



Docker

Mise en oeuvre et déploiement de conteneurs virtuels

Here is where your formation begins

Le Formateur

Guilian GANSTER

Développeur web & mobile

Freelance depuis 4 ans dans les technologies du web & mobile.

Formateur depuis 2 ans.

Développeur fullstack React /
React native
Firebase / Strapi

Couteau suisse de l'informatique:
Devops, sysadmin, électronique,
client lourd, ...



Participants

01 Quelles sont vos attentes par rapport à cette formation ?

02 Les sujets que vous voudriez aborder ?

03 Dans quel cadre allez vous appliquer ces compétences ?



Déroulé de Formation



Horaires & temps de pause

- de 9h à 12h30 et de 13h30 à 17h30
- Une pause de 20mn le matin, et une pause de 20mn l'après midi.
- Mercredi apres midi, examen de validation.
- Fin de la formation mercredi 16h.

- A chaque demi journée, un questionnaire à remplir pour récolter vos retours (équilibre théorie / pratique, notions sur lesquelles revenir, ...)



01

Contexte



compatibilité



PC
Développement
(windows)



PC QA
(mac os)

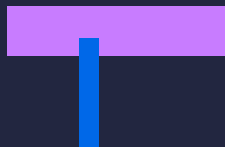


Serveur préprod
Ubuntu S 22.04

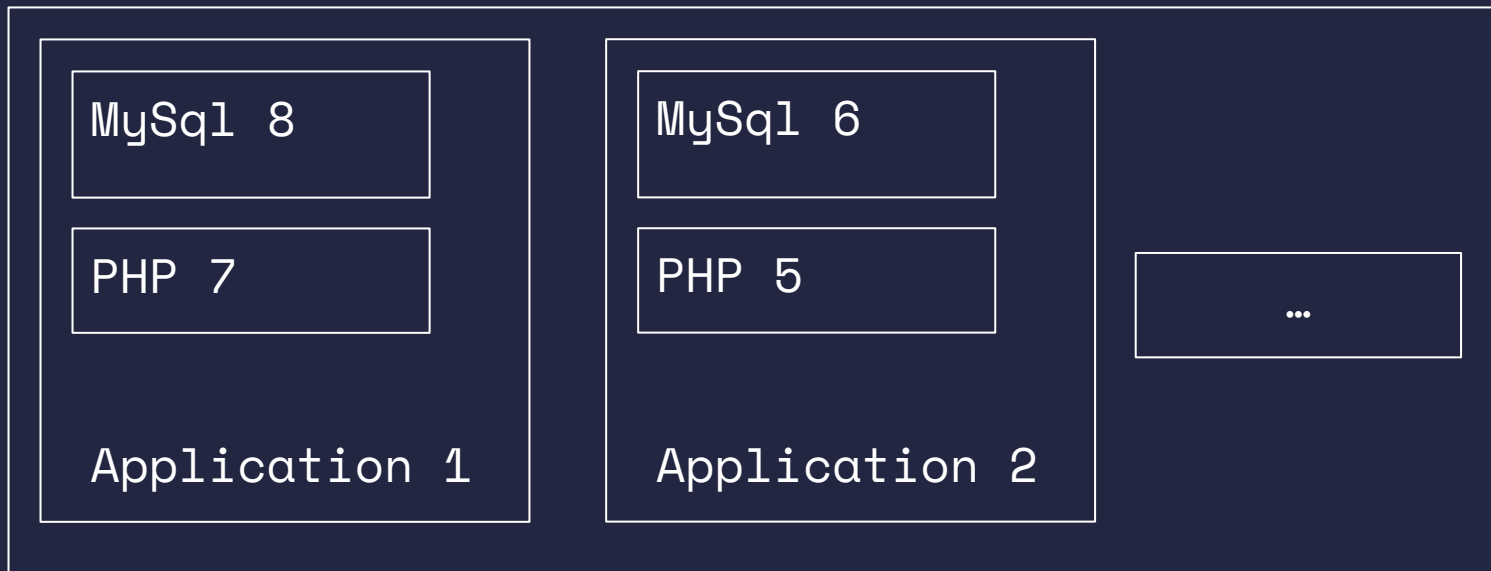


Serveur
production
Ubuntu S 20.04



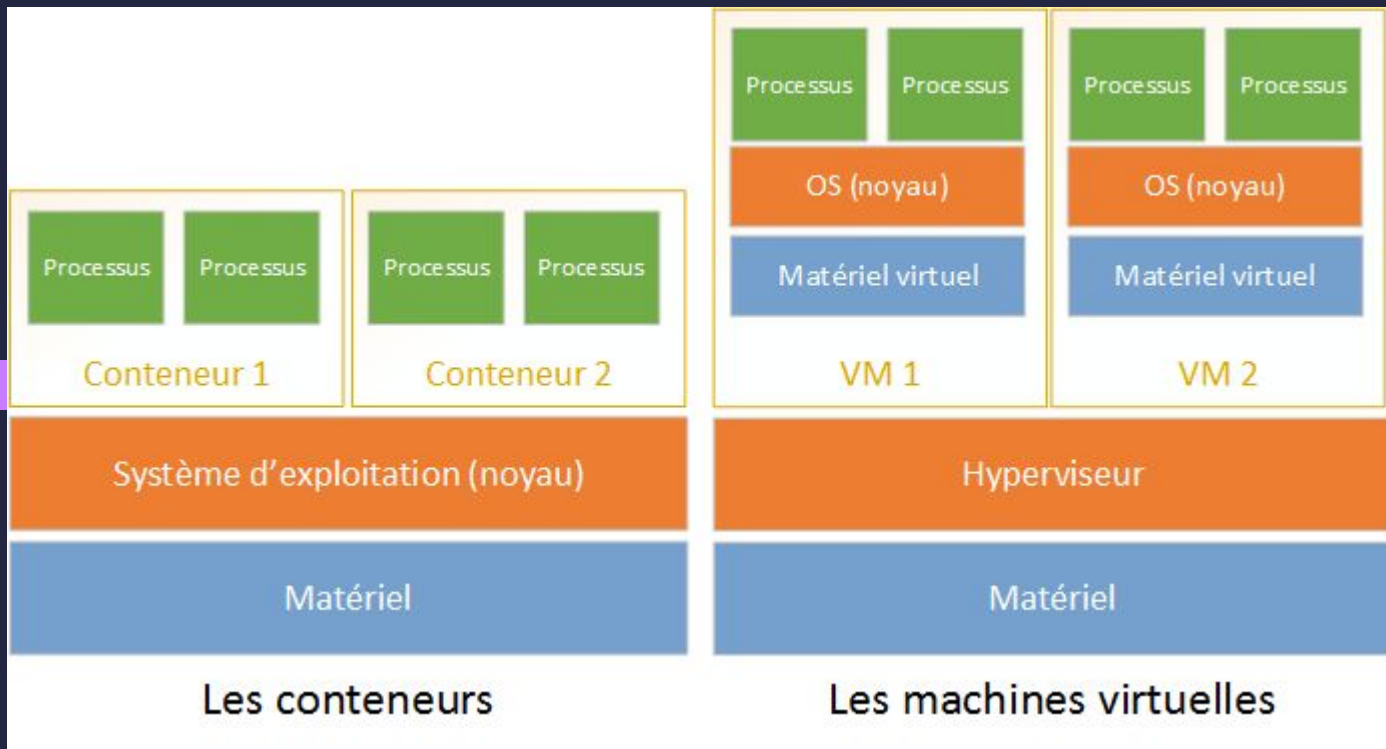


Isolation

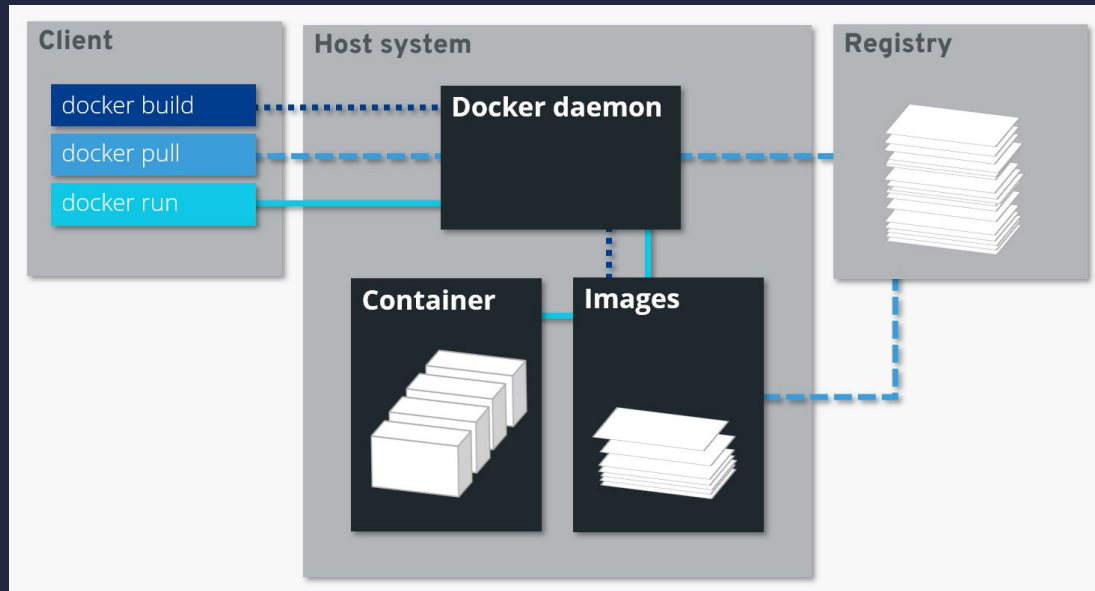




Performances



Simplicité d'utilisation





02

Installation



01

docker run <...> <image>

Crée un conteneur à partir d'une image

Astuces:

-d = mode détaché (daemon)

-name <name> = donne un nom à son container, plus facile à manipuler que les ids



02

docker ps

Liste les conteneurs démarrés

03

docker stop <id>

Stoppe un conteneur

04

docker start <id>

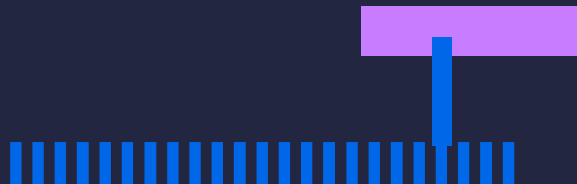
Démarre un conteneur existant

Astuce: -d implicite

05

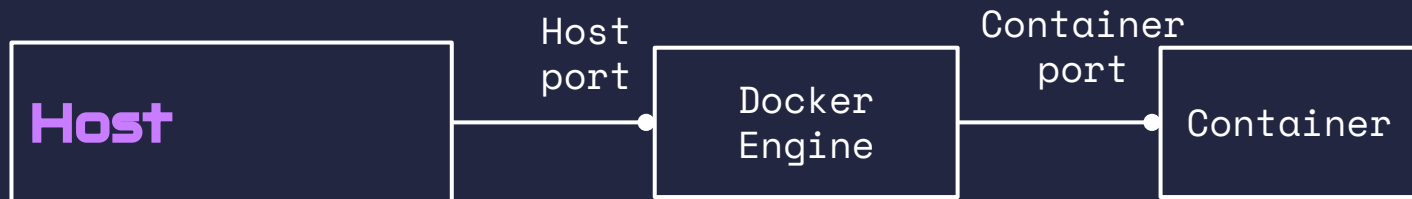
docker rm <id>

Supprime un conteneur



Le port forwarding

```
docker run -p <host port>:<container port> <image>
```



La persistance par les volumes

01 `docker volume create <name>`

Crée un volume qui persistera même si le conteneur est coupé

02 `docker volume ls`

Liste les volumes disponibles

03 `Docker volume rm <name>`

Supprime un volume

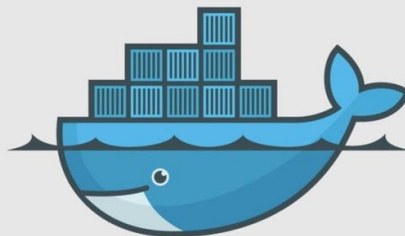
04 `Docker volume inspect`

Donne des informations détaillées sur un volume

`docker run -v <name>:<container path> <image>`

Le registry

Docker Hub



docker



TP



Installer
lu
fi

Avec persistance de données
Accessible par le port 80

Pour les rapides:

Redémarrage automatiquement au boot /
en cas de plantage



Executer une commande dans un container

01

`docker exec -it <container_id> /bin/bash`
`<optional_params>`

Connecte le terminal au container, utilisé couramment pour lancer un shell. Ou un script

Ctrl+d ou "exit" pour quitter

Logs & inspect

01 Docker info

Affiche les infos sur docker. Version, drivers disponibles, plugins installés, ...

02 Docker inspect <container>

Affiche les infos sur un conteneur

02 Docker logs <container>

Visualise les logs du processus lancé dans le container

Astuce:

-f (attache le terminal, visualisation en continu)

L'environnement

01

`docker run -e key=value -e key2=value2 <...>`

Ajoute des entrées à l'environnement d'exécution du container. (cf documentation d'image)

02

`docker run --env-file <path> <...>`

Ajoute à l'environnement toutes les clés valeurs du fichier d'env passé en paramètre.

Ressources limits

`docker run --memory-reservation <value> <...>`

Spécifie la soft limit de mémoire, cette limite peut être dépassé mais lève un warning. Utilisé pour le monitoring

`docker run --memory value <...>`

Spécifie la hard limit de mémoire, le conteneur ne peut pas le dépasser, docker préférera le faire planter que de dépasser cette limite. (OOM error)

Les valeurs sont noté sous le format suivant:

1k (=1ko)

1m (=1mo)

1g (=1go) ...

Fonctionne pareil avec cpu



Le linkage

01


`docker run --link <container_name> <...>`

Connecte le terminal au container, de ce fait on peut exécuter des scripts à la main dans la vm



TP

Démarrer un conteneur mysql
Démarrer un conteneur wordpress lié
Accessible sur port 80

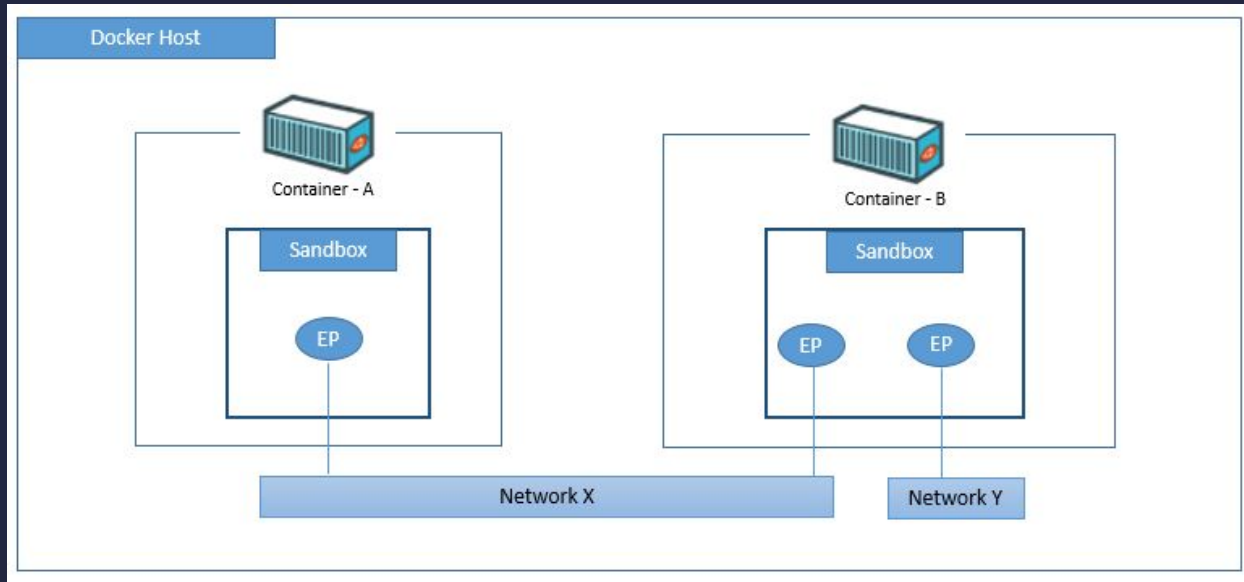


Attention aux volumes de persistences
Mysql ne doit pas être exposé



//////////

Les networks



Les networks

01

Docker network create <name>

Crée un network (même principe que volume)
Driver par défaut: "bridge"

02

Docker run --network <name> <...>

Attache le conteneur au network

Les drivers network

01

bridge

Le driver network par défaut. Utilisé lorsqu'un container doit continuer avec d'autres.

02

host

Supprimer l'isolation de réseau entre l'hôte et le container.

03

overlay

Permet la connection de plusieurs docker engine lors de la mise en place de cluster.

Les volumes drivers

01 local

Driver par défaut, le volume est stocké localement sans modification (bind ou géré)

02 nfs3

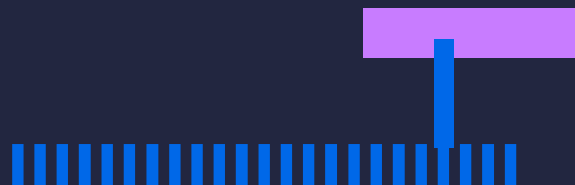
Un volume stocké dans le cloud, appartient à amazon S3.

03 nvieux/sshfs

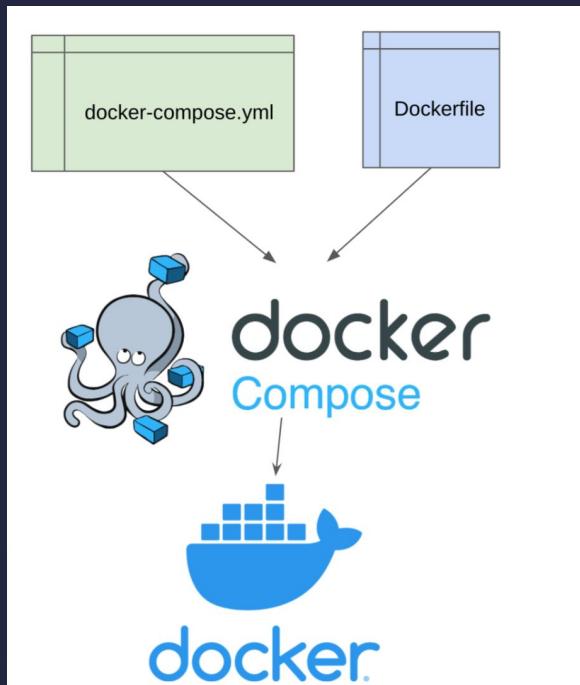
Un volume sur un serveur tiers à travers SSH

04 tmpfs

Un volume stocké dans la RAM.



docker-compose





TP

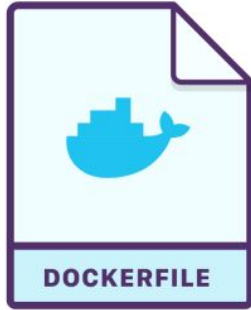


Docker-compose wordpress

• • • • •



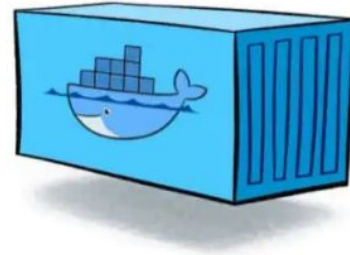
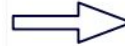
Dockerfile



Docker file



Docker Image



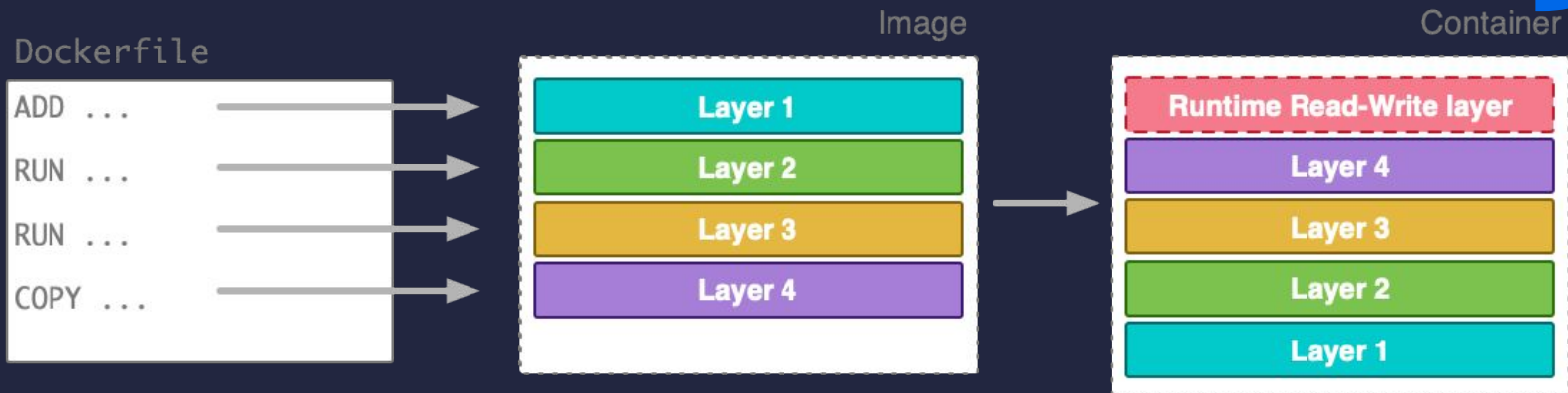
Docker Container

Dockerfile

Builder main commands

command	description
<code>FROM image scratch</code>	base image for the build
<code>MAINTAINER email</code>	name of the maintainer (metadata)
<code>COPY path dst</code>	copy <i>path</i> from the context into the container at location <i>dst</i>
<code>ADD src dst</code>	same as <code>COPY</code> but untar archives and accepts http urls
<code>RUN args...</code>	run an arbitrary command inside the container
<code>USER name</code>	set the default username
<code>WORKDIR path</code>	set the default working directory
<code>CMD args...</code>	set the default command
<code>ENV name value</code>	set an environment variable

Dockerfile - layers



Build des images

01

Docker image build <...> .

Build l'image a partir du Dockerfile

Options importantes:

- no-cache → spécifie que toutes les étapes intermédiaires doivent être rejouées
- t "name" → spécifie le nom de l'image



TP

Créer un DockerFile FROM nginx:latest
Permettant le hosting d'un site
statique



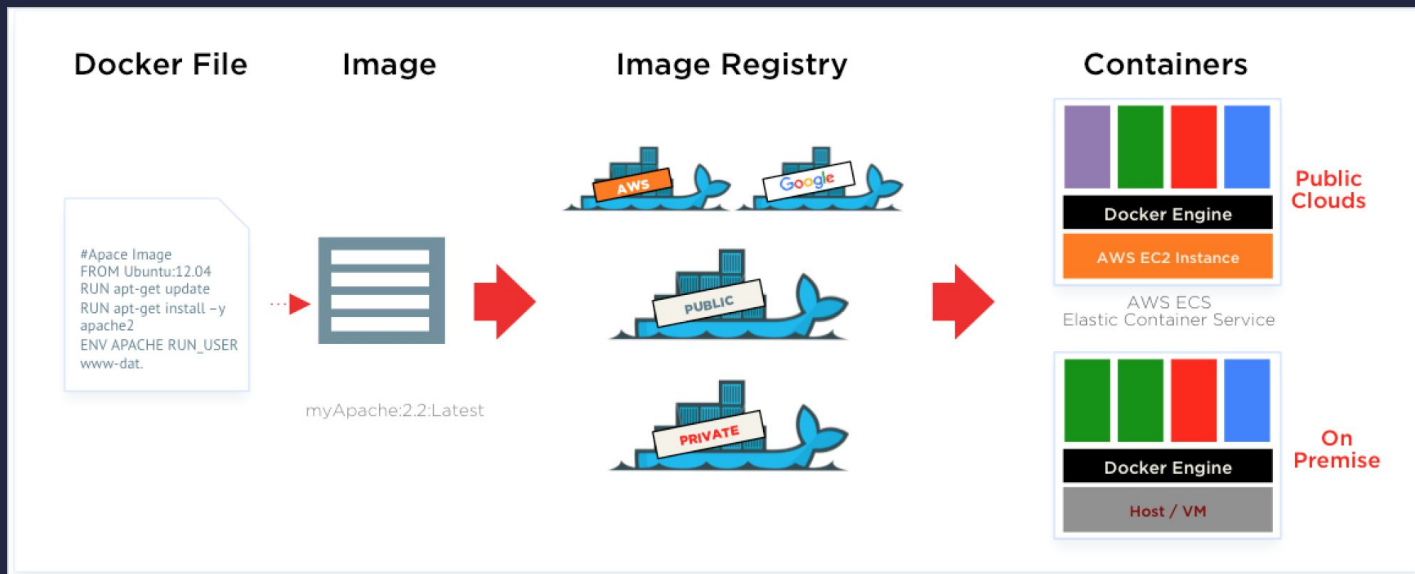
Exécuter votre nouvelle image à
l'aide de docker run ou docker
compose



Créer une image à partir de l'état d'un conteneur

01 Docker commit

Envoi vers le registre

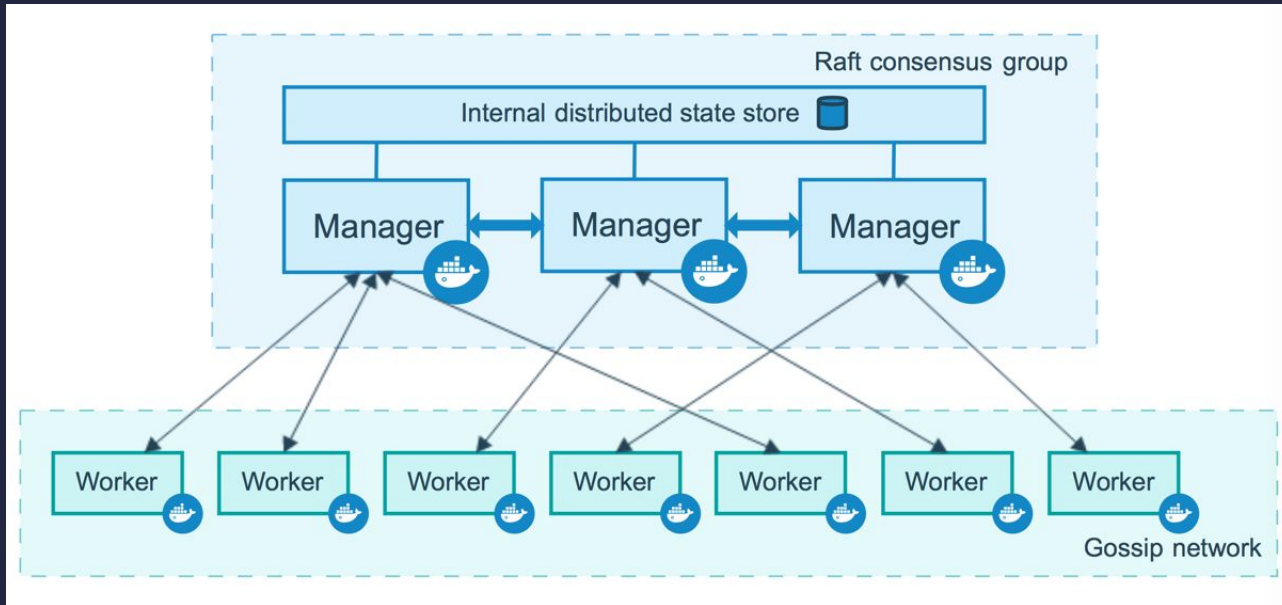


Envoi des images sur le registre

- 01 `Docker image build -t pseudo/repo:version .`
- 02 `Docker push pseudo/repo:version`
- 02 `docker image tag localImage pseudo/repo:version`

Retag une image précédemment créé

Introduction Swarm



Gestion des nodes

01

Docker swarm init

Initialise le swarm, a faire qu'une seule fois.

02

Docker swarm join-token <worker|manager>

Crée un token qui permet à un autre node de rejoindre le swarm

02

Docker node ls

Liste les nodes présent dans le réseau

Gestion des services

01

Docker service ls

Affiche les services

02

docker service create <..>

Equivalent à un docker run, mais pour swarm. Docker service rm/logs/inspect fonctionnent de la même manière.

03

Docker service scale SERVICE=N

Met à jour le nombre d'instance d'un service dans le swarm.

Gestion des stacks

01

Docker stack ls

Affiche les stacks

02

docker service deploy -compose-file <path>

Equivalent docker-compose. Mais ne fait pas les build d'image

03

Docker service scale SERVICE=N

Met à jour le nombre d'instance d'un service dans le swarm.

healthcheck



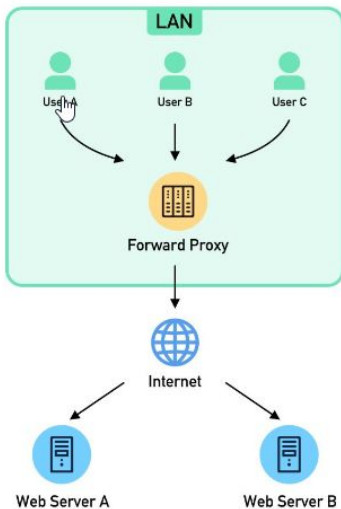
+ Health Check

Docker Compose

Reverse-proxy avec traefik

Forward Proxy

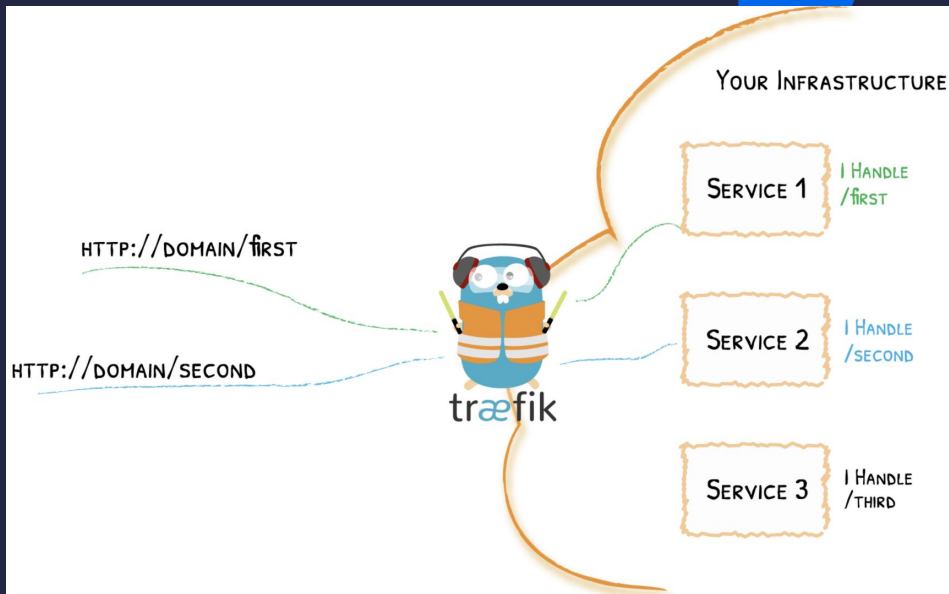
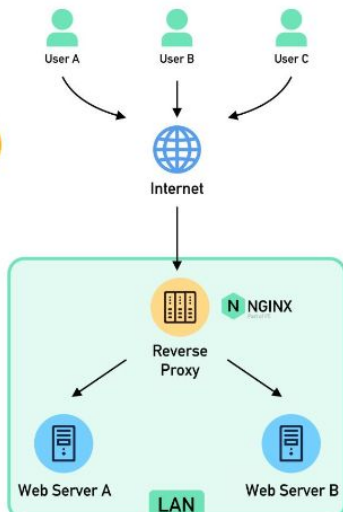
- Avoid browsing restrictions
- Block access to certain content
- Protect user identity online



VS

Reverse Proxy

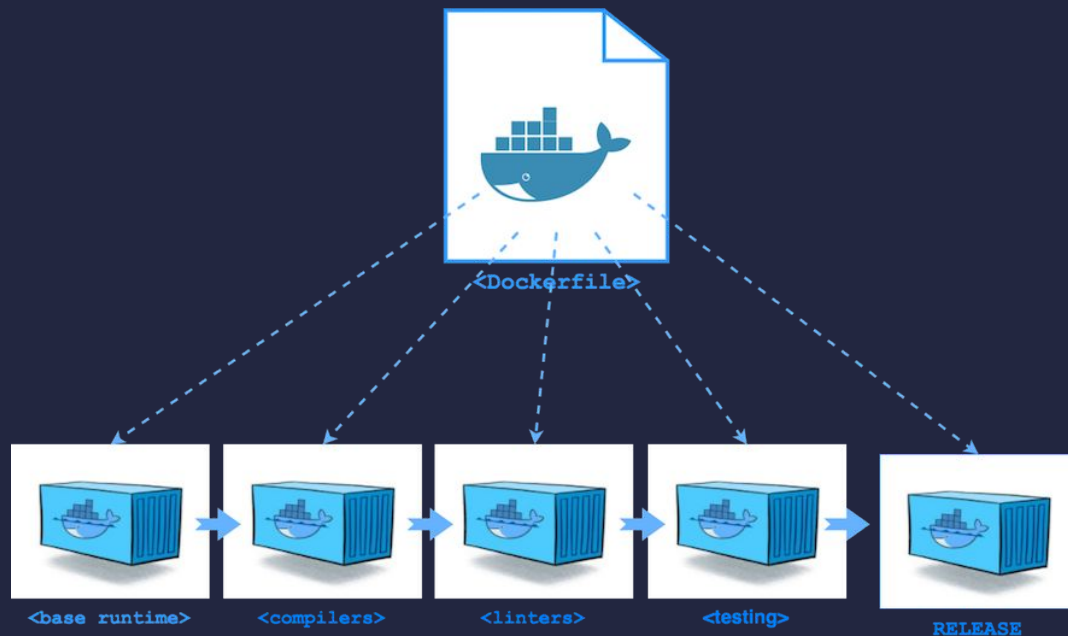
- Load balancing
- Protect from DDos attacks
- Cache static content
- Encrypt and decrypt SSL communications



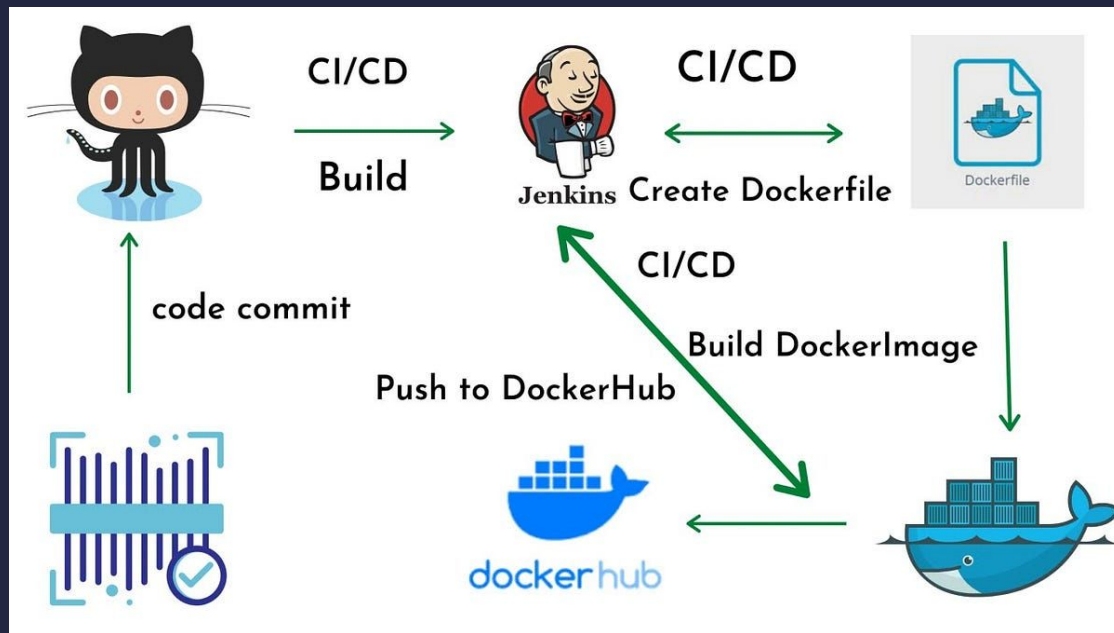
Monitoring



Dockerfile avec build intermédiaire



Intégration dans un pipeline de CI



Bonne pratiques sécurité

