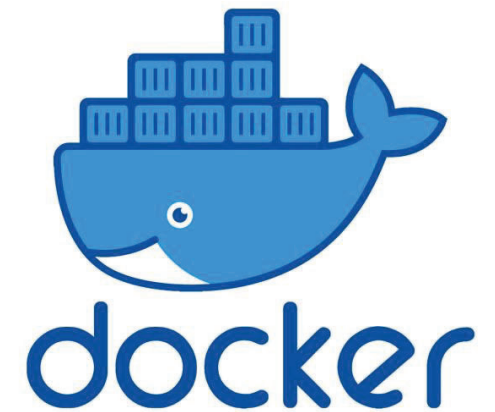


---

# CONTAINÉRISATION DOCKER

VINCENT LAINE



# SOMMAIRE

- Conteneurisation Docker
- Bénéfices des containers
- Apports pour les Dev-Ops
- Docker hub
- NOTIONS DE BASE DOCKER
- OFFRES COMMERCIALES
- DOCKER COMPOSE
- DOCKER MACHINE
- DOCKER SWARM
- DOCKER ENGINE
- DOCKER HUB / REGISTRY
- FONCTIONNEMENT DES VOLUMES DE STOCKAGE EN CLUSTER
- Le réseau dans Docker (Que les 3 principaux)
- Variables d'environnement
- Run – port mapping
- Run – volume mapping
- Docker Commands
- Inspect container
- Container logs
- DOCKER FILE
- Build Multistage
- TAG / PUSH / PULL

# CONTENEURISATION DOCKER

- Qu'est ce que Docker ?
- Docker permet d'embarquer une application dans un ou plusieurs containers logiciels qui pourra s'exécuter sur n'importe quel serveur machine.

Docker est une plate-forme de virtualisation par conteneur qui, à la différence de la virtualisation par hyperviseur qui nécessite de créer une machine virtuelle embarquant un système d'exploitation invité pour s'assurer de son indépendance avec la machine principale.

Il va permettre d'isoler une ou plusieurs applications dans des conteneurs en utilisant les ressources de la machine hôte.

Docker doit pouvoir être exécutée sur n'importe quel environnement, quel que soit son contenu, évitant ainsi divers problèmes de compatibilité entre les applications et l'hôte qui les exécute.

# DOCKER

## Avantages

- Un conteneur n'a besoin que de quelques centaines de Mo d'espace disque pour être déployé, contrairement à une VM. Un conteneur va directement utiliser les ressources dont il a besoin sur le serveur physique.
- La possibilité d'exploiter directement une application sans être contraint de modifier les librairies installées sur la machine hôte ou de se soucier des versions des logiciels installés sur cette dernière.
- C'est un gain de temps non négligeable, l'administrateur système n'a plus autant à s'accorder avec l'équipe de développement pour être certain que son serveur possédera toutes les dépendances nécessaires au déploiement d'une application.
- La possibilité de partager la configuration complète d'une application sur n'importe quel poste grâce aux Dockerfiles.
- Une simple exécution du fichier permet de déployer des conteneurs prêts à l'emploi, avec la possibilité d'en modifier certains aspects sans affecter leur configuration.

# DOCKER

## Inconvénients

- Comme les conteneurs utilisent directement les ressources de la machine hôte, il est par exemple impossible de déployer un conteneur basé sur un système d'exploitation Windows sur un serveur physique Linux.
- Certaines applications ne se prêtent pas à un déploiement morcelé par couche car certaines d'entre elles ne peuvent fonctionner que de façon monolithique, il est donc préférable d'effectuer des tests de conteneurisation avant de se lancer dans un déploiement d'application déjà existante.
- Il peut être parfois difficile de définir une sécurité poussée entre les différents environnements Docker d'une même machine physique, les conteneurs utilisant les ressources du noyau et des composants systèmes de cette dernière, certaines situations ont plus de chance de se répercuter sur le système d'exploitation sous-jacent ou sur d'autres conteneurs.

# BÉNÉFICES DES CONTAINERS

- Agilité
  - Les containers sont rapides et simples à mettre en œuvre
- Portabilité
  - Les containers sont portables sur tout type d'environnement
  - Du poste développeur, aux serveurs de l'entreprise, aux plateformes cloud privées ou publiques
- Consistance
  - Les containers ne subissent aucune modification du poste du développeur à l'environnement de production
  - Ils sont conçus pour être réutilisés
- Optimisation des coûts
  - Que ce soit en termes de ressources nécessaires (RAM, stockage..) ou de management (agilité, portabilité...)
  - Profite des coûts d'investissement plus bas du Cloud provider CAPEX/OPEX
- Elasticité
  - Orchestrateur !
  - Autoscaling

# APPORTS POUR LES DEV-OPS

- Développeurs
  - « build once... run everywhere »
  - Le container est un environnement sécurisé, stable, portable sur tous environnements
  - Il résout les problèmes de dépendances ou de packages manquant lors du déploiement
  - Plusieurs versions de bibliothèques peuvent être utilisées dans différents containers du fait de leur isolation
  - Forte automatisation de toutes les phases (test, intégration, packaging..)
  - Réduit voire élimine les problèmes d'environnement clients différents de ceux de l'éditeur
  - Rapidité de déploiement, de retour arrière comparativement aux VMs
  - En définitif, tout semble simple et rapide !

# APPORTS POUR LES DEVOPS

- Administrateurs
  - « configure once... run everywhere »
  - Rend le cycle de vie des applications plus efficace, consistant, et reproductible
  - Améliore le rendu de la qualité du code développé
  - Élimine définitivement les inconsistances entre les différents environnements de développement, test, et production
  - Apporte une répartition des responsabilités claire avec les développeurs
  - Améliore sensiblement la vitesse et l'agilité des procédés d'intégration et de déploiement continu
  - Améliore beaucoup d'aspects comparativement aux VMs du fait de leur taille réduite : performance, coûts, déploiement, portabilité..



# APPORTS POUR L'ENTREPRISE

- Stratégie cloud
  - Facilite l'évolution vers le(s) cloud(s) public(s)
  - Facilite l'évolution vers l'hybridation et le multi-clouds
- Stratégie de modernisation applicative
  - Containerisation des applications « legacy »
  - Transformation des applications « legacy » en microservices
  - Accélère le développement de nouvelles applications sur le modèle microservices
- Stratégie DevOps
  - Résout les problèmes de gestion des différents environnements par la consistance des containers
  - Facilite les démarches d'intégration et de déploiement continu
  - Facilite l'autoscaling des applications

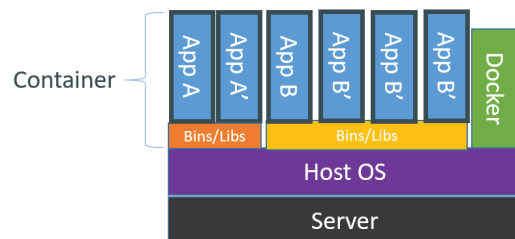
# LIMITES - MOBILITÉ DES CONTAINERS

- Registre
  - Registre Docker public
  - Registre privé
  - Registre en mode SaaS
  - Quay.io
- ...
- Problématique des données
  - Stockage objet
  - Bases de données !
- Kubernetes
  - 2017 meilleure gestion du stockage stateless/statefull sur les pods

# LIMITES - SÉCURITÉ DES CONTAINERS

## ■ Isolation

- Isolation native entre les containers
- Partage de bibliothèques pour l'optimisation mais en lecture seule



## ■ Sécurité

- Kernel « hardened » > Grsecurity / PaX
- Linux Security Module
- SE Linux / AppArmor
- Os Minimaux / réduction de la surface d'attaque > CoreOS / Fedora Atomic / VMWare Photon...

# DOCKER

Docker est une technologie de conteneurisation reposant sur le noyau Linux et ses fonctionnalités de virtualisation par conteneurs (LXC pour Linux Containers), notamment :

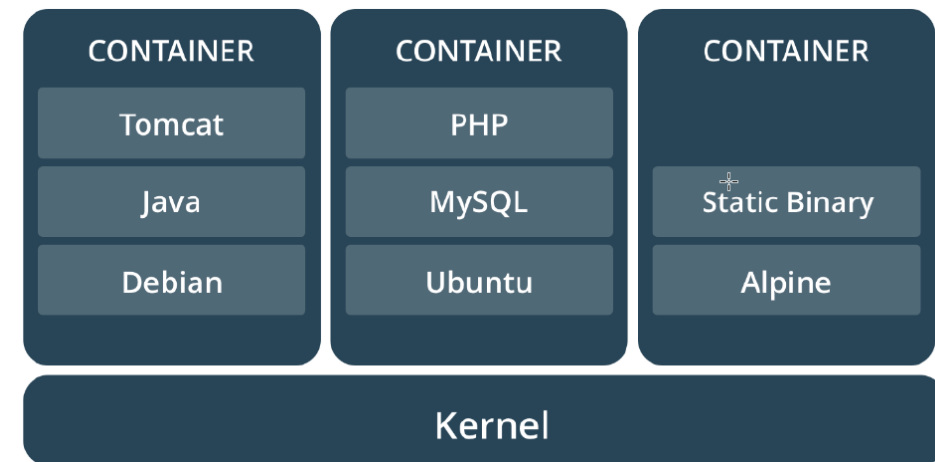
- le composant cgroups pour contrôler et limiter l'utilisation des ressources pour un processus ou un groupe de processus (utilisation de la RAM, CPU entre autres) associé au système d'initialisation systemd qui permet de définir l'espace utilisateur et de gérer les processus associés ;
- les espaces de noms ou namespaces qui permettent de créer des environnements sécurisés de manière à isoler les conteneurs et empêcher par exemple qu'un groupe puisse « voir » les ressources des autres groupes.

Docker utilise des fonctionnalités natives au noyau Linux, comme les cgroups ou les namespaces, mais offre les outils pour le faire de manière simplifiée pour permettre, entre autres :

- la duplication et la suppression des conteneurs ;
- l'accessibilité des conteneurs à travers la gestion des API et CLI ;
- la migration (à froid ou à chaud) de conteneurs.

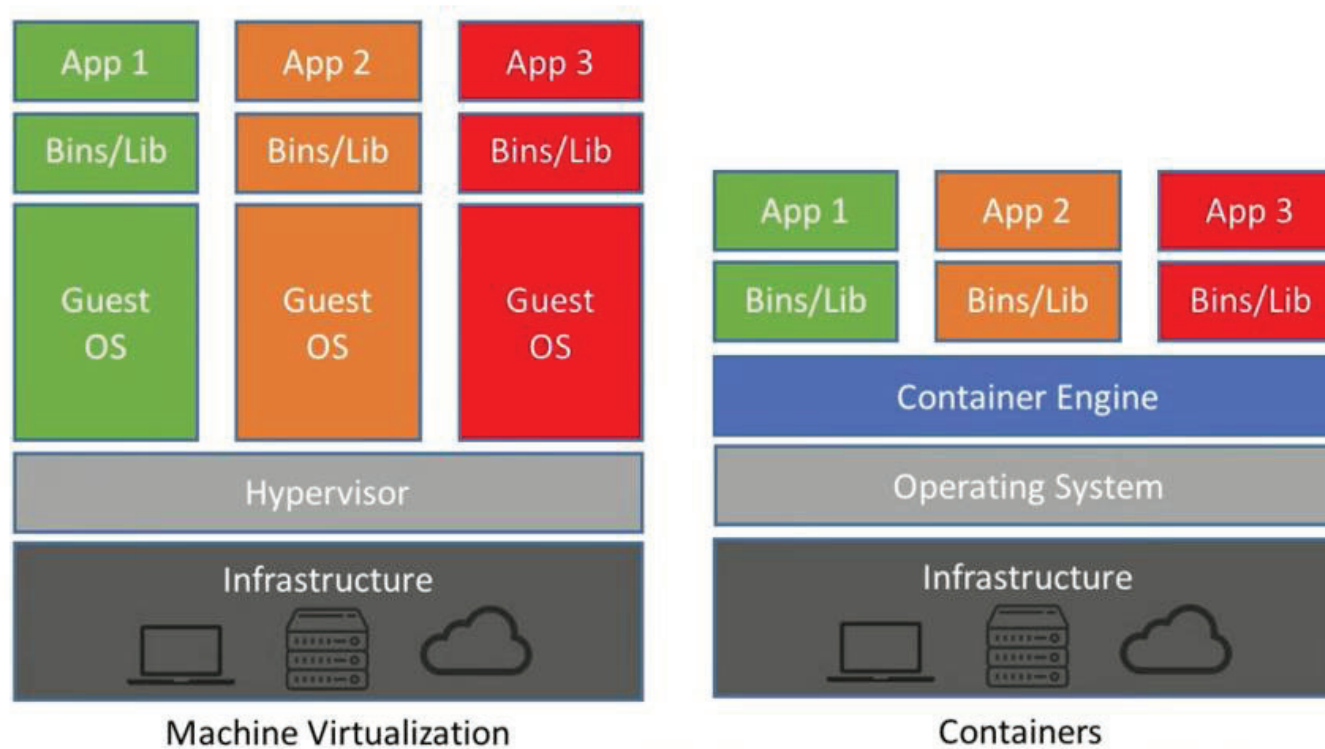
# DOCKER

- Les conteneurs Docker sont construits à partir des images Docker. Une image Docker est un package léger, autonome et exécutable d'un logiciel qui inclut tout ce qui est nécessaire pour l'exécuter :
  - code de l'application,
  - environnement d'exécution (runtime),
  - outils système et librairies, etc.
- Disponible pour les applications basées sur Linux et Windows, le logiciel conteneurisé fonctionnera toujours de la même manière, quel que soit l'environnement.



# DOCKER

- Différence entre Docker et Machine Virtuelle



# DOCKER

## Machine Virtuelle

- Basée sur un Hyperviseur
- Permet d'émuler des machines complètes : un Os avec une ou plusieurs applications
- Machine virtuelles lourdes (plusieurs giga)

## Docker

- Plus léger, basé sur un moteur de conteneur (Docker)
- Sur la couche supérieure, on retrouve les applications installées dans chacun des conteneurs
- Un seul système d'exploitation, celui de la machine hôte qui fait bénéficier de ses ressources aux différents conteneurs
- Conteneur plus léger (rarement au-dessus de 500 Mo)
- Permet de segmenter une application en micro-service (un docker par service : apache, base de données ...)

# POURQUOI TANT D'ENGOUEMENT ? QUELLES SONT LES UTILISATIONS POSSIBLES, EFFICACES ET PERTINENTES DE DOCKER ?

- Déployer rapidement un service lorsque l'on a besoin de le déployer plusieurs fois : cette reproductibilité est la base de docker, c'est typiquement l'utilisation que peut en faire un fournisseur de cloud.
- Distribuer une application : Docker en tant que "système de distribution d'une application" : n'ont plus à packager une application sous différents systèmes (deb, rpm, etc)
- Développer et tester une application, Docker permet :
  - de concevoir une architecture de test plus agile, chaque conteneur de test pouvant par exemple intégrer une brique de l'application (base de données, langages, composants, ...), le développeur pourra tester sur la même machine plusieurs versions d'un même logiciel en inter changeant le conteneur correspondant.
  - de développer une application selon le concept d'architecture de micro-services avec pour chaque couche des conteneurs isolant les composants de l'application ;
  - faciliter le process de mise à jour de l'application : les images Docker sont versionnées et permettent une mise à jour simplifiée et maîtrisée. Le process de rollback est aussi simplifié : on redéploie la version précédente de l'image.
  - d'avoir un environnement de développement identique à l'environnement de production



# DOCKER HUB

- C'est quoi ?
- <https://hub.docker.com/>
- Regroupe toutes les images docker disponibles
- Permet de retrouver toutes les images officielles
- Et celle de la communauté
- Tout le monde peut déposer son propre conteneur (inscription obligatoire)

# TECHNOLOGIES SOUS-JACENTES

- Développement
  - Docker est développé en langage GO
  - Le daemon Docker se base sur le pilote d'exécution RunC
- Control Group CGROUPS
  - Allocation de ressources RAM et CPU aux containers
- Namespaces
  - Isolation des containers
  - système de fichiers, nom d'hôte, utilisateurs, réseau, processus
- UNION FILE SYSTEM UnionFS
  - Stockage sur plusieurs couches en lecture seule
  - Support d'autres pilotes de stockage en fonction des distributions (AUFS, devicemapper, BTRFS, Overlay)
- Format des containers Docker
  - Namespaces + Cgroup + UnionFS
  - Ce format par défaut est appelé **LIBCONTAINER**

# NOTIONS DE BASE DOCKER



- Docker image (Dockerfile)
  - Équivalent à une application complète composée de différentes couches liées.



- Docker container (docker-compose)
  - Correspond à l'exécution d'une image Docker + une couche de lecture/écriture



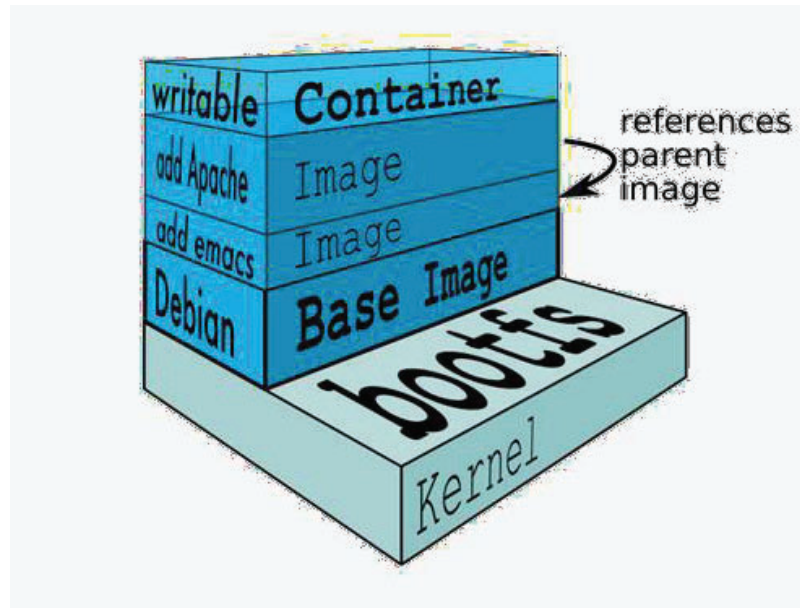
- Docker engine
  - Crée, lance et exécute les containers Docker sur les plateformes physiques, virtuelles, en local, dans le datacenter ou chez un fournisseur de cloud



- Docker hub / Registry service
  - Service de stockage et de distribution des images dans le cloud ou sur un stockage d'entreprise

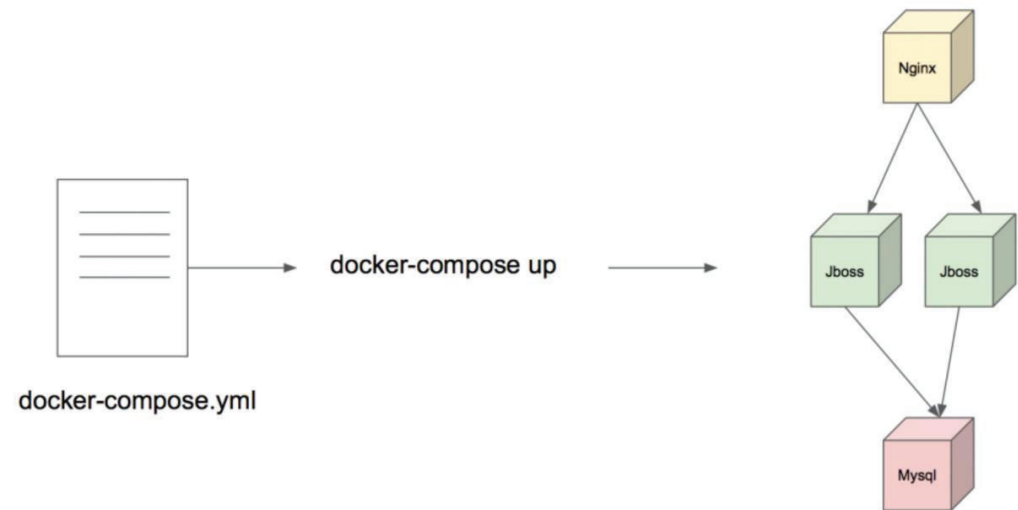
# SYSTÈME EN COUCHES

- Chaque couche est liée à la couche sous-jacente
- Copy-on-WRITE
- 127 couches possibles



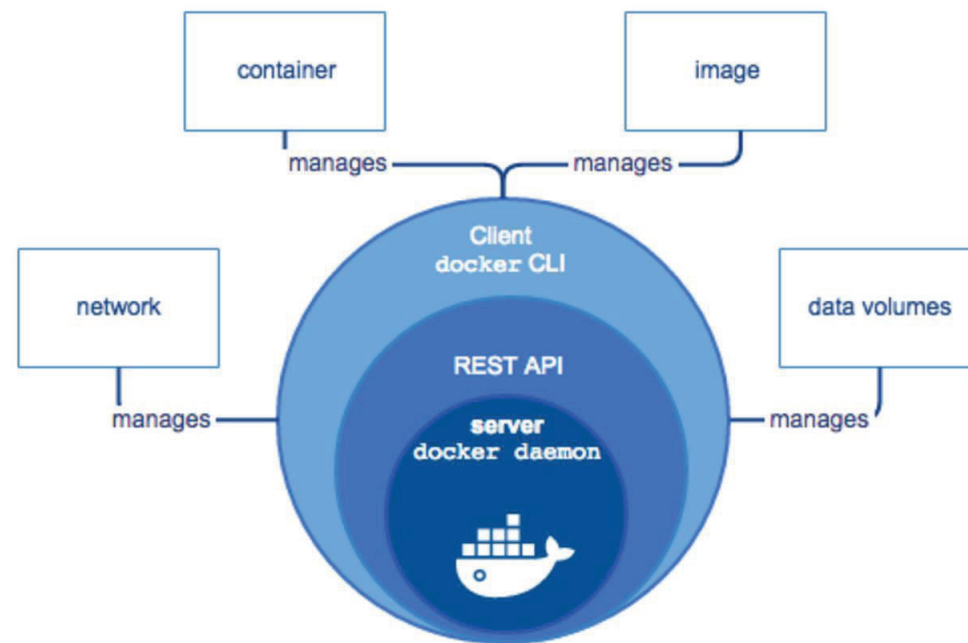
# DOCKER COMPOSE

- Liens de containers
  - Description YAML



# DOCKER ENGINE

- Moteur Docker
  - Docker Daemon
  - REST API pour piloter le daemon
  - Command Line Interface



# QUEL OS POUR L'HÔTE ?

- Distributions minimales
  - Le choix idéal ! CoreOS, Fedora Atomic, RancherOS...
  - • + Sécurisé + Cluster/Réseau + Léger
  - - mature / stable ?
  - Production ?
- Distribution
  - Ubuntu / CentOS / Debian / Suse ....
  - Adapté à la production : en Entreprise & Cloud providers
  - + Support + Maîtrisé + GUI
  - - sécurisé ? - cluster ?
- Poste de développement
  - Poste Linux : Ubuntu ou autre
  - Poste Mac/Windows : VirtualBox / Distri minimale Boot2Docker

# DOCKER HUB / REGISTRY

- Analogie
  - Docker Hub est équivalent à un repository Github mais pour stocker des images de containers.
- Docker Hub
  - Service de stockage d'images Docker en mode SaaS
    - La version publique est gratuite et accessible par tous
    - La version privée est payante et sécurisée
  - API : push/pull/search images
  - Intègre une GUI avec statistiques et gère des droits d'accès utilisateur (version privée)
- Docker registry
  - Repo de container à la Docker Hub
  - Installation Onpremise
  - API : push/pull/Search



# LES VOLUMES DANS DOCKER

- Lorsque vous supprimer le conteneur, toutes vos modifications sont perdus car les fichiers sont stockés par défaut dans ce conteneur
- Pour éviter ce problème, il est possible d'utiliser un volume qui sera situé sur la machine hôte
- Vous pourrez supprimer le conteneur, et le recréer en conservant vos fichiers modifiés

# FONCTIONNEMENT DES VOLUMES DE STOCKAGE EN CLUSTER

- Pour améliorer la performance et la portabilité d'une infrastructure, il est recommandé de ne pas stocker les données importantes directement sur la couche d'écriture des conteneurs.
- Pour cela il faut utiliser les volumes.
- Les volumes sont des espaces de stockages qui peuvent être mappés sur plusieurs conteneurs/services en même temps.
- Docker met à disposition 3 types de volumes :
  - Volume Data
  - Volume Bind
  - Volume tmpfs

# VOLUME DATA

- Description :
- C'est le type de stockage par défaut de docker.
- Elle offre de nombreux avantages :
  - Elle n'augmente pas la taille de la couche d'écriture du conteneur.
  - Cycle de vie indépendant du conteneur/service.
  - Peut être managé directement via la CLI Docker.
  - Fonctionne aussi bien sous Linux que Windows.
  - Facile à déplacer/sauvegarder.
  - Permet l'utilisation de « volumes driver » pour stocker les données sur des espaces de stockages distants de l'hôte (NFS, SAMBA, cloud, etc...).
- Les volumes docker de type data sont visibles dans le répertoire : /var/

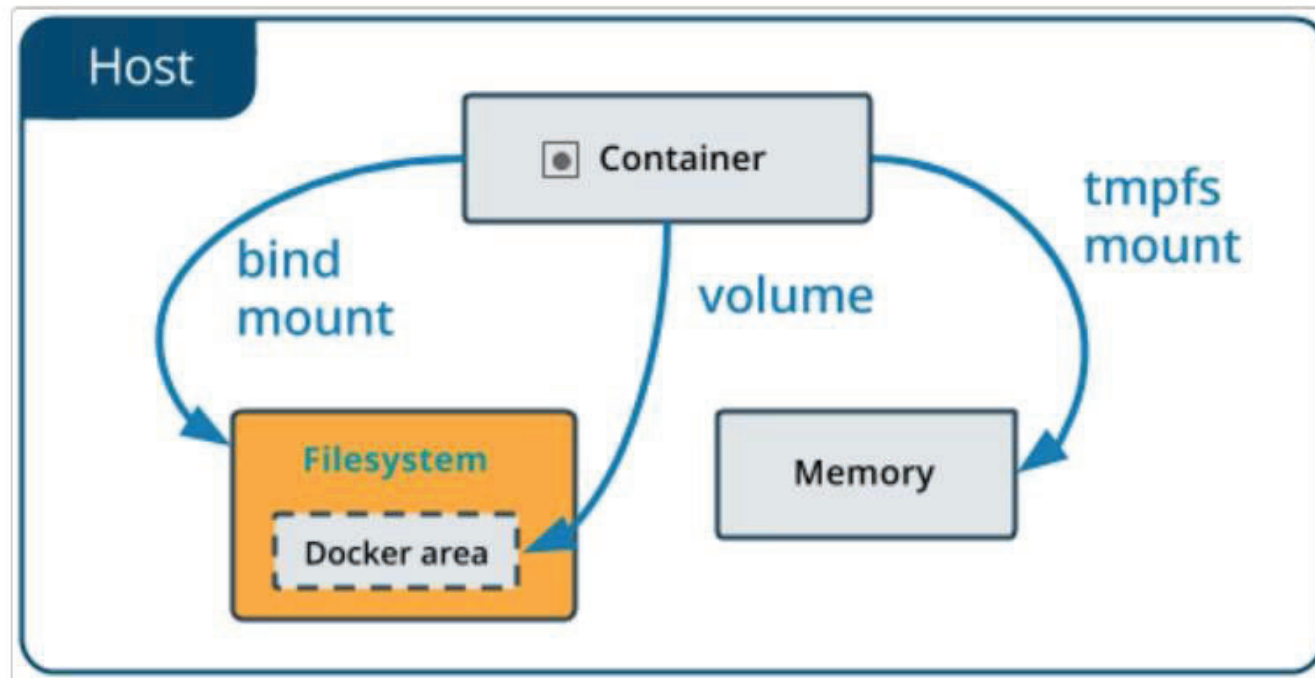
# VOLUME BIND

- Description :
  - Le volume de type « bind » consiste à monter un chemin du filesystem de l'hôte directement dans le conteneur.
  - Ce type de volume est historique à docker, mais il est préférable aujourd'hui d'utiliser les volumes de type « data ».
  - Ce type de volume ne peut pas être géré via la CLI de Docker et est donc plus « difficile » à administrer.
  - Si le chemin du volume « bind » n'existe pas à la création du conteneur, il sera automatiquement créé.
  - Ce type de volume est dépendant de la structure de répertoire du filesystem hôte.

# VOLUME TMPFS

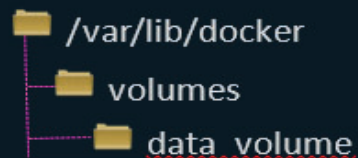
- Description :
  - Ce type de volume permet de stocker de la donnée volatile directement sur la RAM de l'hôte.
  - Ce type de volume dispose d'une performance très élevée.
- Contraintes :
  - Uniquement disponible sur les hôtes docker Linux.
  - Le cycle de vie du volume tmpfs est identique à celui du conteneur qui l'utilise.
  - Un volume tmpfs ne peut être partagé entre plusieurs conteneurs/services.

# RÉSUMÉ



# RÉSUMÉ

```
docker volume create data_volume
```

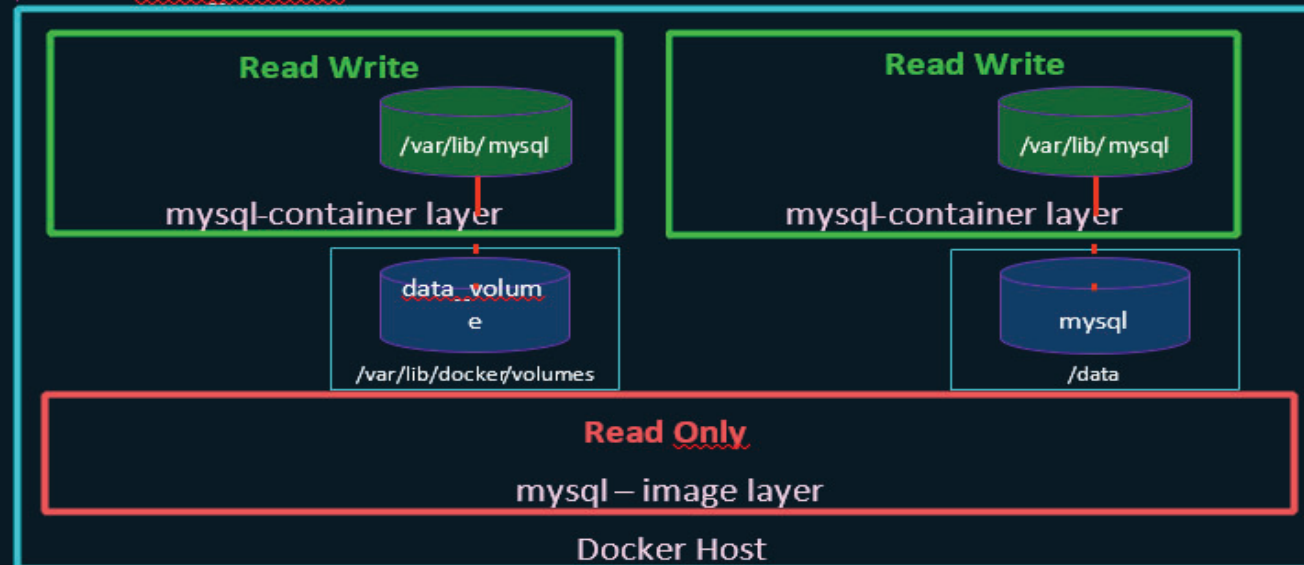


```
docker run -v data_volume:/var/lib/mysql mysql
```

```
docker run -v data_volume2:/var/lib/mysql mysql
```

```
docker run -v /data/mysql:/var/lib/mysql mysql
```

```
docker run \
--mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql
```



# VOLUMES PERSISTANT DANS DOCKER

- La commande suivante va permettre à docker de mapper le répertoire local `/srv/data/html/` dans l'arborescence du container `/usr/share/nginx/html/`
  - `Docker run -tid -p 8080 :80 -v /srv/data/html:/usr/share/nginx/html/ --name nginx nginx:latest`
- Autre manière : créer un volume local directement avec docker :
  - `docker volume create nginxlocal`
- Pour avoir un résumé de la configuration du volume, taper :
  - `docker volume inspect nginxlocal`
- Le volume est situé ici : `/var/lib/docker/nginxlocal/_data`
- Installer un conteneur nginx utilisant ce volume :
  - `docker run -tid --name nginx -p 8080:80 --mountsource=nginxlocal,target=/usr/share/nginx/html nginx:latest`



# VOLUMES PERSISTANT DANS DOCKER

- `docker run -tid --name conteneur1 -v volume01:/usr/share/nginx/html/ -p 80:80 nginx`
- `docker volume inspect volume01`
  - Vous aurez le Mountpoint qui vous dira où est le volume
- `docker inspect conteneur1`
  - Vous retrouverez l'équivalence entre le conteneur et le volume
- On peut faire une sauvegarde tar.gz
  - Lancer un conteneur de sauvegarde léger type alpine
  - Monter le volume à sauvegarder `volume01:/tmp/src`
  - Monter le volume de destination `/tmp:/tmp/dest`
  - Ajouter la commande de sauvegarde type `tar -czvf <dest> <source>`
  - Prévoir la suppression automatique
- Se rendre dans le conteneur01 avec `docker exec` puis aller dans le dossier qui contient `index.html` et le modifier pour tester
- Pour sauvegarder ensuite :
  - `docker run --rm -v volume01:/tmp/src/ -v /tmp:/tmp/dest -u root alpine tar -czvf /tmp/dest/backup.tar.gz /tmp/src/`

# LE RÉSEAU DANS DOCKER

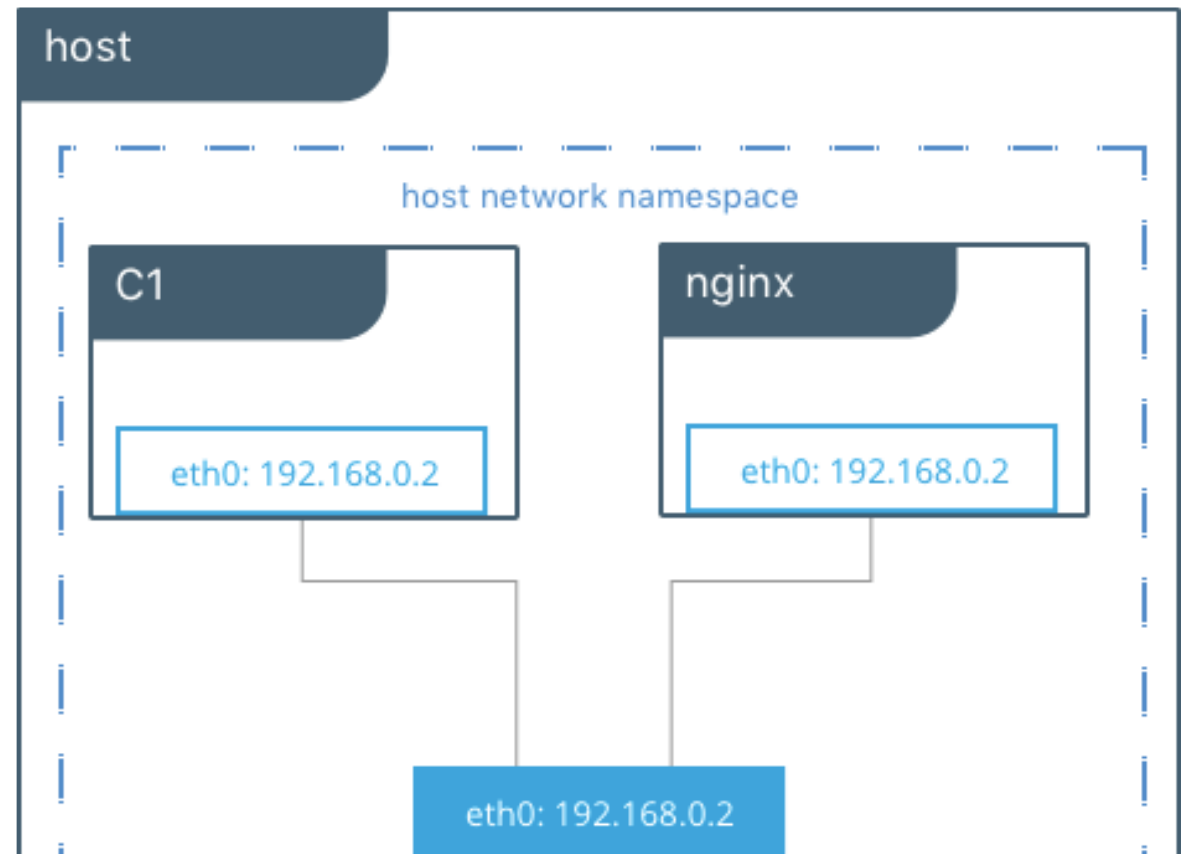
- Par défaut le réseau utilisé pour tous les conteneurs est le 172.17.0.0/16
- Docker assigne une ip automatiquement et dynamiquement dans ce range pour tout conteneur créé.
- Attention, si un conteneur redémarre, son ip peut changer.
- **Par défaut**, tous les conteneurs peuvent communiquer entre eux car ils sont créés dans le bridge .
- Il est possible de créer ses propres réseaux afin d'isoler les conteneurs entre eux.
- Les commandes utiles :
- `docker network ls` => liste les réseaux disponibles
- `docker inspect monréseau` => montre le détail du réseau nommé « monréseau »
- `docker network create -d bridge --subnet 172.18.0.0/24 monréseau` => créé le réseau 172.18.0.0/24 nommé monréseau

# LE RÉSEAU DANS DOCKER

- Il existe 8 types de réseau utilisable dans docker
  - bridge
  - host
  - The NONE Network
  - overlay
  - user defined bridge
  - macvlan (bridge)
  - macvlan (802.1q mode)
  - IPvlan (Layer 2)
  - IPvlan (Layer 3)

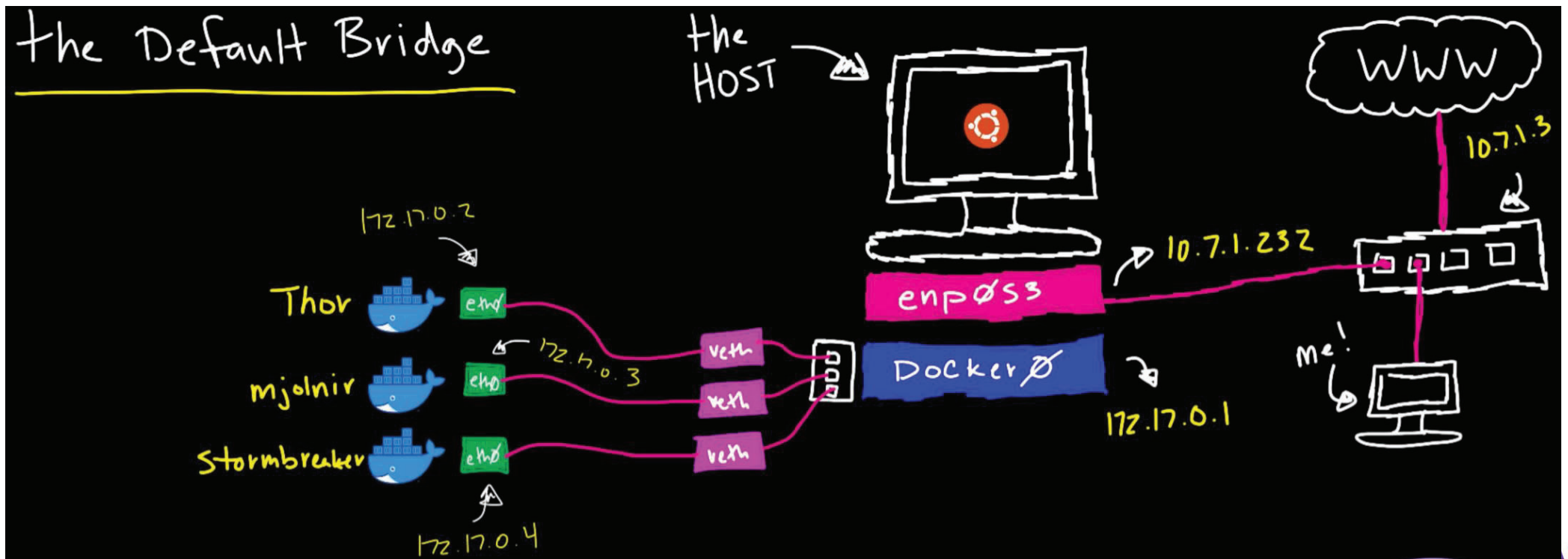
# HOST

- Ce type de réseau permet aux conteneurs d'utiliser la même interface que l'hôte.
- Il supprime l'isolation réseau entre les conteneurs et seront par défaut accessibles de l'extérieur. Il prendra donc la IP que votre machine hôte.



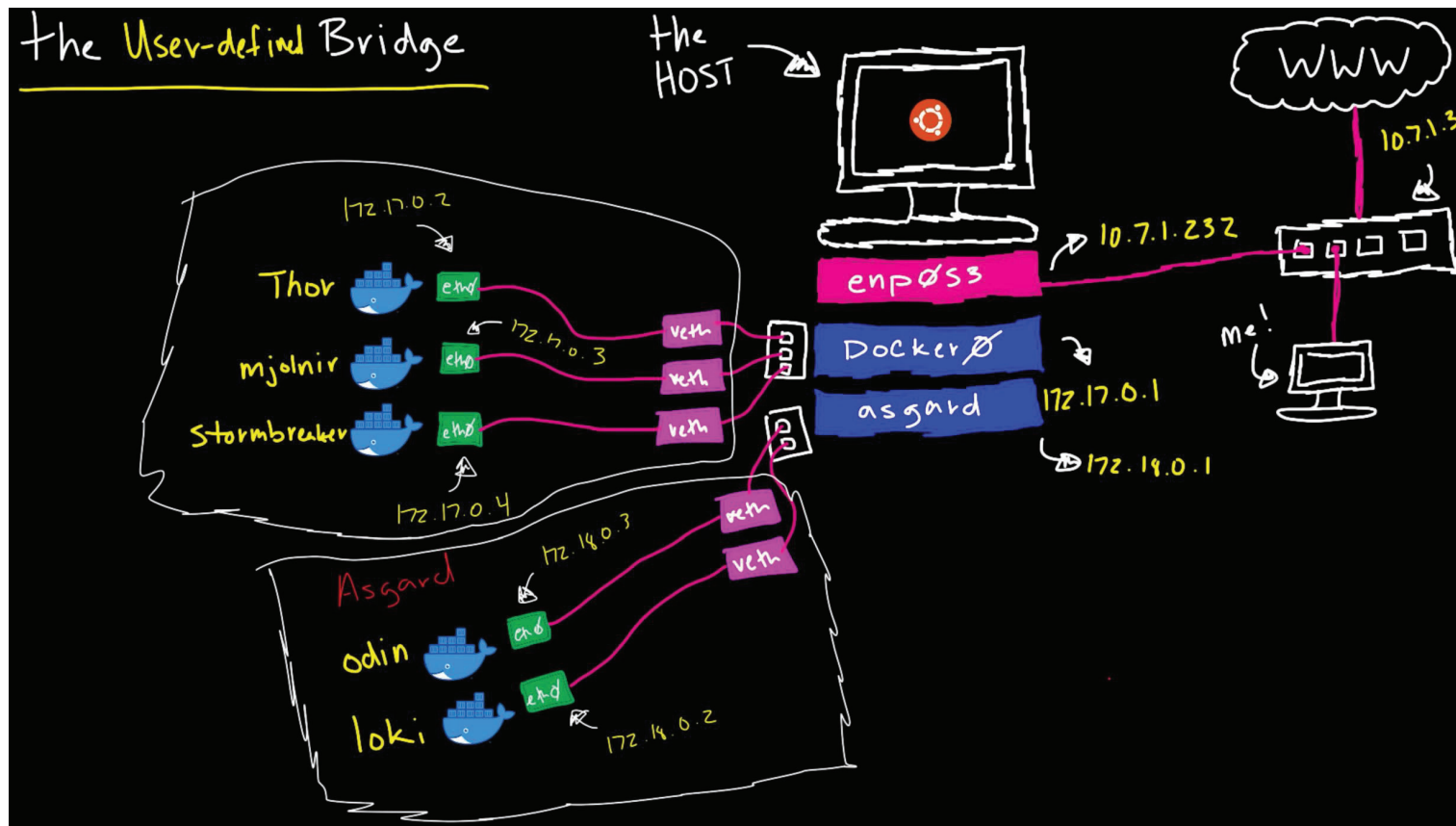
# DEFAULT BRIDGE

Sous réseau naté (NAT) sur le réseau normal (hôte) - 172.17.0.0



# USER DEFINED BRIDGE

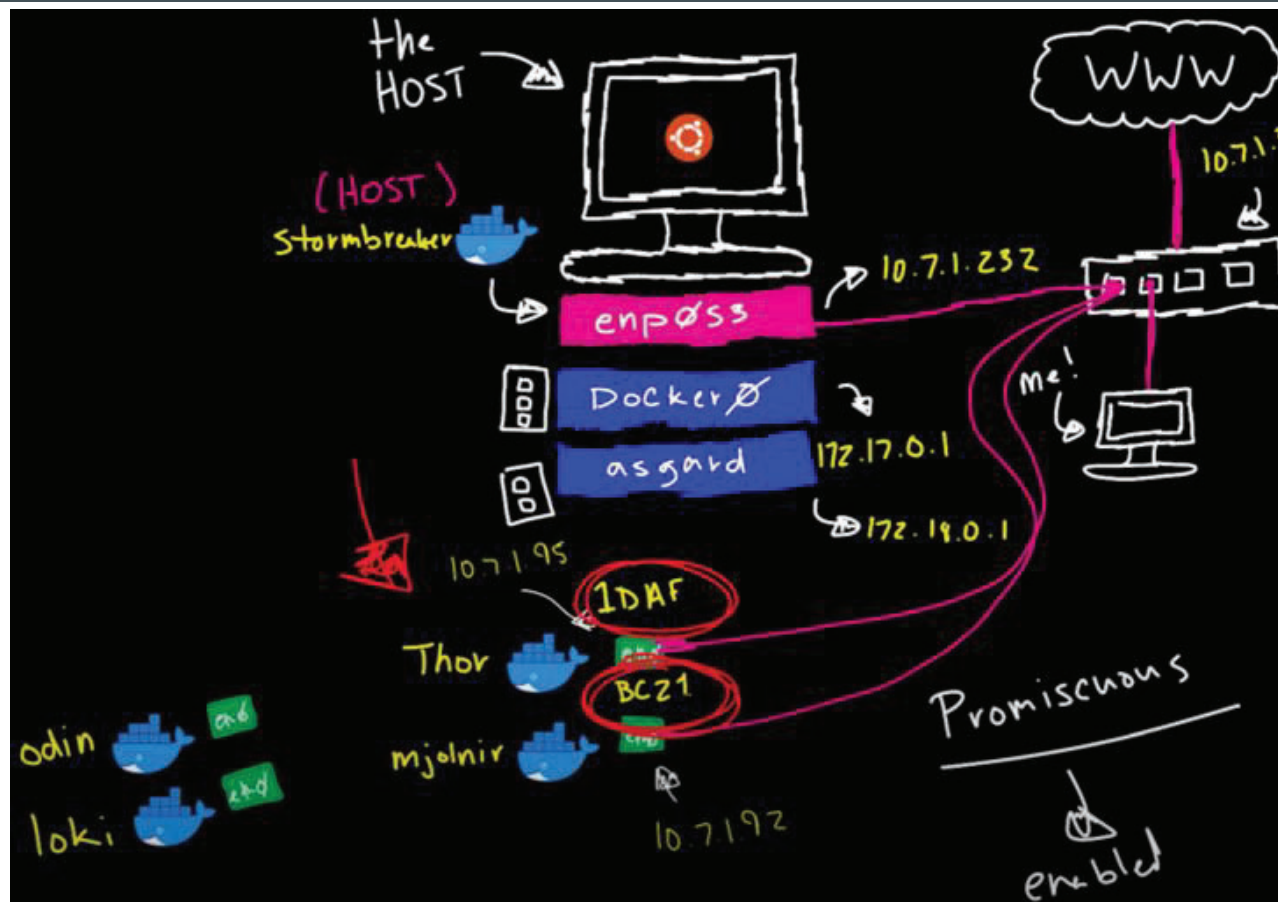
Autre réseau bridge, isolé les un des autres



# MACVLAN (BRIDGE)

- macvlan (bridge mode) - Connecté directement a notre réseau physique, un peu comme une VM - tu peux lui attribuer une adresse de ton réseau local
- Donner son subnet local et gateway (box, routeur) et son parent (l'interface réseau de l'hôte) pour mapper le reseau macvlan dessus
- Issue: Cela distribue une mac address au container et une interface réseau ne peux pas avoir plusieurs mac sur une interface
- Il faut donc accepter le mode promiscuité
  - `ip link set eth0 promisc on`
  - et dans l'hyperviseur (workstation, esxi...)
- pas de dhcp - SPECIFIER UNE IP ADDRESS POUR LE CONTAINER POUR EVITER UN CONFLIT AVEC L'IP DE L'HÔTE (VOIR DOC POUR LES OPTIONS AU DEPLOY)

# MACVLAN (BRIDGE)

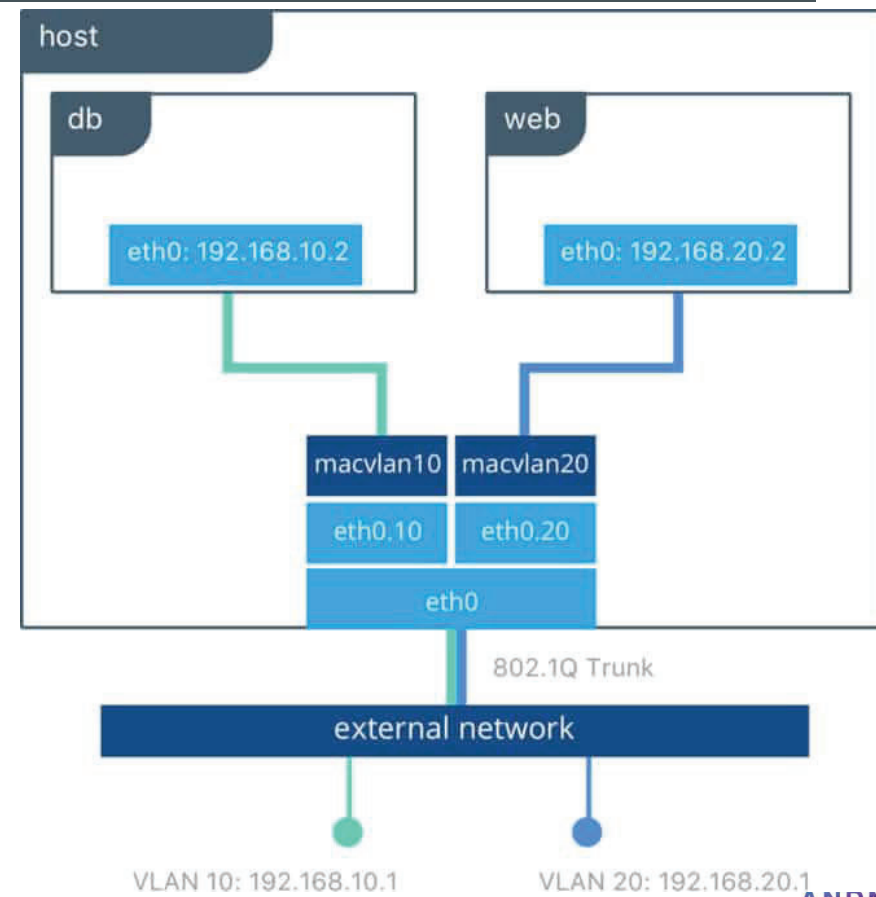
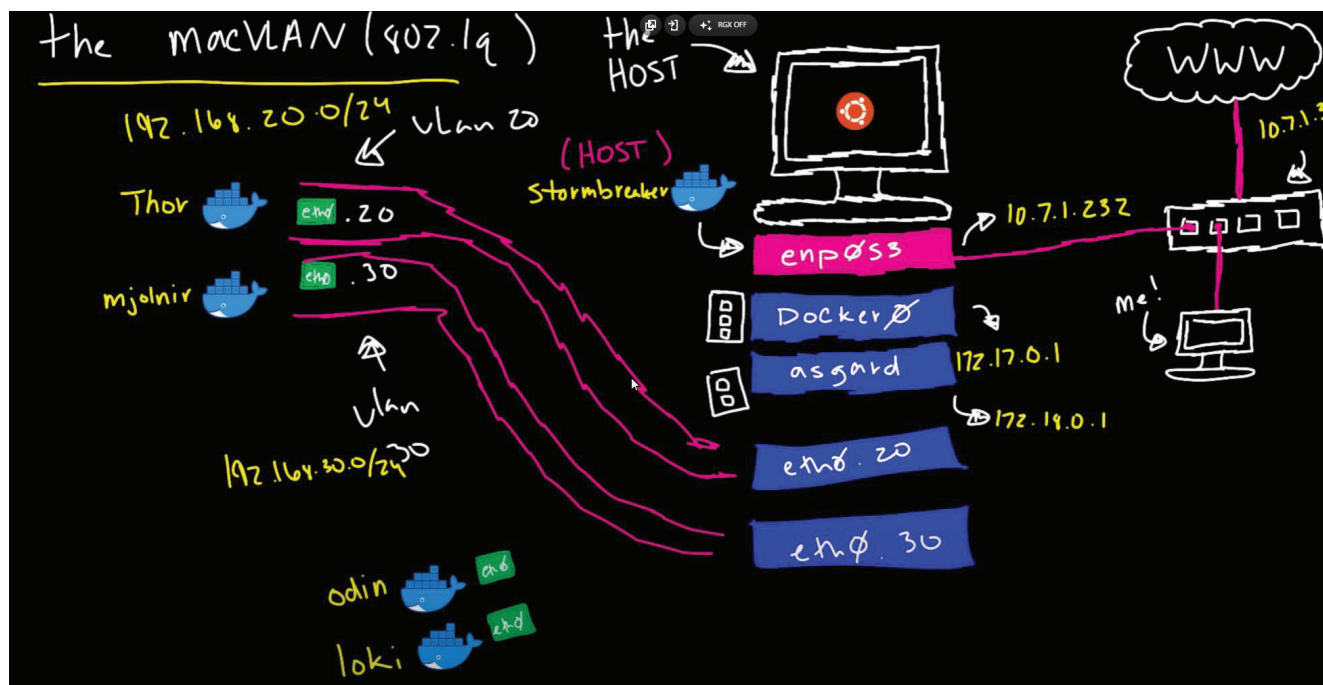




# MACVLAN (802.1Q (TRUNKED))

- macvlan (802.1q mode) - Connecté directement a notre réseau physique, un peu comme une VM - tu peux lui attribuer une adresse de ton réseau local, similaire à macvlan mais avec une notion de VLAN et de sous interfaces.
- Réseau sous forme de vlan : - eth0.20 eth0.30 etc..
- Donner son subnet local et gateway (box) et son parent (l'interface réseau de l'hôte avec .20 (num du vlan) eg: eth0.20) pour la mapper dessus
- Ce réseau crée au hôte des sous interfaces
  - `docker network create -d macvlan \`
  - `--subnet 192.168.20.0/24`
  - `--gateway 192.168.20.1 \`
  - `-o parent=eth0.20 \`
  - `macvlan20`

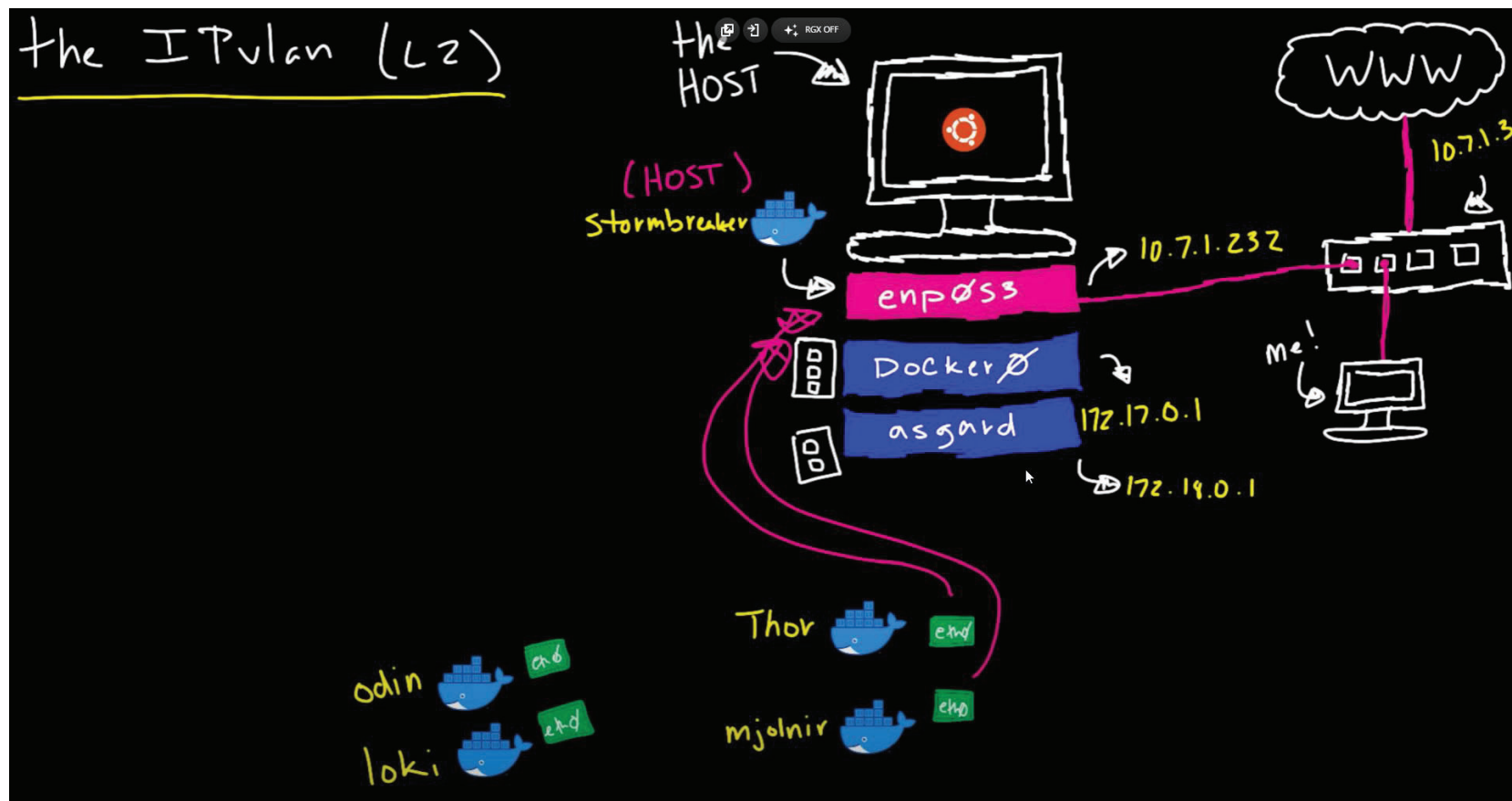
# MACVLAN (802.1Q (TRUNKED))



## IPVLAN (L2)

- Les container vont pouvoir avoir leur propre ip sur notre réseau local mais cette fois en partageant la mac address de l'hôte
  - `docker network create -d ipvlan \`
  - `--subnet 192.168.1.0/24`
  - `--gateway 192.168.1.1 \`
  - `-o parent=eth0 \`
  - `new_ipvlan_network`
- Puis run un container :
  - `docker run -tid --rm --network new_ipvlan_network \`
  - `--ip 192.168.1.4/24`
  - `-name container01 busybox`

# IPVLAN (L2)



# IPVLAN (L3)

- Le container (le network docker) considère l'host (son ip) comme étant le routeur, donc l'ip du host est la gateway
- Nous aurons donc besoin de créer des routes statiques sur l'hôte et/ou routeur pour faire communiquer nos réseaux.

- `docker network create -d ipvlan \`
- `--subnet 192.168.94.0/24` #le réseau que nous créons
- `-o parent=eth0 -o ipvlan_mode=l3 \` #pas de gateway car le parent est la gateway
- `--subnet 192.168.95.0/24` #on fait ça si on veut créer plusieurs réseau sur la même interface
- `new_ipvlan_network`

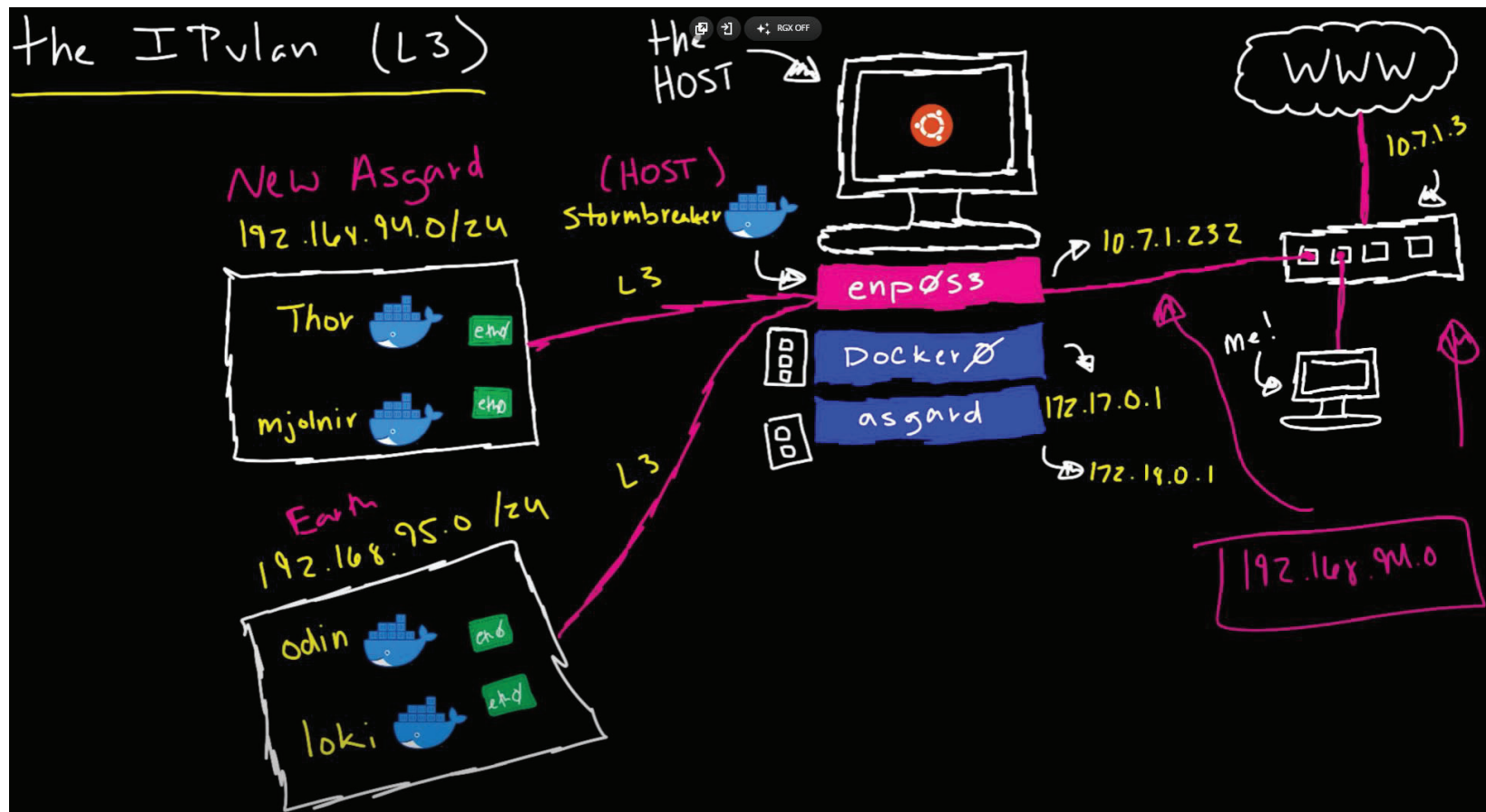
```
docker run -tid --rm --network new_ipvlan_network \
--ip 192.168.94.8/24
-name container01 busybox
```

```
docker run -tid --rm --network new_ipvlan_network \
--ip 192.168.95.9/24
-name container02 busybox
```

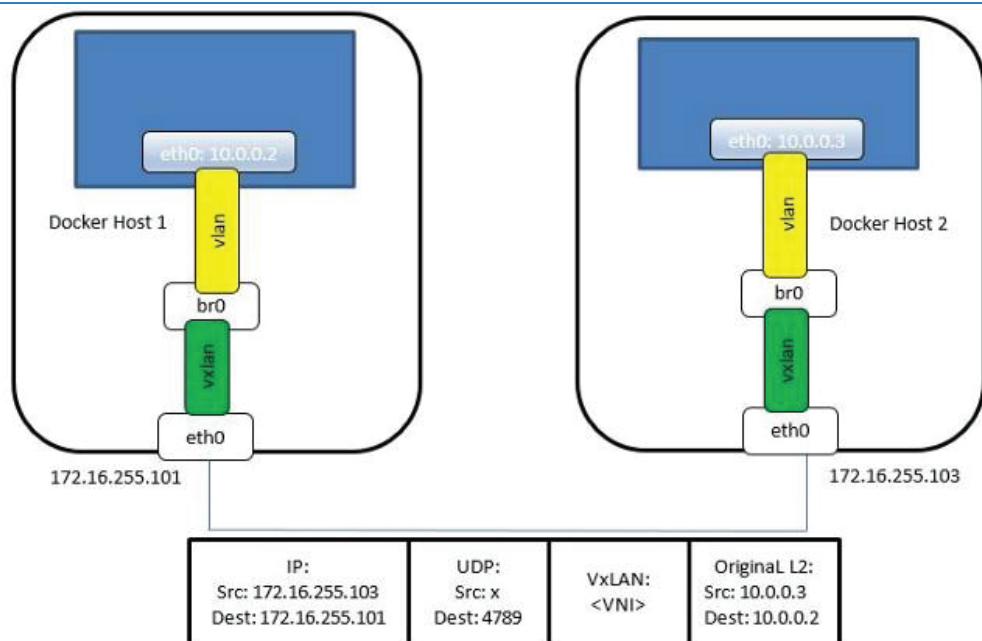
`docker inspect new_ipvlan_network`

- les container ont pas internet, pas de routes
- peut pinger les autres container et par le nom aussi
- donc ajouter une vraie route sur le réseau si on veut faire communiquer à l'extérieur
- no broadcast traffic

# IPVLAN (L3)

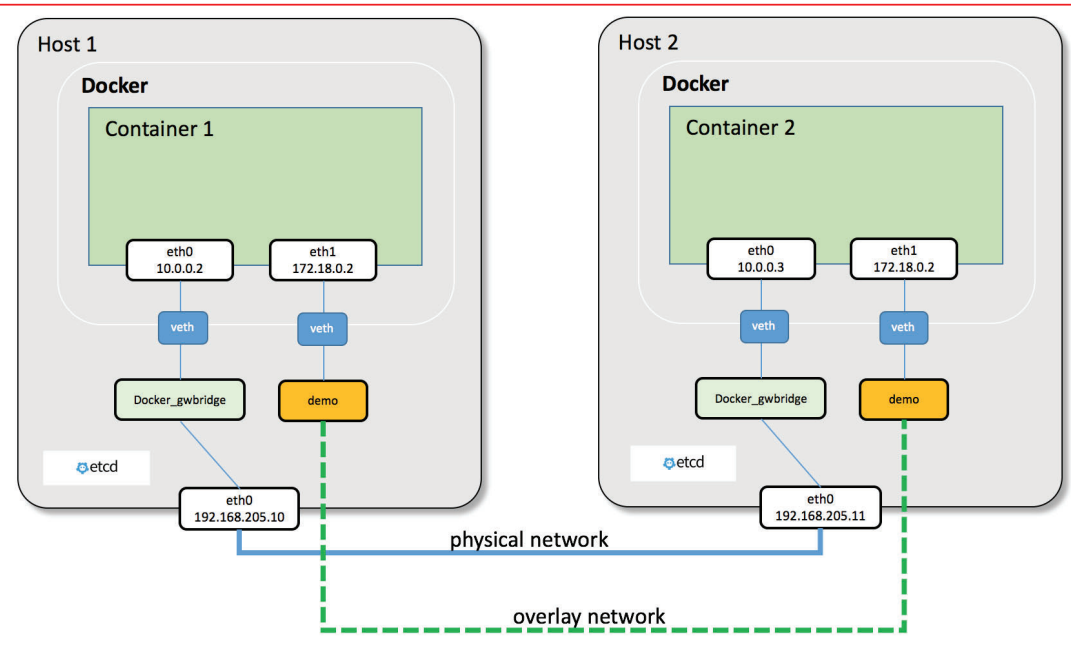


# OVERLAY (UTILISÉ DANS L'ORCHESTRATION) #SWARM



(Host) Docker01 : 172.16.255.101  
Container1 : eth0: 10.0.0.2

(Host) Docker03 : 172.16.255.103  
Container2 : eth0: 10.0.0.3



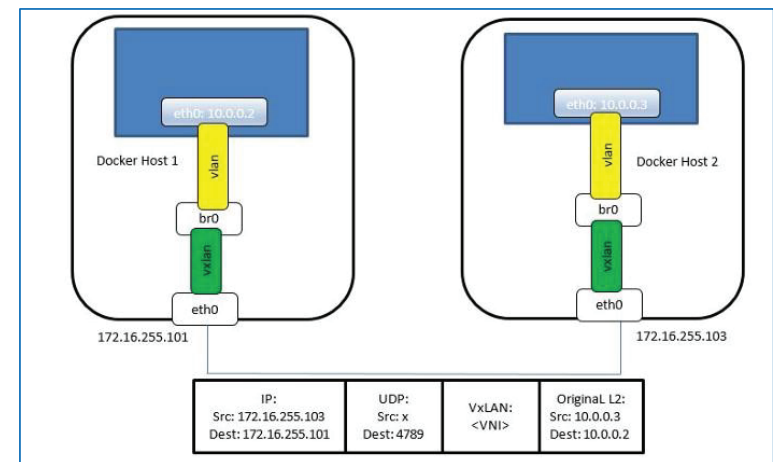
Dans un environnement docker avec 2 hôtes, chaque hôte a 1 conteneur à l'intérieur :

# OVERLAY (UTILISÉ DANS L'ORCHESTRATION) #SWARM

- Lorsque vous créez un réseau overlay, Docker crée un espace de noms pour le réseau sur l'hôte.
  - Il créera ensuite un périphérique de pont (par exemple br0) et une interface vxlan.
  - Lorsque vous créez un conteneur attaché à ce réseau, il sera attaché au bridge.
  - Lorsque vous envoyez ensuite du trafic entre les conteneurs sur différents hôtes, le périphérique réseau sur le conteneur l'envoie au périphérique vxlan et au pont br0, jusqu'à l'hôte.
  - L'hôte utilise l'en-tête vxlan pour acheminer le paquet vers le nœud de destination.
- 
- La méthode recommandée pour créer un réseau superposé de nos jours se fait en 2 étapes :
    1. Créez un réseau swarm entre les nœuds que vous souhaitez mettre en réseau.
    2. Créez un overlay sur le dessus et les nodes de swarm se découvriront automatiquement.

Avant de commencer à créer un réseau superposé à l'aide de Swarm, assurez-vous que les ports suivants sont ouverts et accessibles sur tous les nœuds hôtes Docker :

- Port TCP 2377
- Port TCP et UDP 7946
- Port UDP 4789





# NONE (NULL) NETWORK

- Il n'y a pas d'IP, juste l'adresse du localhost.

# NETWORK COMMAND

## ## Créer un réseau docker

```
docker network create --driver <DRIVER TYPE> <NETWORK NAME>
```

## # Lister les réseaux docker

```
docker network ls
```

## ## Supprimer un ou plusieurs réseau(x) docker

```
docker network rm <NETWORK NAME>
```

## ## Récupérer des informations sur un réseau docker

```
docker network inspect <NETWORK NAME>
```

-v ou --verbose : mode verbose pour un meilleur diagnostic

## ## Supprimer tous les réseaux docker non inutilisés

```
docker network prune
```

-f ou --force : forcer la suppression

## ## Connecter un conteneur à un réseau docker

```
docker network connect <NETWORK NAME> <CONTAINER NAME>
```

## ## Déconnecter un conteneur à réseau docker

```
docker network disconnect <NETWORK NAME> <CONTAINER NAME>
```

-f ou --force : forcer la déconnexion

## ## Démarrer un conteneur et le connecter à un réseau docker

```
docker run --network <NETWORK NAME> <IMAGE NAME>
```

# VARIABLES D'ENVIRONNEMENT

## VARIABLES D'ENVIRONNEMENT EN PYTHON

app.py

```
import os
from flask import Flask

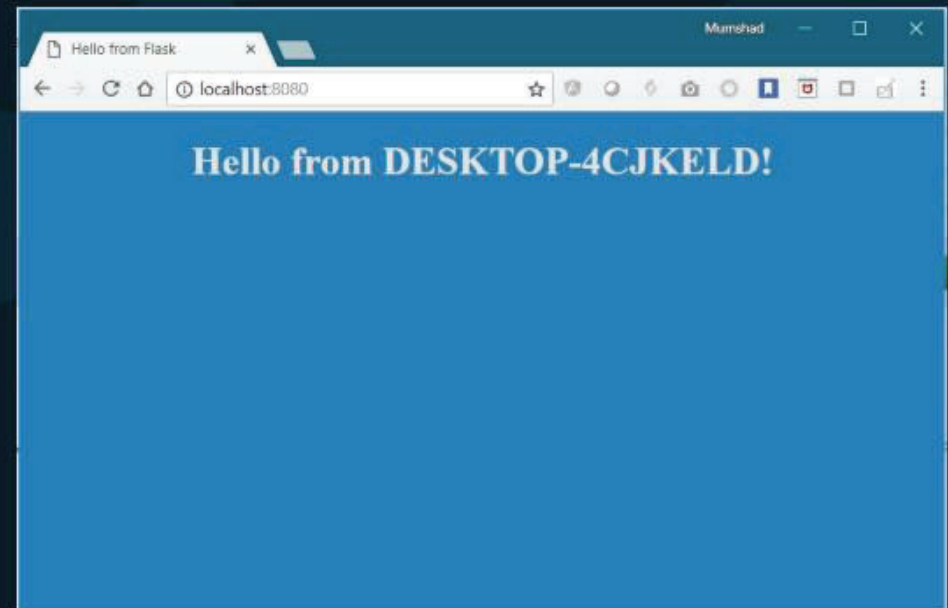
app = Flask(__name__)

...

color = os.environ.get('APP_COLOR')

@app.route("/")
def main():
    print(color)
    return render_template('hello.html', color=color)

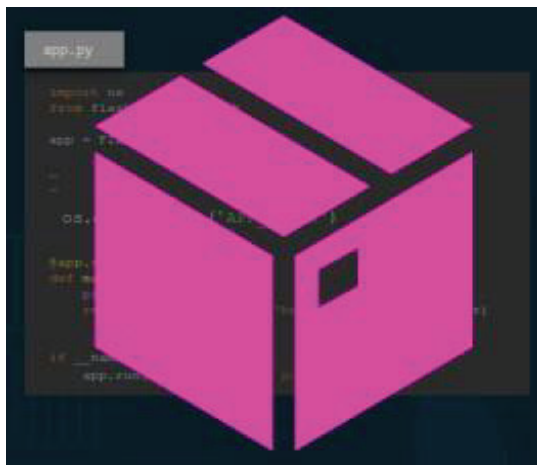
if __name__ == "__main__":
    app.run(host="0.0.0.0", port="8080")
```



```
export APP_COLOR=blue; python app.py
```

# VARIABLES D'ENVIRONNEMENT

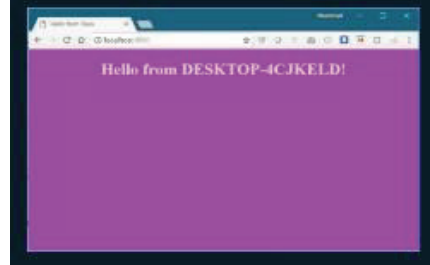
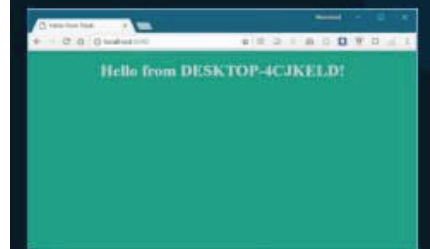
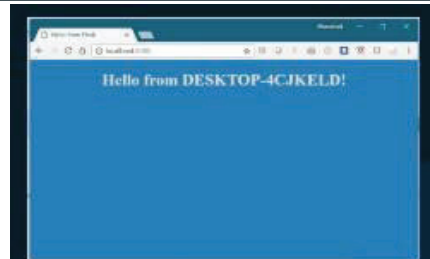
## VARIABLES D'ENVIRONNEMENT DANS DOCKER



```
▶ docker run -e APP_COLOR=blue simple-webapp-color
```

```
▶ docker run -e APP_COLOR=green simple-webapp-color
```

```
▶ docker run -e APP_COLOR=pink simple-webapp-color
```



# VARIABLES D'ENVIRONNEMENT

## INSPECTER LES VARIABLES D'ENVIRONNEMENT

```
▶ docker inspect blissful_hopper

[
  {
    "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
    "State": {
      "Status": "running",
      "Running": true,
    },
    "Mounts": [],
    "Config": {
      "Env": [
        "APP_COLOR=blue",
        "LANG=C.UTF-8",
        "GPG_KEY=0D96DF4D4110E5C43FBFB17F2D347EA6AA65421D",
        "PYTHON_VERSION=3.6.6",
        "PYTHON_PIP_VERSION=18.1"
      ],
      "Entrypoint": [
        "python",
        "app.py"
      ],
    },
  }
]
```

# DOCKER

```
curl -fsSL https://get.docker.com | sh; >/dev/null  
curl -sL "https://github.com/docker/compose/releases/download/v2.17.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
chmod +x /usr/local/bin/docker-compose
```

- Installer Docker :
  - Ajouter les repos suivant votre distribution
    - Exemple Debian: <https://docs.docker.com/engine/install/debian/>
  - Debian/ubuntu : `sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin`
- Installer une image et la lancer :
  - `docker run <image_name>`
  - `docker run -di <image_name>` (d : detach, l : interactive => permet de garder la main sur l'image)
- Lorsqu'on lance un « docker run », docker cherche d'abord l'image en local, si elle n'est pas disponible, il la télécharge depuis Docker Hub (par défaut). Une fois l'image présente en local, le conteneur est lancé.

# DOCKER COMMANDS

## RUN – PORT MAPPING

```
docker run kodecloud/webapp
```

```
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

http://172.17.0.2:5000

Internal IP

```
docker run -p 80:5000 kodecloud/simple-webapp
```

```
docker run -p 8000:5000 kodecloud/simple-webapp
```

```
docker run -p 8001:5000 kodecloud/simple-webapp
```

```
docker run -p 3306:3306 mysql
```

```
docker run -p 8306:3306 mysql
```

```
docker run -p 8306:3306 mysql
```

```
root@osboxes:/root # docker run -p 8306:3306 -e MYSQL_ROOT_PASSWORD=pass mysql
docker: Error response from daemon: driver failed programming external connectivity on endpoint boring_bhabha (5079d342b7e8ee11c71d46): Bind for 0.0.0.0:8306 failed: port is already allocated.
```



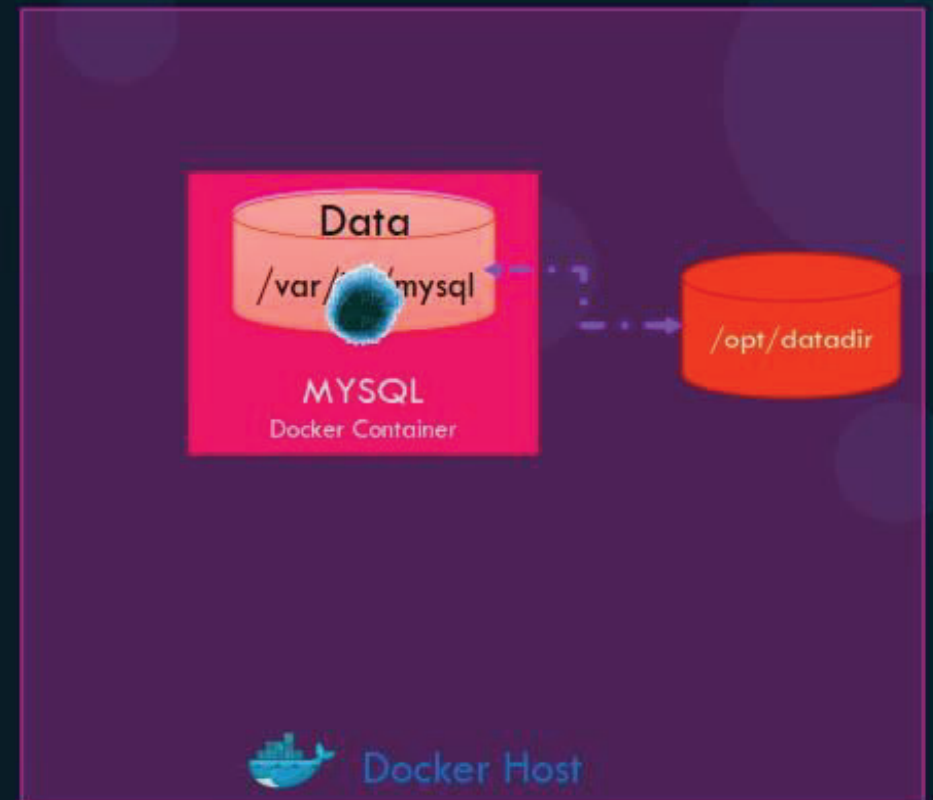
# DOCKER COMMANDS

## RUN – VOLUME MAPPING

```
docker run mysql
```

```
docker stop mysql  
docker rm mysql
```

```
docker run -v /opt/datadir:/var/lib/mysql mysql
```





# DOCKER COMMANDS

## INSPECT CONTAINER

```
docker inspect blissful_hopper

[
  {
    "Id": "35505f7810d17291261a43391d4b6c0846594d415ce4f4d0a6ffbf9cc5109048",
    "Name": "/blissful_hopper",
    "Path": "python",
    "Args": [
      "app.py"
    ],
    "State": {
      "Status": "running",
      "Running": true,
    },
    "Mounts": [],
    "Config": {
      "Entrypoint": [
        "python",
        "app.py"
      ],
    },
    "NetworkSettings": {...}
  }
]
```

# DOCKER COMMANDS

## CONTAINER LOGS

```
▶ docker logs blissful_hopper
```

```
This is a sample web application that displays a colored background.  
A color can be specified in two ways.
```

1. As a command line argument with `--color` as the argument. Accepts one of red,green,blue,blue2,pink,darkblue
  2. As an Environment variable `APP_COLOR`. Accepts one of red,green,blue,blue2,pink,darkblue
  3. If none of the above then a random color is picked from the above list.
- Note: Command line argument precedes over environment variable.

```
No command line argument or environment variable. Picking a Random Color =blue
```

```
* Serving Flask app "app" (lazy loading)
```

```
* Environment: production
```

```
WARNING: Do not use the development server in a production environment.
```

```
Use a production WSGI server instead.
```

```
* Debug mode: off
```

```
* Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```

# DOCKER

- `docker ps` : permet de lister les processus
- `docker ps -a` : permet de lister tous les processus docker, même ceux qui sont terminés
- `docker exec -ti nomduconteneur sh` : Se connecter au conteneur en shell
  
- Exemple avec un serveur web, nginx :
- `docker run -tid -p 8080:80 --name monnginx nginx:latest`
  - `/usr/share/nginx/html/index.html`
    - Modifier le contenu et tester
- Explication :
- Lancer le conteneur nginx, le nommer monnginx, le laisser fonctionner. Rediriger son port 80 sur le 8080 de notre machine locale
  
- Accéder au paramétrage du conteneur :
- `docker inspect nom du conteneur` (permet de voir l'adressage ip entre autre)

# QUELQUES COMMANDES DE BASE

Commande	Explication
<code>docker image ls</code>	Lister les images existantes
<code>docker container ls -a</code>	Lister les conteneurs
<code>docker ps -a</code>	Lister les conteneurs
<code>docker run alpine:latest</code>	Lancer ou créer le conteneur alpine
<code>docker run -di --name alpinetest alpine:latest</code>	Lancer ou créer le conteneur alpine, le nommer alpinetest et le laisser fonctionner (mode Detach Interactive)
<code>docker run -di --name alpinetest alpine:latest sleep infinity</code>	Comme au dessus mais en plus on conserve un process qui tourne sur le conteneur
<code>docker run -tid -p 8080:80 --name monnginx nginx:latest</code>	Lancer le conteneur nginx, le nommer monnginx, le laisser fonctionner. Rediriger son port 80 sur le 8080 de notre machine locale
<code>docker run -tid --name conteneur2 --link conteneur1 alpine</code>	Lancer un conteneur alpine que l'on nomme conteneur2, le linker avec conteneur1 (conteneur2 pourra pinguer conteneur1)
<code>docker start monconteneur</code>	Redémarrer un conteneur arrêté
<code>docker exec -ti monnginx sh</code>	Se connecter au conteneur nommé monnginx avec le shell et le bach
<code>docker stop &lt;nom du conteneur&gt;</code>	Arrêter un conteneur
<code>docker container kill \$(docker ps -q)</code>	Tuer tous les conteneurs
<code>docker start &lt;nom du conteneur&gt;</code>	Redémarrer un conteneur
<code>docker rm -f &lt;nom du conteneur&gt;</code>	Supprimer un conteneur (-f permet de forcer, même si le conteneur tourne)
<code>docker rm \$(docker ps -a -q)</code>	Supprimer tous les conteneurs
<code>docker inspect &lt;nom du conteneur&gt;</code>	Obtenir les infos sur un conteneur
<code>docker inspect -f "{{.NetworkSettings.IPAddress}}" &lt;nom du conteneur&gt;</code>	Obtenir l'ip d'un conteneur

# CHEAT SHEET

Commande	Description
<code>docker ps</code>	Visualiser les conteneurs actifs
<code>docker ps -a</code>	Visualiser tous les conteneurs
<code>docker rm [container]</code>	Supprimer un conteneur inactif
<code>docker rm -f [container]</code>	Forcer la suppression d'un conteneur actif
<code>docker images</code>	Lister les images existantes
<code>docker rmi [image]</code>	Supprimer une image docker
<code>docker exec -t -i [container] /bin/bash</code>	Exécuter des commandes dans un conteneur actif
<code>docker inspect [container]</code>	Inspecter la configuration d'un conteneur
<code>docker build -t [image] .</code>	Construire une image à partir d'un Dockerfile
<code>docker history [image]</code>	Visualiser l'ensemble des couches d'une image
<code>docker logs --tail 5 [container]</code>	Visualiser les logs d'un conteneur (les 5 dernières lignes)

# CHEAT SHEET - REGISTRY

Commande	Description
docker login	Se connecter au registry
docker search [name]	Rechercher une image
docker pull [image]	Récupérer une image
docker push [image]	Pousser une image du cache local au registry
docker tag [UUID] [image]:[tag]	Tagger une image

# DOCKER-COMPOSE

- Compose est un outil pour définir et exécuter des applications multi-conteneurs, aussi appelé "**stack**".
- Les conteneurs étant idéaux pour des applications basées sur des micro services, il devient évident que rapidement on doit faire face à l'interconnexion de plusieurs conteneurs et à la gestion de multi-conteneurs.
- Un fichier docker-compose s'écrit avec le langage YAML.
- Il se présentera sous la forme **docker-compose.yml**
- Références officiels : <https://docs.docker.com/compose/reference/>
- Documentation d'installation : <https://docs.docker.com/compose/install/#install-compose>

# DOCKER-COMPOSE

- Arguments de docker-compose !
  - **image** qui permet de spécifier l'image source pour le conteneur ;
  - **build** qui permet de spécifier le Dockerfile source pour créer l'image du conteneur ;
  - **volume** qui permet de spécifier les points de montage entre le système hôte et les conteneurs ;
  - **restart** qui permet de définir le comportement du conteneur en cas d'arrêt du processus ;
  - **environment** qui permet de définir les variables d'environnement
  - **depends\_on** qui permet de dire que le conteneur dépend d'un autre conteneur ;
  - **ports** qui permet de définir les ports disponibles entre la machine host et le conteneur.



# DOCKER-COMPOSE

- Pour utiliser docker-compose, vous devez d'abord créer un fichier yml, ici quelques exemples de docker-compose.yml

```
version: "3.9" # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "8000:5000"
    volumes: - ./code - logvolume01:/var/log
    depends_on:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

```
services:
  webapp:
    image: examples/web
    ports:
      - "8000:8000"
    volumes:
      - "/data"
```

```
version: "3"

services:
  web:
    image: "apache:${PHP_VERSION}"
    restart: 'always'
    depends_on:
      - mariadb
    ports:
      - '8080:80'
    links:
      - mariadb
  mariadb:
    image: "mariadb:${MARIADB_VERSION}"
    restart: 'always'
    volumes:
      - "/var/lib/mysql/data:${MARIADB_DATA_DIR}"
      - "/var/lib/mysql/logs:${MARIADB_LOG_DIR}"
      - /var/docker/mariadb/conf:/etc/mysql
    environment:
      MYSQL_ROOT_PASSWORD: "${MYSQL_ROOT_PASSWORD}"
      MYSQL_DATABASE: "${MYSQL_DATABASE}"
      MYSQL_USER: "${MYSQL_USER}"
      MYSQL_PASSWORD: "${MYSQL_PASSWORD}"
```

# DOCKER-COMPOSE

- Si par exemple vous voulez installer un wordpress avec docker-compose (en partant du principe que vous n'utilisez pas déjà l'image existante sur le hub ;)
- Vous devrez installer une base de donnée et un serveur web.
- `docker-compose up -d`
- Puis <http://127.0.0.1:8000> pour accéder à votre wordpress

```
version: '3'
services:
  db:
    image: mysql:5.7
    volumes:
      - db_data:/var/lib/mysql
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: somewordpress
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpress
      MYSQL_PASSWORD: wordpress

  wordpress:
    depends_on:
      - db
    image: wordpress:latest
    ports:
      - "8000:80"
    restart: always
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpress
      WORDPRESS_DB_PASSWORD: wordpress
      WORDPRESS_DB_NAME: wordpress

volumes:
  db_data: {}
```

# DOCKER COMPOSE - CHEAT SHEET

Commande	Description
<code>docker-compose up -d</code>	Démarre un ensemble de conteneurs en arrière-plan
<code>docker-compose down</code>	Stoppe un ensemble de conteneurs
<code>docker-compose exec [service] [command]</code>	Exécute une commande au sein d'un service
<code>docker-compose ps</code>	Affiche les stacks
<code>docker-compose logs -f --tail 5</code>	Afficher les logs du stack
<code>docker-compose stop</code>	Arrêter un stack
<code>docker-compose config</code>	Valider la syntaxe de votre fichier docker-compose

# TP DOCKER-COMPOSE

- Monter une stack LAMP avec docker compose.
- LAMP : LINUX APACHE MYSQL PHP
- help :
  - <https://linuxconfig.org/how-to-create-a-docker-based-lamp-stack-using-docker-compose-on-ubuntu-18-04-bionic-beaver-linux>
  - <https://connect.ed-diamond.com/Linux-Pratique/lp-111/creer-son-environnement-de-developpement-lamp-grace-a-docker-compose>
  - <https://github.com/sprintcube/docker-compose-lamp>
  - [https://cours.brosseau.ovh/tp/docker/docker\\_compose.html](https://cours.brosseau.ovh/tp/docker/docker_compose.html)