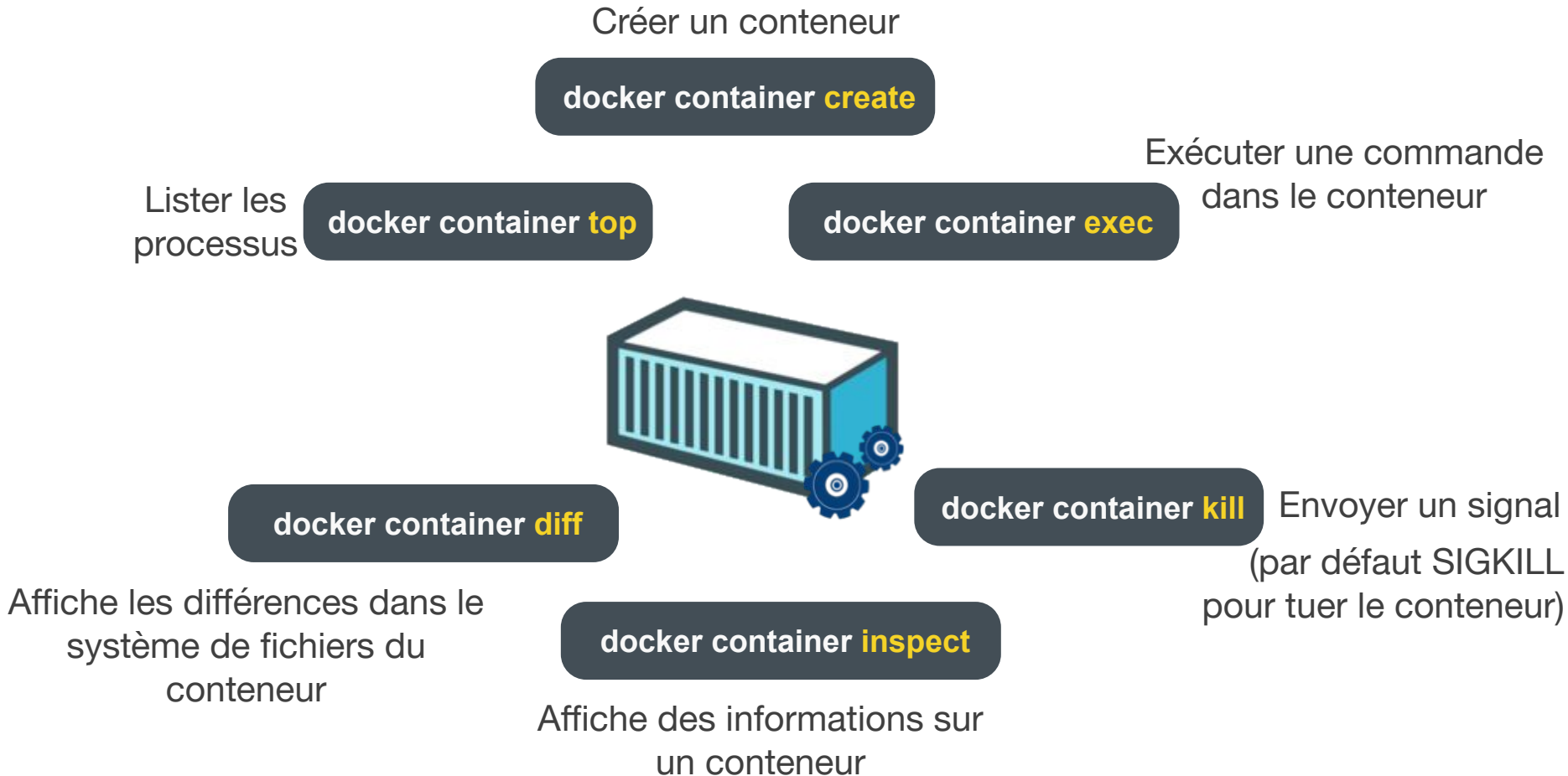
- > par poids relatif ou en absolue
- > selon le type d'accès (lectures, écriture)
- > pour le débit ou pour les entrées/sorties par seconde (IOPS)
- > pour un disque donné ou pour l'ensemble des accès

Exemple de lancement du conteneur toto avec des limites CPU, mémoire et disques

```
$ docker container run  
  --cpus=1 \  
  --memory=1g \  
  --device-write-iops=/dev/sda2:5000\  
  --device-read-bps=/dev/sda1:10mb \  
toto
```



Les commandes avancées

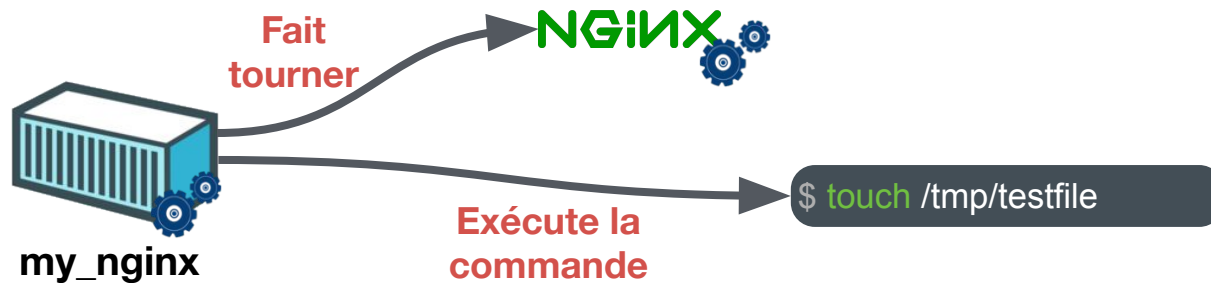


Commande Docker : *docker container exec*

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 diff*

Affiche les différences dans le système de fichiers du conteneur par rapport à l'image d'origine

```
$ docker container diff my_container
```

Exemple de sortie de la commande diff

```
C /tmp/FichierModifié  
D /tmp/FichierSupprimé  
A /tmp/FichierCréé
```

- ▷ **A** = Ajouté (*Added*) **D** = Supprimé (*Deleted*) **C** = Modifié (*Changed*)
- ▷ Très utile pour identifier rapidement les processus qui ne respectent pas les principes des conteneurs
(écriture de données dans le conteneur, utilisation d'un fichier de logs...)



Commande Docker : *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}}'



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ **Les volumes**
- ▷ Création d'images et registres
- ▷ Docker Compose TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

Gestion des données avec Docker

- ▷ Les conteneurs sont **légers et éphémères** et ne sont pas faits pour enregistrer des données de matières 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,
 - > un volume monté dans un conteneur

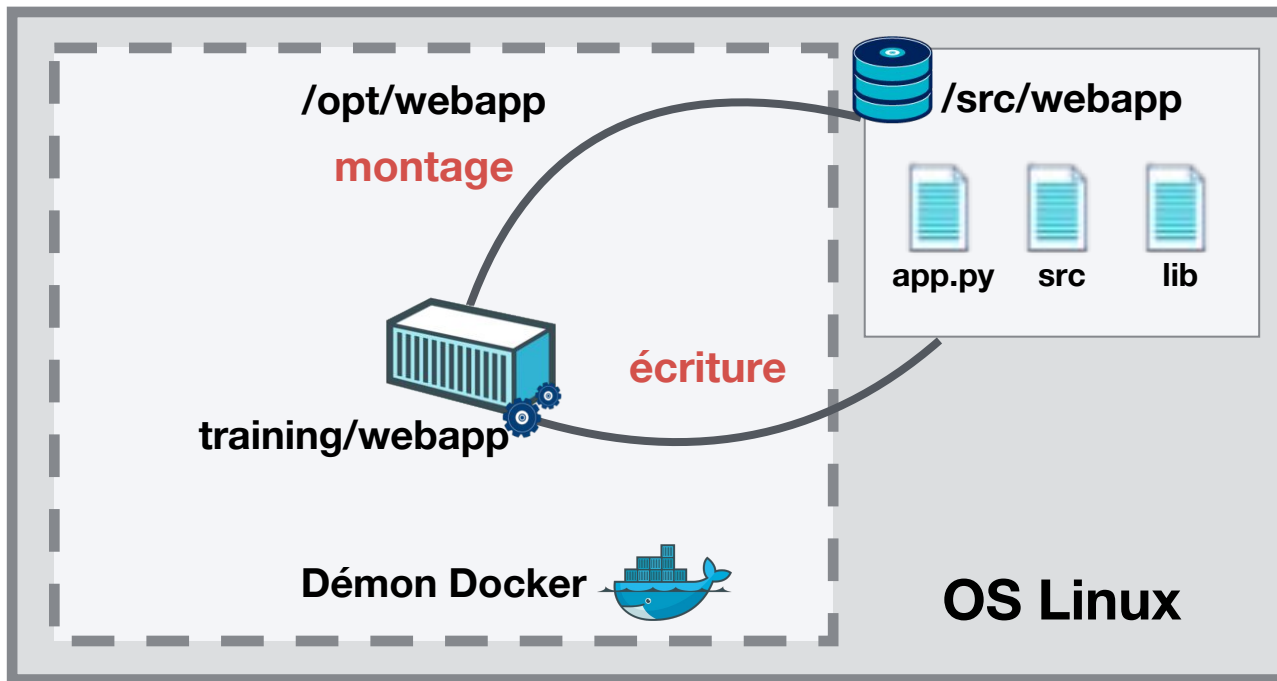


L'accès au système de fichiers de l'hôte

- 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 training/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...



L'essentiel des commandes

Créer un volume de données

`docker volume create`

Supprimer un volume

`docker volume rm`



Obtenir les informations
sur un volume

`docker volume inspect`

`docker volume ls`

Lister les volumes existants



Une gestion de volumes simplifiée

Exemples sur 2 cas d'utilisation:

- ▶ Création d'un volume de données pour MySQL

```
$ docker volume create --name mysql_data
$ docker container run -d -v mysql_data:/var/lib/mysql mysql
```

- ▶ Sauvegarde puis restauration des données de MySQL

```
$ docker volume create --name mysql_backup
$ docker container run -v mysql_data:/var/lib/mysql -v mysql_backup:/backups mysql
tar cjvf /backups/backup.tar.bz2 /var/lib/mysql
```

```
$ docker container run -v mysql_data:/var/lib/mysql -v mysql_backup:/backups mysql
bash -c 'cd /var/lib/mysql && tar xvjf /backups/backup.tar.bz2'
```



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ **Création d'images et registres**
- ▷ Docker Compose TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

Conclusion et Take Away

Le Dockerfile

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

Exemple de fichiers *Dockerfile*

```
FROM ubuntu

LABEL maintainer="Coco LASTICOT"

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

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

EXPOSE 80

CMD ["nginx"]
```



Création des images

Sauvegarder les changements d'un conteneur
dans une nouvelle image

`docker container commit`



`docker image build`

Construire une image à
l'aide d'un Dockerfile

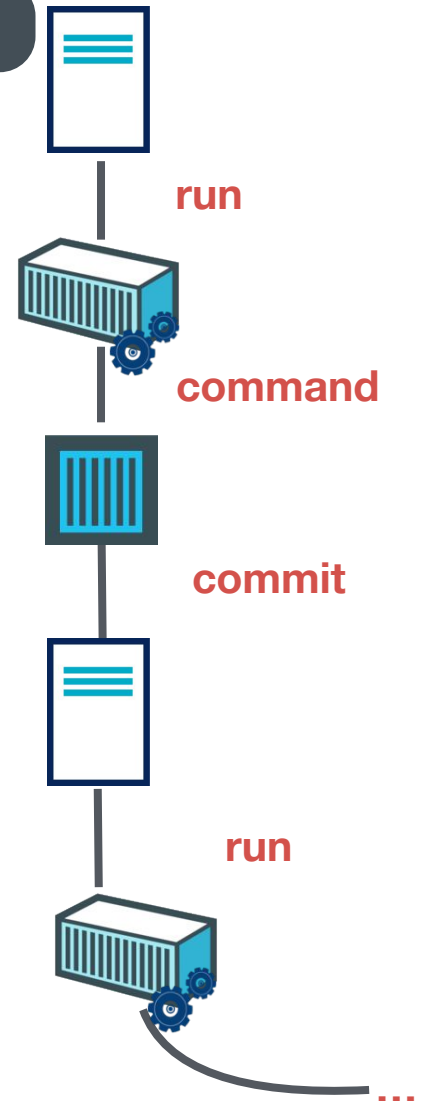


Le workflow d'un Dockerfile

```
$ docker image build -t username/myimage .
```

- ▶ Docker lance l'exécution d'un conteneur à partir d'une image
- ▶ Une instruction est exécutée et change le contenu en termes de fichiers
- ▶ Docker lance l'exécution d'un docker commit pour sauvegarder les changements de la nouvelle couche dans une image
- ▶ Docker lance un nouveau conteneur à partir de cette nouvelle image
- ▶ La prochaine instruction est exécutée... et ainsi de suite
- ▶ L'image finale est taggée "*username/myimage*"

...



Dockerfile : instruction *RUN*

Exécute une commande puis créer une nouvelle image à partir des changements

```
RUN apt-get -y install apt-utils
```



Faire une étape de purge n'a pas de sens, il vaut mieux tout inliner dans une même commande :

```
RUN apt-get -y update && apt-get install -y vim \  
&& rm -rf /var/lib/apt/lists/*
```



Dockerfile : instructions **ENTRYPOINT** et **CMD** ensemble

ENTRYPOINT `["/usr/bin/mongod"]` Exécute un container comme un exécutable, potentiellement avec des arguments

CMD `["--config", "/etc/mongodb.conf"]`

```
$ docker container run hello/mongodb
```

Run

Conteneur
MongoDB



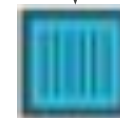
Execute command

```
# $ENTRYPOINT $CMD  
$ /usr/bin/mongod --config /etc/mongodb.conf
```

```
$ docker container run hello/mongodb --help
```

Run

Conteneur
MongoDB



Execute command

```
# $ENTRYPOINT --help  
$ /usr/bin/mongod --help
```



Dockerfile : instruction *WORKDIR*

Place le répertoire courant de travail dans le Dockerfile
Effectue un **syscall** pour changer de dossier

```
WORKDIR /etc/elasticsearch/
```

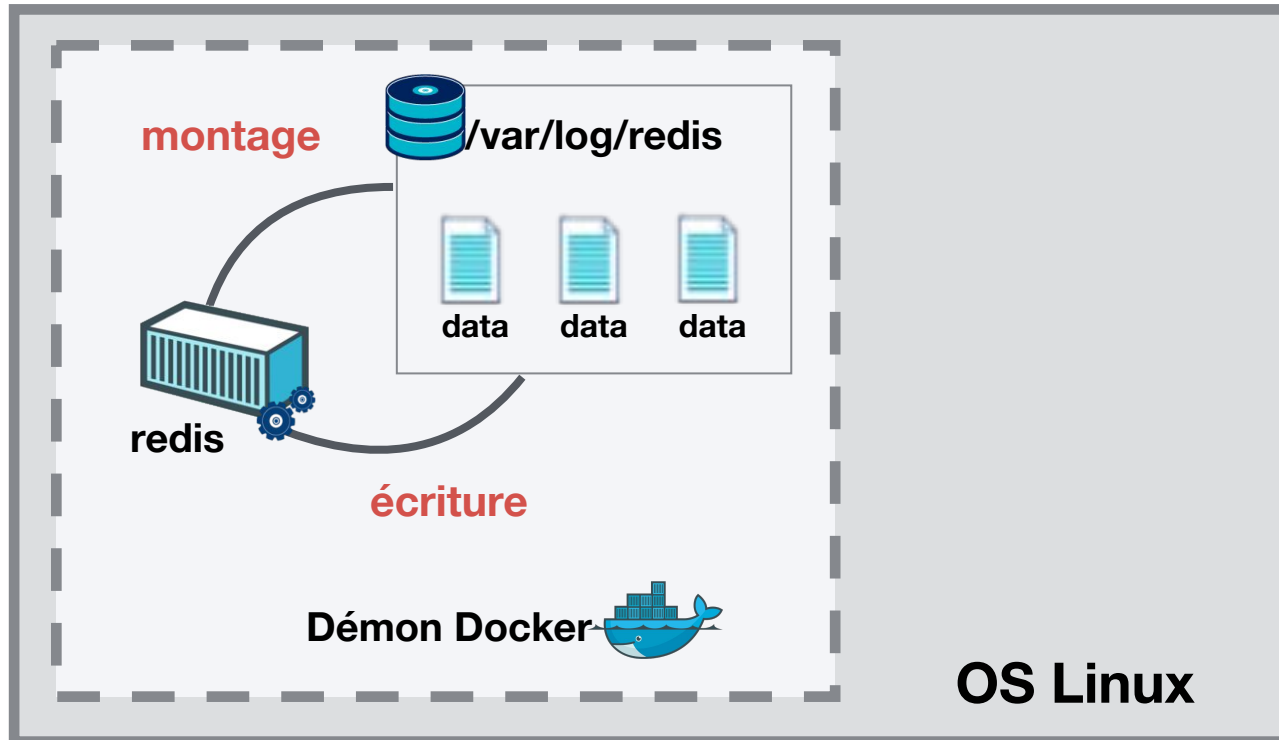
Toutes les commandes postérieur à un **WORKDIR** auront lieu dans le répertoire pointé



Dockerfile : instruction **VOLUME**

Déclare un volume secondaire au sein du conteneur pour sauvegarder des données (même effet que **docker run -v <path>**)

```
VOLUME ["/var/log/redis"]
```



Dockerfile : instruction *ENV*

Place une variable d'environnement pour toutes les commandes **RUN** qui suivent et pour la commande qui sera lancée par le conteneur

```
ENV JAVA_HOME /usr/share/java
```



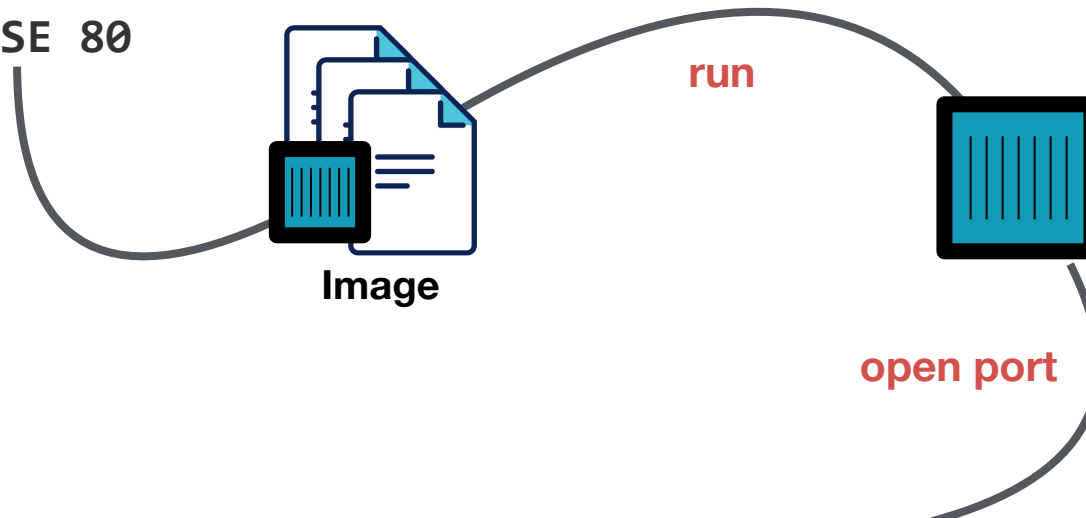
Dockerfile : instruction EXPOSE

Documente le fait que l'application écoute sur ce port

Pour pouvoir l'exposer, le préciser au lancement du container :

```
$ docker container run -d -p 8080:80 nginx
```

EXPOSE 80



```
$ iptables -A PREROUTING -t nat -i eth0 -p tcp --dport 80 -j DNAT --to 192.168.1.2:8080 ACCEPT
```



Dockerfile : instruction *COPY*

Copie un fichier ou un dossier de l'Hôte dans le conteneur au chemin spécifié

```
COPY [--chown=<user>:<group>] <src>... <dest>
```

Dockerfile : instruction *ADD*

Ajoute un fichier ou un dossier dans le conteneur au chemin spécifié.
Le fichier source peut-être une URL.
Décompresse également les archives

```
ADD [--chown=<user>:<group>] <src>... <dest>
```



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. Exemple :





Utiliser le registre Docker

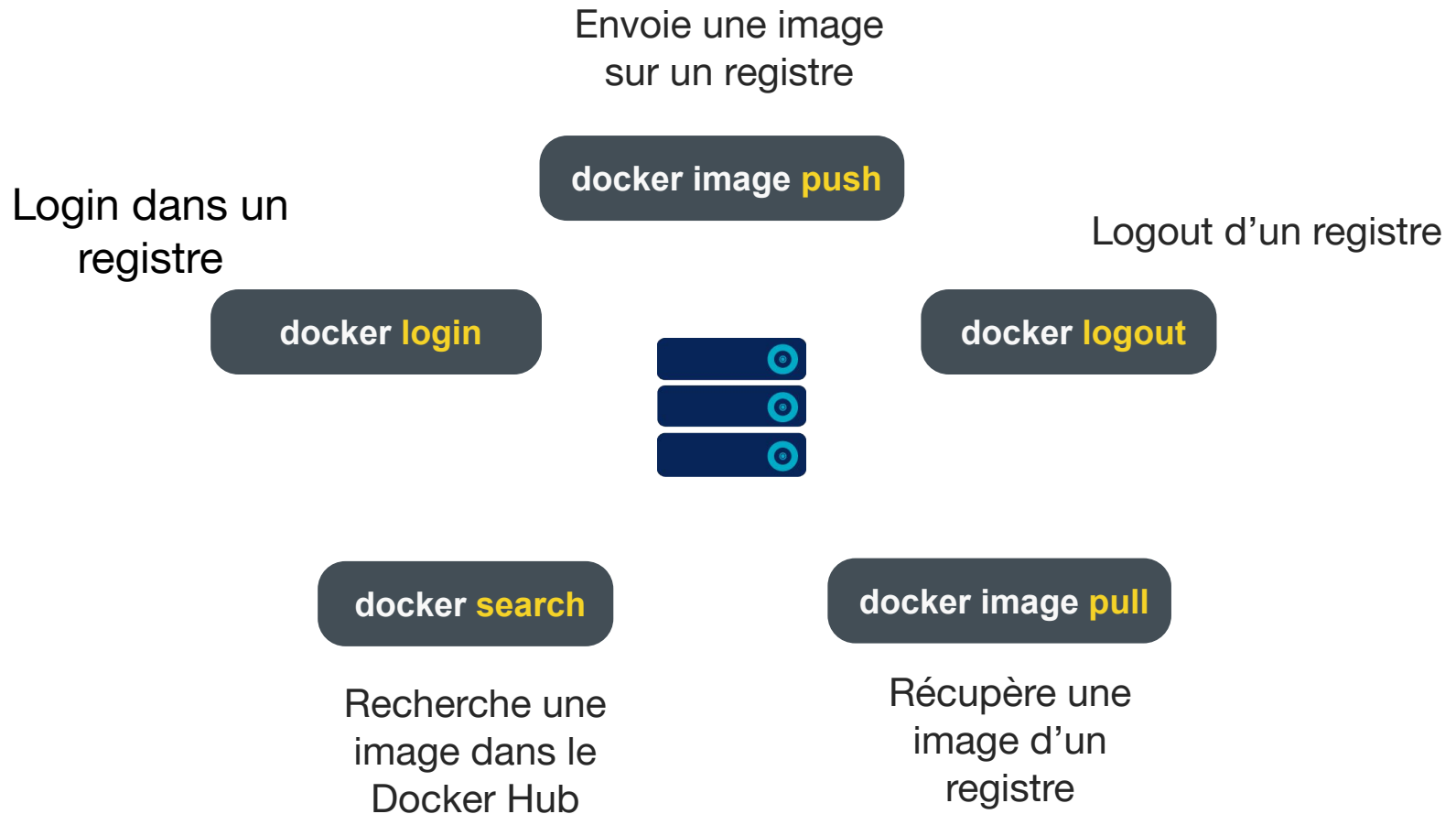


Le registre Docker

- ▷ **Un référentiel centralisé** qui stocke et rend accessible toutes les images avec leur différentes couches
- ▷ **Accessible via une API REST** connue de tous les clients Docker et formalisée dans l'OCI Distribution Specification
- ▷ **Permet le partage d'images** avec communauté ou au sein d'une entreprise
- ▷ Plusieurs implémentations existantes
 - > **Docker Hub** : registre public sur un modèle freemium
 - > **Docker Registry (aka "distribution")** : implémentation open-source
 - > **Registre chez les Cloud providers** : Instanciation à la demande de Registries privées ou publiques (ex: AWS ECR, GCP, Azure ACR)
 - > **Nexus, Artifactory** peuvent également offrir le service de registre Docker
 - > **Gitlab, Github** : un registre Docker pour chaque dépôt de code
 - > **Portus, Harbor** : registres open-source spécialisés



L'essentiel des commandes



Le registre officiel Docker - Docker Hub

- ▷ **Service de registre de Docker en mode SaaS**
- ▷ **Utilisé pour la distribution des images officielles**
(ubuntu, nodejs, ruby, openjdk, gitlab...)
- ▷ **Ouvert aux entreprises et aux utilisateurs**
Gratuit pour un unique dépôt privé (image en plusieurs versions) et pour un nombre illimité de dépôts publics
- ▷ **Quota** en place : **100 pulls/6 heures/IP** pour les requêtes anonymes
- ▷ Les fonctionnalités clés
 - > **Automated builds** : construction automatique des images en cas de mise à jour d'un dépôt de source
 - > **Webhooks** : lancement automatique d'action en cas de changements sur une image
 - > **Dépôts privés** : dépôts non accessibles sans autorisation



Savoir rechercher une image - CLI

```
$ docker search java
```


Exemple de sortie de docker search

NAME	DESCRIPTION	STARS	OFFICIAL	AUTOMATED
node	Node.js is a JavaScript-based platform for...	1465	[OK]	
java	Java is a concurrent, class-based, and obj...	555	[OK]	
develar/java		24		[OK]
anapsix/alpine-java	Oracle Java8 with GLIBC 2.21 over AlpineLinux	17		[OK]
isuper/java-oracle	This repository contains all java releases...	16		[OK]
netflixoss/java	Java Base for NetflixOSS container images	8		[OK]
nimmis/java-centos	This is docker images of CentOS 7 with dif...	7		[OK]
maxexcloo/java	Docker framework container with the Oracle...	7		[OK]
nimmis/java	This is docker images of Ubuntu 14.04 LTS ...	6		[OK]
1science/java	Java Docker images based on Alpine Linux	5		[OK]
andreluiznsilva/java	Docker images for java applications	5		[OK]
lwieske/java-8	Oracle Java 8 Container	5		[OK]
aerath/java	Ubuntu with latest oracle java jdk, git, a...	1		[OK]
dwolla/java	Dwolla's custom Java image	1		[OK]





- ▶ **Stars** : nombre de votes par la communauté
- ▶ **Official** : dépôt dont la qualité a été vérifiée par Docker Inc et qui est mis à jour régulièrement
- ▶ **Automated** : image construite par Docker Hub à partir d'un processus automatique et vérifiable



Savoir rechercher une image - Interface web



Explore Sign In Pricing [Get Started](#)

 Docker EE  Docker CE  Containers  Plugins

Filters


Docker Certified ⓘ
☐ ☒ Docker Certified

Images
☐ Verified Publisher ⓘ
Docker Certified And Verified Publisher Content
☐ Official Images ⓘ
Official Images Published By Docker

Categories ⓘ
☐ Analytics
☐ Application Frameworks
☐ Application Infrastructure
☐ Application Services
☐ Base Images
☐ Databases
☐ DevOps Tools

1 - 25 of 31 598 results for **alpine**. [Clear search](#)

Most Popular


 **alpine**
Updated 30 minutes ago

OFFICIAL IMAGE ⓘ

10M+ 5.6K
Downloads Stars

A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!


Container Linux PowerPC 64 LE ARM 64 IBM Z 386 ARM x86-64 Featured Images Base Images
Operating Systems

 **alpine/git**
By [alpine](#) • Updated 7 months ago

10M+ 98
Downloads Stars

A simple git container running in alpine linux, especially for tiny linux distro.

Container Linux x86-64

 **alpine/socat**
By [alpine](#) • Updated 19 days ago

5M+ 36
Downloads Stars



Registre et workflow de développement

- ▷ **Le registre est le pivot central de Docker** de toute utilisation industrialisée de Docker
- ▷ Devient **un référentiel dev/ops des images à déployer**
- ▷ **Nécessite d'être intégré dans les workflows de développement**

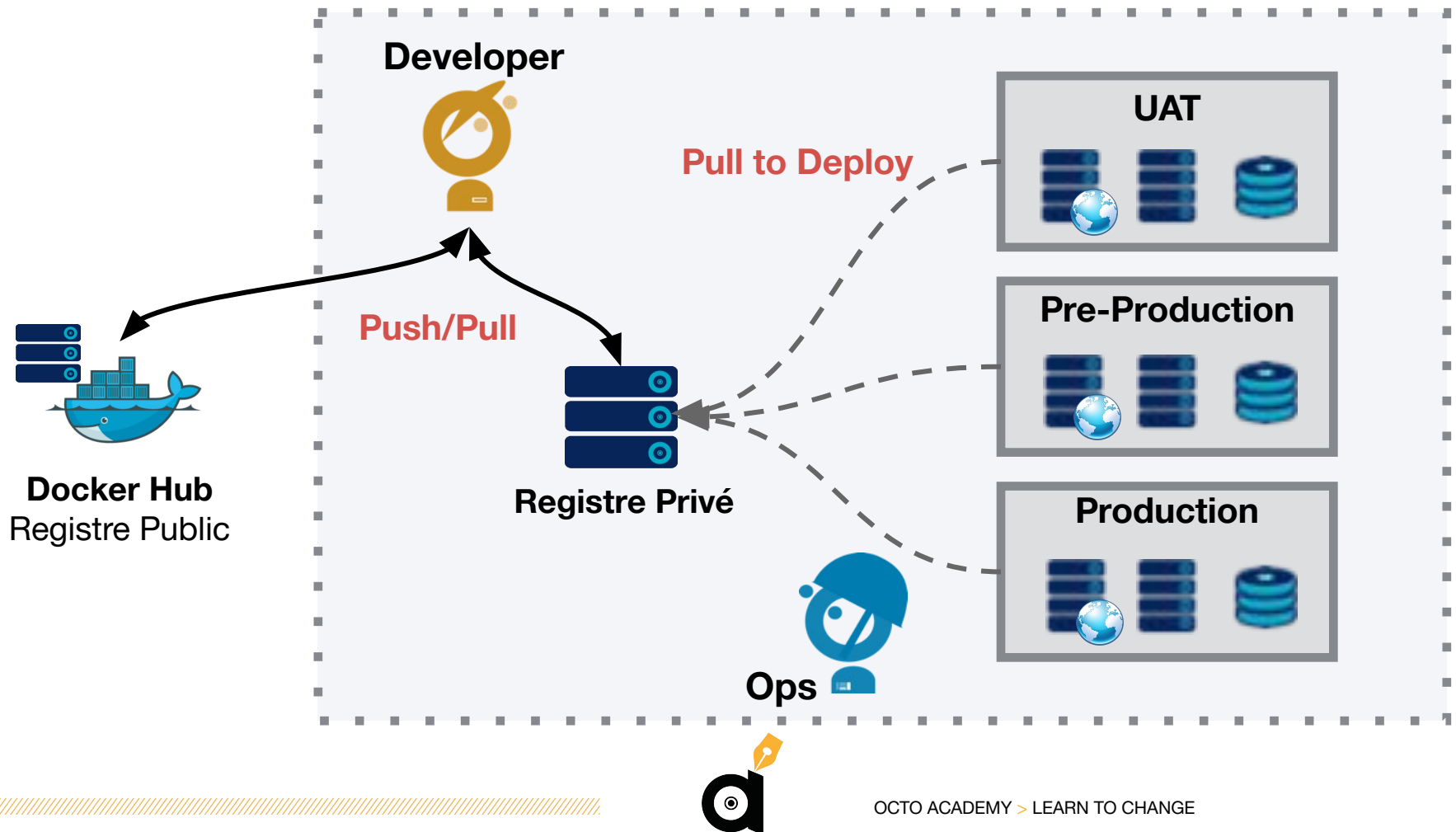
Exemple de *workflow* de développement simplifié

- > Chargement d'une image de base (Pull)
- > Développement en local
- > Packaging de l'application dans une image (Build)
- > Upload de l'image sur le Registre (Push)
- > Téléchargement de l'image pour le déploiement sur les serveurs (Pull)



Registre et workflow de développement

- **Collaboration Dev/Ops** dans le cadre d'un workflow de développement avec Docker



AGENDA

Les bases de Docker

- ▷ Introduction : l'avant Docker
- ▷ Qu'est ce que Docker TP#1
- ▷ Architecture et concepts Docker

Docker en pratique

- ▷ Les images Docker
- ▷ Utilisation de Docker
- ▷ Les volumes
- ▷ Création d'images et registres
- ▷ Docker Compose TP#2

Les bases de Kubernetes

- ▷ Introduction & historique de K8s
- ▷ Utilisation du client kubectl TP#3

Manipulation simple de Kubernetes

- ▷ Concepts de base de Kubernetes

Mettre son application en prod dans K8s

- ▷ Secrets et Configmaps TP#4
- ▷ Liveness et Readiness
- ▷ Routes HTTP TP#5
- ▷ Maîtrise des capacités
- ▷ Monitoring applicatif TP#6
- ▷ Log Management TP#7

Gestion des conteneurs à état

- ▷ Les volumes, PV et PVC
- ▷ Les statefulsets
- ▷ CRD et opérateurs TP#8

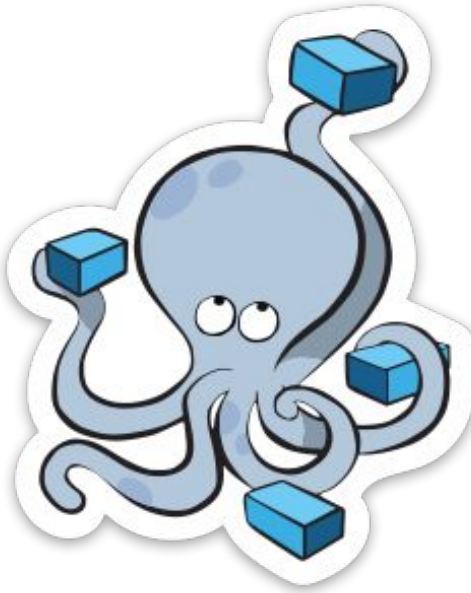
Le Continuous Delivery avec Kubernetes

- ▷ Exemples de Continuous Integration
- ▷ Exemples de Continuous Deployment

Eco-conception

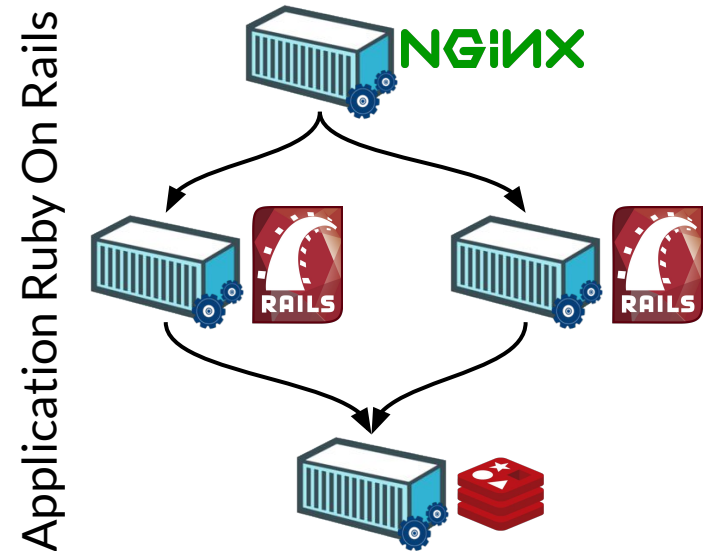
Conclusion et Take Away

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 plugin à Docker CLI 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 Python en dehors de Docker CLI sous le nom `$ fig`. Acquis par Docker Inc en 2014 il est alors renommé en `$ docker-compose`
- ▷ V2 sortie en 2022, iso-fonctionnelle complètement ré-implémentée en Go
- ▷ Permet notamment
 - > le partage facile d'un **environnement de développement** complet
 - > le lancement de **multiples versions d'un environnement**
 - > **raccourcir** des commandes Docker fréquentes
 - > **le scaling (manuel)** des conteneurs stateless



Le format de description

 `docker-compose.yml`

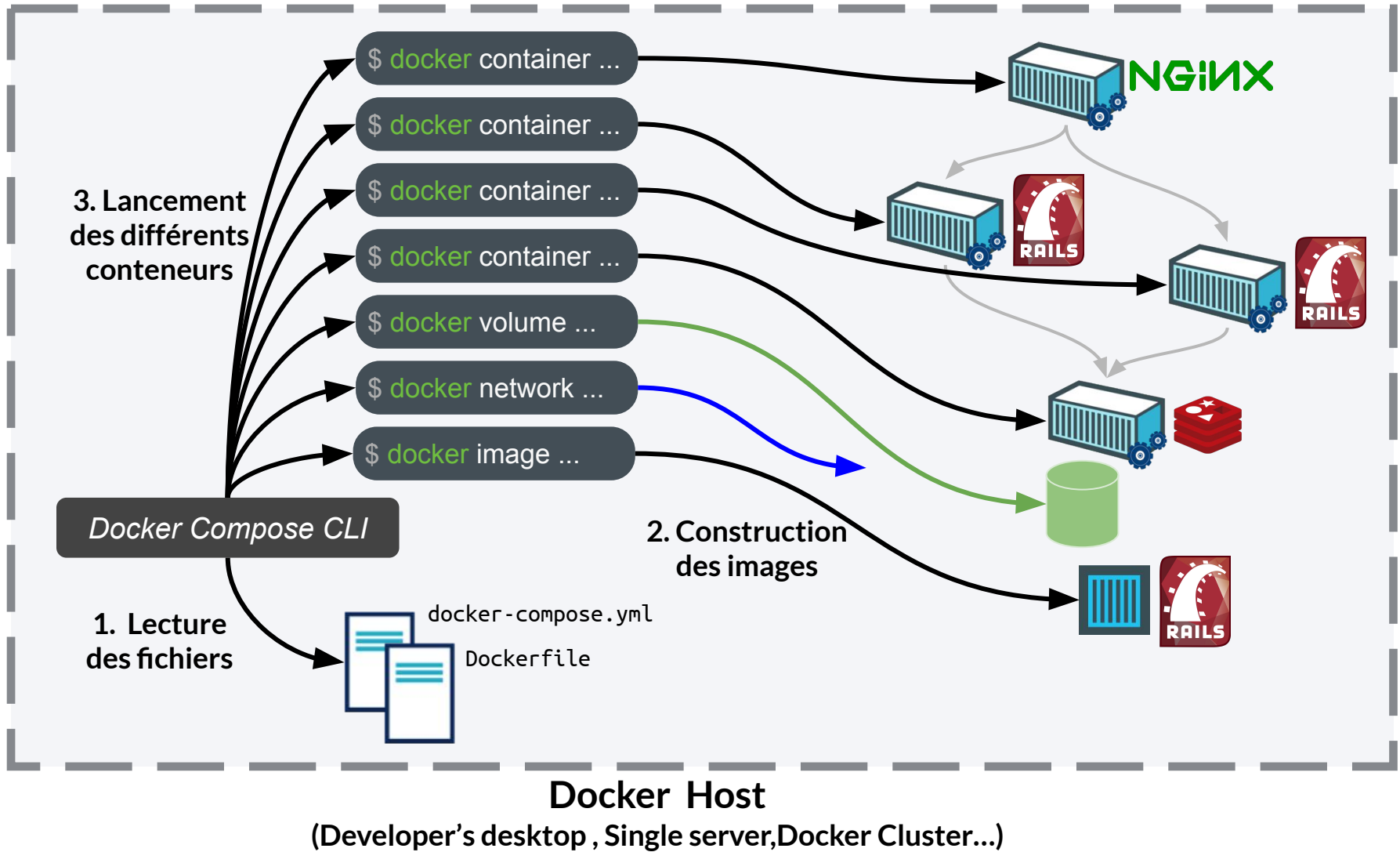
```
version: "3"
services:
  web:
    build: .
    command: python app.py
    ports:
      - "5000:5000"
    volumes:
      - ./code
    depends_on:
      - redis
    networks:
      - mynet
  redis:
    image: orchardup/redis
    volumes:
      - redis-data:/var/lib/redis
    networks:
      - mynet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

Déclaration de chaque
conteneur : créé un alias
DNS

Instructions
pour construire ou
récupérer l'image



Fonctionnement de Docker Compose



L'essentiel des commandes

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

docker compose up [-d]

Accéder aux logs des
conteneurs

docker compose logs

Stoppe/Démarre/Redémarre les services

docker compose stop|start|restart

Pousse les
images
buildées
localement

docker compose push

Supprimer les
conteneurs, réseaux, volumes

docker compose down

Exécuter une
commande à l'intérieur
d'un conteneur

docker compose exec

docker compose ps

Lister les conteneurs
lancés

docker compose scale

Ajouter des instances d'un conteneur

docker compose build

Construit les images



Les limitations de Docker Compose

- ▷ **Pas de démon**, la surveillance de la santé des conteneurs n'est faite qu'au lancement d'une commande *up* ou *run*
- ▷ Le **scaling des conteneurs** ne peut être fait qu'*a posteriori*
- ▷ Une logique d'orchestration de déploiement simpliste et gérée en dehors de la plateforme de gestion des conteneurs : c'est le poste client qui orchestre !



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



TP #2