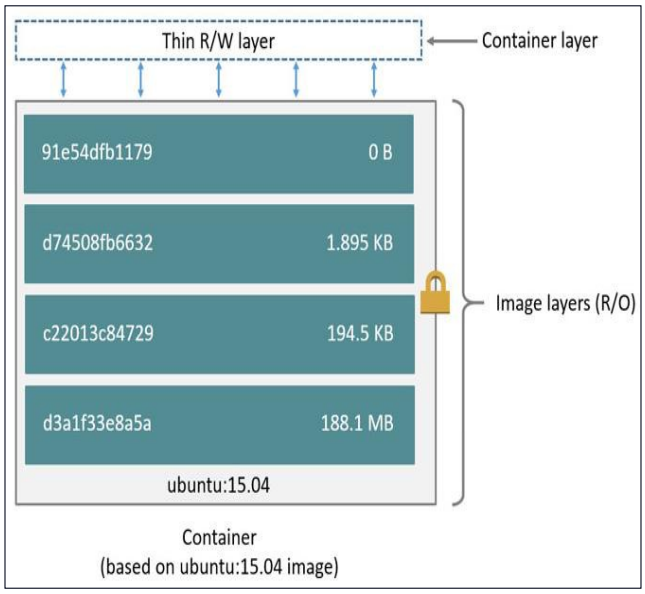
Docker Storage

Docker Storage

Docker Image:



Que se passe't'il quand un conteneur s'arrête ?

- La couche de conteneur est tout simplement perdue.

Si cette couche comprend des données à conserver entre deux lancements de conteneurs ou bien à partager entre plusieurs conteneurs, il faut utiliser un volume (ou data volume).

Image Source: https://docs.docker.com/storage/storagedriver/#images-and-layers

Docker Storage

Support de stockage

- Fournit un stockage interne temporaire pour les conteneurs.
- Gère et contrôle la façon dont les images et les conteneurs sont stockés sur votre hôte Docker.

Stockage Docker :

- Stockez et gérez les données des conteneurs.

Deux types de stockage :

- Non persistante
- Persistant

Reference : https://docs.docker.com/storage/storagedriver/select-storage-driver/

Docker Storage

Stockage non persistant :

- Les données résident dans le conteneur
- Sont supprimés lorsque le conteneur est supprimé
- Tous les conteneurs l'ont par défaut.

Pilotes de stockage :

- RHEL/Les derniers Ubuntu et CentOS utilisent Overlay2
- CentOS 7 et versions antérieures utilisent DeviceMapper
- Windows utilise son propre système.

Emplacement de stockage:

- Linux : /var/lib/docker/[STORAGE-DRIVER]/
- Windows : C:\ProgramData\Docker\windowsfilter\

Docker Storage

Stockage persistant:

- Les données ne résident pas dans le conteneur
- Ils ne sont pas supprimés lorsque le conteneur est supprimé
- Deux types de stockage persistant :

1.Volumes:

- Monté sur un répertoire dans un conteneur.
- Emplacement de stockage:
  - Linux : /var/lib/docker/volumes/
  - Windows : C:\ProgramData\Docker\volumes
- Prend en charge les pilotes tiers :
  - Stockage en bloc, par ex. Amazon AWS EBS.
  - Stockage de fichiers, par ex. AmazonAWSEFS.
  - Stockage d'objets, par ex. AmazonAWS S3.

2.Bind Mounts :

- Le fichier ou le répertoire sur le système hôte est monté dans le fichier ou le répertoire d'un conteneur.

Reference Doc :

Docker Storage : Volume

Volumes:

- Monté un répertoire dans un conteneur.

Volume CLI:

- Creer un Volume.
  - docker volume create [**volume name**]
- Lister les Volumes.
  - docker volume ls
- Inspecter un Volume.
  - docker volume inspect [**volume name**]
- Supprimer un volume.
  - docker volume rm [**volume name**]
- Supprimez tous les volumes inutilisés
  - docker volume prune

Reference Doc: https://docs.docker.com/storage/volumes/

Docker Storage : Volume

Deux façons de monter un volume dans un conteneur :

1. **- - mount**

   Syntaxe:

```
docker container run -d \
        --name mynginx1 \
        --mount type=volume,\
          source=nginxvolume,\
          target=/usr/share/nginx/html/ \
          nginx
```

docker volume create nginxvolume

directory inside mynginx1 container

2. **- - volume or - v**

   Syntaxe:

```
docker container run -d \
        --name mynginx2 \
        -v nginxvolume:/usr/shared/nginx/html/ \
        nginx
```

Volume name should NOT start with slash (/)
Correct: -v nginxvolume:/usr/shared/nginx/html/
Wrong: -v /nginxvolume:/usr/shared/nginx/html/

Reference Doc : https://docs.docker.com/storage/volumes/

Docker Storage : Bind Mounts

Bind Mounts:

- Le fichier ou le répertoire sur le système hôte est monté dans le fichier ou le répertoire d'un conteneur.

Deux façons de créer des Bind Mounts:

1. **- - mount**

   Syntaxe:

```
docker container run -d \
        --name nginxbind1 \
        --mount type=bind,\
          source="$(pwd)"/bindexample,\ target=/app \
          nginx
```

mkdir bindexample

directory inside nginxbind1 container

2. **--volume or –v**

   Syntaxe :

```
docker container run -d \
        -name nginxbind2 \
        -v /user/username/bindexample2:/app \
         nginx
```

Bind Mount is a file or directory on the host system.
Therefore, Bind Mount name should start with slash (/)
Correct: -v /user/username/bindexample2:/app
Wrong: -v user/username/bindexample2:/app

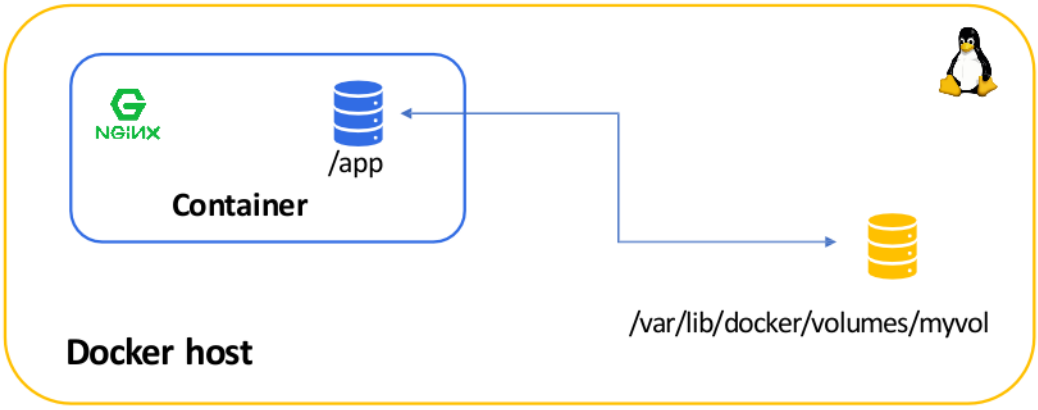
Reference Doc : https://docs.docker.com/storage/bind-mounts/

Docker Storage : Volume

Volume:

- Mounting a volume created using 'docker volume create' command
- mounting it from default volume location /var/lib/docker/volumes

```
docker volume create myvol
docker run -d --name nginx -v myvol:/app nginx
docker run -d --name nginx --mount source=myvol2,target=/app nginx
```

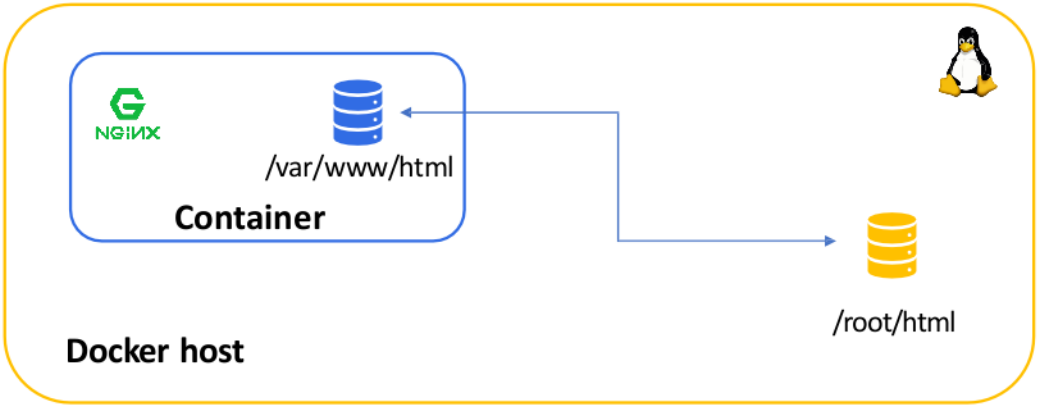


Docker Storage : Volume

Bind Mpunts:

- External mounting(external hard disks etc.)
- Bind mounts may be stored anywhere on the host system.
- They usually start with '/'

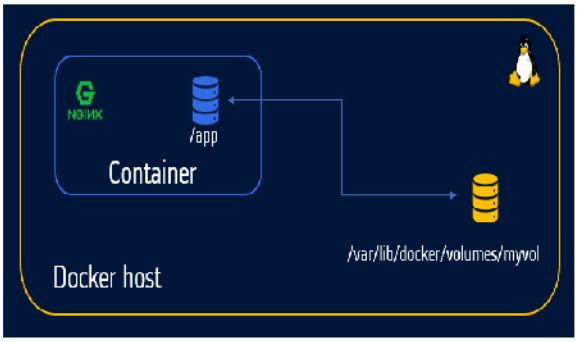
```
docker run –name web -v /root/html:/var/www/html/ nginx
```



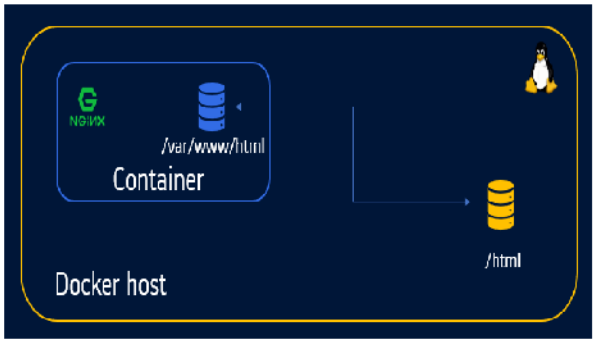
Docker Storage : Volume

Docker volumes commands

docker volume create <volume_name>
docker volume ls
docker volume inspect <volume_name>
docker volume rm <volume_name>
docker volume prune



Named Volume



Bind Volume

Docker Storage : Volume

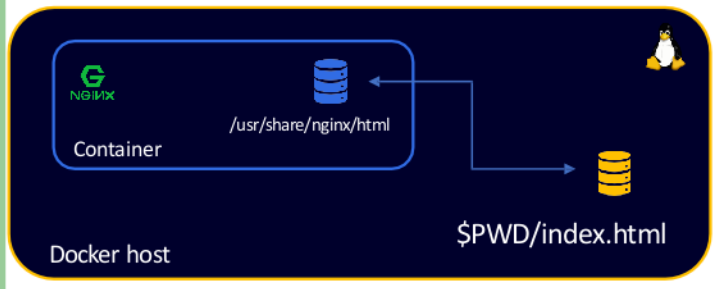
Demo Hosting a static website using nginx

docker run -d --name web –p 80:80 nginx

docker exec -it web bash
root@768faf801706:/# ls /usr/share/nginx/html
50x.htm index.html

docker run -d --name web -p 80:80 -v

$PWD:/usr/share/nginx/html nginx



CPU & Memory

Docker propose plusieurs options pour la limitation de la mémoire:

- **--memory / -m :** mémoire maximum qu'un container peut utiliser. C'est une "hard limit" que le container ne pourra pas dépasser
  - docker container run -m 8m mongo
  - -m 4m : conteneur ne doit pas depasser 8 mega
- **--memory-reservation :** mémoire maximum qu'un container peut utiliser si Docker détecte une utilisation élevée de la RAM sur la machine hôte. Cette valeur doit être inférieur à celle spécifiée dans --memory. C'est une "limite soft", on ne peut pas être sur que le container ne la dépassera pas.
- **--oom-kill-disable :** utilisée conjointement avec l'option --memory, elle permet de s'assurer que le container ne sera pas tué par le kernel si une exception Out Of Memory est levée.

Reference https://docs.docker.com/config/containers/resource_constraints/

CPU & Memory

Docker propose plusieurs options pour la limitation de CPU:

- **--cpus=value** : permet de spécifier les ressources CPU qu'un container peut utiliser
  - --cpus="2", sur une machine contenant 4 CPUs, assure que le container ne pourra pas utiliser plus de 2 CPUs.
- **--cpuset-cpus** : spécifie les CPUs ou coeurs qu'un container peut utiliser.
  - --cpuset-cpus=0-7 assure que le container utilisera les 8 coeurs (index de 0 à 7 car Le premier CPU est numéroté 0)
  - --cpuset-cpus=3,4 assure que le container n'utilisera que les coeurs 4 et 5 (index 3 et 4)

Reference https://docs.docker.com/config/containers/resource_constraints/

DockerFile

DockerFile

Dockerfile:

Dockerfile est un ensemble d'instructions et de commandes utilisées pour créer une image.

Build Image:

- docker image build -t [TAG] . ⇐ n'oubliez pas le point
- docker image build -t [TAG] -f [Dockerfile Name] .

- Exemple :
  - Docker image build myubuntu:1.0 .

```
FROM ubuntu:22.04
LABEL version="1.0"
RUN apt update -y
```

Reference Doc : https://docs.docker.com/engine/reference/builder/

DockerFile

Instructions de Dockerfile:

- **FROM :** spécifier l'image Docker parente à utiliser.
- **LABEL :** ajoute des métadonnées à l' image (descriptions).
- **RUN :** Permet d'exécuter des commandes, utilisées généralement pour construire l'image (new layer)
- **EXPOSE :** Spécifier les port(s) écouté(s) par le conteneur.
- **CMD :** permet d'exécuter une commande au démarrage du conteneur résultant
- **ENTRYPOINT :** permet , comme CMD, d'exécuter une commande au démarrage du conteneur

	CMD	ENTRYPOINT
Spécifier les commandes par défaut à exécuter lorsque l'image Docker est exécutée en tant que conteneur	oui	oui
Les arguments peuvent être remplacés par l'utilisateur	oui	non

Reference Doc : https://docs.docker.com/engine/reference/builder/

DockerFile

Instructions de Dockerfile:

- **WORKDIR :** permet de changer le chemin courant (appelé dossier de travail) pour les instructions RUN, CMD, ENTRYPOINT, COPY et ADD. Elle peut être utilisée plusieurs fois dans un fichier Dockerfile. Son effet s'applique à toute instruction qui suit.
- **COPY et ADD :**
- ADD et COPY permettent de copier des fichiers qui se trouvent dans notre machine locale vers l'image
- La principale différence entre les deux, est que
- ADD permet également de copier des fichiers via une URL.
- si la source est un fichier local compressé de format reconnu (tar, gzip,..), alors le fichier est automatiquement décompressé en un dossier.
- Lorsque la source est un fichier défini par une URL, il ne sera pas décompressé, et ce même s'il correspond à une archive de format reconnu.

- **Remarque :** Les instructions ne sont pas sensibles à la casse. Cependant, la convention veut qu'elles soient en MAJUSCULES pour les distinguer plus facilement des arguments.

DockerFile

Sample Dockerfile:

```
# Pull the minimal Ubuntu image
FROM ubuntu

# Install Nginx
RUN apt update -y && apt install nginx -y

# Change Directory
WORKDIR /var/www/html

# Copy the Nginx config
COPY index.html .

# Expose the port for access
EXPOSE 80/tcp

# Run the Nginx server
CMD ["/usr/sbin/nginx", "-g", "daemon off;"]
```

```
7   RUN apt-get update -y && \
8       apt-get install curl -y && \
9       apt-get install nginx -y
```

Combining RUN instructions into one line.

```
10  WORKDIR /var/www
11  WORKDIR html
```

In the above snapshot, html does not start with / (slash) so it becomes **relative** to */var/*www

Which is same as WORKDIR /var/www/html

après la commande nginx se trouve l'option -g 'daemon off;' qui exécute le processus principal du serveur nginx au premier plan.

DockerFile

Instructions de Dockerfile:

ARG et ENV

- permettent de créer des variables
- La principale différence entre les deux, est que
  - ARG définit des variables disponibles uniquement pendant la phase de construction d'une image.
  - ENV définit des variables d'environnement qui restent disponibles même après la construction, lors de l'exécution du conteneur.

USER

- permet de définir l'utilisateur qui exécute les commandes issues des instructions RUN, CMD et ENTRYPOINT. Son effet s'applique à toute instruction qui suit et à toute image enfant.

```
ARG APP_VERSION=1.0
FROM ubuntu:${APP_VERSION
ENV myName="ali ahmed " myJob=Agent\ secret
CMD echo $myNam
USER ali
CMD sudo ls /etc
```

DockerFile

Sample Dockerfile: ENV and USER Instructions.

```
1  FROM ubuntu
2
3  LABEL description="My image for nginx Web Server"
4  LABEL version="V1.0"
5
6  ENV PORT="80"
7  ENV ENVIRONMENT="development"
8
9  RUN apt-get update -y && \
10     apt-get install curl -y
11
12 RUN useradd -ms /bin/bash devopsuser
13 USER devopsuser
14 WORKDIR /home/devopsuser
```

```
[root@Devops4Beginners ~]# #vi Dockerfile
[root@Devops4Beginners ~]# #docker image build -t myubuntu .
[root@Devops4Beginners ~]# #docker container run -d -it --name myUbuntuUser  myubuntu
```

```
[root@Devops4Beginners ~]# docker container exec -it myUbuntuUser /bin/bash
devopsuser@1179b62a016a:~$ whoami
devopsuser
devopsuser@1179b62a016a:~$ pwd
/home/devopsuser
devopsuser@1179b62a016a:~$
```


DockerFile

Deposer l'image dans Docker Hub :

- Creer un compte dans docker hub
- Créer un repository dans docker hub
  - Exemple : aliali/images
- Accder à docker hub par la commande
  - Docker login (entrer votre login et password)
  - Taguer votre image
    - Exemple : docker tag nginximage:latest aliali/myimages:nginxubuntu
- Deposer l'image dans docker hub
  - docker push aliali/myimages:nginxubuntu

DockerFile

Les couches de l'image après avoir été publiée sur docker hub

IMAGE LAYERS

1	ARG RELEASE	0 B
2	ARG LAUNCHPAD_BUILD_ARCH	0 B
3	LABEL org.opencontainers.image.ref.name=ubuntu	0 B
4	LABEL org.opencontainers.image.version=22.04	0 B
5	ADD file ... in /	29.02 MB
6	CMD ["/bin/bash"]	0 B
7	**/bin/sh -c apt update -y**	**54.14 MB**
8	WORKDIR /var/www/html	0 B
9	COPY file:09bea733e4e8a73da2dfa893d0e567aed5fb1…	284 B
10	EXPOSE 80/tcp	0 B
11	CMD ["/usr/sbin/nginx" "-g" "daemon	0 B

Command

```
/bin/sh -c apt update -y && apt install nginx -y
```

DockerFile

Multi-Stage Builds:

Les constructions en plusieurs étapes auront plus d'une instruction FROM dans le Dockerfile.

Chaque instruction FROM crée une nouvelle construction.

```
# syntax=docker/dockerfile:1
FROM golang:1.21 as build
WORKDIR /src
COPY <<EOF /src/main.go
package main

import "fmt"

func main() {
  fmt.Println("hello, world")
}
EOF
RUN go build -o /bin/hello ./main.go

FROM scratch
COPY --from=build /bin/hello /bin/hello
CMD ["/bin/hello"]
```

- Par défaut, les stages ne sont pas nommées et vous les désignez par leur numéro, commençant par 0 pour la 1er instruction FROM.
- Mais, vous pouvez nommer les stages en ajoutant un AS <NAME> à l'instruction FROM.

```
FROM alpine:latest AS builder
RUN apk --no-cache add build-base

FROM builder AS build1
COPY source1.cpp source.cpp
RUN g++ -o /binary source.cpp

FROM builder AS build2
COPY source2.cpp source.cpp
RUN g++ -o /binary source.cpp
```

```
FROM ubuntu AS base
RUN echo "base"

FROM base AS stage1
RUN echo "stage1"

FROM base AS stage2
RUN echo "stage2"
```

Reference Doc : https://docs.docker.com/develop/develop-images/multistage-build/

Docker Storage : DockerFile

VOLUME:

- permet de monter un répertoire de machine hôte sur le conteneur.
- Lorsque vous exécutez le conteneur basé sur cette image, Docker crée un volume anonyme (volume avec un ID unique comme nom) et le monte sur le chemin spécifié.

```
1  ## Sample Dockerfile
2  FROM nginx
3  LABEL description="Using Volume Instruction"
4  VOLUME ["/usr/share/nginx/html/"]
```

Reference Doc :
https://docs.docker.com/engine/reference/builder/#volume

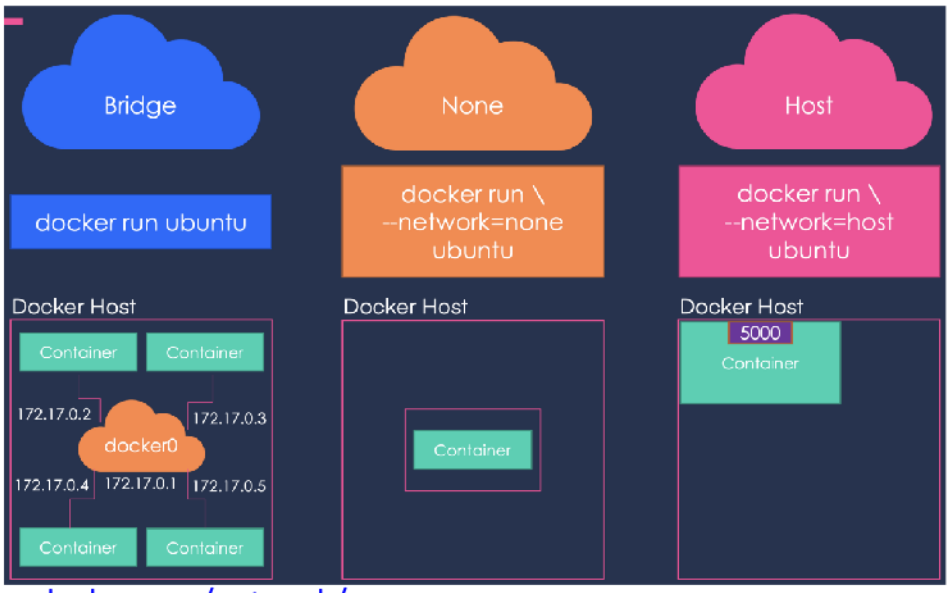
Docker Networking

Docker Networking

Différents type de réseau sur Docker

- Bridge
- Host
- None
- Overlay
- MACVLAN

```
$ docker network ls
NETWORK ID      NAME      DRIVER    SCOPE
5077a7b25ae6    bridge    bridge    local
7e25f334b07f    host      host      local
475e50be0fe0    none      null      local
```



Reference : https://docs.docker.com/network/

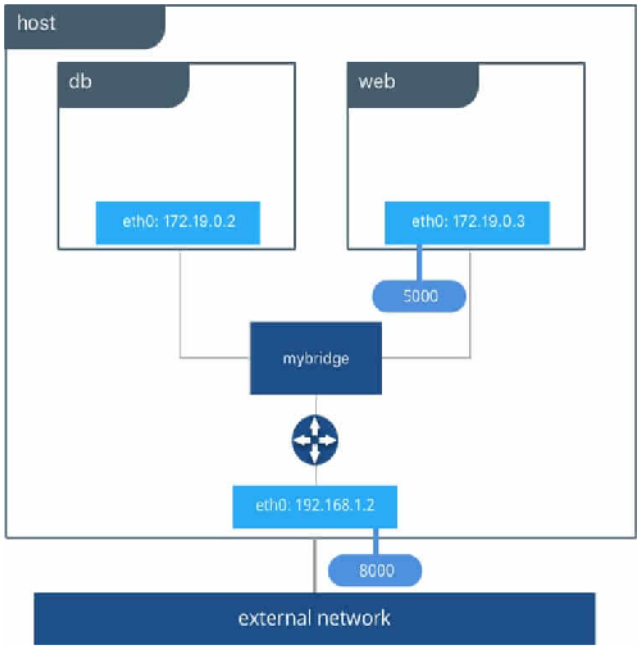
Docker Networking

Driver Bridge

- c'est le plus couramment utilisé. Les conteneurs qui utilisent ce driver, ne peuvent communiquer qu'entre eux, cependant ils ne sont pas accessibles depuis l'extérieur si un mappage de port n'est pas mise en place.

- **Créer un réseau bridge**
- docker network create [**Network Name**]

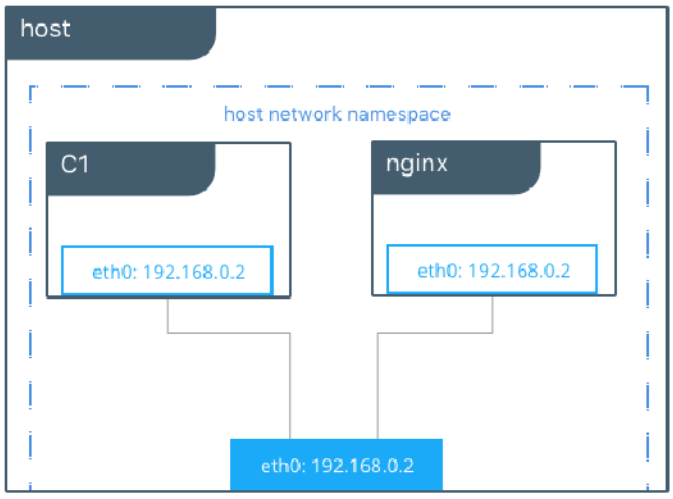
- Pour plus de details :
- docker network inspect bridge



Docker Networking

Driver Host

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

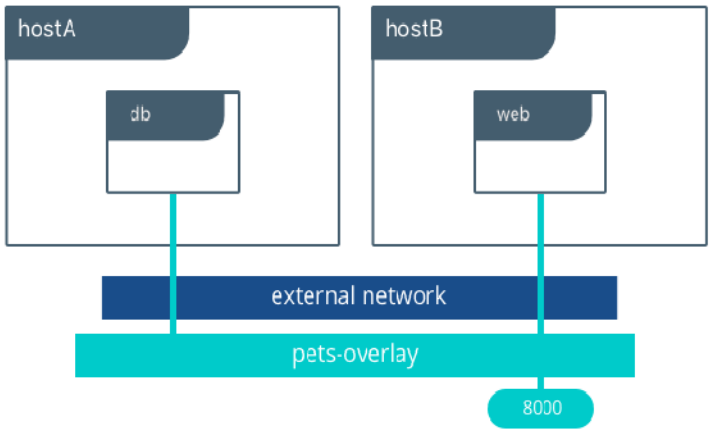


- **Créer le réseau Host**
  - docker container run -d --name mynginx **--network host** nginx

Docker Networking

Driver Overlay

- permet aux conteneurs exécutés sur les même nœuds ou différents nœuds (hôtes multiples) de communiquer entre eux.

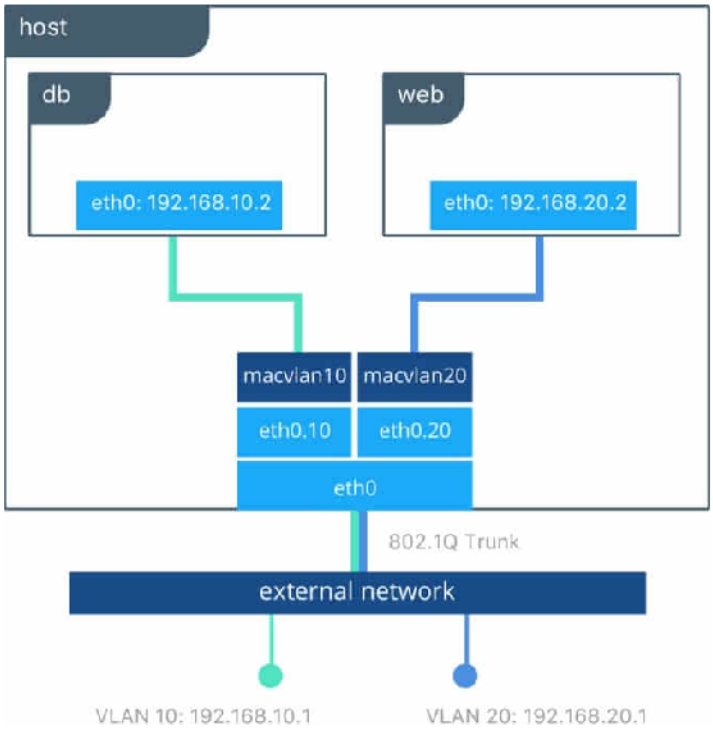


- **Créer le réseau Overlay**
  - docker network create --driver overlay [**Network Name**]

Docker Networking

Driver MacVlan

- permet d'attribuer une adresse MAC à un conteneur et le faisant apparaître comme un périphérique physique sur votre réseau. Le moteur Docker route le trafic vers les conteneurs en fonction de leurs adresses MAC.

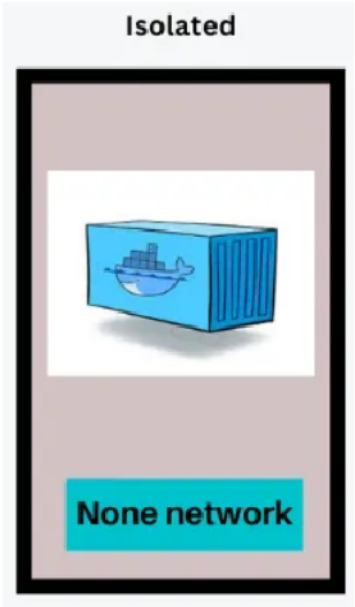


Docker Networking

Driver None

- permet d'interdire toute communication interne et externe avec votre conteneur, car votre conteneur sera dépourvu de toute interface réseau (sauf l'interface loopback).

- **Creer un reseau none**
  - docker container run -d --name mynginxnone **--network none** -p 8080:80 nginx



Docker Networking

Docker Networking Commands:

- **List Networks**
  - docker network ls
- **Create a network**
  - docker network create [**Network Name**]
- **Inspect a network**
  - docker network inspect [**Network Name**]
- **Connect a container to a network**
  - docker network connect [**Network Name**] [**Container Name**]
- **Disconnect a container from a network**
  - docker network disconnect [**Network Name**] [**Container Name**]

Docker Networking

Docker Networking Commands (Contd..):

- **Create a subnet and gateway**
  - docker network create --subnet 10.1.0.0/24 --gateway 10.1.0.1 [**Network Name**]
- **Assign a specific IP to a container**
  - docker container run -d --name [**Container Name**] \
    --ip [**IP Address**] \
    --network [**Network Name**] \
    nginx
- **Remove a network**
  - docker network rm [**Network Name**]
- **Remove unused networks**
  - docker network prune