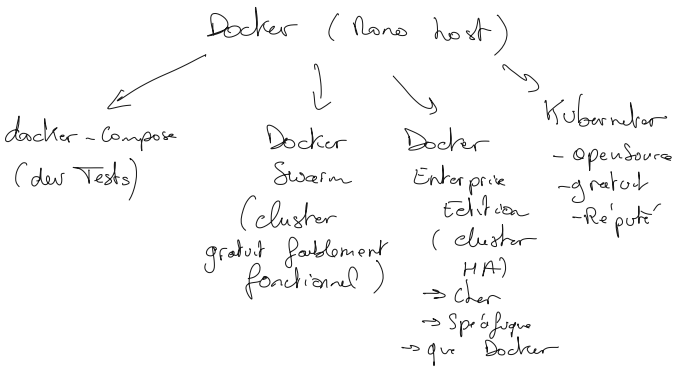
Copier de guacamole -> votre host
↳ direct

-> Copiez du texte depuis votre Terminal vers votre host / notepad

---



Utiliser docker directement sans être root :
sudo usermod -aG docker student

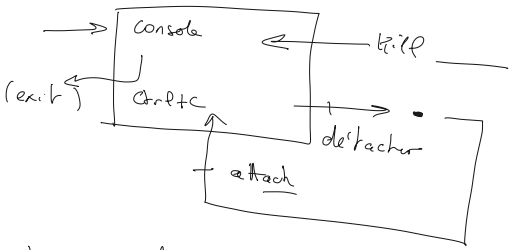
docker container run -it ubuntu -> ajoutez le paquet iputils-ping pour faire un ping.

Charges de travail type en Prod :

- Serveurs de base de données :
  - redis
  - mongodb
  - sqlserver
  - postgres
  - ...
- Serveurs web
  - nginx
  - apache
  - tomcat
  - jboss wildfly
  - ...
- Front/back web
  - .Net core/6 asp.net mvc
  - node.js
  - php fpm symfony
  - java springboot
  - ...
- ELT / ETL
  - Nifi
  - jobs Talend
  - ...
- BUS
  - Kafka
- Datascience
  - conteneur spark
  - ...

Cycle de vie de la console

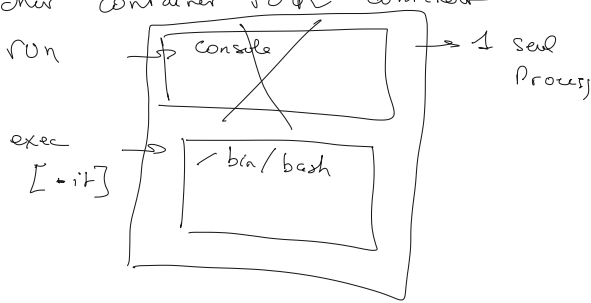
docker container run -it



docker container | attach
exec → "un autre canal"

---

docker container run Container

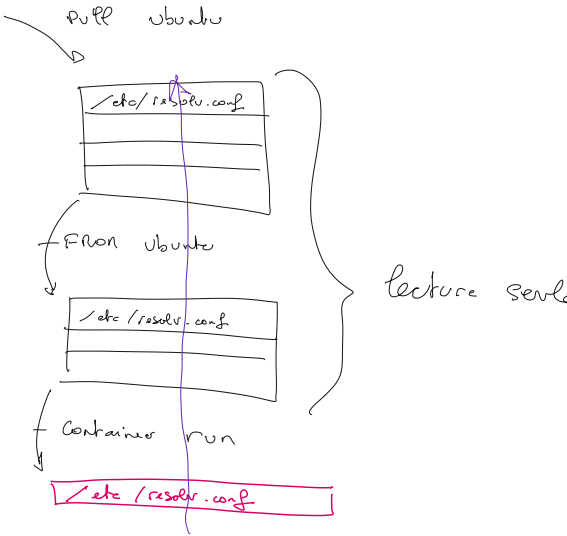


→ Lancer un serveur | Redis
Ubuntu

① - Vous détacher (Ctrl+P, Ctrl+Q)
- vous rattacher

② Lancer un autre shell côte à côte
(container exec -it <conteneur>)

Créez un compte sur hub.docker.com
Poussez votre image
$ docker image tag myubuntu nboost/myubuntu
$ docker login
$ docker image push nboost/myubuntu



student@u20:~/myubuntu$ cat Dockerfile
FROM ubuntu

 # recommandé sauf si vous voulez bénéficier du cache
RUN apt update && apt -y upgrade && apt install -y procps net-tools
#RUN apt -y upgrade
#RUN apt install procps
#RUN apt install net-tools
student@u20:~/myubuntu$ cat *.sh
docker image build -t myubuntu .
docker container run -it --rm myubuntu
student@u20:~/myubuntu$ cat 00-build.sh
docker image build -t myubuntu .
student@u20:~/myubuntu$ cat 01-test.sh
docker container run -it --rm myubuntu

$ cat compress.sh
cd files
tar czf ../files.tar.gz *
cd ..
tar tzf files.tar.gz

Exercice :
Vous allez conteneuriser un site web sous nginx

vos fichiers html (au moins 2) sont dans un dossier
Vous allez vous baser sur l'image officielle nginx qui écoute sur le port 80 et stocke les fichiers de base dans un certain dossier.
vous injectez vos fichiers html à la place de ceux de l'image de base.
et voilà.

-> mysite
docker container run -it -p 80:80 mysite

Serveur web texte :
apt install lynx
lynx http://localhost

La démarche :

1. Lancez un conteneur nginx pour découvrir ou se trouve le dossier html
2. créez un dossier pour votre projet
3. créez y un dossier dans lequel stocker les fichiers html à injecter dans votre future image
4. créez un Dockerfile avec le contenu nécessaire
5. fabriquez et lancez un conteneur de test (spécifiez le paramètre " **-p 80:80** " !!!)
6. connectez vous avec lynx pour vérifiez que votre nginx serve bien votre contenu :
   lynx http://localhost

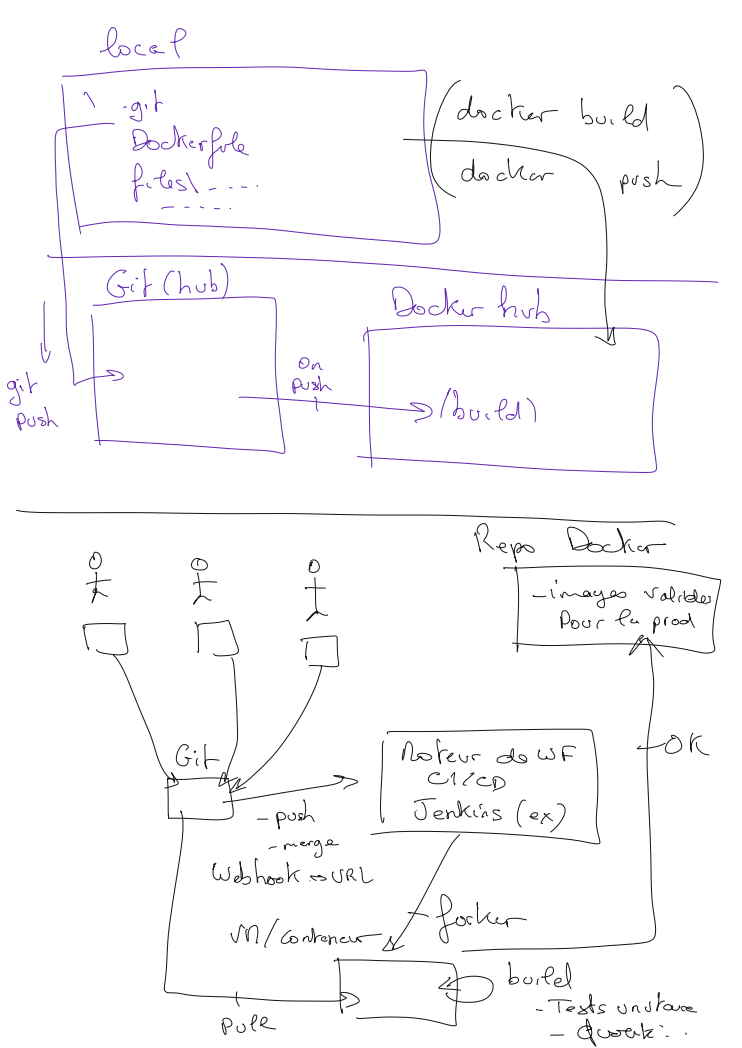
Solution : Rechercher le dossier www de nginx :

1. docker container exec -it n1 find / -name index.html
2. docker container run -it --rm --name n1 nginx find / -name index.html

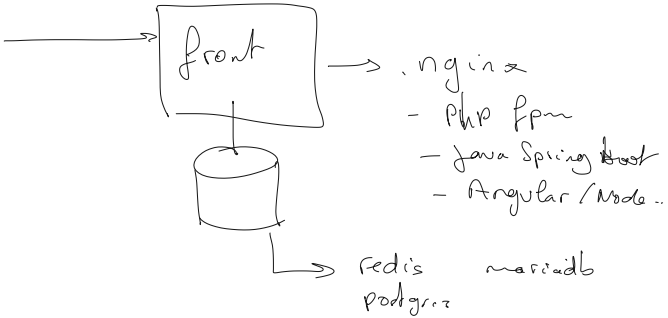
Distinction CMD / ENTRYPOINT :

student@u20:~/cmd$ cat Dockerfile
FROM busybox

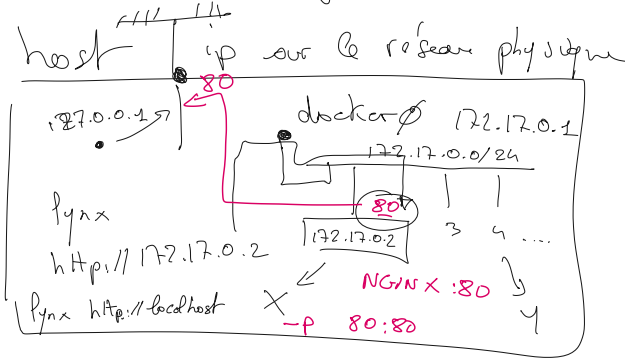
EXERCICE : Réaliser une image qui affiche Bonjour suivi d'un prénom par défaut (Tout le monde), mais qui soit remplaçable au
lancement du conteneur
ex : Bonjour tout le monde
Bonjour machin
CMD ["Tout le monde"]
ENTRYPOINT ["echo","Bonjour"]
student@u20:~/cmd$ cat 00-build.sh
docker image build -t s .
student@u20:~/cmd$ cat 01-test.sh
docker container run -it --rm s **vous !**
student@u20:~/cmd$ docker container run -it --rm **--entrypoint=/bin/sh** s
/ # exit
student@u20:~/cmd$

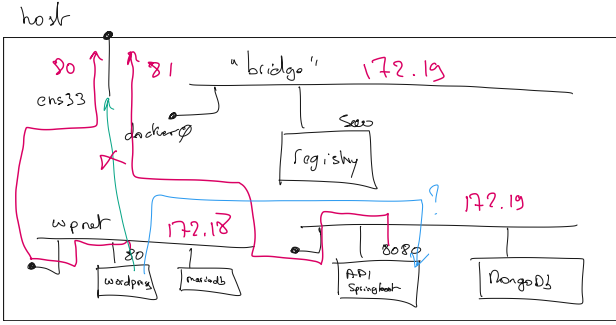

local
.git
Dockerfile
files\ ----
docker build
docker push
Git (hub)
Docker hub
git push
On push
/build
Repo Docker
- images validées Pour la prod
Git
- push
- merge
Webhook → URL
Moteur de WF CI/CD Jenkins (ex)
OK
VM/container
docker
build
- Tests unitaires
- Qwerk...
Pull

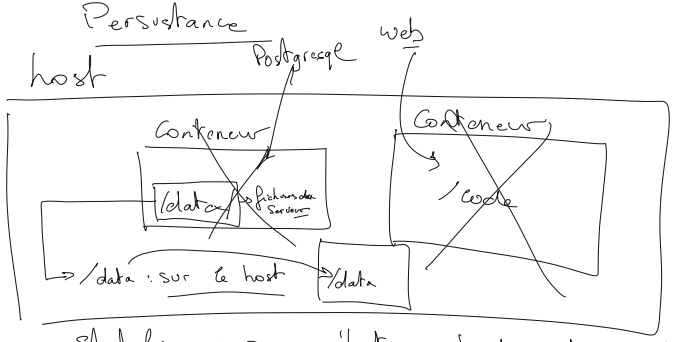
```
student@u20:~$ docker image pull mcr.microsoft.com/mssql/server
Using default tag: latest
latest: Pulling from mssql/server
ea362f368469: Pull complete
dc034f624aa1: Pull complete
cafda714f10f: Pull complete
c6af4ce68233: Pull complete
2e5e63d166b4: Pull complete
Digest: sha256:fb5277e7a3cc53f7d2230ed089ed60849f79567ebb0aae8f41ceb85879e9e09d
Status: Downloaded newer image for mcr.microsoft.com/mssql/server:latest
mcr.microsoft.com/mssql/server:latest
student@u20:~$ docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
myubuntu                      latest    3e01df2b4875   3 minutes ago   72.8MB
ubuntu                        latest    825d55fb6340   9 days ago      72.8MB
registry                      latest    2e200967d166   9 days ago      24.2MB
mcr.microsoft.com/mssql/server   latest    d78e982c2f2b   2 months ago    1.48GB
student@u20:~$ hostname
u20.stage.lan
student@u20:~$ docker image tag myubuntu u20.stage.lan/library/myubuntu
student@u20:~$ docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
u20.stage.lan/library/myubuntu   latest    3e01df2b4875   4 minutes ago   72.8MB
myubuntu                      latest    3e01df2b4875   4 minutes ago   72.8MB
ubuntu                        latest    825d55fb6340   9 days ago      72.8MB
registry                      latest    2e200967d166   9 days ago      24.2MB
mcr.microsoft.com/mssql/server   latest    d78e982c2f2b   2 months ago    1.48GB
student@u20:~$ docker image push u20.stage.lan/library/myubuntu
Using default tag: latest
The push refers to repository [u20.stage.lan/library/myubuntu]
Get "http://u20.stage.lan/v2/": dial tcp 127.0.1.1:80: connect: connection refused
student@u20:~$ docker image tag myubuntu u20.stage.lan:5000/library/myubuntu
student@u20:~$ docker image ls
REPOSITORY                         TAG       IMAGE ID       CREATED         SIZE
u20.stage.lan/library/myubuntu     latest    3e01df2b4875   5 minutes ago   72.8MB
u20.stage.lan:5000/library/myubuntu   latest    3e01df2b4875   5 minutes ago   72.8MB
myubuntu                           latest    3e01df2b4875   5 minutes ago   72.8MB
ubuntu                             latest    825d55fb6340   9 days ago      72.8MB
registry                           latest    2e200967d166   9 days ago      24.2MB
mcr.microsoft.com/mssql/server     latest    d78e982c2f2b   2 months ago    1.48GB
student@u20:~$ docker image push u20.stage.lan:5000/library/myubuntu
Using default tag: latest
The push refers to repository [u20.stage.lan:5000/library/myubuntu]
ce64cbb45822: Pushed
c5ec52c98b31: Pushed
latest: digest:
sha256:ecc84d929635e4598b73afdab1f9ae083712ad9a7712760d3a42937b70483ef7 size:
736
student@u20:~$ docker container stop registry
registry
student@u20:~$ docker image push u20.stage.lan:5000/library/myubuntu
Using default tag: latest
The push refers to repository [u20.stage.lan:5000/library/myubuntu]
Get "http://u20.stage.lan:5000/v2/": dial tcp 127.0.1.1:5000: connect: connection refused
student@u20:~$ docker container start registry
registry
student@u20:~$ docker image push u20.stage.lan:5000/library/myubuntu
Using default tag: latest
The push refers to repository [u20.stage.lan:5000/library/myubuntu]
ce64cbb45822: Layer already exists
c5ec52c98b31: Layer already exists
latest: digest:
sha256:ecc84d929635e4598b73afdab1f9ae083712ad9a7712760d3a42937b70483ef7 size:
736
student@u20:~$
```



Réseau par défaut

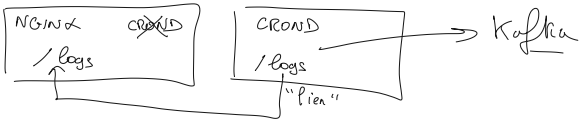




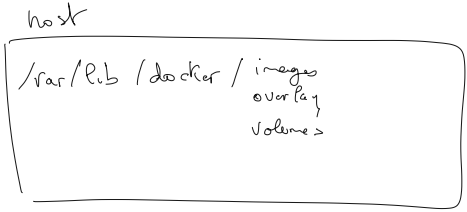


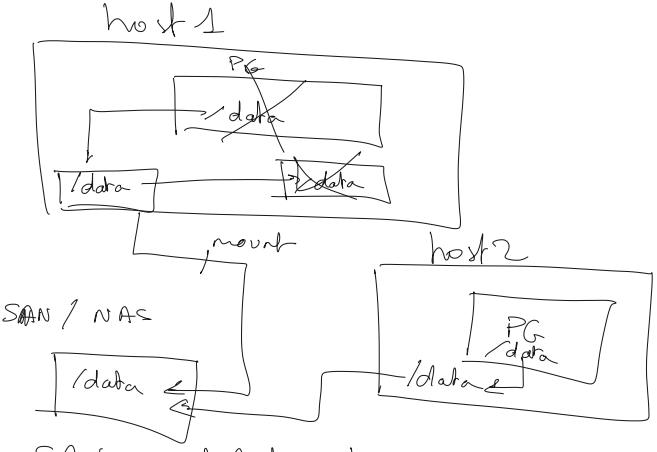
Stateless → sans état = perte du conteneur OK

Stateful → avec état = modification de fichiers =
besoin de ne pas perdre le contenu
exemple = Base de données



docker container run -d --name pg -e POSGRESQL_PASSWORD=password -v var/lib/postgresql/data postgres





SAN → block → baies de disque — iSCSI
NAS → fileserver → SMB ou NFS

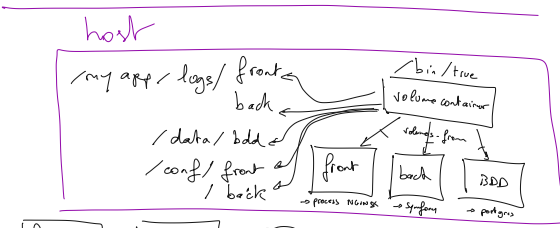
docker container run -it --rm --name pg2 -e POSTGRES_PASSWORD=password -v /myvolumes/pg:/var/lib/postgresql/data postgres

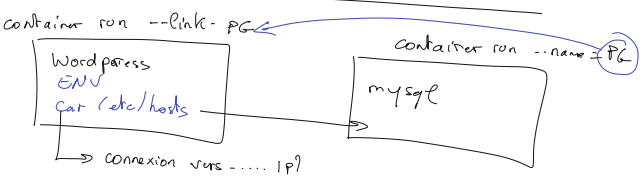
Exercice :

1. Créez un conteneur ubuntu et créez un fichier /data/ok avec un contenu
2. Ce fichier ne doit pas être perdu quand vous supprimez le conteneur
3. Relancez un autre conteneur qui accède à nouveau à ce fichier.

docker container run -it **--name nginx** --rm -v myvol ubuntu
docker container run -it --rm **--volumes-from=nginx** ubuntu

apparition dans (bien)





Cas n°1 (legacy) → network bridge et --link

Cas n°2 → via network create et DNS gérer par docker

Object f= découverte de service (fqdn= ip)

https://docs.docker.com/samples/django/

dev

1 Conteneur → docker

n Conteneurs → docker-compose

→ Tests d'intégration

Prod

→ Mono host
→ docker
→ docker-compose
→ Multi-host (cluster)
→ Docker Swarm
→ Kubernetes (↗ Openshift)

YAML

YAML

Tutoriel Azure Kubernetes Service :
https://docs.microsoft.com/fr-fr/azure/aks/tutorial-kubernetes-prepare-app

Site de tutos :
https://www.server-world.info/en/