ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

docker

Docker Introduction

Pierre COURBIN
pierre.courbin@devinci.fr

L'ESILV ou l'auteur n'ont aucun lien d'intérêt avec l'entreprise Docker

Références et sources

Si vous repérez une erreur dans les affirmations suivantes, n'hésitez pas à me l'indiquer en commentaire de ce Deck !

- Sur chaque image, vous trouverez le lien vers sa source
- Suite à chaque citation, vous trouverez le lien vers sa source
- De façon générale, je n'ai aucun lien avec les différentes sources listées, ce sera sinon indiqué clairement.
  J'ai parfois simplement trouvé leur image particulièrement pertinente...

Quelques références spécifiques :

- La plupart du contenu est basé sur le site Docker.com
- Je me suis aussi inspiré de plusieurs autres présentations, notamment :
  - Présentation de Rama Krishna B, Openstack Architect at Hewlett Packard Enterprise
  - L'excellent Grafikart, qui m'a aussi fait découvrir Slides.com
  - Une présentation de rmkpatchaa

À propos de ce cours

Ce cours est donné sous forme de MasterClass ouvert à tous ou intégré dans certains cours plus institutionnels.

À propos de l'enseignant

Informations professionnelle sur LinkedIn et sur l'aspect recherche sur ResearchGate.

À propos de l'ESILV

L'ESILV, Ecole Supérieure d'Ingénieurs Léonard de Vinci est une école d'ingénieurs généraliste au cœur des technologies du numérique.

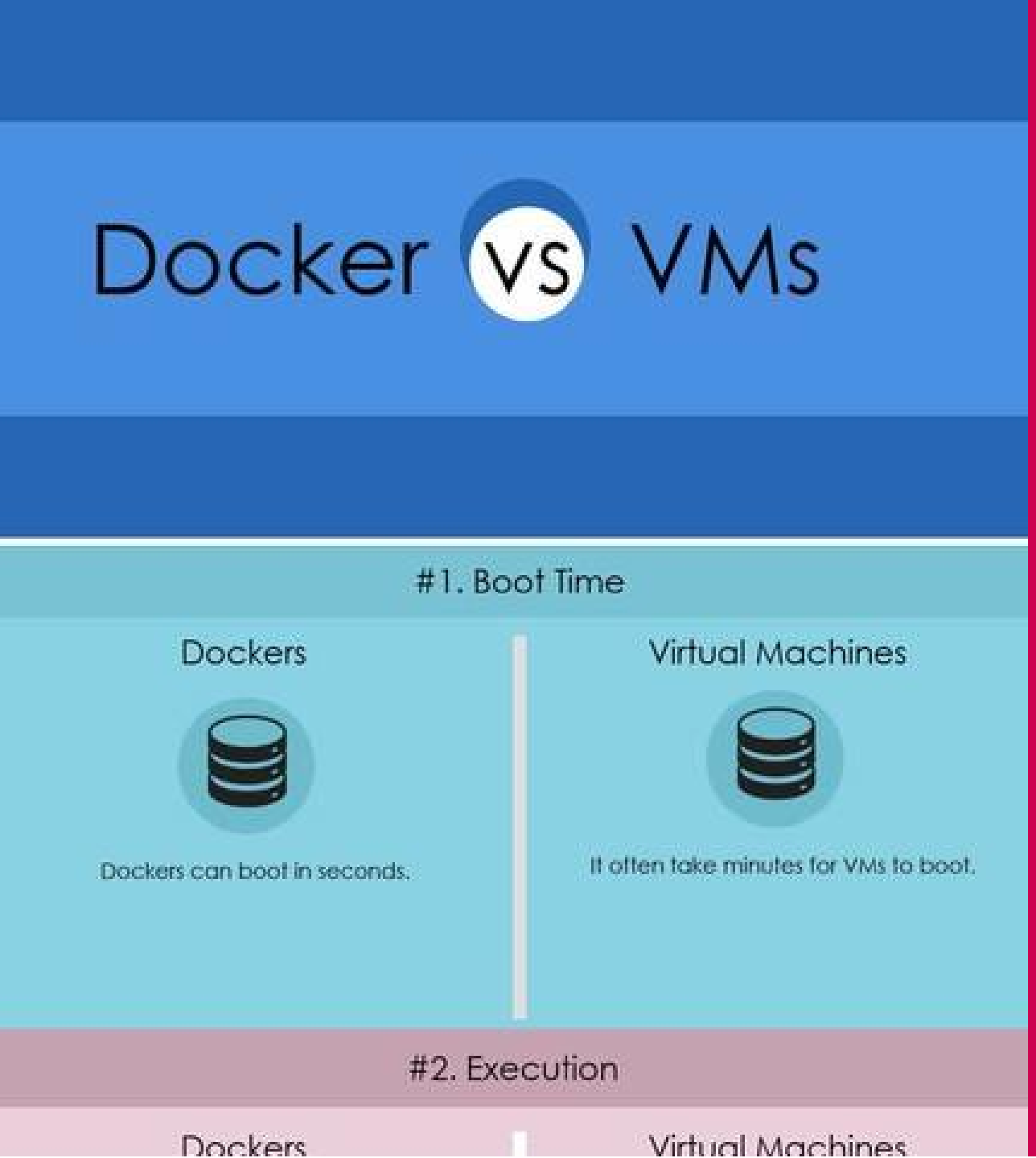
Elle recrute principalement au niveau Baccalauréat (S et STI2D) et forme en 5 ans des ingénieurs opérationnels s'insérant parfaitement dans le monde professionnel.

Le projet pédagogique de l'ESILV s'articule autour **des sciences et des technologies numériques** combinées à 5 thématiques (Informatique, Finance, Mécanique, Energie/Ville, Santé) développées dans **9 grandes spécialisations** : IOCS (Informatique, objets connectés et sécurité), DIA (Data et intelligence artificielle), IF (Ingénierie financière), ACT (Actuariat), FIN (Fintech), MMN (Modélisation et mécanique numérique), IND (Industrie 4.0), EVD (Énergie et villes durables), SB (Santé biotech).

L'ESILV s'appuie aussi sur une **approche pédagogique transverse** avec 20% de son cursus en commun avec une école de management (EMLV) et une école du digital (IIM) dont un parcours Ingénieur Manager en 5 ans, double diplômant.

Elle propose également un Bachelor Ingénierie Numérique et un Bachelor Technology & Management.

2600 élèves. Labellisée EESPIG, l'ESILV est membre de la CGE, de l'UGEI, de la CDEFI, de Campus France, de Talents du Numérique et de LearningLab Network.



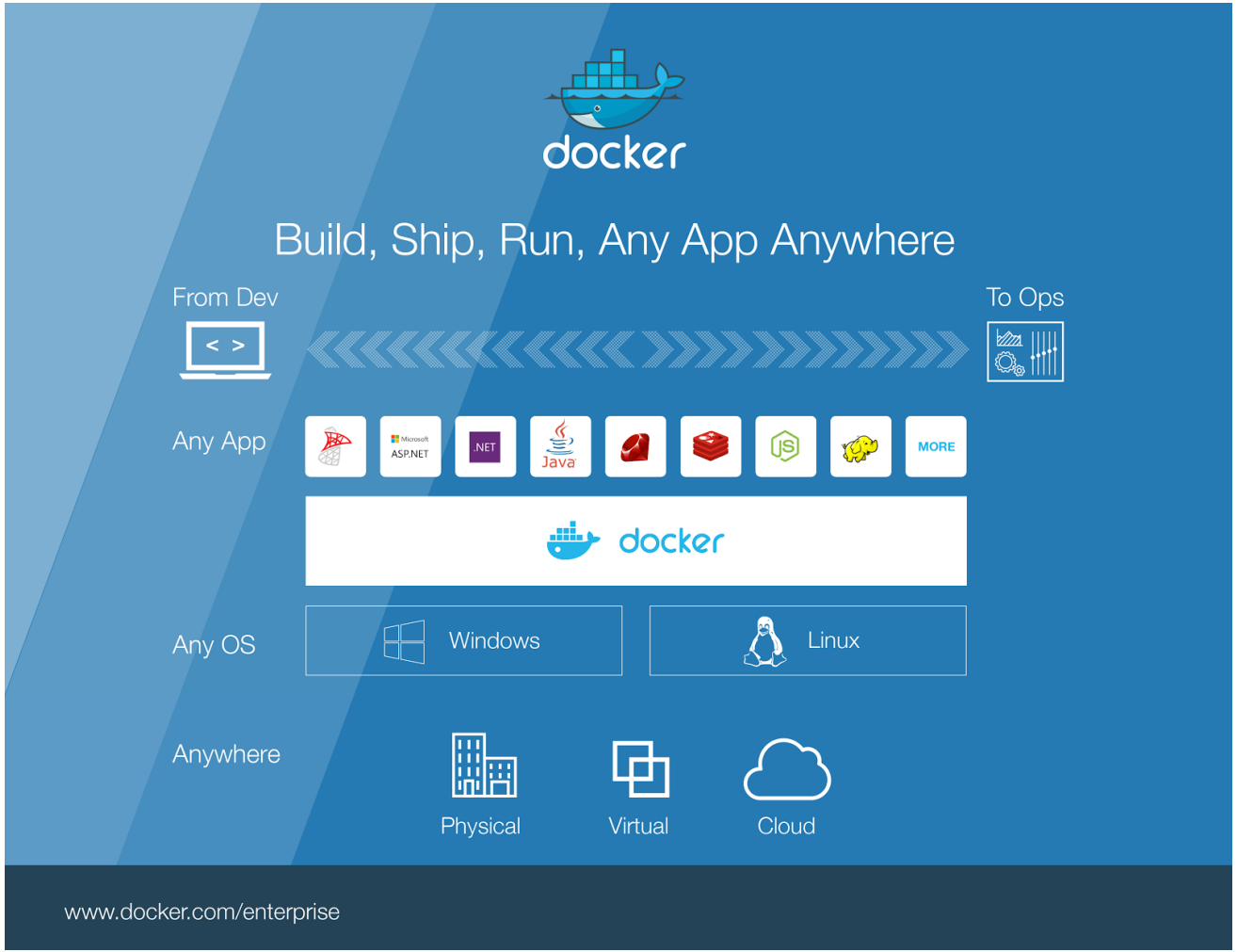
ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. **Introduction générale**
   Conteneurs et VMs
2. Utilisation basique
   docker run
3. Utilisation simplifiée
   docker-compose
4. Utilisation avancée
   (logs, données, réseaux, ressources, etc)
5. Création d'image
   (Commit et Dockerfile)
6. Aller plus loin
   (Accéder à des devices, l'orchestration etc)

Conteneurs et VMs

Quelle(s) promesse(s) ?



Conteneurs et VMs

Définitions

Virtualisation

La **virtualisation** consiste à exécuter sur une machine hôte dans un environnement isolé des systèmes d'exploitation — on parle alors de **virtualisation système** — ou des applications — on parle alors de **virtualisation applicative**. Ces ordinateurs virtuels sont appelés serveur privé virtuel (*Virtual Private Server* ou VPS) ou encore environnement virtuel (*Virtual Environment* ou VE).

Machine Virtuelle

En informatique, une **machine virtuelle** (anglais *virtual machine*, abr. *VM*) est une illusion d'un appareil informatique créée par un logiciel d'émulation. Le logiciel d'émulation simule la présence de ressources matérielles et logicielles telles que la mémoire, le processeur, le disque dur, voire le système d'exploitation et les pilotes, permettant d'exécuter des programmes dans les mêmes conditions que celles de la machine simulée.

Hyperviseur

En informatique, un **hyperviseur** est une plate-forme de virtualisation qui permet à plusieurs systèmes d'exploitation de travailler sur une même machine physique en même temps.

Ex : VMWare, Microsoft Hyper-V, Oracle VirtualBox etc

Pourquoi Virtualiser ?

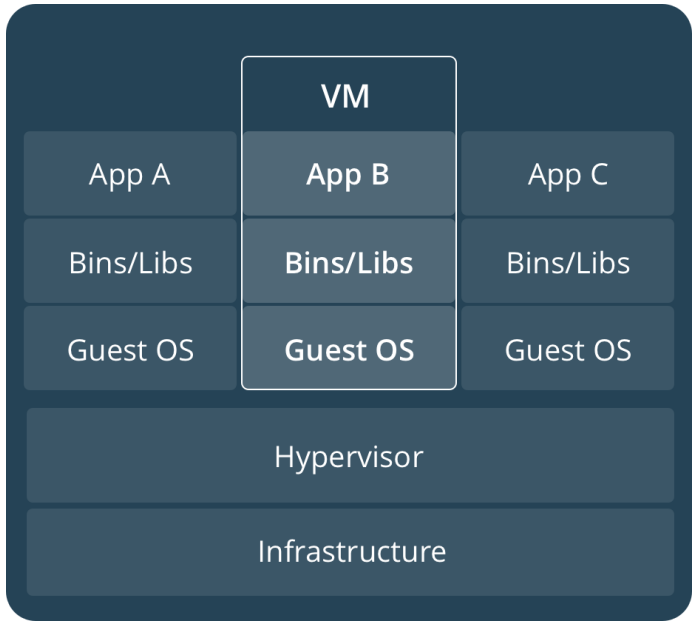
- Plus flexible
- Plus rapide à déployer et migrer
- Évite d'être trop dépendant d'un vendeur
- Meilleure utilisation des serveurs
- Coûts réduits
- ...

Limites ?

- Une VM est lourde, il faut un OS complet chaque fois
- L'hyperviseur doit simuler le matériel pour chaque VM
- En cas de mise à jour d'OS, il faut mettre à jour chaque VM individuellement

Conteneurs et VMs

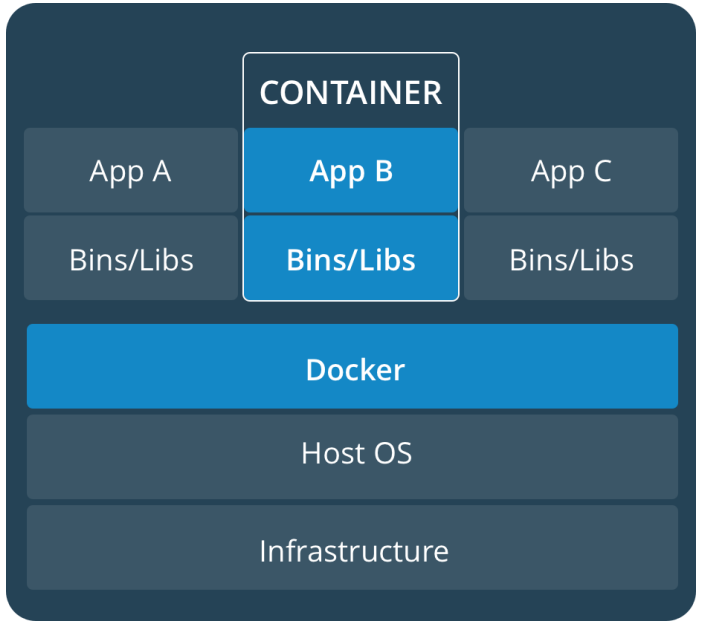
Machines Virtuelles



- Infrastructure -- Serveur, Cloud, Ordinateur
- Hyperviseur -- VMWare, Hyper-V etc

Un Guest OS (Windows, Linux etc) pour chaque application.

Conteneurs

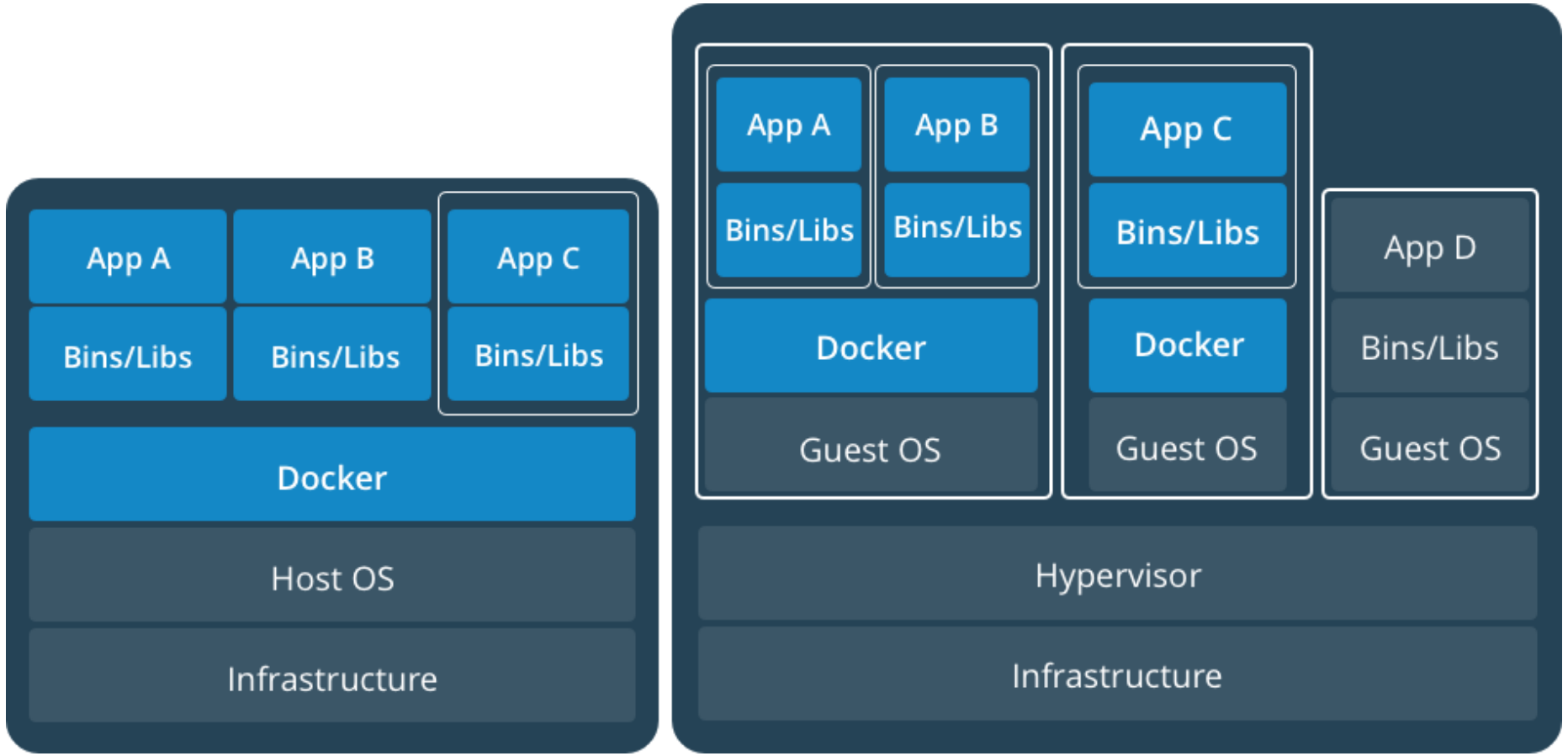


- Infrastructure -- Serveur, Cloud, Ordinateur
- Host OS -- Linux, Windows ou MacOS (avec VM Linux généralement), etc

Les Bins/Libs peuvent aussi être partagées, suivant l'application, à plusieurs instances de la même application.

Conteneurs et VMs

Machines Virtuelles ET Conteneurs



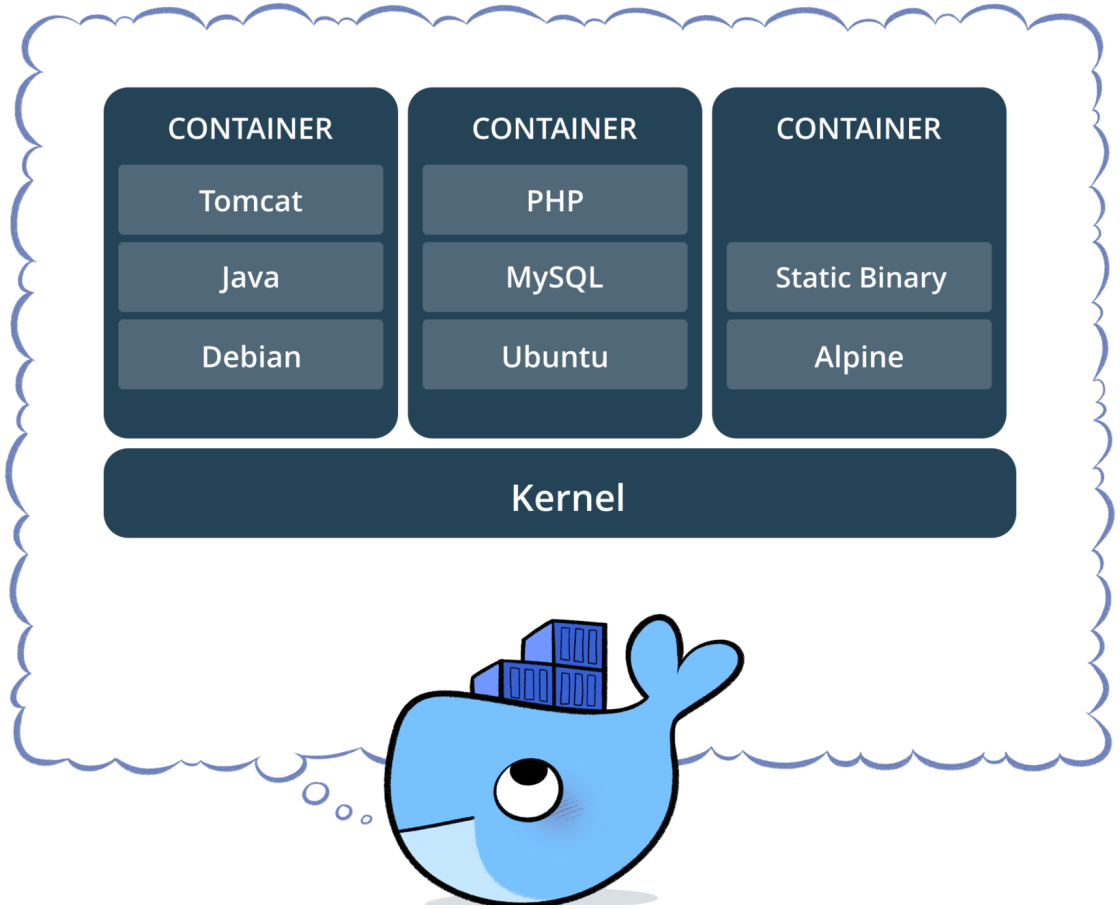
Conteneurs et VMs

Quelques avantages

- Grâce au partage du kernel, moins d'overhead dans la gestion de la "virtualisation"
- Possibilité de lancer un container en quelques millisecondes (plusieurs minutes pour une VM)
- Possibilité de lancer un container ponctuellement pour exécuter une simple tâche
- Les images reposent sur des couches qui se superposent
- ...

et limites...

- Il faut un hôte reposant sur le kernel Linux (utilisation de VM pour lancer des Containers Docker sur Windows ou MacOS), ou une adaptation de Docker (travail entre Microsoft et Docker notamment)
- L'isolation est au niveau des processus, il peut donc y avoir plus de risques au niveau sécurité



Autres sources :

- Présentation de Rama Krishna B, Openstack Architect at Hewlett Packard Enterprise

Conteneurs et VMs

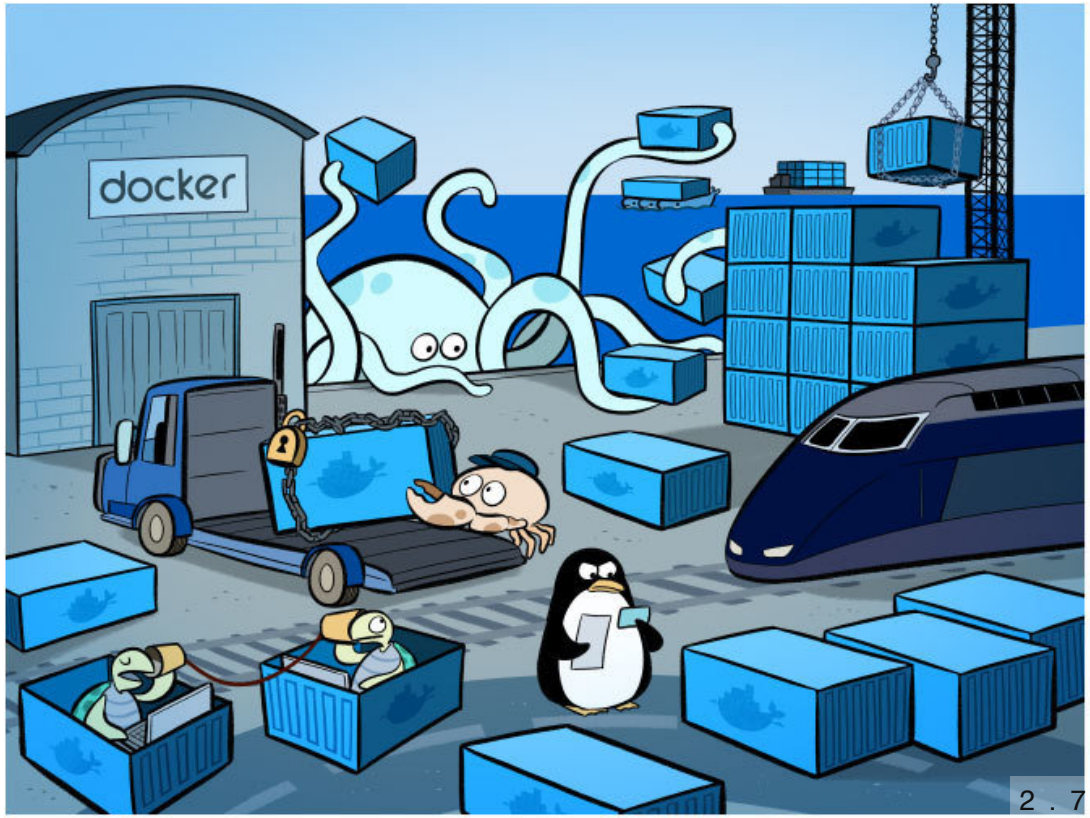
Pourquoi "Docker" ?

En 1953, Malcom MacLean reprend le principe de "conteneurs" imaginé dans le transport ferroviaire au début du XXe siècle pour l'appliquer au transport naval.

> *[...] alors que les autoroutes reliant les différents ports de la côte Ouest sont frappées par les bouchons de plus en plus importants, il image le moyen d'embarquer directement les remorques de ses camions sur des bateaux afin que ceux-ci se chargent du trafic d'un port à l'autre sans qu'il soit besoin, à chaque fois, de décharger et de recharger les véhicules. Des terminaux pourraient alors être construits sur le front de mer où seraient effectuées les opérations de transbordement des remorques. [...]*

Soit :

- Isoler du reste des marchandises
- Standardiser : Forme, taille, volume, poids
- Économie d'échelle notamment en termes de transport



Soit en informatique, Dockeriser c'est

"isoler" :

- Du système de fichier de l'hôte et des autres conteneurs
- Du réseau (possibilité de redéfinir un routage, d'ouvrir des ports spécifiques etc)
- Des droits d'utilisateurs et groupes (indépendant pour chaque conteneurs)
- Des autres processus...

Et *"standardiser"* :

- Possibilité de déplacer "facilement" un conteneur entre serveurs et avoir la garantie d'un fonctionnement cohérent
- Imposer des limites liées aux ressources (CPU, RAM, Stockage etc)

Autres sources :

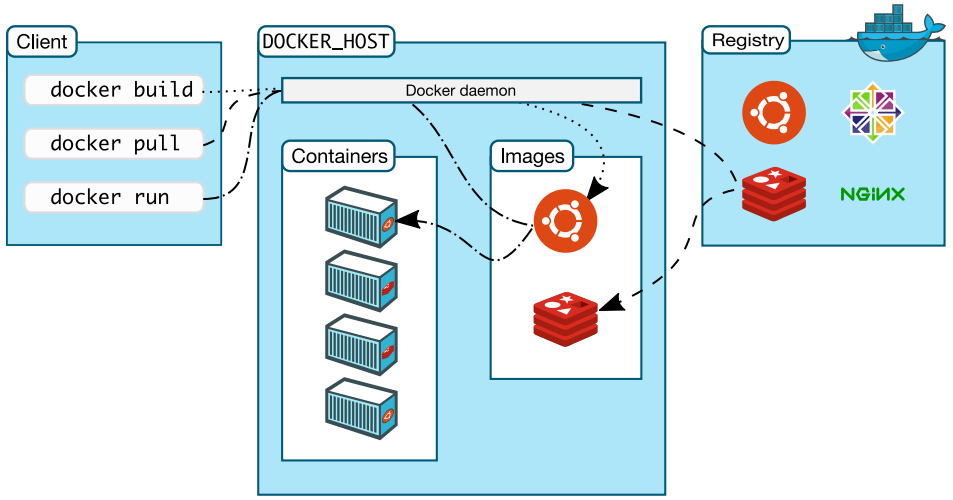
- Présentation de Rama Krishna B, Openstack Architect at Hewlett Packard Enterprise

Conteneurs et VMs

C'est quoi "Docker" ?

Package software into standardized units for development, shipment and deployment

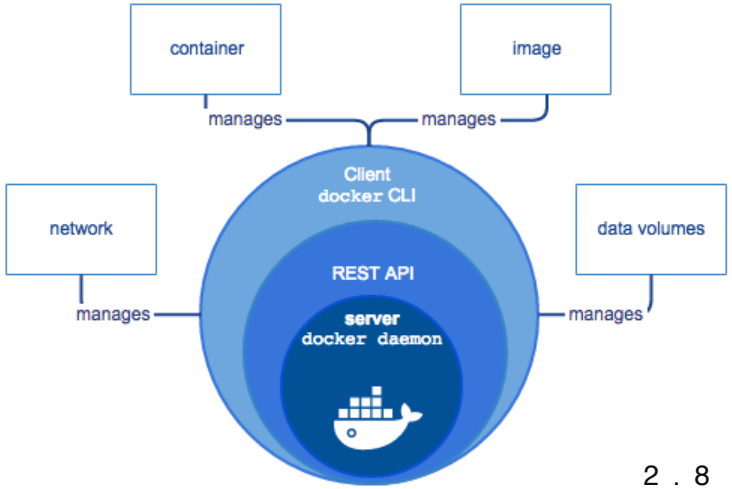
“ A container image is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, settings. Available for both Linux and Windows based apps, containerized software will always run the same, regardless of the environment. Containers isolate software from its surroundings, for example differences between development and staging environments and help reduce conflicts between teams running different software on the same infrastructure.

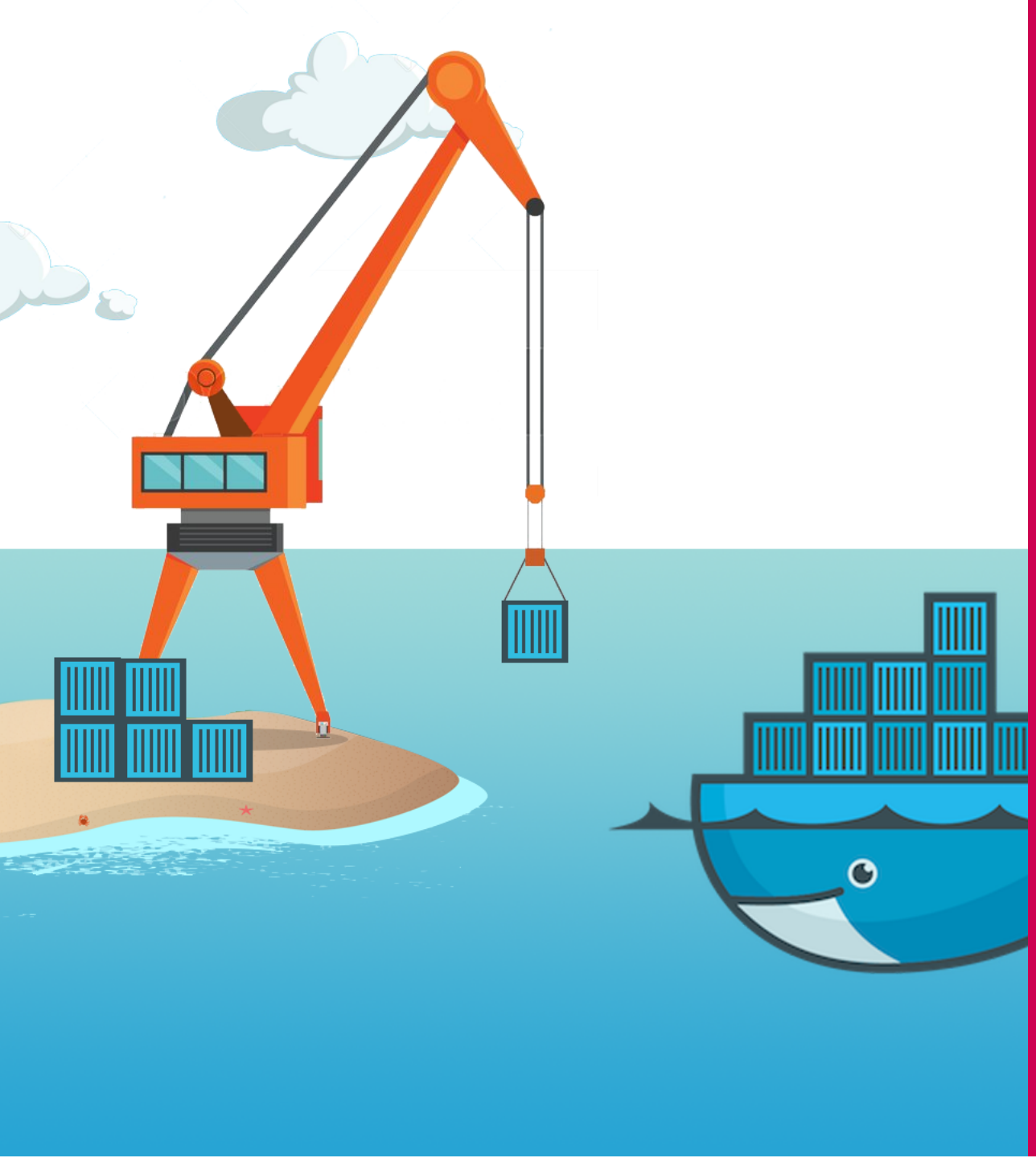


Ça marche comment ?

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a daemon process (the dockerd command).
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).





ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. Introduction générale
   Conteneurs et VMs
2. **Utilisation basique**
   docker run
3. Utilisation simplifiée
   docker-compose
4. Utilisation avancée
   (logs, données, réseaux, ressources, etc)
5. Création d'image
   (Commit et Dockerfile)
6. Aller plus loin
   (Accéder à des devices, l'orchestration etc)

Lancer un conteneur

- Prenons l'exemple de "tensorflow docker". (ou "docker hub")
- Le premier résultat devrait être lié à DockerHub.
- Il faut donc utiliser "tensorflow/tensorflow" pour récupérer l'image de "tensorflow" créée sur Docker.
- Plusieurs versions (tags) sont disponibles. Nous voulons avoir une ancienne version, nous allons donc utiliser :

```
#Dans un terminal
docker run tensorflow/tensorflow:1.10.1
```

- Lien ne fonctionne pas, pourquoi ?
- Arrêtons le conteneur, la console indique "Use Control-C to stop this server [...]" ...
  Ça ne fonctionne pas non plus...

Il faut en fait pouvoir interagir avec le retour du conteneur, donc activer :

- l'option de connexion au terminal (**option "tty", soit "-t"**),
- ainsi que l'option d'interactivité (**soit "-i"**).

Et donc, pour arrêter un conteneur ?

```
user@user-vm:~$ docker run tensorflow/tensorflow:1.10.1
Unable to find image 'tensorflow/tensorflow:1.10.1' locally
1.10.1: Pulling from tensorflow/tensorflow
3b37166ec614: Pull complete
ba077e1ddb3a: Pull complete
34c83d2bc656: Pull complete
84b69b6e4743: Pull complete
0f72e97e1f61: Pull complete
6086c6484ab2: Pull complete
25817b9e5842: Pull complete
5252e5633f1c: Pull complete
8de57ae4ad7d: Pull complete
4b7717108c3b: Pull complete
b65e9e47e80a: Pull complete
006d31e013ea: Pull complete
700521cc53f3: Pull complete
Digest: sha256:f45d87bd473bf999241afe444748a2d3a9be24f8d736a808277b4f3e32159566
Status: Downloaded newer image for tensorflow/tensorflow:1.10.1
[I 09:48:03.627 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/
[W 09:48:03.649 NotebookApp] WARNING: The notebook server is listening on all IP addresses and not using
[I 09:48:03.661 NotebookApp] Serving notebooks from local directory: /notebooks
[I 09:48:03.661 NotebookApp] The Jupyter Notebook is running at:
[I 09:48:03.662 NotebookApp] http://(716bec8f8301 or 127.0.0.1):8888/?token=c5556797dc167b2f802cad83c8f16
[I 09:48:03.662 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip c
[C 09:48:03.663 NotebookApp]

    Copy/paste this URL into your browser when you connect for the first time,
    to login with a token:
        http://(716bec8f8301 or 127.0.0.1):8888/?token=c5556797dc167b2f802cad83c8f16f5126e0320d36fc7ccf
```

Téléchargement de l'image "tensorflow/tensorflow" qui n'était pas disponible en local

Exécution du conteneur, log d'exécution. Lien à tester : http://127.0.0.1:8888/?tocken=??

```
#Dans un terminal, pour lister les conteneurs
docker ps
```

```
user@user-vm:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
716bec8f8301        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   5 minutes ago       Up 5 minutes        6006/tcp, 8888/tcp       optimistic_keller
```

```
#Dans un terminal, pour arrêter un conteneur
docker stop nom_conteneur

#soit ici :
docker stop optimistic_keller
```

Lancer un conteneur

```
user@user-vm:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                    NAMES
716bec8f8301        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   5 minutes ago       Up 5 minutes        6006/tcp, 8888/tcp       optimistic_keller
```

- Relançons le conteneur "tensorflow/tensorflow:1.10.1" avec les options d'interactivité (-i) avec le terminal (-t)
- Et retentons le lien "http://127.0.0.1:8888/?token=??"

```
#Dans un terminal
docker run -it tensorflow/tensorflow:1.10.1
```

- Lien ne fonctionne toujours pas... pourquoi ?
- Cette fois-ci, on peut au moins arrêter le conteneur avec "Control+C"

Cependant, si on tape le lien indiqué sur notre navigateur "http://127.0.0.1:8888" :

- "127.0.0.1" correspond à notre machine hôte, pas au conteneur
- "8888" correspond à un port de notre machine hôte, vers où pointe-t-il ?

Prenons un peu plus de temps pour comprendre le retour de "docker ps".

Nom	Exemple	Signification
CONTAINER ID	716bec8f8301	ID unique du conteneur
IMAGE	tensorflow/tensorflow:1.10.1	Image utilisée pour créer ce conteneur
COMMAND	"/run_jupyter.sh --a…"	Commande lancée au démarrage du conteneur
CREATED	5 minutes ago	Depuis quand le conteneur a été créé
STATUS	Up 5 minutes	S'il est arrêté ou, s'il est lancé, depuis quand.
PORTS	**6006/tcp, 8888/tcp**	Ports déclarés par défaut par le conteneur
NAMES	optimistic_keller	Nom du container, par défaut composé d'un couple de personnage illustre et de qualificatif

Configurer un conteneur

Le conteneur déclare bien le port "8888".
Cherchons des détails...

```
#Dans un terminal
docker inspect nom_conteneur

#soit ici :
docker inspect optimistic_keller
```

On voit beaucoup d'informations, notamment :

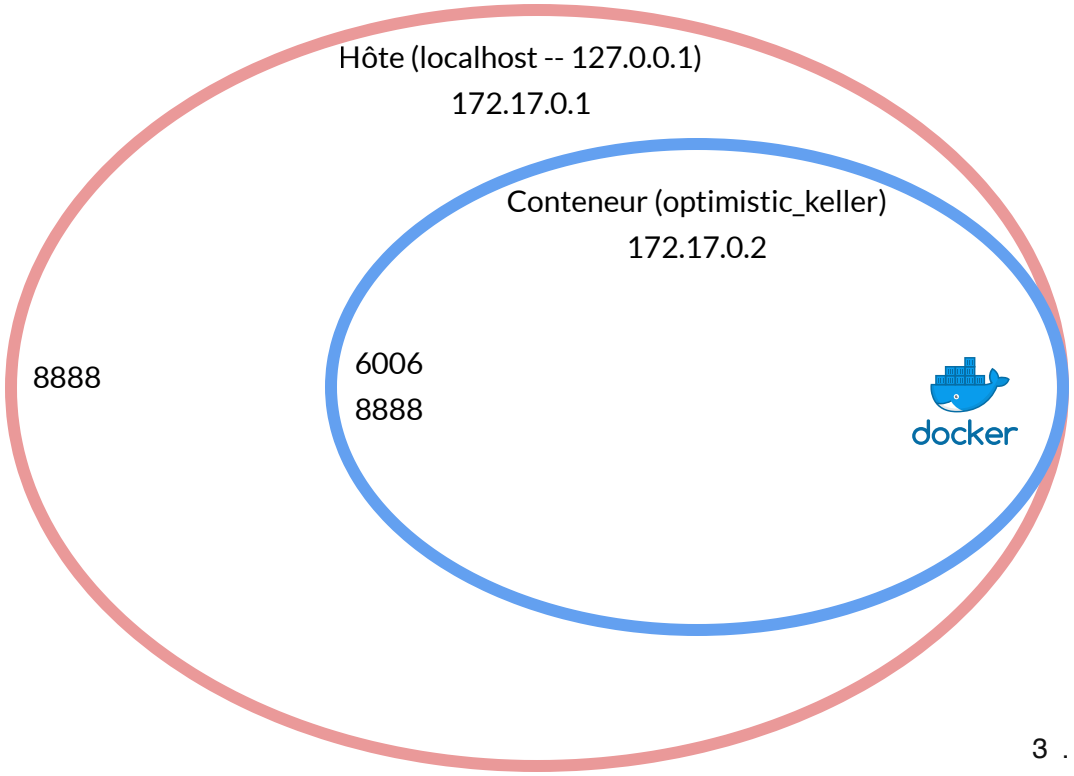
- la commande lancée au démarrage ("Path": "/run_jupyter.sh") et ses paramètres ("Args": ["--allow-root"]) (C'était donc ça la fin du "/run_jupyter.sh --a..." renvoyé par "docker ps" !)
- son état, son nom, sa réservation de processeur et mémoire éventuellement etc

Et quelques autres informations qui semblent pertinentes pour notre problème :

```
    "ExposedPorts": {
            "6006/tcp": {},
            "8888/tcp": {}
    },
    "Tty": true,
    "OpenStdin": true,
```

Et dans la section "NetworkSettings" :

```
    "Ports": {
            "6006/tcp": null,
            "8888/tcp": null
    },
    "Gateway": "172.17.0.1",
    "IPAddress": "172.17.0.2",
```



Configurer un conteneur

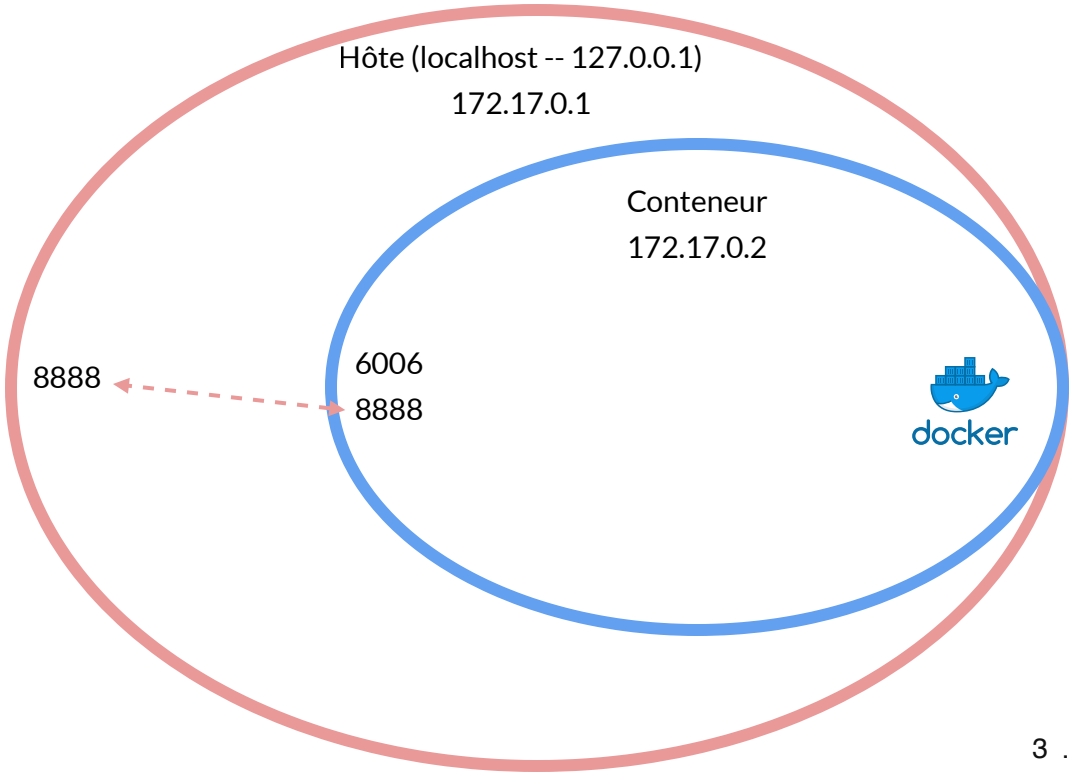
Le conteneur déclare bien le port "8888".
Mais ce n'est lié à rien sur notre hôte.

```
    "PortBindings": {},
    "ExposedPorts": {
            "6006/tcp": {},
            "8888/tcp": {}
    },
    "Tty": true,
    "OpenStdin": true,
```

La partie "PortBindings" est vide...

```
#Dans un terminal, pour arrêter un conteneur
docker stop nom_conteneur

#Relancer un conteneur avec bind de port
docker run -it -p Hote:Conten tensorflow/tensorflow:1.1

#Soit ici, par exemple :
docker run -it -p 8888:8888 tensorflow/tensorflow:1.10

    "PortBindings": {
        "8888/tcp": [
            {
                "HostIp": "",
                "HostPort": "8888"
            }
        ]
    },
```

```
user@user-vm:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
bab8d04ff48e        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   52 seconds ago

          STATUS              PORTS                              NAMES
          Up 51 seconds       6006/tcp, 0.0.0.0:8888->8888/tcp   busy_jepsen
```



Configurer un conteneur

On peut donc lancer plusieurs instances d'une même image, et lier différents ports...

```
#Dans un terminal, lancer un nouveau conteneur
# mais utilisant le port 8800 par exemple
docker run -it -p 8800:8888 tensorflow/tensorflow:1.10

#Vérifions les conteneurs lancés
docker ps
```

On a maintenant deux instances de Tensorflow :

- L'une accessible à https://localhost:8888/?token=??
- L'autre avec le lien http://localhost:8800/?token=??

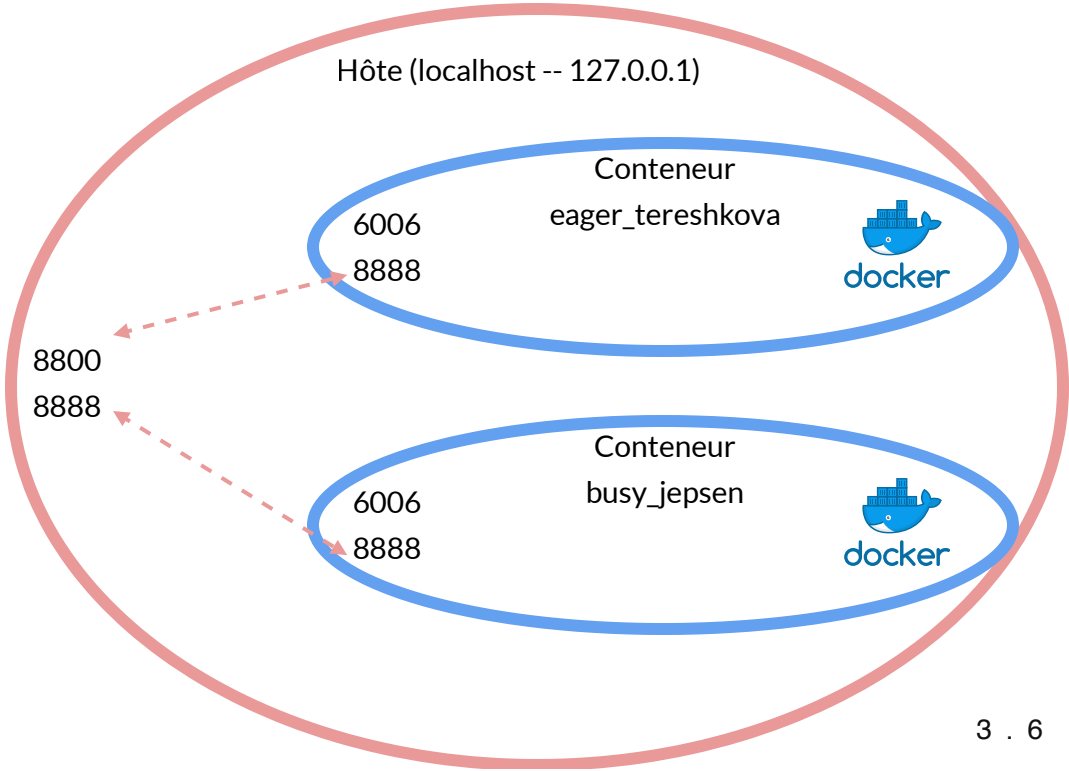
Chacune ne sait pas qu'une autre instance est lancée. Ils pourraient notamment être de versions différentes, par exemple pour tester les évolutions...

Pour cela, on peut donc indiquer un autre "tag" à l'image, comme ceux proposés sur le dépôt DockerHub, ici. Par défaut, c'est le tag "latest" qui est utilisé.

Par exemple, pour avoir la version "1.3.0" :

```
#Dans un terminal, lancer un nouveau conteneur
# mais utilisant le port 8800 par exemple
docker run -it -p 88:8888 tensorflow/tensorflow:1.3.0
```

```
user@user-vm:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED
4eacd174d510        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   46 seconds ago
bab8d04ff48e        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   8 minutes ago

STATUS              PORTS                              NAMES
Up 46 seconds       6006/tcp, 0.0.0.0:8800->8888/tcp   eager_tereshkova
Up 8 minutes        6006/tcp, 0.0.0.0:8888->8888/tcp   busy_jepsen
```



Nettoyer des conteneurs

On commence à en avoir pas mal de conteneurs...

```
#Dans un terminal,
# pour voir les conteneurs
docker ps
```

```
user@user-vm:~$ docker ps
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS              PORTS                              NAMES
c4f9e6e26b3a        tensorflow/tensorflow:1.3.0    "/run_jupyter.sh --a…"   2 minutes ago       Up 2 minutes        6006/tcp, 0.0.0.0:88->8888/tcp     relaxed_moser
4eacd174d510        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   7 minutes ago       Up 7 minutes        6006/tcp, 0.0.0.0:8800->8888/tcp   eager_tereshkova
bab8d04ff48e        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   15 minutes ago      Up 15 minutes       6006/tcp, 0.0.0.0:8888->8888/tcp   busy_jepsen
```

```
#Pour vraiment tous
# les voir... même les anciens
docker ps -a
```

```
user@user-vm:~$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS                      PORTS                              NAMES
c4f9e6e26b3a        tensorflow/tensorflow:1.3.0    "/run_jupyter.sh --a…"   2 minutes ago       Up 2 minutes                6006/tcp, 0.0.0.0:88->8888/tcp     relaxed_moser
4eacd174d510        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   8 minutes ago       Up 8 minutes                6006/tcp, 0.0.0.0:8800->8888/tcp   eager_tereshkova
bab8d04ff48e        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   15 minutes ago      Up 15 minutes               6006/tcp, 0.0.0.0:8888->8888/tcp   busy_jepsen
00244554500a        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   31 minutes ago      Exited (137) 16 minutes ago                                    nervous_mestorf
716bec8f8301        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   About an hour ago   Exited (137) 36 minutes ago                                    optimistic_keller
```

On fait le ménage ?

```
#Pour supprimer un conteneur
docker rm nom_conteneur

#Il faut l'arrêter si ce
# n'est pas fait...
docker stop nom_conteneur
```

```
user@user-vm:~$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS                      PORTS                              NAMES
c4f9e6e26b3a        tensorflow/tensorflow:1.3.0    "/run_jupyter.sh --a…"   2 minutes ago       Up 2 minutes                6006/tcp, 0.0.0.0:88->8888/tcp     relaxed_moser
4eacd174d510        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   8 minutes ago       Up 8 minutes                6006/tcp, 0.0.0.0:8800->8888/tcp   eager_tereshkova
bab8d04ff48e        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   15 minutes ago      Up 15 minutes               6006/tcp, 0.0.0.0:8888->8888/tcp   busy_jepsen
00244554500a        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   31 minutes ago      Exited (137) 16 minutes ago                                    nervous_mestorf
716bec8f8301        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   About an hour ago   Exited (137) 36 minutes ago                                    optimistic_keller
user@user-vm:~$ docker rm optimistic_keller
optimistic_keller
user@user-vm:~$ docker rm relaxed_moser
Error response from daemon: You cannot remove a running container c4f9e6e26b3a1d99d4565b29154ecaee093fcd2a8ba5ee6e29f335a0c0d14371. Stop the container before attempting removal o
r force remove
user@user-vm:~$ docker stop relaxed_moser
relaxed_moser
user@user-vm:~$ docker rm relaxed_moser
relaxed_moser
user@user-vm:~$ docker ps -a
CONTAINER ID        IMAGE                          COMMAND                  CREATED             STATUS                      PORTS                              NAMES
4eacd174d510        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   9 minutes ago       Up 9 minutes                6006/tcp, 0.0.0.0:8800->8888/tcp   eager_tereshkova
bab8d04ff48e        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   17 minutes ago      Up 17 minutes               6006/tcp, 0.0.0.0:8888->8888/tcp   busy_jepsen
00244554500a        tensorflow/tensorflow:1.10.1   "/run_jupyter.sh --a…"   33 minutes ago      Exited (137) 17 minutes ago                                    nervous_mestorf
user@user-vm:~$
```

Un par un ?!

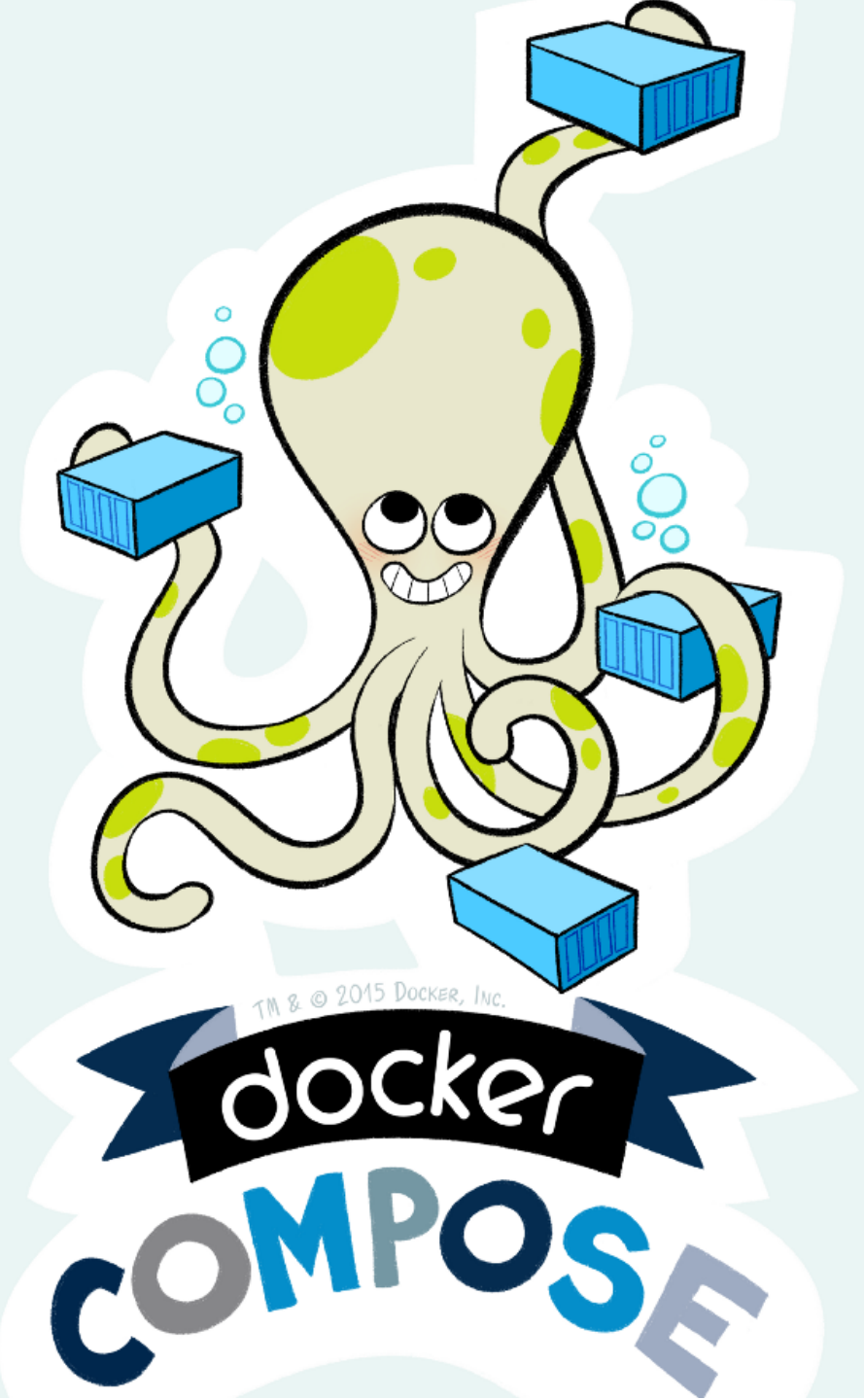
```
#Nettoyer le système
docker system prune
```

Voir ici pour plus d'informations.

Conclusion

Vous savez...

```
#Lancer un conteneur
# d'une image spécifique venant de DockerHub
#    Ici, tensorflow/tensorflow
# avec une version/tag particulière
#    Ici, 1.3.0
# en liant les ports
#    Ici, l'hôte port "80" vers le port conteneur "88
# en se connectant au terminal de façon interactive
#    Ici, -it
docker run -it -p 80:8888 tensorflow/tensorflow:1.3.0

#Afficher les conteneurs lancés
docker ps

#Afficher tous les conteneurs
docker ps -a

#Inspecter un conteneur
docker inspect nom_conteneur

#Arrêter un conteneur
docker stop nom_conteneur

#Supprimer un conteneur arrêté
docker rm nom_conteneur

#Supprimer tous les conteneur arrêtés
# (mais aussi les réseaux, volumes, on verra...)
docker system prune
```



ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. Introduction générale
   Conteneurs et VMs
2. Utilisation basique
   docker run
3. **Utilisation simplifiée**
   docker-compose
4. Utilisation avancée
   (logs, données, réseaux, ressources, etc)
5. Création d'image
   (Commit et Dockerfile)
6. Aller plus loin
   (Accéder à des devices, l'orchestration etc)

Docker-compose

Pour simplifier la configuration d'un conteneur mais aussi faciliter le déploiement et la migration, il existe un outils permettant de décrire un conteneur : docker-compose

La commande :

```
docker run -it -p 8880:8888 tensorflow/tensorflow
```

est équivalente à écrire un fichier "docker-compose.yml" :

```
version: '3'
services:
  tensorflow:
    image: 'tensorflow/tensorflow'
    ports:
      - 8880:8888
    stdin_open: true
    tty: true
```

Objectif	**docker run**	**docker-compose**
Indiquer l'image/tag	tensorflow/tensorflow	image: 'tensorflow/tensorflow'
Spécifier les binds de ports	-p 8880:8888	ports: - 8880:8888
Connexion au terminal	-t	tty: true
Mode interactif du terminal	-i	stdin_open: true

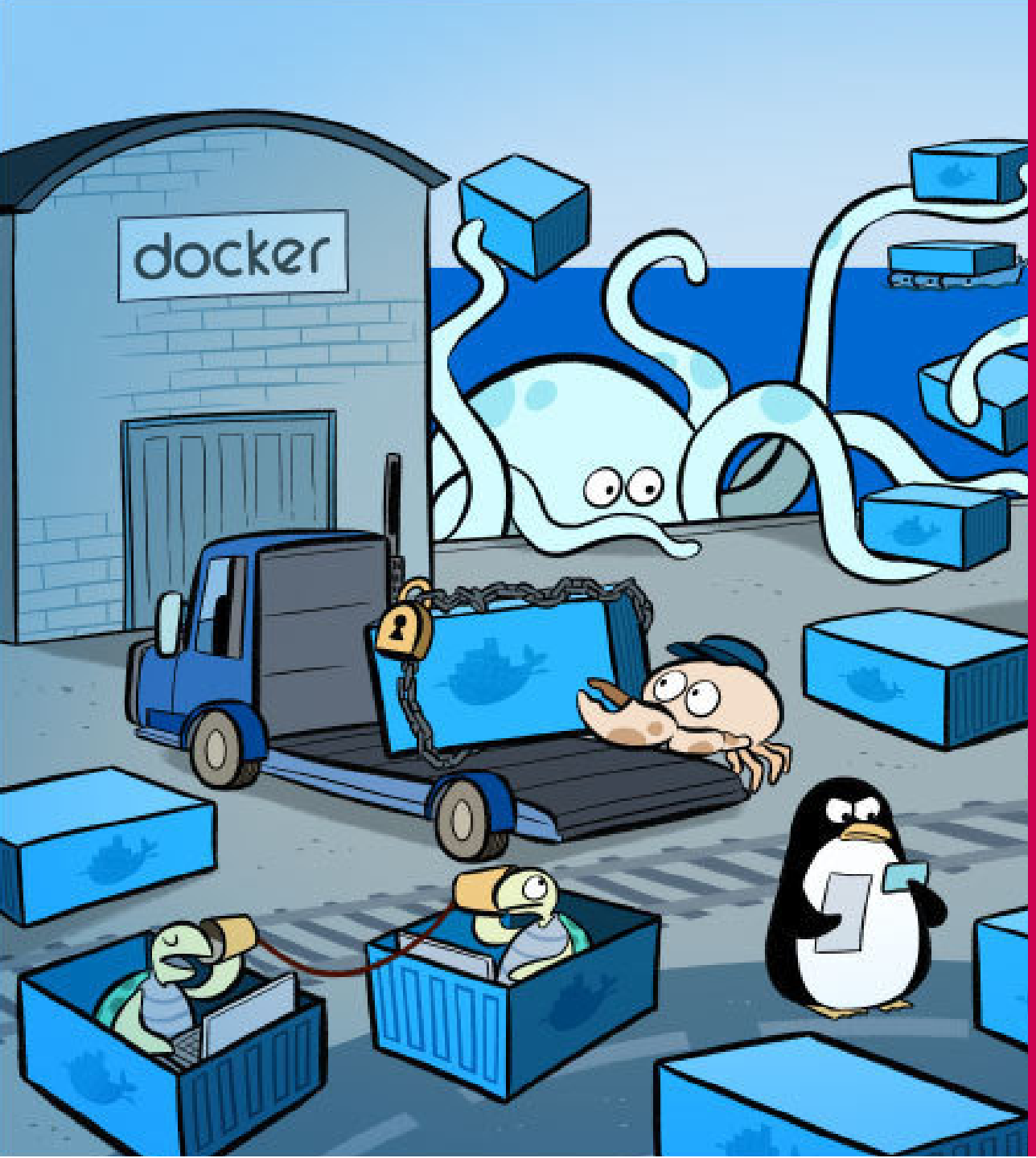
Docker-compose

Puis exécuter, dans le dossier du fichier :

```
#Dans un terminal
docker-compose up
```

Quelques commandes supplémentaires :

Commande	Signification
docker-compose up	Lancer le docker run. Possible d'ajouter les commandes : -d => lancement en "deamon", plus d'accroche au terminal --force-recreate => forcer à recréer le container s'il existe déjà.
docker-compose start	Démarrer les containers. S'ils n'existent pas, un "docker-compose up –d" est exécuté implicitement.
docker-compose stop	Arrêter les containers, s'ils sont déjà lancés
docker-compose rm	Supprimer les containers s'ils existent et sont arrêtés.



ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. Introduction générale
   Conteneurs et VMs

2. Utilisation basique
   docker run

3. Utilisation simplifiée
   docker-compose

4. **Utilisation avancée**
   (logs, données, réseaux, ressources, etc)

5. Création d'image
   (Commit et Dockerfile)

6. Aller plus loin
   (Accéder à des devices, l'orchestration etc)

Logs

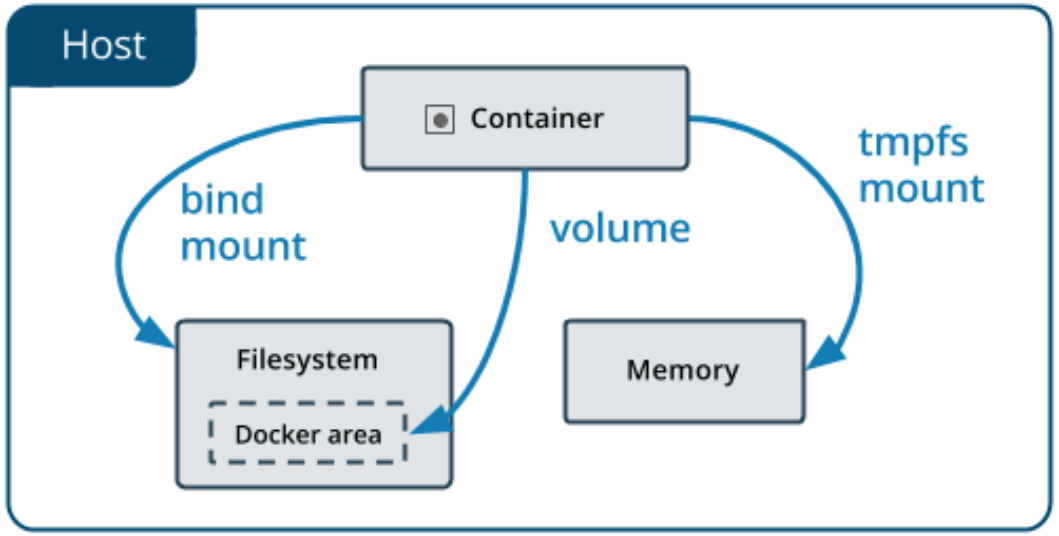
Pas vraiment une fonctionnalité avancée, au contraire, **un outils de base** !

Quand vous lancez un conteneur, toutes informations de sortie standard et d'erreur sont connectées au gestionnaire de logs de Docker.

Vous pouvez y accéder à partir de "docker logs" ou de "docker-compose logs" :

```
#Afficher les logs d'un conteneur
docker logs nom_conteneur

#Afficher les logs de tous les services d'un docker-compose.yml
#Chaque service aura alors une couleur dédiée
docker-compose logs

#Afficher les logs d'un service en particulier
docker-compose logs mon_service

#Afficher, en continu, les logs de tous le services
#d'un docker-compose.yml
docker-compose logs -f

#Afficher les 10 dernières lignes des logs de tous
#les services d'un docker-compose.yml
docker-compose logs --tail=10
```

Stockage de données



D'après cette page de documentation :

- Volumes are stored in a part of the host filesystem which is managed by Docker (/var/lib/docker/volumes/ on Linux). Non-Docker processes should not modify this part of the filesystem. Volumes are the best way to persist data in Docker.
- Bind mounts may be stored anywhere on the host system. They may even be important system files or directories. Non-Docker processes on the Docker host or a Docker container can modify them at any time.
- tmpfs mounts are stored in the host system's memory only, and are never written to the host system's filesystem.

Stockage de données

Exemple de "bind mounts"

Sur notre conteneur "tensorflow", on voudrait pouvoir déposer un dossier contenant un exemple de code.

On fait ça où ?!

En creusant, on trouve que le Docker tensorflow déclare un dossier à sa racine "**/notebooks**" qui contiendra les différents codes créés.

L'idée ? Lier ce dossier qui se trouve dans le conteneur, à un dossier se trouvant sur l'hôte.

```
version: '2'
services:
  tensorflow:
    image: 'tensorflow/tensorflow:1.3.0'
    ports:
      - 8880:8888
    stdin_open: true
    tty: true
    volumes:
      - ./shared:/notebooks
```

Maintenant, le dossier "./shared" se trouvant à côté du fichier "docker-compose.yml" est un point de montage du dossier "/notebooks" interne au conteneur.

Réseaux

Si je place plusieurs "services" dans un docker-compose, ils feront tous partis d'un même réseau virtuel spécifique. Je peux aussi le nommer afin d'y connecter d'autres services qui seraient déclarés hors de ce docker-compose.

Ici, un exemple avec NextCloud, nous avons :

- Un service "db", basé sur l'image "mariadb"
  - Les variables d'environnement permettent de pré-configurer la base de données.
  - Aucun port n'est "bindé", je ne peux donc pas accéder à la base de données si je ne suis pas sur le réseau virtuel
- Un service "nextcloud" basé sur l'image "nextcloud"
  - Un port (80) est "bindé" sur l'hôte (8082)
- Un réseau est déclaré "nextcloud-esilv" et est utilisé dans chaque service. Ce nom de réseau pourrait être utilisé ailleurs pour y connecter d'autres services.

```
#Voir les informations d'un réseau
docker network inspect nom_reseau
```

Notez l'utilisation de "links" qui permet de "lier" le service "db" au service "nextcloud", donc si ce dernier est recréé, la base de données sera aussi recréée.

```yaml
version: '2'
services:

    db:
        image: mariadb:10.4.7-bionic
        container_name: nextcloud_db
        hostname: nextcloud_db
        restart: always
        networks:
            - nextcloud-esilv
        volumes:
            - ./dbdata:/var/lib/mysql
        environment:
            - MYSQL_ROOT_PASSWORD=pX6V38AzO6O2
            - MYSQL_PASSWORD=82vDQro1JJP6
            - MYSQL_DATABASE=nextcloud
            - MYSQL_USER=nextcloud

    nextcloud:
        image: nextcloud:16.0.4-apache
        container_name: nextcloud_esilv
        hostname: nextcloud_esilv
        restart: always
        ports:
            - 8082:80
        networks:
            - nextcloud-esilv
        links:
            - db
        volumes:
            - ./nextclouddata:/var/www/html

networks:
    nextcloud-esilv:
        driver: bridge
```

Ressources

Pour chaque service, on peut indiquer plusieurs paramètres pour limiter/réserver l'utilisation de la mémoire, des CPUs etc.

Ici, un exemple avec GitLab, nous avons :

- Un service "gitlab", basé sur l'image "gitlab-ce"
  - Les variables d'environnement permettent de pré-configurer le serveur
  - Plusieurs ports et volumes sont "bindés"
  - Une **limite de mémoire** utilisée est fixée à 768Mo, le service ne voit que cette capacité disponible
  - Une **réservation de mémoire** de 512Mo permet de garantir que le système conservera cette quantité disponible pour le service
  - On **limite l'utilisation en CPU** de ce service à 35% de l'ensemble des processeurs disponibles sur l'hôte

On peut vérifier tout cela avec la commande :

```
#Dans un terminal, pour afficher les statistiques
#d'un conteneur
docker stats nom_conteneur
```

```
#Attention, ces options sont valables à partir de la version 2.2,
#puis modifiées à partir de la version 3.0

version: '2.2'
services:

    gitlab:
        image: 'gitlab/gitlab-ce:12.2.4-ce.0'
        restart: always
        hostname: gitlab_esilv
        container_name: gitlab_esilv
        networks:
            - gitlab-esilv
        environment:
            GITLAB_OMNIBUS_CONFIG: |
                external_url 'http://git.esilv.org'
                # Add any other gitlab.rb configuration here,
                # each on its own line
                gitlab_rails['backup_keep_time'] = 604800
        ports:
            - '8083:80'
            - '4433:443'
            - '223:22'
        volumes:
            - './config:/etc/gitlab'
            - './logs:/var/log/gitlab'
            - './data:/var/opt/gitlab'
        mem_limit: 768m
        mem_reservation: 512m
        cpus: 0.35

networks:
    gitlab-esilv:
        driver: bridge
```

Adapter un conteneur

Voyons un cas particulier. Une image disponible n'est pas complète et il faut la modifier légèrement...

Ici, un exemple avec Wordpress. Notez qu'il existe une image docker spécifique à Wordpress, mais nous allons ici plutôt déployer :

- Un service "site", basé sur l'image "php" en version 5 avec un serveur Apache.
  - On va "binder" un dossier local "wordpress" vers le dossier du conteneur "/var/www/html" qui correspond au dossier chargé par le serveur Apache.
    Dans ce dossier local, on va simplement décompresser le dossier téléchargé sur Wordpress.com.
- Un service "db", basé sur l'image "mysql" en version 5.7.
  - Avec les variables d'environnement, on prépare une base de données pour notre installation de Wordpress.

Malheureusement, si on lance cela, des erreurs apparaitront lors de l'installation de Wordpress...

```
version: '2'

services:

    site:
      image: php:5-apache
      ports :
       - "8080:80"
      volumes:
       - ./wordpress:/var/www/html/
      links:
       - db
      networks:
       - website-network

    db:
      image: mysql:5.7
      volumes:
#       - ./data-init:/docker-entrypoint-initdb.d
       - ./database:/var/lib/mysql
      environment:
          MYSQL_ROOT_PASSWORD: myrootpassword
          MYSQL_DATABASE: wordpress
          MYSQL_USER: wordpress
          MYSQL_PASSWORD: mypassword
      networks:
       - website-network

networks:
  website-network:
      driver: bridge
```

Adapter un conteneur

Il manque en fait à l'image "php:5-apache" quelques modules, notamment :

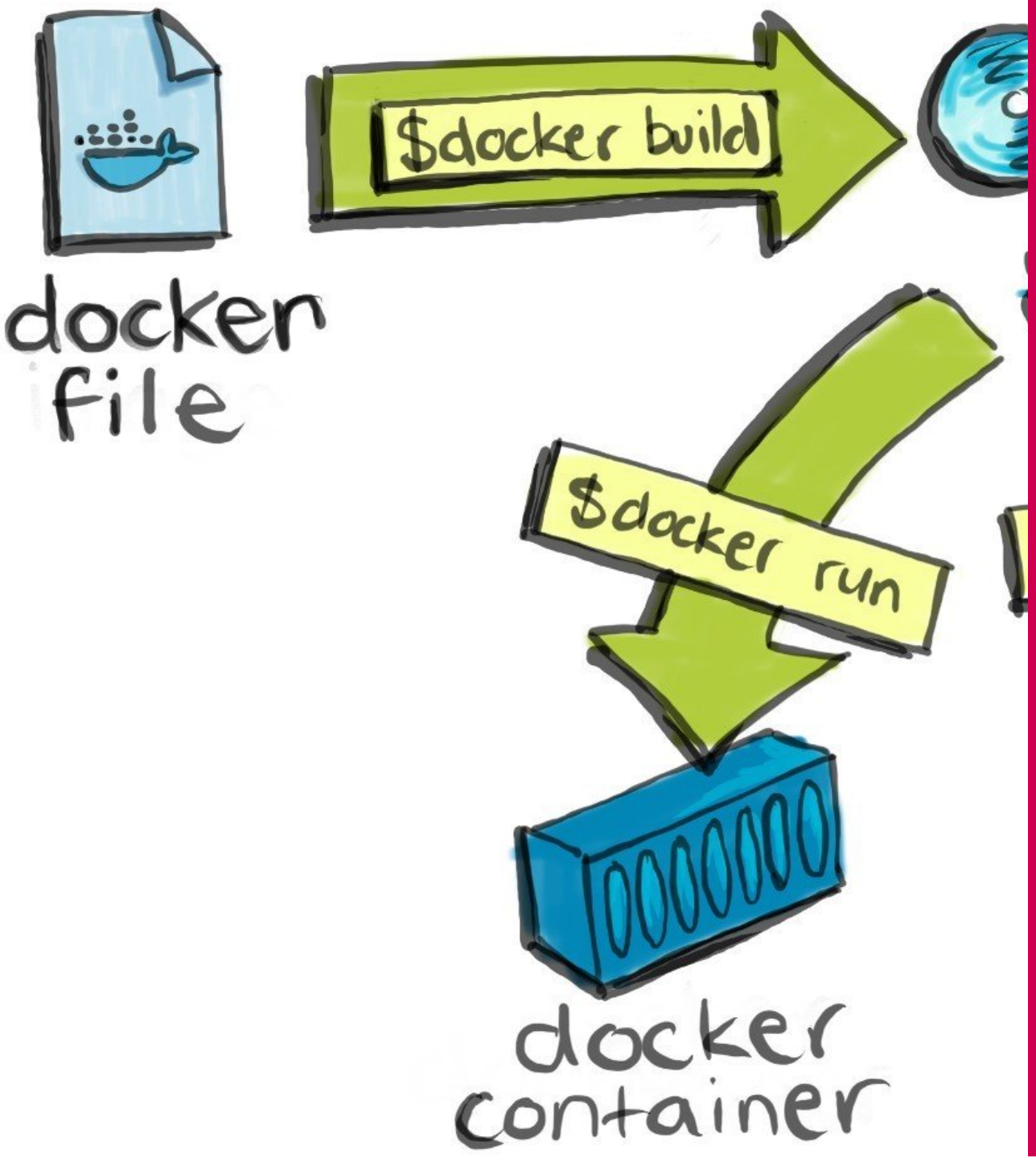
- Les extensions php : "pdo", "mysql" et "mysqli"
- L'activation du module "rewrite"
- Le droit pour l'utilisateur "www-data" qui lance le serveur Apache de modifier les fichiers contenus dans le dossier local "./wordpress"

```
#Pour exécuter une commande dans un conteneur
docker exec nom_conteneur ma_commande

#Par exemple, pour lancer la commande... bash
#et pouvoir y saisir des commandes
docker exec -it nom_conteneur bash
```

Puis je peux taper les commandes que je souhaite, notamment :

```
#Pour installer les modules php manquant
#spécifiques à ce conteneur
docker-php-ext-install pdo mysql mysqli

#Pour activer le module php "rewrite"
#comme sur n'importe quel serveur Apache
a2enmod rewrite

#Pour redémarrer le conteneur
#Je sors de la commande "bash" puis
docker-compose restart site
```

```
version: '2'

services:

    site:
      image: php:5-apache
      ports :
       - "8080:80"
      volumes:
       - ./wordpress:/var/www/html/
      links:
       - db
      networks:
       - website-network

    db:
      image: mysql:5.7
      volumes:
#       - ./data-init:/docker-entrypoint-initdb.d
       - ./database:/var/lib/mysql
      environment:
          MYSQL_ROOT_PASSWORD: myrootpassword
          MYSQL_DATABASE: wordpress
          MYSQL_USER: wordpress
          MYSQL_PASSWORD: password
      networks:
       - website-network

networks:
  website-network:
      driver: bridge
```



ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. Introduction générale
   Conteneurs et VMs
2. Utilisation basique
   docker run
3. Utilisation simplifiée
   docker-compose
4. Utilisation avancée
   (logs, données, réseaux, ressources, etc)
5. **Création d'image**
   (Commit et Dockerfile)
6. Aller plus loin
   (Accéder à des devices, l'orchestration etc)

A partir d'un conteneur

Notre conteneur basé sur l'image "php:5-apache" a été modifié pour faire fonctionner Wordpress. Mais si on redémarre le conteneur, tout sera effacé.

```
#Pour recréer un conteneur à partir de l'image
docker-compose up --force-recreate
```

Donc, pour déplacer mon conteneur, il faudra chaque fois que je relance les commandes dans le conteneur.
Il faudrait "sauvegarder" notre conteneur et le transformer en image. À l'image de "git", on a une commande "docker commit" :

```
#Transformer un conteneur en image
docker commit id_conteneur new_image:new_tag

#Pour vérifier la liste des images en local :
docker images
```

Je peux ensuite la sauvegarder en fichier (docker save) ou l'envoyer sur un dépôt d'images, par exemple DockerHub.

Après vous avoir créé un compte, vous pourrez enregistrer autant d'images publiques que vous le souhaitez et une image privée (payant pour plus d'images privées).

```
#Transformer une image en fichier déplaçable
docker save -o my_backup.tar new_image:new_tag

#Par exemple :
docker save -o wordpress.tar pcourbin/wordpress:test
```

Pour l'envoyer sur DockerHub, il faut s'identifier puis utiliser la commande "docker push"

```
#Se connecter à son compte DockerHub
docker login mon_id

#Envoyer une image sur DockerHub
docker push new_image:new_tag

#Soit ici :
docker login pcourbin
docker push pcourbin/wordpress:test
```

On peut alors (et, si l'image est publique, n'importe qui peut), réutiliser cette image en utilisant le nom "pcourbin/wordpress" et le tag "test".

```
version: '2'

services:
    site:
      image: pcourbin/wordpress:test
      ports :
       - "8080:80"
      volumes:
       - ./wordpress:/var/www/html/
      links:
       - db
      networks:
       - website-network
    db:
```

A partir d'un Dockerfile

La procédure précédente est assez simple, mais :

- Il faut avoir un endroit où stocker les images déjà créées (DockerHub, Registry privé)
- Surtout, il n'est pas évident de comprendre ce qui a changé dans cette nouvelle image par rapport à son image de base...
- Et si elle contenait d'autres choses dont je n'ai pas besoin ? Est-ce que je peux faire confiance à la personne qui a créé l'image ?

Une fois qu'on a "joué" à créer une image à la main, on peut écrire un "Dockerfile", un simple fichier texte qui décrit les commandes à exécuter pour créer notre image.

Avantages :

- On garde une trace de la façon dont est construite l'image
- On peut la faire évoluer facilement
- On peut stocker ce fichier sur un git, plus léger qu'une image complète.
- Il est possible de recréer l'image n'importe où avec la commande "docker build"

```
#Dans le même dossier que le fichier "Dockerfile"
docker build --tag new_image:new_tag .
```

```
#Fichier "Dockerfile"
FROM php:5.6-apache

LABEL maintainer="pierre.courbin@gmail.com"

# Install PDO MySQL driver
# See https://github.com/docker-library/php/issues/62
RUN docker-php-ext-install pdo mysql mysqli

# Workaround for write permission on write
# See https://github.com/boot2docker/boot2docker/pull/534
RUN usermod -u 1000 www-data

# Enable Apache mod_rewrite
RUN a2enmod rewrite
```

Ou utiliser directement un build de docker-compose

```
version: '2'

services:
    site:
      build: .
      image: new_image:new_tag
      ports :
       - "8080:80"
      volumes:
```

Puis lancer la commande :

```
#Dans le même dossier que le fichier "docker-compose.yml"
docker-compose build
```

Je peux alors vérifier les changements apportés à mon image avec la commande "docker history" :

```
#Voir l'historique de changement d'une image
docker history new_image:new_tag
```

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

1. Introduction générale
   Conteneurs et VMs
2. Utilisation basique
   docker run
3. Utilisation simplifiée
   docker-compose
4. Utilisation avancée
   (logs, données, réseaux, ressources, etc)
5. Création d'image
   (Commit et Dockerfile)
6. **Aller plus loin**
   (Accéder à des devices, l'orchestration etc)

Accéder à des devices

Il est possible d'ouvrir votre docker à différents devices de votre hôte grâce à l'option "--device" de la commande "docker run"

Notamment des clés USB :

```
#Connecter un lecteur USB à un conteneur
docker run -t -i --device=/dev/ttyUSB0 mon_image
```

Ou de connecter la sortie graphique de votre hôte sur un conteneur. Par exemple, pour lancer un programme en mode isolé (exemple ici pour Firefox et NetBeans)

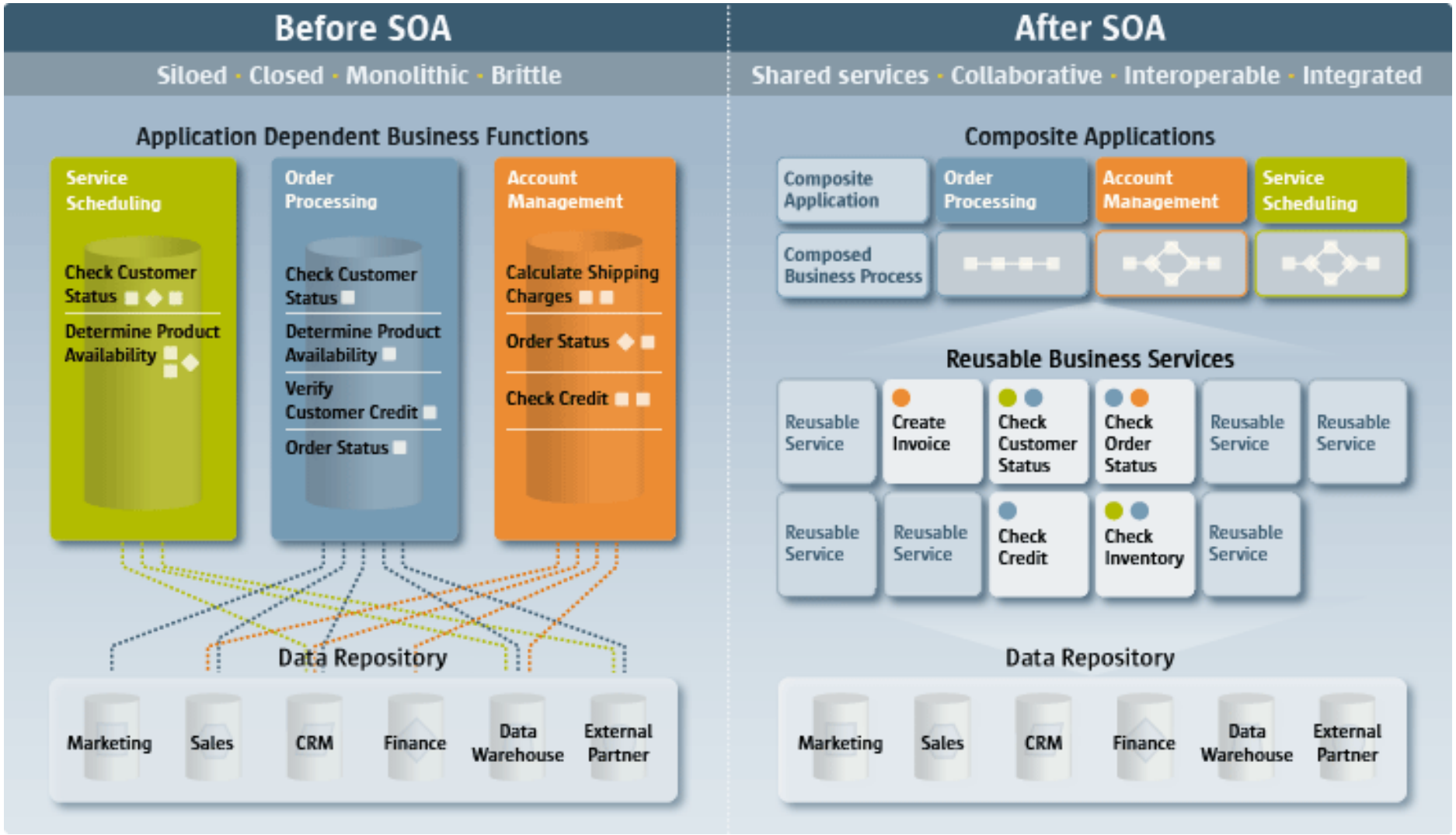
```
#Image contenant Netbeans et s'affichant comme
#une application classique
docker run -ti --rm \
          -e DISPLAY=$DISPLAY \
          -v /tmp/.X11-unix:/tmp/.X11-unix \
          -v `pwd`/.netbeans-docker:/home/developer/.netbeans \
          -v `pwd`:/workspace \
          fgrehm/netbeans:v8.0.1
```

Ou encore créer un Docker pour jouer avec les GPIO d'une Raspberry...

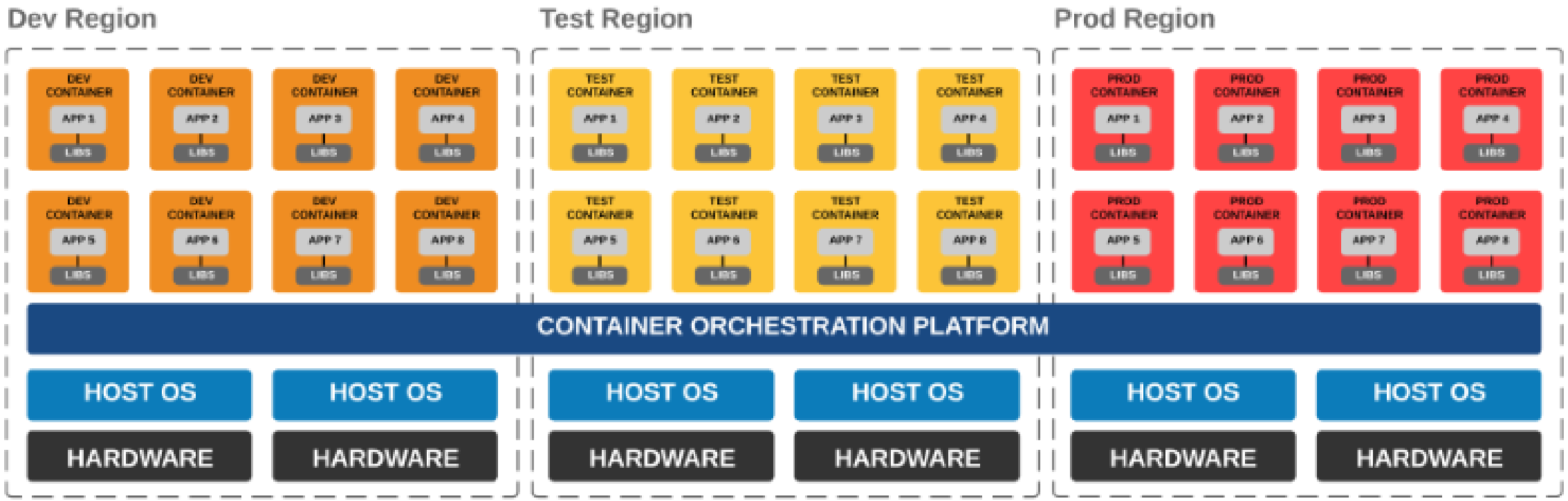
```
#Donner accès aux GPIO d'une Raspberry
docker run --device /dev/gpiomem mon_image
```

L'Orchestration -- Ecosystème Docker

L'Orchestration -- Pourquoi ?



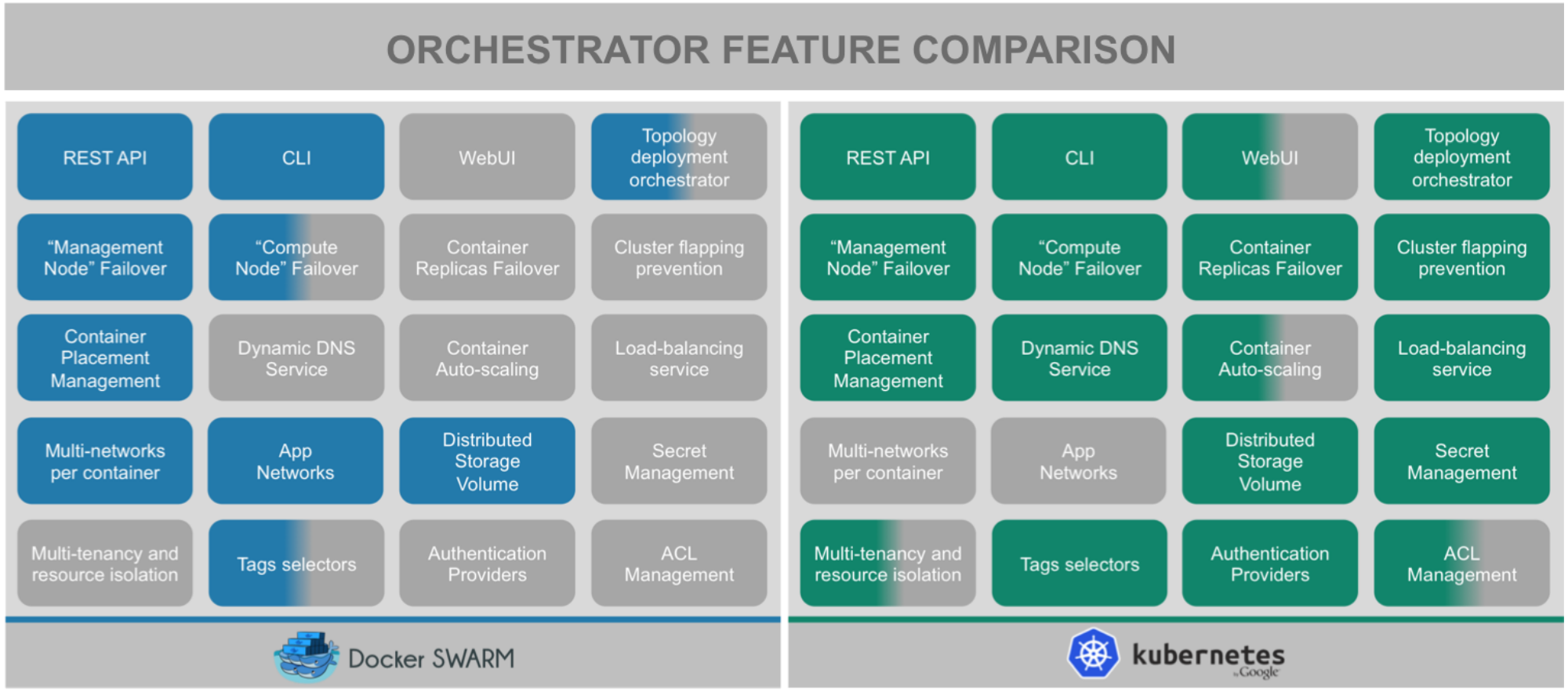
L'Orchestration -- Objectifs



L'Orchestration -- Exemples

Plusieurs technologies s'affrontent, en particulier :

- Docker Swarm, créé par Docker
- Google Kubernetes, créé par Google

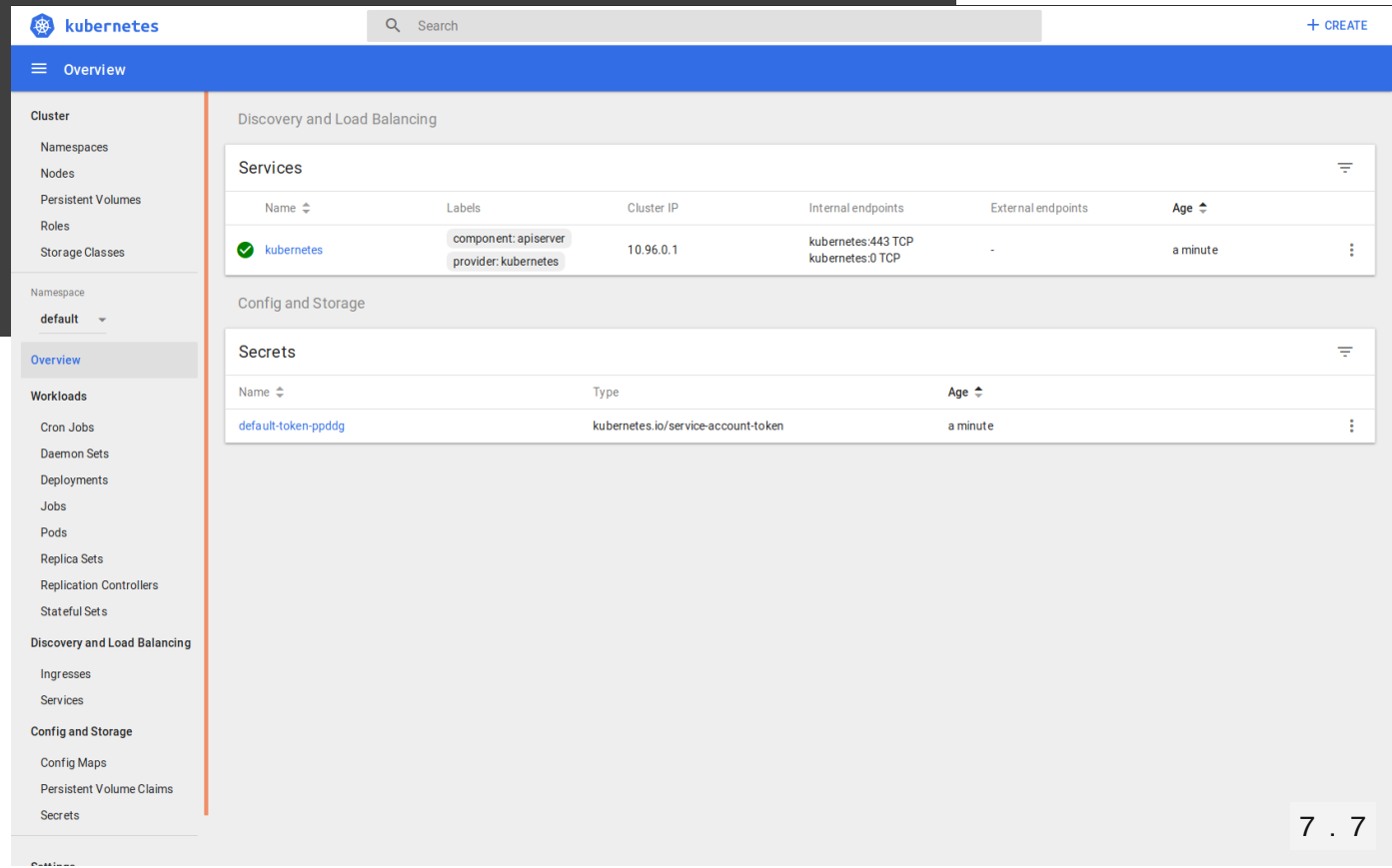


Comparaison datant de 03/2016

L'Orchestration -- Exemples

Google Kubernetes peut être testé grâce à "Minikube".
Un tutoriel est disponible ici. En résumé, sur Linux :

```
#Téléchargement de Minikube
curl -Lo minikube https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64 && \
    chmod +x minikube && \
    sudo mv minikube /usr/local/bin/

#Lancement de Minikube
minikube start --vm-driver=none

#Lancement du Dashboard
#L'adresse s'affichera a la fin de la commande
minikube dashboard
```

Tester le déploiement d'un "nginx" :

```
#Déploiement d'un nginx
kubectl apply -f https://k8s.io/docs/tasks/run-application/deployment-update.yaml
```