

# Docker - Mise en œuvre du déploiement d'applications en conteneur



Support de cours

Réf. T115-172



Module 0

# A propos de cette formation

# Votre formateur

- Son nom
- Ses activités
- Ses domaines de compétence
- Ses qualifications et expériences pour ce cours

# Votre formation - Présentation

- Description
  - Docker est une plate-forme de conteneurs logiciels virtualisés, qui permet d'empaqueter des applications et leurs dépendances systèmes, afin de les exécuter sur n'importe quel serveur Linux ou Windows.
- Profil des stagiaires
  - Développeurs, architectes techniques, administrateurs et responsables d'exploitation et de production, chefs de projet.
- Connaissances préalables
  - Notions sur les réseaux TCP/IP
  - Utilisation de la ligne de commande et du script Shell en environnement Linux
- Objectifs à atteindre
  - Installer Docker sous Linux et Windows
  - Travailler avec des conteneurs et images
  - Construire des images et les publier dans un registre
  - Configurer le réseau et les volumes
  - Orchestrer des conteneurs avec Docker Compose
  - Assurer la disponibilité des conteneurs avec Docker Swarm

# Votre formation - Programme

- Module 1 : Introduction & Installation
- Module 2 : Docker en production
- Module 3 : Conception de conteneur
- Module 4 : Exploitation de Docker
- Module 5 : Chainage de conteneurs avec Docker Compose
- Module 6 : Orchestration de conteneurs avec Docker Swarm

# Votre formation - Ressources à votre disposition

- Le présent support de cours
- La plateforme technique
  - 4 VM et un réseau virtuel
  - Outils : Putty, Visual Studio Code, ...

# Tour de table - Présentez-vous

- Votre nom
- Votre société
- Votre métier
- Vos compétences dans des domaines en rapport avec cette formation
- Les objectifs et vos attentes vis-à-vis de cette formation

# Logistique

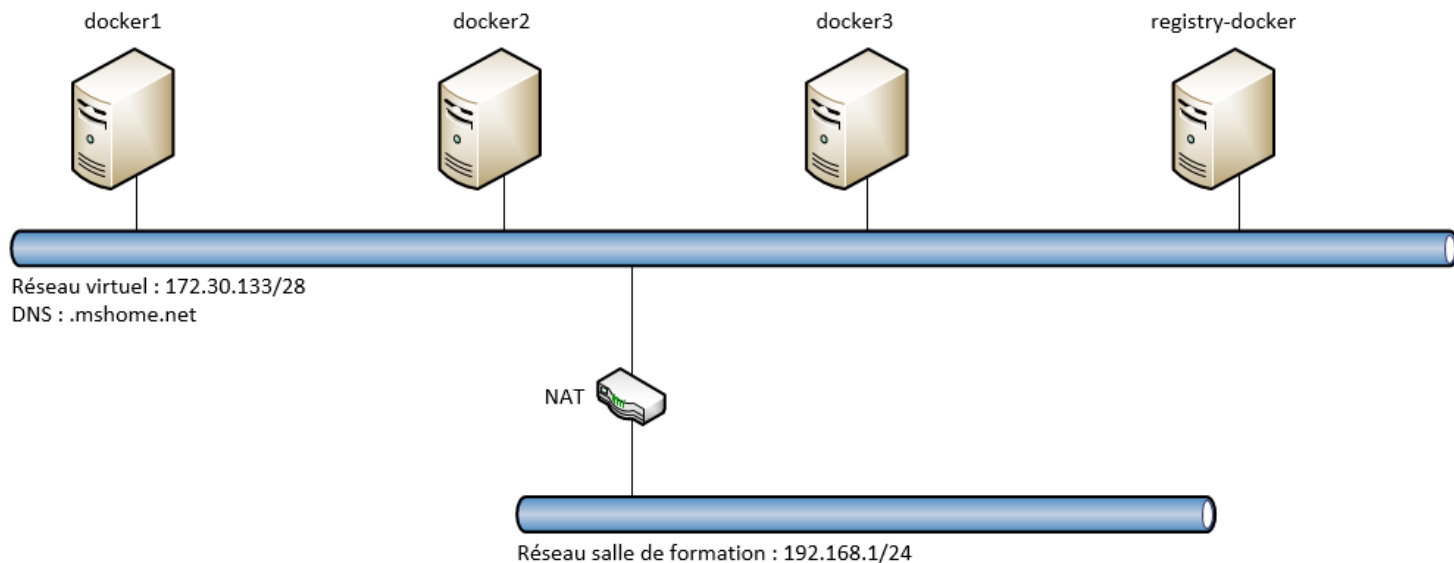
- Horaires de la formation
  - 9h00-12h30
  - 14h00-17h30
- Pauses

Merci d'éteindre vos téléphones portables



# La plateforme technique de formation

Machines Virtuelles Hyper-V (Ubuntu)





# Module 1

## Introduction & Installation

# Contenu du module

- Présentation
- Motivations
- Architecture
- Installation

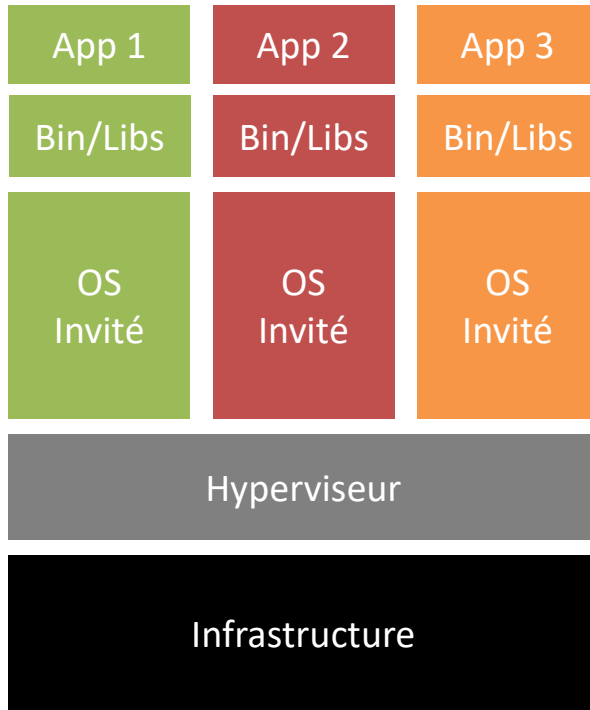
# Présentation

- Docker est une plate-forme de conteneurs logiciels virtualisés qui permet d'empaqueter des applications et leurs dépendances systèmes afin de les exécuter sur n'importe quel serveur Linux ou Windows.
- On parle alors de « container ».
  - Technique qui permet d'exécuter des processus de façon « isolée »
  - Fonctionnalité de LXC (container Linux Standard) dont Docker étend le format grâce à une API de haut niveau
- Docker utilise LXC, les cgroup, et le noyau Linux.
- Sous Windows, des fonctionnalités bas niveau équivalentes existent depuis Windows Server 2016.
- L'approche d'une architecture applicative en containers est celle des micro-services.

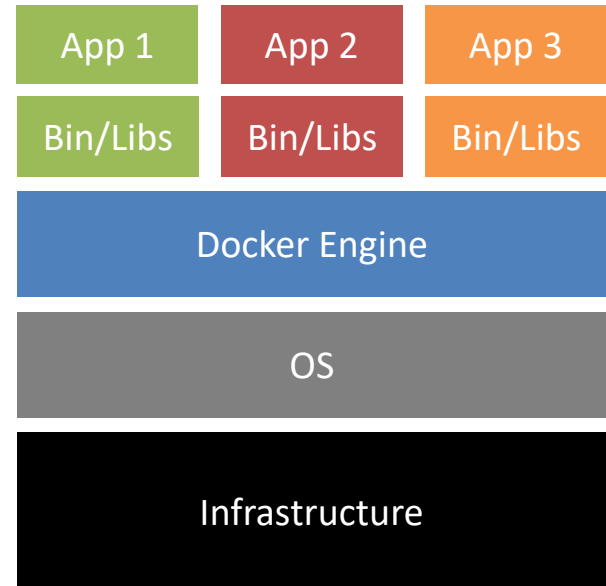
# Motivations

- Fourniture d'un environnement équivalent aux machines virtuelles (isolation, configuration...) mais moins volumineux en espace disque
  - Seules l'application et ses dépendances (autres applications et bibliothèques) sont dans l'image Docker.
- Invariance
  - Chaque exécution d'un container depuis la même image crée les mêmes processus dans le même état de départ (si on n'utilise pas de volume).
- Environnement de développement/test plus facile à distribuer que les VM
- En production
  - Déploiement, initialisation et mise à jour de version rapide
  - Docker est souvent présent dans une approche « DevOps »
  - Mise à l'échelle (augmentation/réduction du nombre d'instances d'un container) simple dans un cluster (Swarm)

# Architecture – Virtualisation vs. Conteneurs

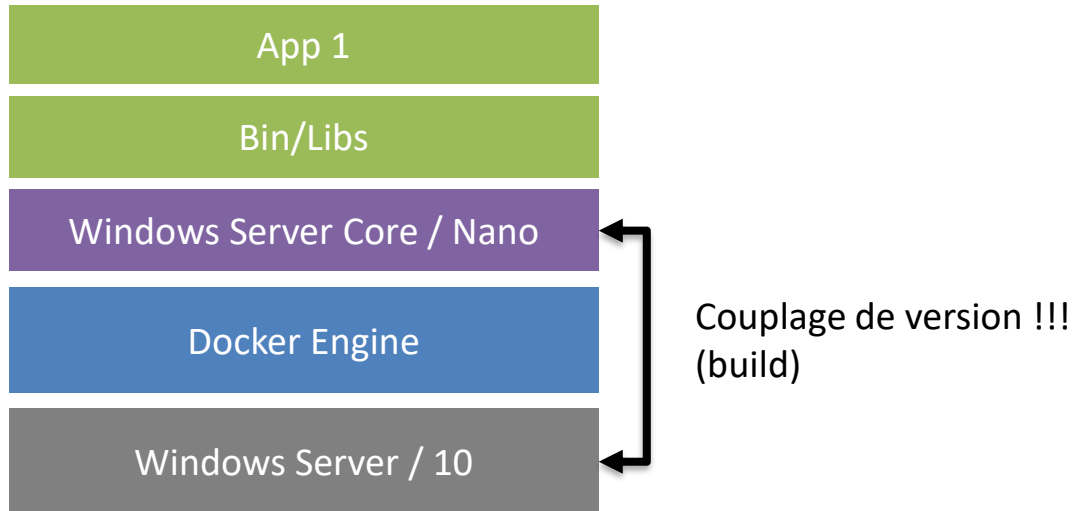


Virtualisation



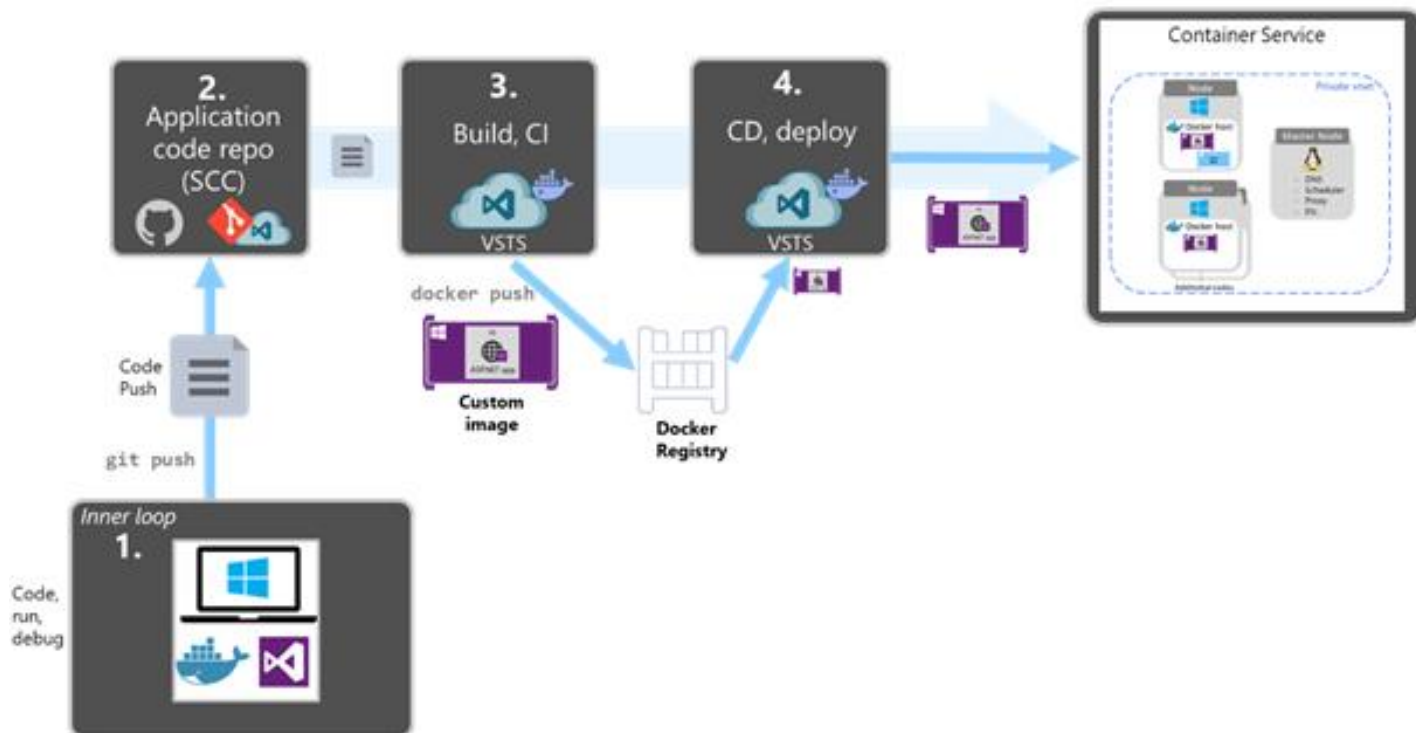
Conteneurs

# Architecture – Conteneurs Windows sous Windows



- Nano Server
  - Pas de GUI.
  - Pas de Framework .NET mais .NET Core.
  - N'est plus un OS, il est réservé à Docker.

# DevOps





# Installation

- Procédure :  
<https://docs.docker.com/engine/install/>
- Linux : Support de multiples distributions
- Windows : Server 2016 ou Windows 10 Build 15063 ou + (pas Home Edition)
  - 2 modes de fonctionnement :
    - Container Windows sous Windows (natif).
    - Container Linux
      - WSL 2
      - VM HyperV
- Possibilité d'installer le client uniquement.
  - Commande docker
- Docker Compose s'installe séparément.
  - <https://docs.docker.com/compose/install/>

# Docker Desktop

- Docker Desktop est un package d'installation tout en un destiné à déployer Docker
- Il est disponible pour :
  - Windows
  - macOS
    - C'est la technique d'installation recommandée pour ces 2 systèmes d'exploitation
  - Linux
    - Nouveau ! (mai 2022)
    - Mais l'environnement Docker tourne dans une VM !
      - Performances dégradées...
    - On préférera installer le démon Docker sur ces systèmes



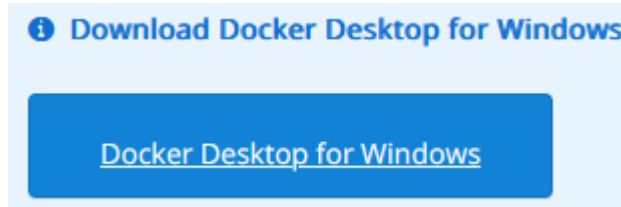
Commercial use of Docker Desktop in larger enterprises (more than 250 employees OR more than \$10 million USD in annual revenue) now requires a paid subscription.

# Installation sous Linux

- Installation du démon Docker selon sa distribution :
  - <https://docs.docker.com/engine/install/>
- Des étapes supplémentaires sont nécessaires après l'installation.
  - Exécuter les commandes Docker en tant qu'utilisateur standard (non-root).
  - Lancer le moteur Docker au démarrage.
- Procédure :
  - <https://docs.docker.com/engine/install/linux-postinstall/>

# Installation sous Windows – WSL 2

- <https://docs.docker.com/desktop/windows/install/>

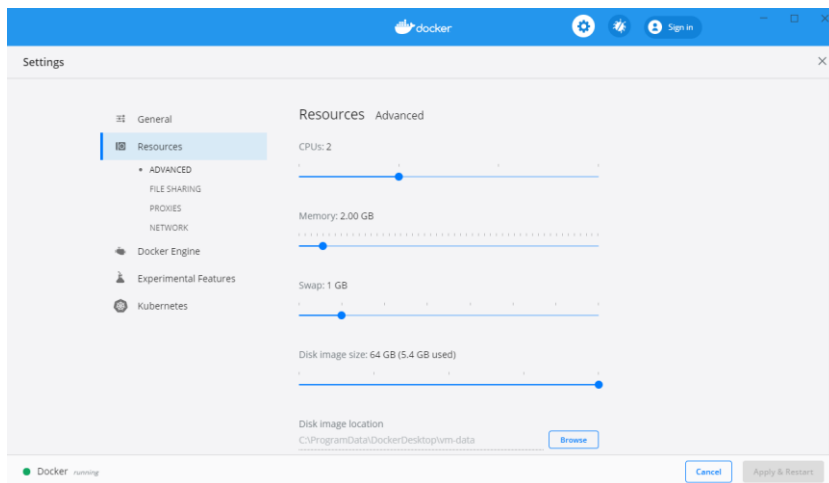


- Sous Windows, Docker peut profiter des fonctionnalités du noyau Linux grâce à WSL 2 (Windows Subsystem for Linux)
  - Windows 10, build 18362.1040+ or higher / Windows 11
  - L'installation proposera d'activer la fonctionnalité si elle n'est pas active
  - Il pourra être nécessaire d'installer un package de mise à jour
    - <https://docs.microsoft.com/windows/wsl/wsl2-kernel>
- Sur les versions antérieures, Docker fonctionne dans une machine virtuelle Hyper-V

☐ Use the WSL 2 based engine (requires Win 10 build 18362.1040+)  
WSL 2 provides better performance than the legacy Hyper-V backend. [Learn more.](#)

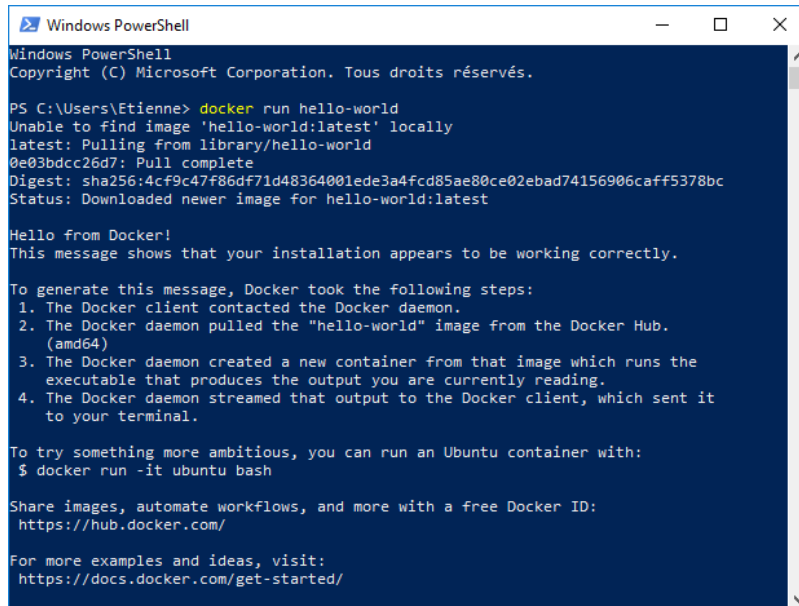
# Installation sous Windows – *Hyper-V backend*

- L'installation de Docker Desktop sous Windows crée une nouvelle VM Hyper-V nommée **DockerDesktopVM**.
  - Elle permet l'exécution de conteneurs Linux sous Windows.
- Les propriétés de cette VM peuvent être ajustées via le menu contextuel « *Settings* » de l'icône Docker situé dans la zone de notification de Windows.
- **Elles ne doivent pas être modifiées via l'interface d'Hyper-V !!!**



# Vérifier l'installation

- Commande :
  - `docker run hello-world`



```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

PS C:\Users\Etienne> docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete
Digest: sha256:4cf9c47f86df71d48364001ede3a4fcd85ae80ce02ebad74156906caff5378bc
Status: Downloaded newer image for hello-world:latest

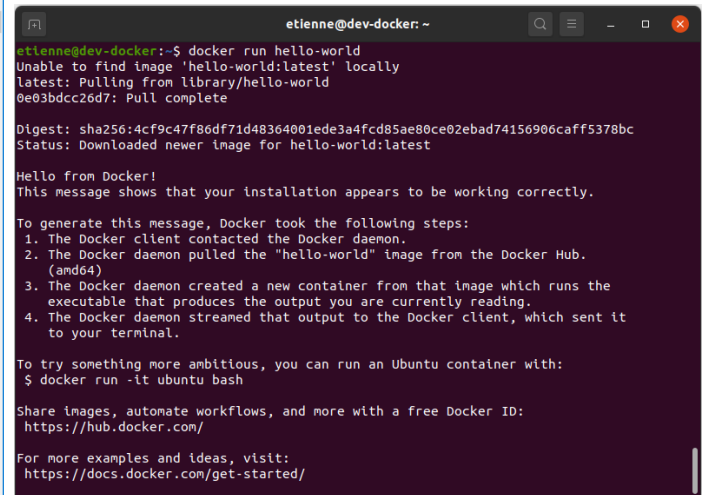
Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```



```
etienne@dev-docker: ~
etienne@dev-docker:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
0e03bdcc26d7: Pull complete

Digest: sha256:4cf9c47f86df71d48364001ede3a4fcd85ae80ce02ebad74156906caff5378bc
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```

# Outillage pour Docker

- Visual Studio Code
  - Permettra de créer et d'éditer les fichiers `Dockerfile` et les fichiers `docker-compose.yml`.
  - Terminal intégré.
  - Possède des extensions pour Docker !
    - **Docker**
      - Pour gérer les images et conteneurs locaux.
    - **Remote Container**
      - Manipuler les fichiers à l'intérieur d'un conteneur
- Un émulateur de terminal (sous Windows)
  - Putty
  - MobaXterm

# Travaux Pratiques







# Module 2

## Docker en production

# Contenu du module

- Les commandes Docker
- Images, Conteneurs et Registre
- Etats d'un container
- Nommage
- Stockage

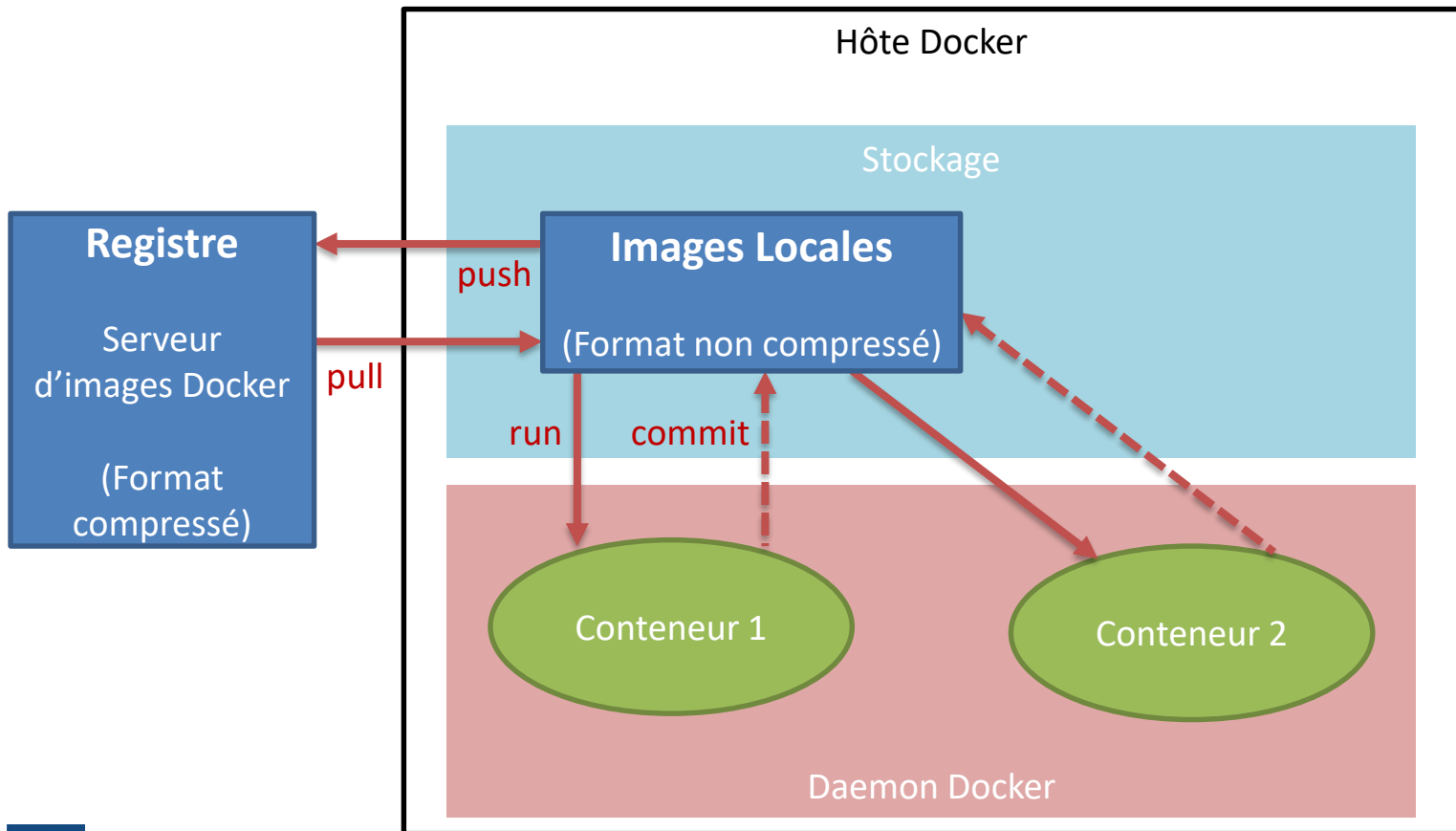
# Les commandes Docker

- Docker s'utilise en ligne de commande
  - Shell sous Linux, Power Shell sous Windows
    - Complétion des commandes natives sur un shell Bash
    - Projets disponibles pour Power Shell
      - <https://github.com/matt9ucci/DockerCompletion>
      - <https://github.com/samneirinck/posh-docker>
- Usage :
  - `docker [options] commande`
- Aide :
  - `docker --help`
- Aide sur les commande :
  - `docker <commande> --help`

# Images, Conteneurs et Registre

- Les images
  - Une image Docker est un fichier statique non modifiable.
  - L'image inclut les bibliothèques, les outils système et la configuration nécessaires à l'exécution d'un processus.
  - Une image est stockée sur un registre.
- Les registres
  - Les registres Docker sont des serveurs d'images.
  - Ils stockent les images au format compressé.
  - Docker héberge un tel registre en ligne : Docker Hub (<https://hub.docker.com>)
    - Référence de base.
  - Il est possible de créer un registre privé pour héberger ses propres images.
- Les conteneurs
  - Ils sont créés à partir des images.
  - Ce sont des instances actives modifiables des images.
  - Ils exécutent des processus.

# Images, Conteneurs et Registre - Commandes

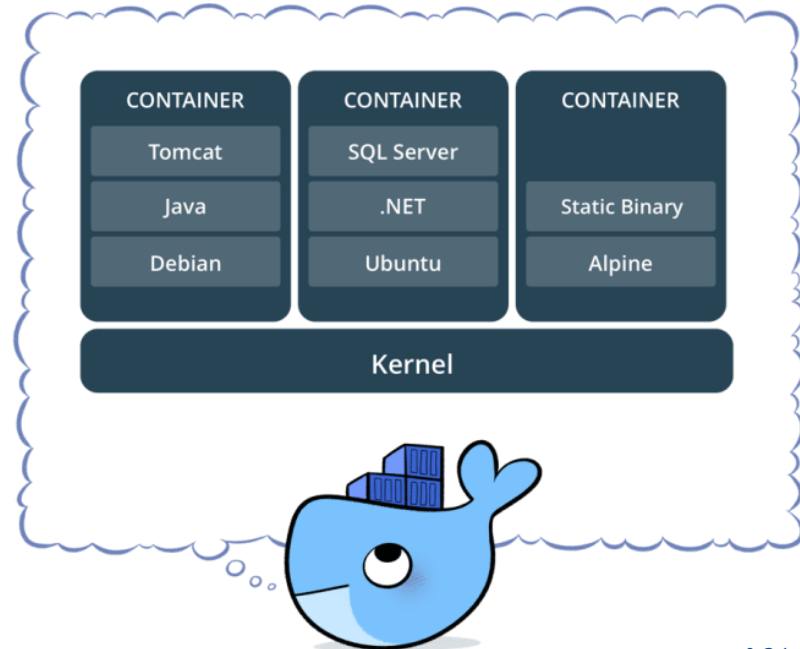


# Les images – Résumé des commandes

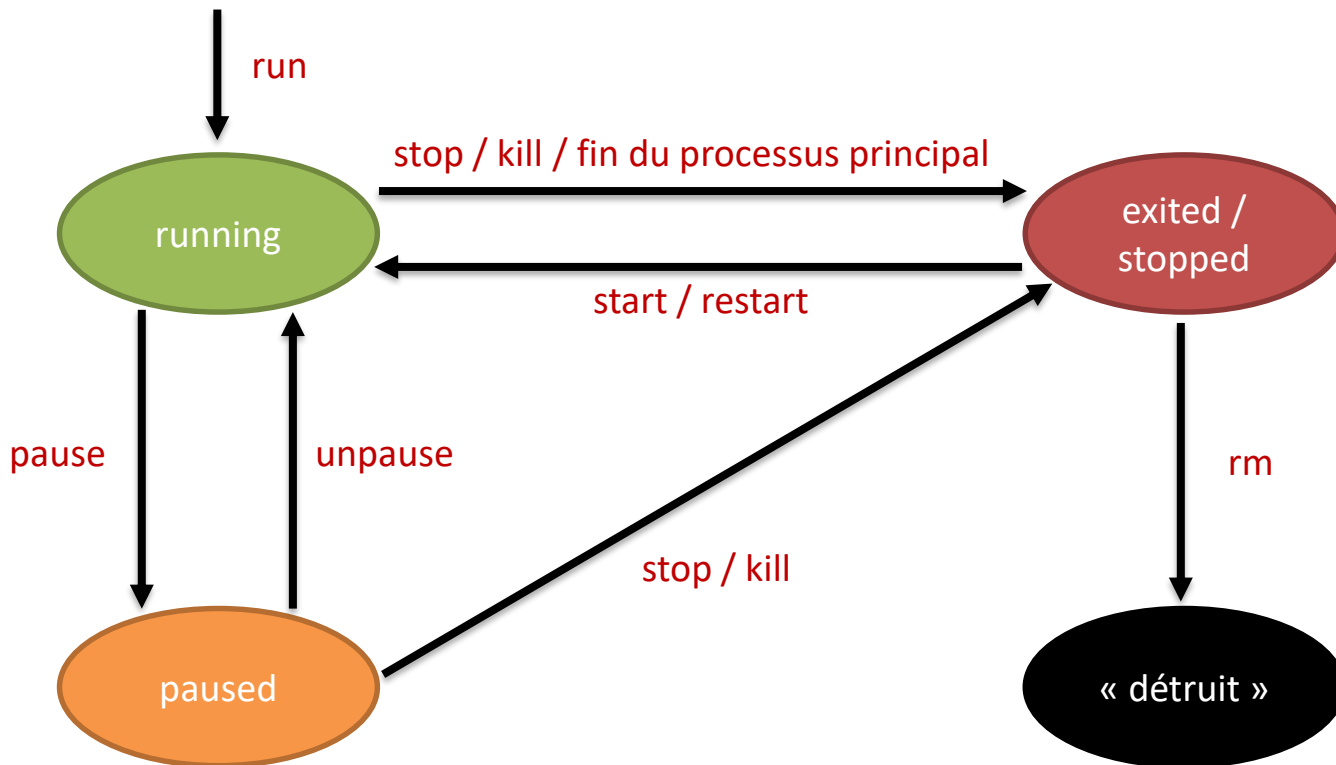
- `docker images [-a]`
  - Lister les images locales.
  - L'option `-a` permet d'afficher également les images intermédiaires (liées aux constructions)
- `docker rmi <image>`
  - Supprimer une image locale.
- `docker pull <image>`
  - Récupérer une image en local depuis un registre.
- `docker push <image>`
  - Publier une image locale sur un registre.
- `docker inspect <image>`
  - Obtenir des informations de bas niveau sur la configuration de l'image.

# Les conteneurs

- Ils contiennent :
  - Une base de système d'exploitation partageant un noyau.
  - Des bibliothèques et dépendances.
  - La/Les application(s) à exécuter.



# Les états d'un container (et commandes associées)



**restart** fonctionne aussi sur un container « running »



# Création d'un conteneur

- La commande `docker run` permet de créer et démarrer un conteneur à partir d'une image.
- Au démarrage, le conteneur exécute son processus principal (« entryptpoint »), selon le conteneur et son processus, il peut être nécessaire de :
  - Démarrer le conteneur en mode interactif pour obtenir un shell dessus.
    - `docker run -it <image>`
  - Démarrer en mode « daemon » pour pouvoir récupérer la main sur le terminal depuis lequel le conteneur est lancé.
    - `docker run -d <image>`
- Il est également possible que le conteneur soit automatiquement détruit après son arrêt.
  - Lorsqu'un conteneur est utilisé pour exécuter des tests logiciels par exemple.
    - `docker run --rm <image>`

# Redémarrage automatique d'un conteneur

- Docker fournit des politiques de redémarrage pour contrôler si les conteneurs démarrent automatiquement à leur sortie ou au redémarrage de Docker (ou de la machine hôte)
  - Ex. Si un conteneur est en cours d'exécution au moment de l'arrêt de l'hôte, il pourra être redémarré automatiquement lorsque l'hôte est redémarré
- Pour configurer la stratégie de redémarrage d'un conteneur, il faut utiliser l'option `--restart` lors de l'utilisation de la commande `docker run`
  - L'option peut également être utilisée avec la commande `docker update` pour mettre à jour la stratégie
- Valeurs possibles pour l'option :

Valeur	Description
<b>no</b>	Ne redémarre pas automatiquement le conteneur. (Par défaut)
<b>on-failure[:max-retries]</b>	Redémarre le conteneur s'il s'arrête en raison d'une erreur (Code de sortie différent de zéro). On peut limiter le nombre de tentatives de redémarrage avec l'option <code>:max-retries</code> .
<b>always</b>	Redémarre systématiquement le conteneur s'il s'arrête.
<b>unless-stopped</b>	Comme pour <b>always</b> , sauf que lorsque le conteneur est arrêté manuellement, il n'est pas redémarré même après le redémarrage du démon Docker.

# Un conteneur démarré...

- Lors de la création d'un conteneur, un nom par défaut lui est attribué.

```
etienne@dev-docker: ~  
etienne@dev-docker:~$ docker run -d httpd  
49664aae34a2a6b6abfe33cf58c41c86e23ad999cbd3ead48153a5e138e61a6d  
etienne@dev-docker:~$ docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES  
49664aae34a2   httpd     "httpd-foreground"      3 seconds ago Up 2 seconds  80/tcp       reverent_dubinsky  
etienne@dev-docker:~$
```

- Un nom plus explicite peut lui être donné avec l'option `--name` de la commande `run`.

```
etienne@dev-docker: ~  
etienne@dev-docker:~$ docker run -d --name mon-apache httpd  
053e1a9dbcf6ad56e3ce9fb06f7fd99347c0789448a529b2b67162db2cc21207  
etienne@dev-docker:~$ docker ps  
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS        NAMES  
053e1a9dbcf6   httpd     "httpd-foreground"      3 seconds ago Up 2 seconds  80/tcp       mon-apache  
etienne@dev-docker:~$
```

# Arrêt et suppression d'un conteneur

- Lorsque le processus principal d'un conteneur se termine, le conteneur se termine avec.
  - Il apparaît « **Exited** »
- Il peut être :
  - Relancé avec `docker start` ;
  - Supprimé avec `docker rm` ;
  - Supprimé automatiquement s'il a été démarré avec l'option `--rm`.
- Lorsqu'un conteneur est créé avec `docker run`, il l'est avec l'état de départ défini dans l'image (système de fichiers, processus, ...).
  - Tous les conteneurs créés avec une image sont identiques au départ.
  - Tant qu'un conteneur n'est pas supprimé, ses modifications internes sont conservées.

# Les conteneurs – Commandes du cycle de vie

- `docker run <image>`
  - Créer et démarrer un conteneur.
- `docker start <conteneur>`
  - Démarrer un conteneur.
- `docker stop <conteneur>`
  - Arrêter un conteneur.
- `docker restart <conteneur>`
  - Redémarrer un conteneur.
- `docker kill <conteneur>`
  - Tuer un conteneur.
- `docker pause <conteneur>`
  - Mettre un conteneur en pause.
- `docker unpause <conteneur>`
  - Réactiver un conteneur en pause.
- `docker rm <conteneur>`
  - Supprimer un conteneur (arrêté).

# Les conteneurs – Commandes de gestion

- `docker ps [-a]`
  - Visualiser la liste des conteneurs actifs.
  - L'option `-a` permet également d'afficher les conteneurs arrêtés.
- `docker top <conteneur> <options>`
  - Visualiser les processus actifs d'un conteneur.
  - Docker envoie une commande `ps` au conteneur. Les options servent à transmettre les options à `ps`.
- `docker logs <conteneur>`
  - Voir les commandes (et leur résultats) passées dans le conteneur.
- `docker inspect <conteneur>`
  - Obtenir des informations de bas niveau (configuration et runtime) sur le conteneur.
- `docker exec [options] <conteneur> <commande>`
  - Exécuter une commande dans un conteneur.
  - Options `-it` pour exécuter un shell (Bash)

# Le nommage dans Docker

- Tout objet Docker (container, image, volume, réseau...) possède un ID unique (son empreinte SHA).  
C'est le seul élément obligatoire pour une image, les autres objets ont un nom.
- Les images
  - Elles peuvent posséder un nom sous la forme `repo:tag`
  - ⚠️ ▪ Si le tag n'est pas précisé il vaut alors « latest »
    - A ne jamais utiliser en production, les versions bougent !
  - Le tag sert (par convention) à préciser un environnement et/ou la version
  - Exemples :
    - `hello-world:latest`
    - `jcd717/mon-site:latest`
    - `10.1.2.3:5000/image1:alpine-1.0`
    - `registry.eni.fr/image2:busybox-1.1` (port par défaut=443)

# Le nommage des images

- Les images Docker sont associées à un ID unique, mais plus communément, elles sont qualifiées par un nom qui prend la forme suivante :
  - registre/nom:tag
- Le nom et le tag de l'image servent à l'identifier lors de la création de conteneurs par exemple.
- Le registre permet d'indiquer dans quel registre elle sont publiées et donc disponible.
  - Par défaut le registre est Docker Hub.
- Exemple :
  - `registry.eni.fr/image2:busybox-1.1`

Registre                  Nom                  Tag



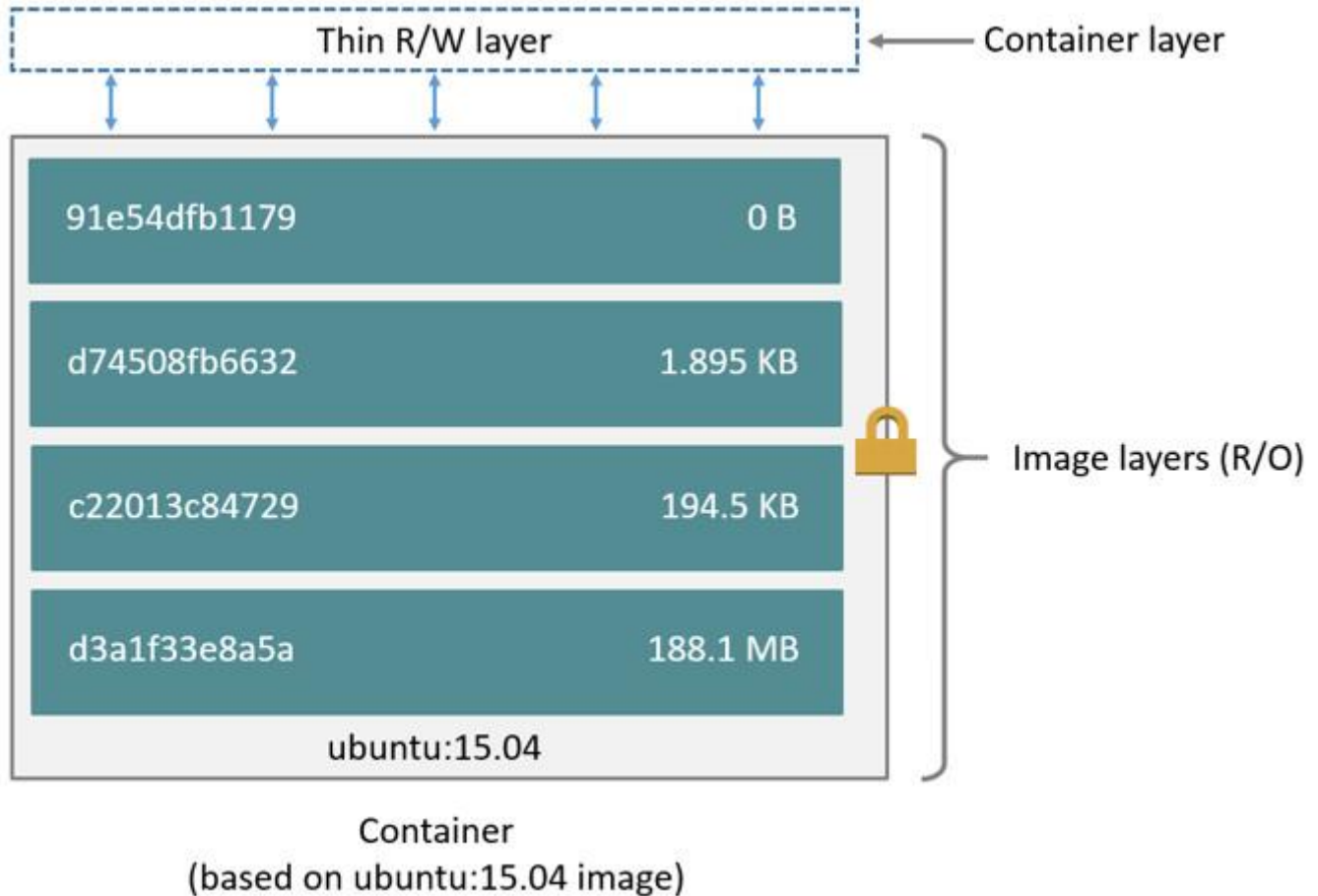
# Le renommage des images

- La commande `docker tag` permet de modifier le nom d'une image.
- Cela est nécessaire :
  - Pour ajouter une information de registre afin de la publier ;
  - Pour appliquer un tag différent et suivre des évolutions de versions ;
  - ...
- `docker tag <source image> <target image>`
  - L'image source est dupliquée et un nouveau tag est appliqué à la cible.
  - L'image source peut éventuellement être supprimée.

# Le stockage

- Chaque image est constituée de couches (« *layers* ») en lecture seule.
- Chaque couche possède un ID unique.
- Le contenu d'une couche n'est jamais dupliqué dans un moteur de stockage Docker.
  - Plusieurs images peuvent donc avoir des couches en commun.
- Un container crée une nouvelle couche en lecture/écriture au-dessus de celles de son image.
  - La commande `docker commit` copie cette couche, la positionne en lecture seule et l'intègre à la nouvelle image.
- Les couches sont stockées avec un moteur de stockage nommé `overlay2`.

# Stockage



# Visualisation des couches

- **Dive** est un outil permettant de visualiser les différentes couches d'une image Docker.
  - <https://github.com/wagoodman/dive>
  - Peut être lancé dans un conteneur ! 😊
    - `docker run --rm -it -v /var/run/docker.sock:/var/run/docker.sock wagoodman/dive:latest <dive arguments...>`

```
1/1 + + + Titix Default
Terminal -
[Layers]
Cmp Image ID Size Command [Current Layer Contents]
sha256:cd710ba72410606589 4.1 MB FROM sha256:cd710ba72410606589 drwxr-xr-x 0:0 805 kB --@ bin
sha256:f03b1ccbacac8c82e 2.1 kB #(nop) ADD file:63d4894bd0857354b drwxr-xr-x 0:0 0 B --@ dev
sha256:62820523d8d8c4350 0 B mkdir /root/example drwxr-xr-x 0:0 251 kB --@ etc
sha256:9f95061518484bb62a 2.1 kB cp /somefile.txt /root/example/so drwxr-xr-x 0:0 0 B --@ home
sha256:edf209d9263ab03c7 2.1 kB cp /somefile.txt /root/example/so drwxr-xr-x 0:0 2.7 MB --@ lib
sha256:bb70c7aab83602e8a6 2.1 kB cp /somefile.txt /root/example/so drwxr-xr-x 0:0 0 B --@ media
sha256:bba712d3d80c83c48 2.1 kB mv /root/example/somefile3.txt /r drwxr-xr-x 0:0 0 B --@ cdrom
sha256:05e8d6d8f9e77b03 0 B rm -rf /root/example/ drwxr-xr-x 0:0 0 B --@ floppy
drwxr-xr-x 0:0 0 B --@ usb
[Image 8 Layer Details]
Layer Command drwxr-xr-x 0:0 0 B --@ mnt
/bin/sh -c #(nop) ADD file:63d4894bd0857354beaed303367ca145cabd131e8583 dr-xr-xr-x 0:0 0 B --@ proc
37d8d4dca2d6e7103d in /somefile.txt drwxr-xr-x 0:0 0 B --@ root
drwxr-xr-x 0:0 0 B --@ run
drwxr-xr-x 0:0 213 kB --@ sbin
-rw-rw-r-- 0:0 2.1 kB --@ somefile.txt
drwxr-xr-x 0:0 0 B --@ srv
drwxr-xr-x 0:0 0 B --@ sys
drwxrwxrwx 0:0 0 B --@ tmp
drwxr-xr-x 0:0 176 kB --@ usr
drwxr-xr-x 0:0 0 B --@ var
Image efficiency score: 99 %
Potential wasted space: 6.2 kB
Count Total Space Path
2 4.2 kB /root/example
2 2.1 kB /root/example/somefile3.txt
```

# Application des modifications dans une image

- Lorsqu'un le contenu d'un conteneur est modifié, il peut être intéressant de garder ces modifications dans une nouvelle image.
  - Pour permettre de créer de nouveaux conteneurs sur cette base.
  - C'est un premier moyen de créer des images personnalisées !
    - Les fichiers Dockerfile seront un moyen plus souple.
- La commande `docker commit` permet de créer une image à partir d'un conteneur existant.
  - `docker commit [options] <container> <repository/image:tag>`
- Les couches R/W du conteneur sont transformées en couches RO et ajoutées à l'image.

# Import / Export d'images et conteneurs

- Sauvegarder et restaurer des images :
  - Sauvegarder une image dans une archive (.tar.gz)
    - `docker save -o <fichier.tar.gz> <image>`
  - Restaurer une image depuis une archive (.tar.gz)
    - `docker load -i <fichier.tar.gz>`
- Avec des conteneurs :
  - Exporter un conteneur dans une archive d'image
    - `docker export -o <fichier.tar.gz> <container>`
  - Restaurer une image créée à partir d'un conteneur
    - `docker import <fichier.tar.gz> <repository/nom:tag>`

# Les registres

- Un registre est un serveur de stockage et de diffusion d'images Docker
  - Il peut être implémenté en tant que serveur On-Premise ou dans le Cloud
- Docker fournit un registre en ligne contenant une multitudes d'images, officielles ou non
  - Docker Hub : <https://hub.docker.com>
  - Un compte avec souscription permettra de contourner les limitations de nombres de téléchargement d'images
  - Pour s'authentifier avec son compte sur un registre, on utilise la commande `docker login <registre>`
- Excellente alternative à Docker Hub (sans limitations)
  - <https://quay.io/> (RedHat)
- Pour implémenter un registre On-Premise, Docker fournit une image permettant de créer un registre dans un conteneur !
  - [https://hub.docker.com/\\_/registry](https://hub.docker.com/_/registry)

# Fin du module

- Commandes Docker abordées :
  - pull, images, rmi, tag
  - run, ps, top
  - inspect
  - commit
  - stop, start, restart, kill, pause, unpause
  - logs
  - history
  - exec
  - cp
  - load/save, import/export
- Configurations effectuées :
  - Lancement d'un service (Apache ou autre) en container
  - Visual Studio Code sous Windows vers hôte Docker sous Linux



# Travaux Pratiques



[www.eni-service.fr](http://www.eni-service.fr)



# Module 3

## Conception de conteneur

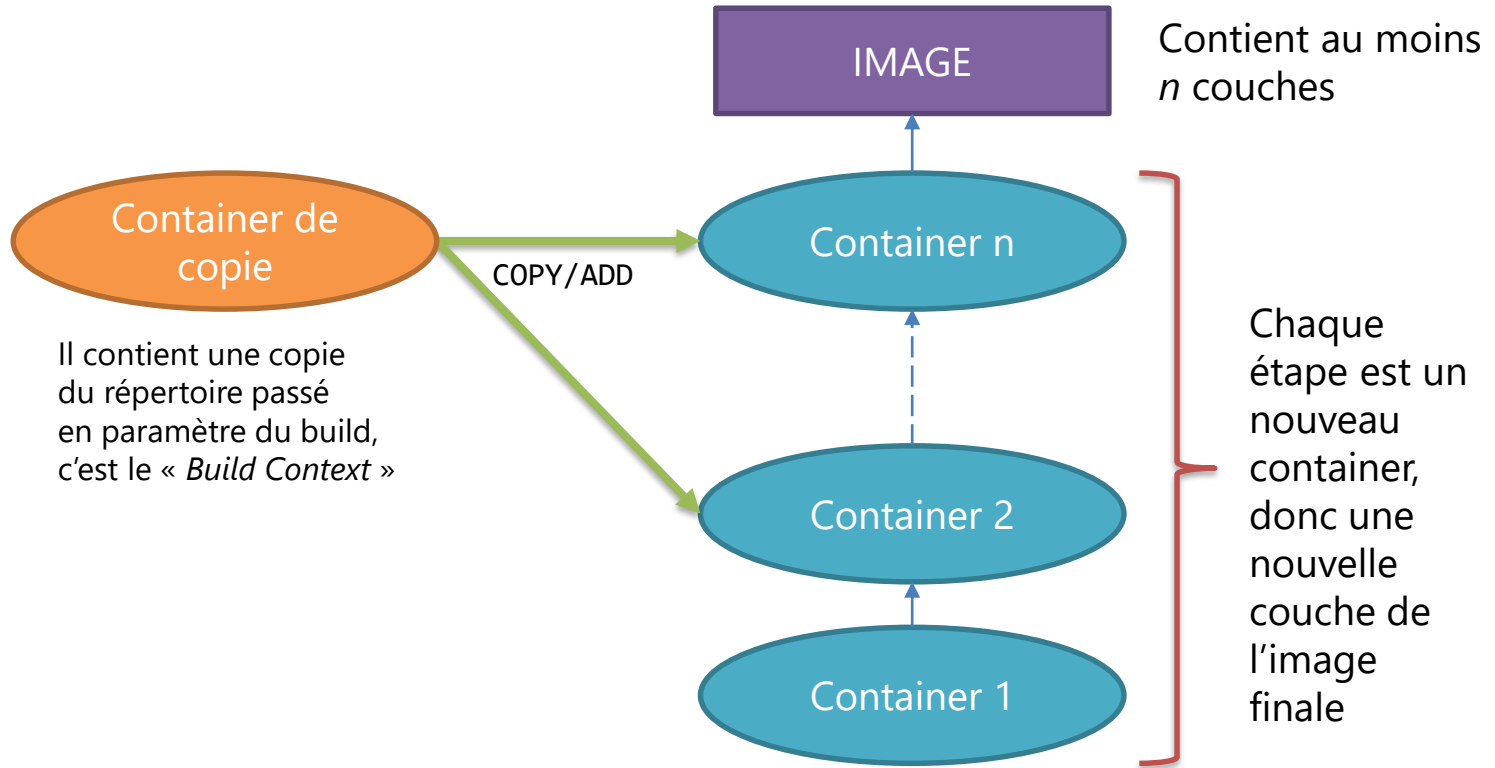
# Contenu du module

- Le Dockerfile
- Le processus de construction
- Structure d'un Dockerfile
- La construction – « build »
- Le Docker Hub et la construction automatique
  - Construction sur Docker Hub
  - Docker Hub et GitHub

# Dockerfile - Présentation

- En pratique, `docker commit` est à éviter pour créer une image personnalisée...
  - On ne garde aucune trace des modifications effectuées sur l'image de base !
  - Le processus est difficilement répétable
- La construction d'une image personnalisée se fera plutôt en utilisant un **Dockerfile**.
  - C'est le nom conventionnel du fichier de construction (*build*) d'une nouvelle image.
- Contient des instructions, écrites en majuscules par convention, permettant de décrire la réalisation de l'image.
  - La commande `docker build` permettra ensuite de construire l'image.

# Le processus de construction



Les *n* containers intermédiaires restent dans le cache (images cachées).

# Structure d'un Dockerfile

- Un Dockerfile contient des instructions de construction, elle sont conventionnellement écrites en majuscules.
  - <https://docs.docker.com/engine/reference/builder/>
- Les lignes commençant par # sont des commentaires.
- Instructions essentielles :
  - FROM
    - Indique l'image de base sur laquelle on s'appuie. En général, on part d'une image existante dans un registre et on la personnalise. Cette image de base contient le strict minimum d'un Linux par exemple. (On peut partir de zéro mais le processus est compliqué.).
  - COPY/ADD
    - Permet de copier/ajouter des fichiers locaux dans l'image au moment de la construction. Pour passer des fichiers de configuration, des archives, ...
    - ADD décompresse une archive source vers l'image.
  - RUN
    - Exécute une commande sur le conteneur de création de l'image.
  - ENTRYPOINT/CMD
    - La commande à exécuter lorsque le conteneur va démarrer.
  - ARG
    - Déclare un argument dont la valeur pourra être transmise lors du *build* (un numéro de version par exemple...)

# Autres instructions du Dockerfile

- Documentation du Dockerfile :
  - LABEL
    - Indique l'auteur, une URL, de la documentation,...
- Déclaration de ressources disponibles dans l'image :
  - EXPOSE
    - Spécifie les ports réseaux accessibles dans l'image.
  - VOLUME
    - Spécifie les volumes accessibles dans l'image.
  - ENV
    - Permet de déclarer une variable d'environnement
- Manipulation du conteneur intermédiaire lors de la création de l'image :
  - WORKDIR
    - Change le répertoire courant.
  - USER
    - Change l'utilisateur courant.

# La construction – « Build »

- La construction d'une image Docker à partir d'un Dockerfile se fait grâce à la commande `docker build`.
- Elle doit recevoir en paramètre l'emplacement du répertoire contenant le Dockerfile.
  - C'est le « *Build Context* ».
- Il est également nécessaire de spécifier le nom complet de l'image à produire : `<repository/nom:tag>`
  - Option `-t`
- Exemple :
  - `docker build -t monimage:1.0 docker-project/`
- Les différentes étapes du Dockerfile donneront lieu à autant de couches intermédiaires dans l'image.
  - Pertinence de regrouper des instructions pour limiter le nombre de couches
  - Les couches déjà existantes ne sont pas recréées -> système de cache.



# Dockerfile - Bonnes pratiques

- Regrouper (si possible) les instructions :

```
RUN mkdir /data  
RUN chmod 0700 /data
```

*devient*

```
RUN mkdir /data ; chmod 0700 /data  
ou  
RUN mkdir /data && chmod 0700 /data
```

- Les instructions « stables » sont au début du fichier pour profiter du cache
  - Exemple :
    - RUN apt install ...

# Hébergement des images

- Pour pouvoir être distribuées, les images doivent être stockées sur un registre.
- Docker Hub permet d'héberger ses propres images !
  - La création d'un compte est nécessaire.
  - Des limitations sont appliquées pour les comptes sans souscriptions.
- Pour pouvoir « pousser » une image sur Docker Hub, elle doit être nommée avec l'identifiant Docker Hub.
  - Exemple :
    - `identifiant/nom:tag`
- Une fois l'image correctement nommée, elle pourra être envoyée vers Docker Hub.
  - `docker login`
    - Pour s'authentifier sur Docker Hub.
  - `docker push ...`
    - Pour envoyer l'image.

# Docker Hub et GitHub

- GitHub est un système d'hébergement de code en ligne implémentant le SCM Git.
- Des projets Docker, contenant des Dockerfile, peuvent être hébergés sur GitHub.
- Il est ensuite possible de lier un projet dans GitHub avec Docker Hub !
  - Docker Hub pourra ainsi construire une image Docker sur la base d'un Dockerfile à chaque fois que ce dernier est modifié.
- Démarche :
  - Créer un repository GitHub.
  - Le remplir avec un Dockerfile et ses ressources.
  - Créer un repository sur Docker Hub et le connecter au repository GitHub.
    - La construction s'effectue alors chez Docker Hub.
    - Toute modification dans GitHub notifiera Docker Hub pour reconstruction.

# Travaux Pratiques



[www.eni-service.fr](http://www.eni-service.fr)



# Module 4

## Exploitation de Docker

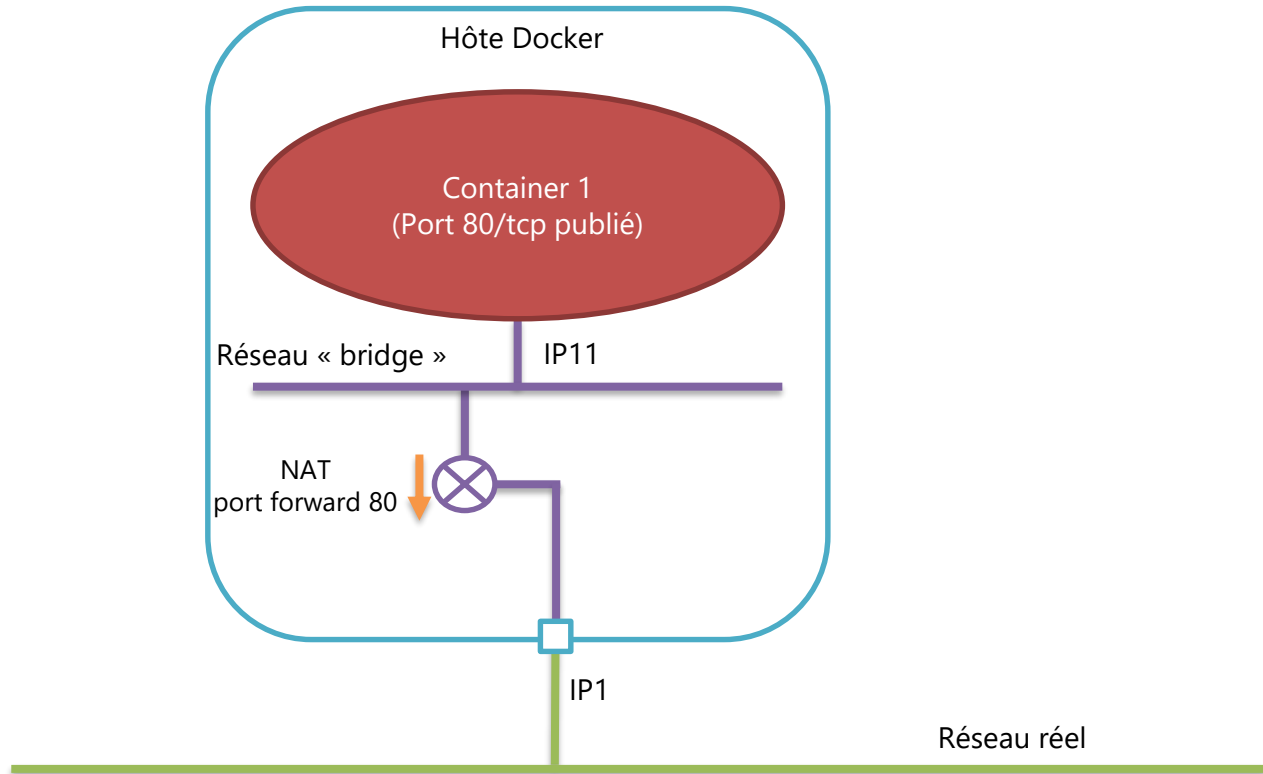
# Contenu du module

- Les réseaux
  - Principes et drivers
  - Création de réseaux
  - Attachement de réseaux
- Les volumes
  - Principes
  - Les volumes nommés
  - Les volumes de conteneurs

# Réseau

- Docker définit des réseaux virtuels par défaut caractérisés par leur nom et leur pilote (drivers).
- 2 containers dans 2 réseaux différents ne pourront pas communiquer directement.
- Drivers
  - **None** : pas de réseau pour le container (Docker Network Connect ne fonctionnera pas).
  - **Host** : le container et l'hôte Docker partagent exactement la même configuration réseau.
  - **Bridge** : le plus pratique et le plus utilisé.
    - Le container est dans un plan d'adressage propre au réseau.
    - Le réseau virtuel Docker est « natté » avec le réseau physique de l'hôte Docker.
- Docker crée 3 réseaux par défaut avec chacun des 3 drivers.

# Le driver « bridge »





# Mapping de port

- Avec un réseau créé avec le driver « bridge », il est possible de mapper un port du conteneur sur un port de l'hôte Docker.
  - Le service réseau du conteneur sera alors accessible via l'hôte.
- Exemple :
  - Un conteneur fait tourner un serveur Web Apache sur le port 80.
  - On souhaite rendre accessible le serveur Web via le port 8080 de l'hôte.
  - A la création du conteneur, on utilisera l'option -p pour définir le mapping.
  - Commande :
    - `docker run -d -p 8080:80 httpd:2.4.46`

# Gestion des réseaux Docker

- Lors de la création d'un nouveau conteneur, il est automatiquement attaché au réseau « bridge » par défaut.
  - De fait, tous les conteneurs créés sont dans le même réseau par défaut.
- Il peut être nécessaire de créer des réseaux supplémentaires pour isoler les conteneurs.
- La commande `docker network` permet de gérer les réseaux. Elle utilise des sous commandes.

```
etienne@dev-docker:~$ docker network

Usage:  docker network COMMAND

Manage networks

Commands:
  connect      Connect a container to a network
  create       Create a network
  disconnect   Disconnect a container from a network
  inspect      Display detailed information on one or more networks
  ls           List networks
  prune        Remove all unused networks
  rm           Remove one or more networks

Run 'docker network COMMAND --help' for more information on a command.
```

# Création d'un réseau

- La commande `docker network create` permet de créer un nouveau réseau.
  - `docker network create mon-reseau`
    - Avec un plan d'adressage affecté automatiquement.
    - Utilisant le driver « bridge »
- De nombreuses options sont disponibles pour personnaliser la création du réseau.
  - `docker network create --subnet 172.28.0.0/16 --gateway 172.28.0.1 mon-reseau`
- `docker network` possède d'autres sous commandes :
  - `docker network ls`
    - Lister les réseaux.
  - `docker network inspect <réseau>`
    - Afficher la configuration du réseau.
  - `docker network rm <réseau>`
    - Supprimer un réseau.

# Gestion et attachement du réseau

- Pour créer un conteneur sur un réseau particulier, il suffit d'utiliser l'option `--network` de la commande `docker run`.
  - `docker run -d --network mon-reseau httpd:2.4.46`
- Il est également possible d'attacher et de détacher des conteneurs actifs à un ou des réseaux.
  - `docker network connect [options] <réseau> <conteneur>`
    - Les options permettent notamment de préciser l'adresse IP à affecter.
  - `docker network disconnect [options] <réseau> <conteneur>`
- Il est ainsi possible de changer la configuration réseau d'un conteneur rapidement.

# Les volumes

- Permettent d'avoir du stockage persistant hors du conteneur accessible dans le conteneur.
  - Un volume peut représenter un répertoire ou un fichier.
  - Plusieurs volumes peuvent être définis pour un container.
- L'emplacement physique du volume peut être géré par :
  - Le moteur Docker : volume nommé
  - L'option de la commande run via un chemin complet
- Un volume utilise un driver de volume.
  - Un seul driver par défaut : local
    - Il permet de stocker sur le système de fichiers local, ou sur un partage SMB ou NFS
  - D'autres drivers peuvent être installés via les plugins :
    - docker plugin install vieux/sshfs, netapp
    - <https://netappdvp.readthedocs.io/en/stable/use/index.html#volume-driver-cli-options>

# Création d'un volume nommé

- La commande `docker volume` permet de gérer les volumes. Elle utilise des sous commandes.

```
etienne@dev-docker:~$ docker volume

Usage:  docker volume COMMAND

Manage volumes

Commands:
  create      Create a volume
  inspect     Display detailed information on one or more volumes
  ls          List volumes
  prune       Remove all unused local volumes
  rm          Remove one or more volumes

Run 'docker volume COMMAND --help' for more information on a command.
```

- Création d'un volume :
  - `docker volume create mon-volume`
- Visualiser les informations sur le volume :
  - `docker volume inspect mon-volume`

# Gestion des volumes nommés

- Les volumes nommés stockent les données sur l'hôte dans le répertoire de données du démon Docker.

```
etienne@dev-docker:~$ docker volume create mon-volume
mon-volume
etienne@dev-docker:~$ docker volume inspect mon-volume
[
  {
    "CreatedAt": "2020-11-19T13:39:27+01:00",
    "Driver": "local",
    "Labels": {},
    "Mountpoint": "/var/lib/docker/volumes/mon-volume/_data",
    "Name": "mon-volume",
    "Options": {},
    "Scope": "local"
  }
]
```

- Pour utiliser le volume dans un conteneur, il faut utiliser l'option -v au moment de la création du conteneur.
  - v <nom volume>:<chemin conteneur>
- Exemple :
  - docker run -it -d -v mon-volume:/sites/site1 httpd:2.4.46

# Les volumes créés au démarrage d'un conteneur

- L'option `-v` de la commande `docker run` permet aussi de monter un répertoire du conteneur sur un répertoire de l'hôte, sans avoir besoin de créer un volume nommé.
  - Usage différent par rapport à un volume nommé.
- Syntaxe :
  - `-v <chemin hôte>:<chemin conteneur>`
- Exemple :
  - `docker run -it -d -v $PWD/webcontent:/sites/site1 httpd:2.4.46`



# Les volumes : Exemples

- `docker run -it --rm -v $PWD/contenu:/sites/site1 alpine`
- `docker run -d --rm -e MYSQL_ROOT_PASSWORD=pass -v mariadb:/var/lib/mysql mariadb`

---

- `docker run -it --rm --mount 'type=volume,src=smbBruno,dst=/smb,volume-driver=local,volume-opt=type=cifs,volume-opt=device=//192.168.200.51/tmp,\"volume-opt=o=username=user1,password=a,vers=3.0,file_mode=0777,dir_mode=0777\",readonly' alpine`

---

- `docker volume create -d local -o type=cifs -o device=//192.168.200.51/tmp -o \"o=username=user1,password=a,vers=3.0,file_mode=0777,dir_mode=0777\" smbBruno`
- `docker run -it --rm -v smbBruno:/smb alpine`
- `docker run -it --rm --mount 'type=volume,src=smbBruno,dst=/smb,readonly' alpine`
  
- Remarque : *docker inspect image1* informe des volumes souhaités par l'image.

# Travaux Pratiques



[www.eni-service.fr](http://www.eni-service.fr)



## Module 5

# Chainage de conteneurs avec Docker Compose

# Contenu du module

- Docker Compose
- Installation de Docker Compose
- Le fichier `docker-compose.yml`
- La commande `docker compose`
- Gestion des conteneurs avec Docker Compose

# Docker Compose

- Vous pouvez avoir une infrastructure logicielle composée de plusieurs services (n-tiers).
  - Exemple : frontal web, service applicatif métier, base de données
- La commande `docker-compose` permet de démarrer plusieurs containers en une commande simple.
- Tous les paramètres habituellement passés à `run` sont présents dans un fichier de configuration YAML appelé, par convention, `docker-compose.yml`.
- Ce fichier configure (entre autres) les containers (caractérisés par un nom), les volumes, les réseaux.
  - En ce qui concerne le réseau, si rien n'est défini, un réseau dédié est créé au démarrage (*up*) et détruit à l'arrêt (*down*).
  - Le nom du répertoire contenant le fichier YAML est le préfixe de tous les objets créés par Docker Compose.

# Installation de Docker Compose

- Docker Compose s'installe séparément du démon Docker
  - Sauf si l'installation de Docker est faite avec Docker Desktop !
- Il s'agit maintenant d'un plugin pour Docker
  - Une extension de la commande docker
    - docker **compose** ...
  - Depuis la version 2 de Docker Compose
- Auparavant (Compose v1), c'était une commande différente
  - docker **-**compose
- <https://docs.docker.com/compose/install/>

# Le fichier docker-compose.yml

```
services:
  wordpress:
    build: ../wordpress-mysite
    image: wordpress-mysite:1.0
    depends_on:
      - db
    ports:
      - 80:80/tcp
    networks:
      - infraweb-network
    environment:
      WORDPRESS_DB_HOST: db
      WORDPRESS_DB_NAME: wordpress
      WORDPRESS_DB_PASSWORD: pwd
    volumes:
      - $PWD/wordpress:/var/www/html

  db:
    build: ../mariadb-wp
    image: mariadb-wp:1.0
    ports:
      - 3306:3306
    networks:
      - infraweb-network
    environment:
      MYSQL_ROOT_PASSWORD: pwd
    volumes:
      - $PWD/db:/var/lib/mysql

networks:
  infraweb-network:
    driver: bridge
    name: infraweb-network
    ipam:
      config:
        - subnet: 172.28.0.0/16
```

- Le fichier docker-compose.yml est constitué de :
  - services
    - Déclaration des différents conteneurs et de leurs options. On utilise sous forme de propriétés, les informations habituellement transmises sur la ligne de commande Docker.
    - Chaque service possède un nom qui **sera le nom d'hôte** du conteneur associé.
    - Les services peuvent utiliser des volumes et des réseaux déclarés dans ce fichier.
  - networks
    - Déclaration de réseaux pour les services. Un réseau par défaut est cependant créé automatiquement par Docker Compose.
  - volumes
    - Des volumes nommés.

Référence pour la syntaxe :

<https://docs.docker.com/compose/compose-file/>

# La commande docker compose

- La commande `docker compose` reprend les principales fonctionnalités de la commande `docker` en ce qui concerne la gestion des conteneurs.
  - `docker compose -help`
- Les commandes doivent être passées depuis le répertoire qui contient le fichier `docker-compose.yml`.
  - `docker compose up -d`
    - Démarrer l'ensemble des services en arrière-plan.
  - `docker compose ps`
    - Statut de l'ensemble des services.
  - `docker compose logs -f --tail 5`
    - Afficher les logs des services.
  - `docker compose stop`
    - Arrêter l'ensemble des services.
  - `docker compose down`
    - Détruire l'ensemble des ressources.
  - `docker compose config`
    - Valider la syntaxe du fichier `docker-compose.yml`.



# Gestion des conteneurs avec Docker Compose

- Lorsqu'un fichier `docker-compose.yml` déclare des services, il fait référence à des images pour la construction des conteneurs.
- Un répertoire de « build » peut être spécifié si l'on souhaite utiliser une image personnalisée et un `Dockerfile`.

```
wordpress:  
  build: ../wordpress-mysite  
  image: wordpress-mysite:1.0
```

- Lors du lancement des services avec `docker compose up`, il faudra ajouter l'option `--build` pour provoquer la construction de l'image.
- La commande `docker compose up` permet également de redémarrer les services pour appliquer les modifications du fichier `docker-compose.yml`.

# Travaux Pratiques



[www.eni-service.fr](http://www.eni-service.fr)



## Module 6

# Orchestration de conteneurs avec Docker Swarm

# Contenu du module

- Préambule - registre privé
- Présentation de Swarm
- Réseau dans Swarm
- Placement des containers dans le cluster
- Service simple et architecture multi-services avec Swarm

# Préambule - Registre privé

- Avec Swarm, l'utilisation d'un registre est essentiel.
  - On peut bien sûr toujours se référer au Docker Hub mais parfois on préférera stocker les images en interne.
- Ce service peut être installé via l'image **registry** présente sur Docker Hub.
- Par défaut, il fonctionne en HTTP sans authentification.
  - Le client Docker s'interdit de communiquer en HTTP avec un registre autre que localhost. Si nécessaire, il faut alors modifier le fichier `/etc/docker/daemon.json` des clients comme suit :

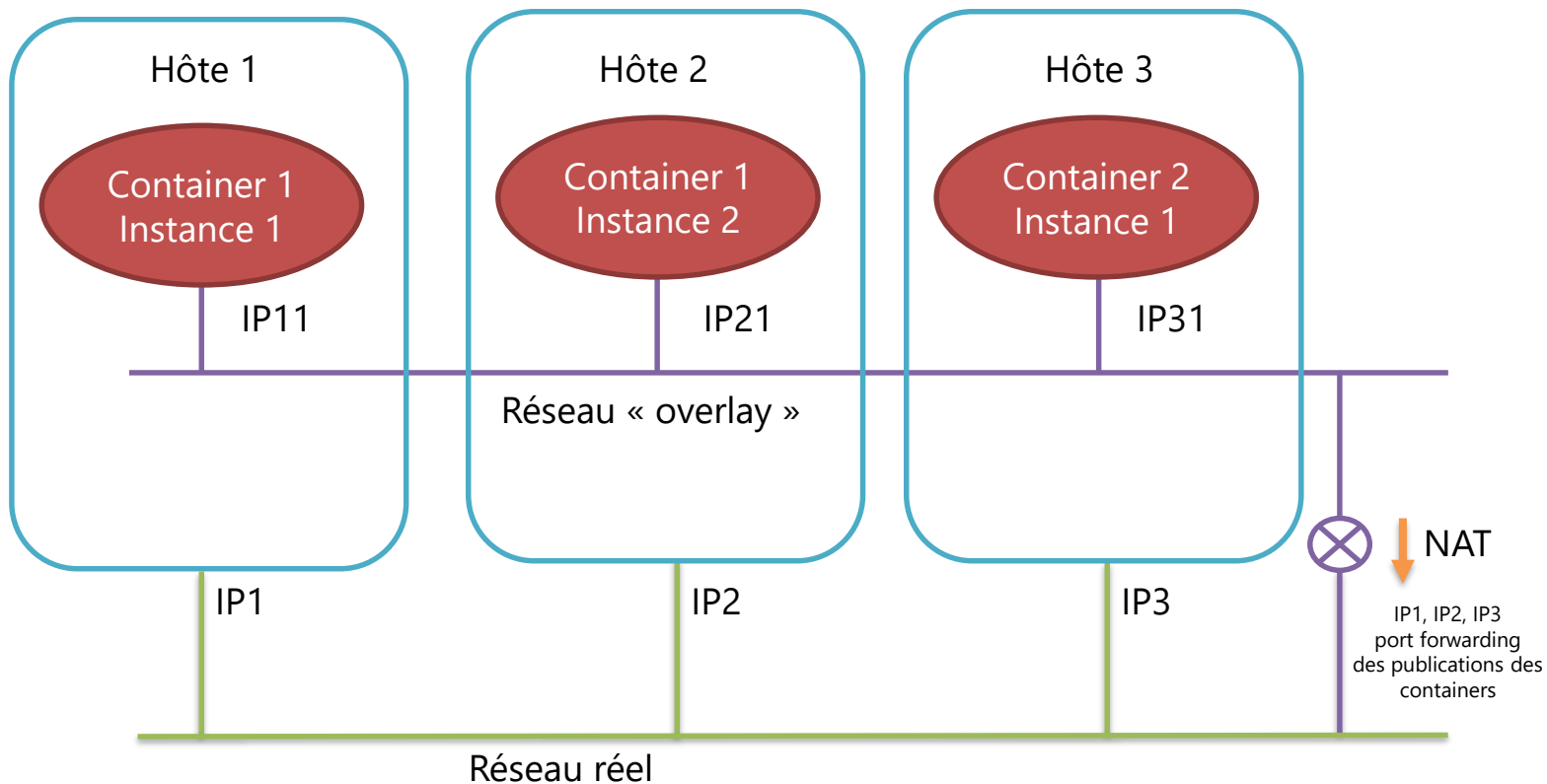
```
{  
  "insecure-registries": [  
    "registry-docker:5000", "registry-docker.mshome.net:5000"  
  ]  
}
```

- Il est également nécessaire de redémarrer le daemon Docker !
- Ce service est seulement une API (aucune GUI).
  - [joxit/docker-registry-ui](https://joxit.github.io/docker-registry-ui/) fournit une GUI

# Présentation de Swarm

- Si vous avez plusieurs hôtes Docker, vous pouvez avoir besoin des fonctionnalités de haute disponibilité (HA) et/ou de répartition de charge.
- Swarm fournit ces fonctions. On pourra parler de cluster et de nœuds.
- Chaque nœud dans le cluster a un rôle :
  - Worker : le nœud fait tourner des containers et leurs objets associés (volumes, réseaux) -> des services.
  - Manager : le nœud est un worker avec la possibilité de configurer le cluster, les nœuds et les services.
    - Un seul manager dans le cluster est le « leader », qui est responsable de la gestion de la réplication dans le cluster. Sans leader plus aucune configuration n'est possible.
    - Le leader est élu par une élection à la majorité absolue (quorum) à laquelle seuls les managers votent => au moins 3 managers pour tolérer une panne de manager
- REMARQUE : un cluster Swarm possédant 1 seul nœud est possible pour tester certaines fonctionnalités uniquement disponibles dans ce mode.

# Réseau dans Swarm - driver « overlay »

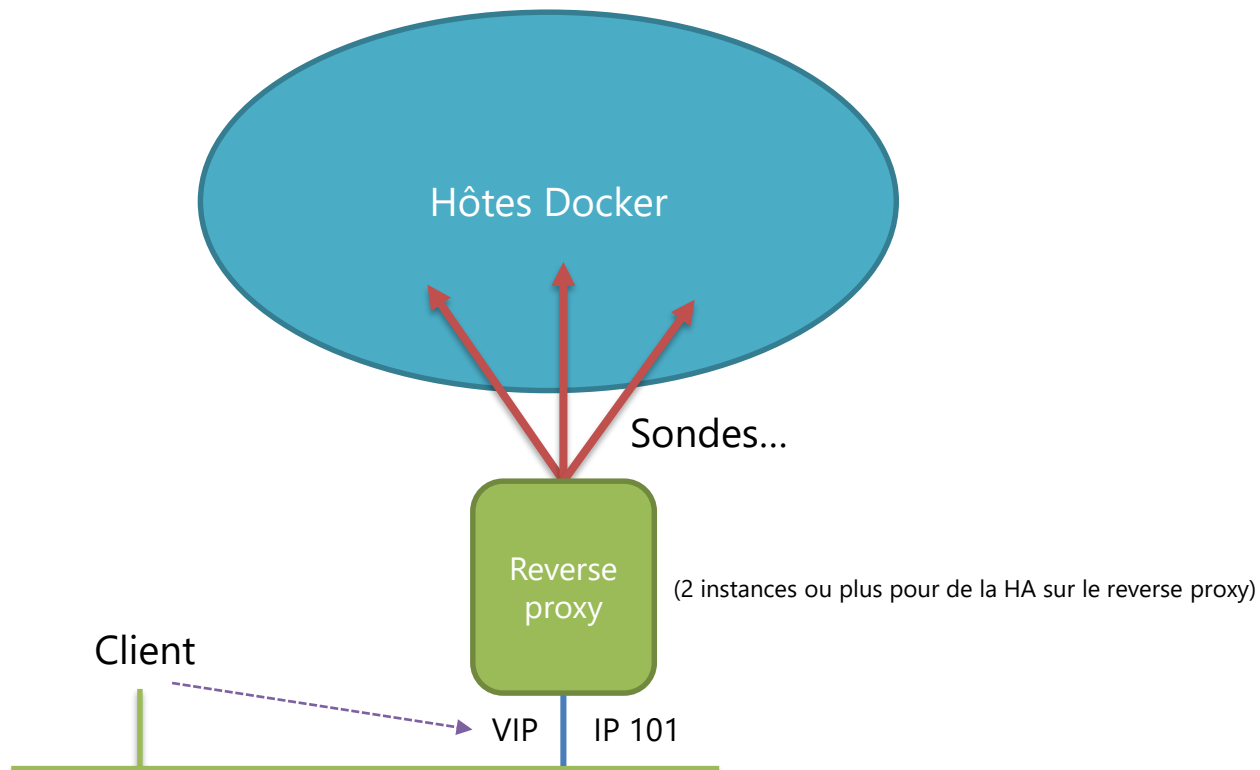


# Réseau dans Swarm - disponibilité d'accès

- Dans le schéma précédent, la disponibilité du service est garantie du point de vue du cluster.
- Si l'un des hôtes auquel un client est connecté directement via l'IP de cet hôte tombe en panne, le client ne peut plus se connecter.
- Il faut donc mettre en œuvre de la haute disponibilité des IPs des hôtes à l'aide d'un produit de type Reverse Proxy en haute disponibilité.
  - HAPROXY
  - Traefik
  - ...



# Réseau dans Swarm - disponibilité d'accès



# Initialisation d'un cluster Swarm

- Pour initialiser un cluster Swarm, il faut, sur un hôte initialement élu comme manager, utiliser la commande `docker swarm init`.

```
etienne@docker1:~$ docker swarm init
Swarm initialized: current node (ndkvpzgdhvw30io3n9zcpmka6) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1m9p88wixtx3bykfc9dwtd2uih3t57pi26ostl3c810ucovlsi-d4vozta4g06l6brl0636bmlfv 172.17.47.9:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.

etienne@docker1:~$
```

- Le résultat de la commande indique comment ajouter des nœuds au cluster.

```
etienne@docker1:~$ docker swarm join-token manager
To add a manager to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1m9p88wixtx3bykfc9dwtd2uih3t57pi26ostl3c810ucovlsi-8gkhhtt0eqozpatl05l39zsaj 172.17.47.9:2377
```

- Ajouter des nœuds :
  - `docker swarm join...`
- Supprimer un nœud :
  - `docker swarm leave...`

# Gestion des nœuds dans un cluster Swarm

- La gestion des nœuds dans un cluster Swarm ne peut se faire que depuis un manager.
- Commande `docker node`.
  - Elle utilise des sous commandes.

```
etienne@docker1:~$ docker node

Usage:  docker node COMMAND

Manage Swarm nodes

Commands:
  demote      Demote one or more nodes from manager in the swarm
  inspect     Display detailed information on one or more nodes
  ls          List nodes in the swarm
  promote     Promote one or more nodes to manager in the swarm
  ps          List tasks running on one or more nodes, defaults to current node
  rm          Remove one or more nodes from the swarm
  update      Update a node

Run 'docker node COMMAND --help' for more information on a command.
```

# Orchestration des conteneurs dans Swarm

- La commande `docker service` permet de gérer les services/conteneurs dans un cluster Swarm.

```
etienne@docker1:~$ docker service
Usage:  docker service COMMAND

Manage services

Commands:
  create   Create a new service
  inspect  Display detailed information on one or more services
  logs     Fetch the logs of a service or task
  ls       List services
  ps       List the tasks of one or more services
  rm       Remove one or more services
  rollback Revert changes to a service's configuration
  scale    Scale one or multiple replicated services
  update   Update a service

Run 'docker service COMMAND --help' for more information on a command.
```

- `docker service create [options] <image>`
  - Créer et démarrer un service.
  - L'option `--replicas` permet de préciser le nombre d'instance au démarrage.
- `docker service ls`
  - Lister les services.
- `docker service ps <service>`
  - Lister les instances d'un service.
- `docker service rm <service>`
  - Arrêter un service.

# Gestion des instances de service

- Il est possible de gérer très facilement le nombre d'instances d'un service et de les répartir sur les différents nœuds d'un cluster Swarm.
- La commande `docker service scale` permet d'indiquer le nombre d'instance souhaité pour un service, l'ajustement se fait sur les différents nœuds.
  - `docker service scale <service>=<nb. instances>`
- Pour rééquilibrer un cluster Swarm (répartir les instances) après une perte de nœuds :
  - `docker service update --force <service>`

# docker service scale

```
eni@docker1:~/docker/monSitePHP$ docker service scale mon-site-php=10
mon-site-php scaled to 10
overall progress: 10 out of 10 tasks
1/10: running [=====>]
2/10: running [=====>]
3/10: running [=====>]
4/10: running [=====>]
5/10: running [=====>]
6/10: running [=====>]
7/10: running [=====>]
8/10: running [=====>]
9/10: running [=====>]
10/10: running [=====>]
```

verify: Service converged

```
eni@docker1:~/docker/monSitePHP$ docker service ps mon-site-php
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
1onem7zqbhv7	mon-site-php.1	registry-docker:5000/mon-site-php:1.0	docker3	Running	Running 4 minutes ago		
muu7a3zeb0uo	mon-site-php.2	registry-docker:5000/mon-site-php:1.0	docker1	Running	Running 4 minutes ago		
mmigxkdxhaj	mon-site-php.3	registry-docker:5000/mon-site-php:1.0	docker1	Running	Running 9 seconds ago		
m4jb6b21s64p	mon-site-php.4	registry-docker:5000/mon-site-php:1.0	docker2	Running	Running 9 seconds ago		
ikyncjrusslla	mon-site-php.5	registry-docker:5000/mon-site-php:1.0	docker3	Running	Running 8 seconds ago		
u9k7o8d05lui	mon-site-php.6	registry-docker:5000/mon-site-php:1.0	docker1	Running	Running 8 seconds ago		
p6qjoobc12oa	mon-site-php.7	registry-docker:5000/mon-site-php:1.0	docker2	Running	Running 7 seconds ago		
4dj2wv0iy2kz	mon-site-php.8	registry-docker:5000/mon-site-php:1.0	docker2	Running	Running 8 seconds ago		
4bycuwlp0fy2	mon-site-php.9	registry-docker:5000/mon-site-php:1.0	docker2	Running	Running 9 seconds ago		
bp6k0meq6hxl	mon-site-php.10	registry-docker:5000/mon-site-php:1.0	docker3	Running	Running 9 seconds ago		

```
eni@docker1:~/docker/monSitePHP$ docker service scale mon-site-php=2
```

mon-site-php scaled to 2

overall progress: 2 out of 2 tasks

```
1/2: running [=====>]
2/2: running [=====>]
```

verify: Service converged

```
eni@docker1:~/docker/monSitePHP$ docker service ps mon-site-php
```

ID	NAME	IMAGE	NODE	DESIRED STATE	CURRENT STATE	ERROR	PORTS
1onem7zqbhv7	mon-site-php.1	registry-docker:5000/mon-site-php:1.0	docker3	Running	Running 5 minutes ago		
muu7a3zeb0uo	mon-site-php.2	registry-docker:5000/mon-site-php:1.0	docker1	Running	Running 5 minutes ago		

# Placement des containers dans le cluster

- Par défaut, Swarm place un nouveau container sur n'importe quel nœud (il tente d'équilibrer le nombre de containers par nœud).
- `docker service create` possède une option `--constraint` permettant d'imposer un seul hôte ou un sous-ensemble d'hôtes du cluster.
  - `docker service update` possède `--constraint-add` et `--constraint-rm`.
- Exemple :
  - `docker service create --name nginx-workers-only --constraint node.role==worker nginx`
- Tests possibles :
  - `node.id==2ivku8v2gvtg4`
  - `node.hostname!=node2`
  - `node.role==manager`
  - `engine.labels.operatingsystem==ubuntu 16.10 LTS`
  - `node.labels.location==north`
    - `docker node update --label-add` et `--label-rm` pour gérer ce test.

# Visualisation d'un cluster Swarm

- dockersamples/visualizer
  - <https://hub.docker.com/r/dockersamples/visualizer>
- Une image Docker contenant une application Web de visualisation d'un cluster Swarm !



# Chainage de conteneurs avec Docker Swarm

- La déclinaison de Docker Compose en mode Swarm : `docker stack` !
- On peut utiliser Docker Compose ou d'autres outils...
- Conventionnellement, le fichier `docker-compose.yml` est remplacé par `docker-stack.yml`.
  - Le contenu est le même !
  - On y ajoute des informations de réplicas sur les services.

```
etienne@docker1:~$ docker stack
Usage:  docker stack [OPTIONS] COMMAND

Manage Docker stacks

Options:
  --orchestrator string  Orchestrator to use (swarm|kubernetes|all)

Commands:
  deploy  Deploy a new stack or update an existing stack
  ls      List stacks
  ps      List the tasks in the stack
  rm      Remove one or more stacks
  services List the services in the stack

Run 'docker stack COMMAND --help' for more information on a command.
```

# Création de stack dans Swarm

```
etienne@docker1:~$ docker stack deploy --help
Usage:  docker stack deploy [OPTIONS] STACK

Deploy a new stack or update an existing stack

Aliases:
  deploy, up

Options:
  --bundle-file string      Path to a Distributed Application Bundle file
  -c, --compose-file strings Path to a Compose file, or "-" to read from stdin
  --orchestrator string     Orchestrator to use (swarm|kubernetes|all)
  --prune                   Prune services that are no longer referenced
  --resolve-image string    Query the registry to resolve image digest and supported platforms ("always"|"changed"|"never") (default "always")
  --with-registry-auth      Send registry authentication details to Swarm agents
```

- Exemple :
  - `docker stack deploy --compose-file ./docker-stack.yml MyStack`

# Service simple et architecture multi-services avec Swarm

- En synthèse...
  - Gestion simple de conteneur
    - Standalone ou bien en cluster Swarm
  - Gestion chaînée (Docker Compose)
    - Standalone ou bien en cluster Swarm

Standalone	Swarm
<code>docker run</code>	<code>docker service</code>
<code>docker compose</code>	<code>docker stack</code>

# Travaux Pratiques





[www.eni-service.fr](http://www.eni-service.fr)



Fin de la formation

# Docker – Mise en œuvre du déploiement d'applications en conteneur

# Pour aller plus loin

- ENI Service sur Internet
  - Consultez notre site web [www.eni-service.fr](http://www.eni-service.fr)
    - Les actualités
    - Les plans de cours de notre catalogue
    - Les filières thématiques et certifications
  - Abonnez-vous à nos newsletters pour rester informé sur nos nouvelles formations et nos événements en fonction de vos centres d'intérêts.
  - Suivez-nous sur les réseaux sociaux
    -  Twitter : <http://twitter.com/eniservice>
    -  Viadeo : <http://bit.ly/eni-service-viadeo>

# Pour aller plus loin

- Notre accompagnement
  - Tous nos Formateurs sont également Consultants et peuvent :
    - Vous accompagner à l'issue d'une formation sur le démarrage d'un projet.
    - Réaliser un audit de votre système d'information.
    - Vous conseiller, lors de vos phases de réflexion, de migration informatique.
    - Vous guider dans votre veille technologique.
    - Vous assister dans l'intégration d'un logiciel.
    - Réaliser complètement ou partiellement vos projets en assurant un transfert de compétence.

# Votre avis nous intéresse

Nous espérons que vous êtes satisfait de votre formation.

Merci de prendre quelques instants pour nous faire un retour en remplissant le questionnaire de satisfaction.

Merci pour votre attention,  
et au plaisir de vous revoir prochainement.