Docker

Introduction

Objectifs

- ✓ Établir en quoi Docker pour vous aider en tant que développeur
- ✓ Définir la différence entre virtualisation et conteneurisation.
- ✓ Lister des cas d'utilisation de Docker.
- ✓ Installer Docker sur mon ordinateur.

Définition

Docker est une plateforme logicielle permettant de créer, déployer et exécuter des applications dans des conteneurs.

Les conteneurs encapsulent le code et les dépendances d'une application, garantissant une exécution cohérente dans différents environnements.

Docker utilise des images comme modèles pour créer des conteneurs et le Docker Engine pour les gérer.

Quelques cas d'utilisation

Environnements de développement cohérents

Les développeurs peuvent créer des environnements de développement standardisés pour tous les membres de l'équipe, évitant les problèmes de "ça fonctionne sur ma machine".

La définition du conteneur étant textuelle elle peut être intégrée dans un outil de gestion de version tel que Git

L'isolation des conteneurs évite la prolifération des versions et des logiciels sur les ordinateurs des développeurs. On passe plus facilement d'un projet à l'autre et aucun risque qu'un projet ne pollue la configuration d'un autre projet

Importance de l'isolation

Projet A	Projet B	Projet C	Projet D
PHP 8.1	PHP 8.3	Python	Java
Nginx	Nginx	Apache	Apache Tomcat
Mariadb	MySQL	PostgreSQL	Oracle

Conséquences du manque d'isolation ?

Questions

Combien de temps mettez-vous à installer un nouvel environnement de production quand vous arrivez sur un projet existant ?

Que ce passe-t-il en cas de changement dans la stack technique (configuration, nouvelle version, ajout d'un outil) ?

On multiplie ce temps par le nombre de développeur intervenant sur le projet.

Tests automatisés

Les tests peuvent être exécutés dans des environnements isolés et reproductibles, garantissant que les résultats des tests sont cohérents.

Déploiements cohérents

Les mêmes images Docker peuvent être utilisées tout au long du pipeline de déploiement, garantissant que le code testé en développement est le même que celui déployé en production.

Microservices

Isolation des services : Chaque microservice peut être exécuté dans son propre conteneur, avec ses propres dépendances et configurations, facilitant la gestion et le déploiement de microservices.

Scalabilité : Les conteneurs peuvent être facilement mis à l'échelle pour répondre à la demande, en démarrant ou arrêtant des instances de conteneurs en fonction des besoins.

Applications multiplateformes

Les applications conteneurisées peuvent être exécutées de manière cohérente sur différentes plateformes, y compris les environnements de développement, de test et de production, ainsi que sur différents fournisseurs de cloud.

Isolation des applications legacy

Les applications anciennes peuvent être conteneurisées pour être isolées des nouvelles applications et des systèmes d'exploitation sous-jacents, facilitant leur gestion et leur migration vers de nouvelles infrastructures.

En résumé

- ✓ Isolation
- ✓ Fiabilité
- ✓ Déploiement
- ✓ Gain de temps
- ✓ Journalisation de la stack technique

Installation

Windows

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

Attention : vous devez avoir les droits administrateur sur votre ordinateur pour installer Docker

Pour l'exécuter en revanche il suffit que vous soyez membre du groupe docker-users

Mac

https://docs.docker.com/desktop/install/mac-install/

Linux

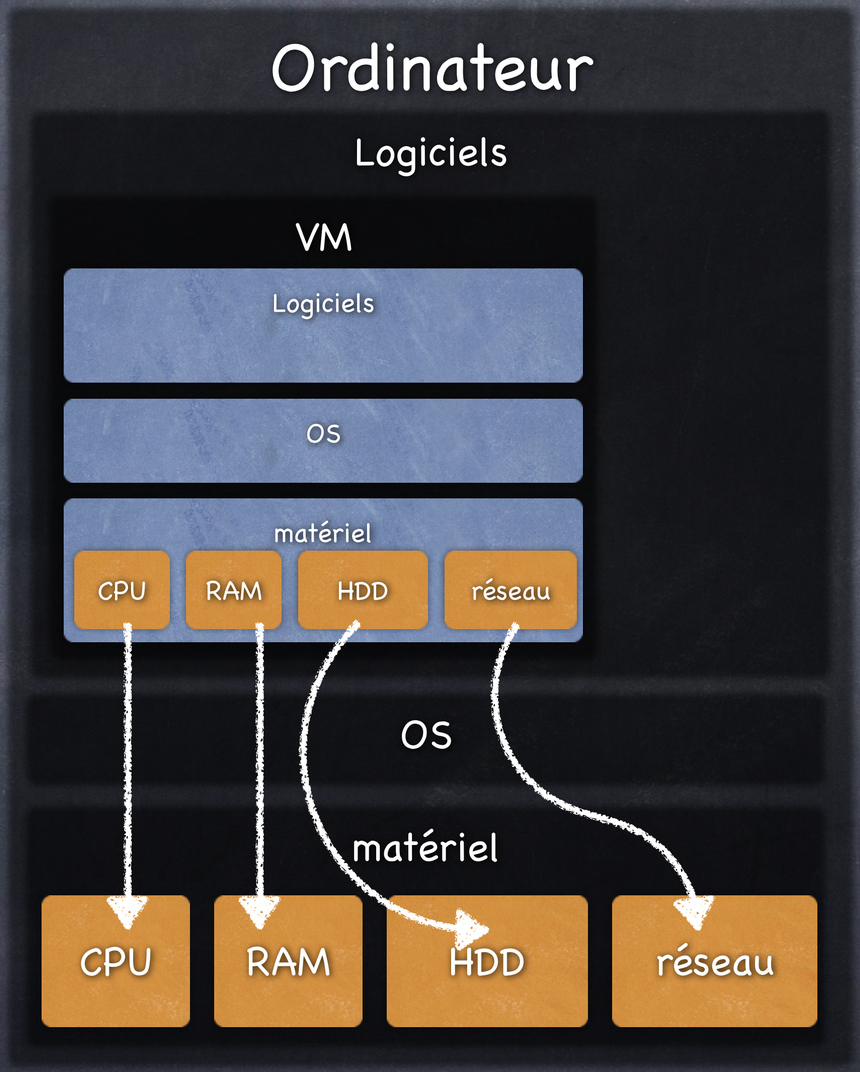
https://docs.docker.com/desktop/install/linux-install/

Conteneur vs virtualisation

La virtualisation

Machine physique

Virtualisation



La conteneurisation

Le conteneur s'appuie sur le système d'exploitation de la machine hôte

Le conteneur contient tous les éléments pour exécuter l'application

Conclusison

- ✓ La virtualisation offre une meilleure isolation
- ✓ La virtualisation est généralement plus lente et plus couteuse en ressources

Docker sur Windows

Docker s'appuie sur un hôte Linux, dans le cas d'une installation sous Windows, cet hôte sera virtualisé (WSL ou Hyper V).

Avec un seul conteneur la performance peut alors être inférieure à celle de la simple virtualisation.

L'hôte virtualisé étant partagé, Docker reprend l'avantage quand on exécute plusieurs conteneurs en même temps

Quelques principes d'utilisation

- ✓ Un conteneur est atomique, un conteneur = un service
  - ✓ une stack complète implique plusieurs conteneur
- ✓ Un conteneur est jetable
  - ✓ on doit pouvoir recréer le conteneur rapidement et sans perte de données

Conclusion

Dans votre pratique professionnelle voyez-vous des cas concrets où vous pourriez utiliser Docker ?

Quel avantage par rapport à la pratique actuelle ?

Si vous avez déjà adopté Docker, comparez l'avant et l'après

Les conteneurs

N'oubliez pas de lancer Docker Desktop

Objectifs

- ✓ Créer un conteneur Docker
- ✓ Identifier le rôle des images dans la création des conteneurs
- ✓ Relancer un conteneur
- ✓ Ouvrir les ports d'un conteneur vers l'extérieur
- ✓ Gérer la persistance et le partage des données entre le conteneur et la machine hôte
- ✓ Lancer un conteneur en mode interactif

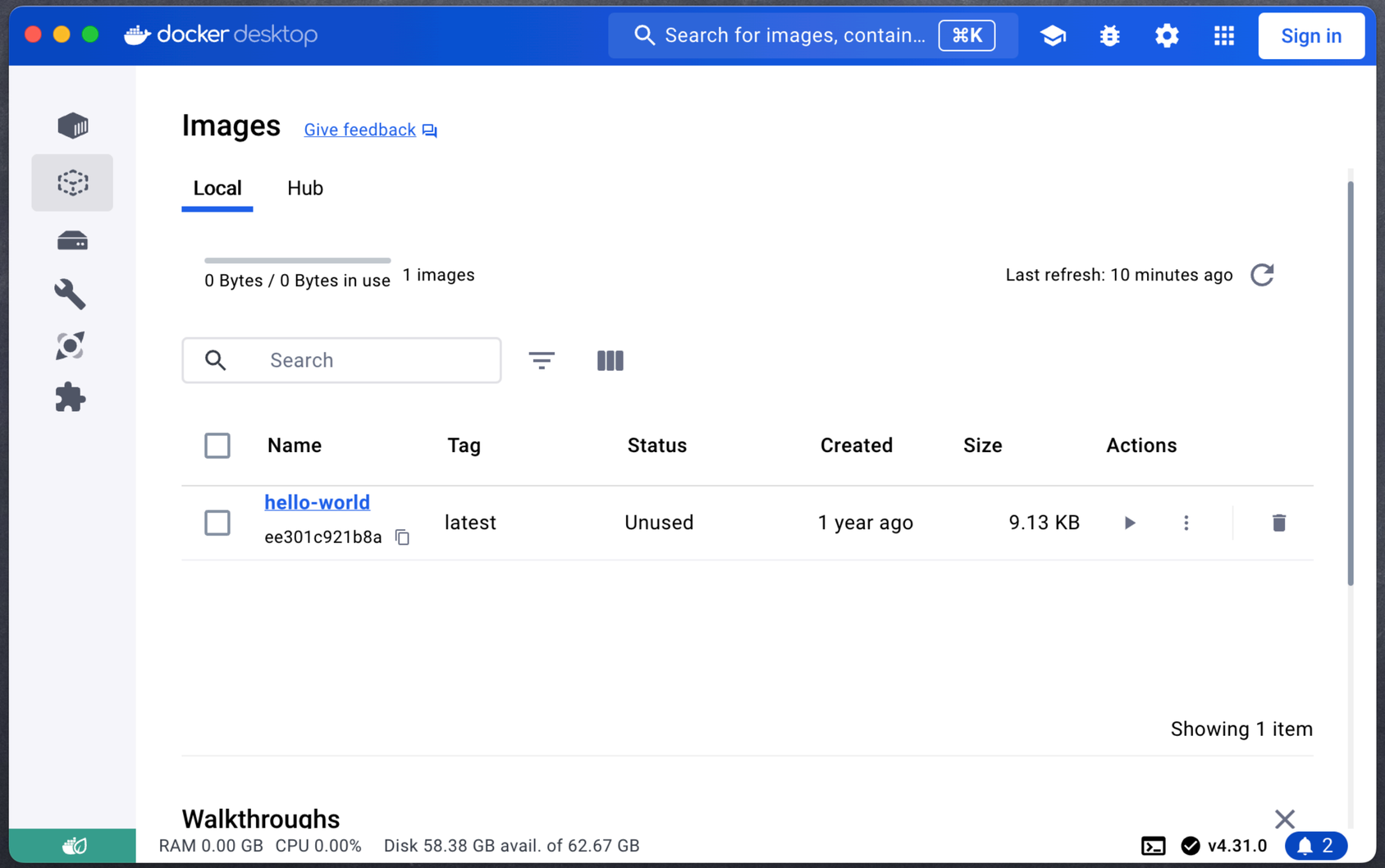
Premier conteneur

seb — -zsh — 81×29

```
seb@mo-ghra-leictreonach-1 ~ % docker container run --rm hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
478afc919002: Pull complete
Digest: sha256:94323f3e5e09a8b9515d74337010375a456c909543e1ff1538f5116d38ab3989
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.
    (arm64v8)
 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/

seb@mo-ghra-leictreonach-1 ~ %
```

Quels constats peut-on faire ?

Docker Desktop



Essayer

La commande docker-run hello-world sans le paramètre --rm

Que constatez-vous ?

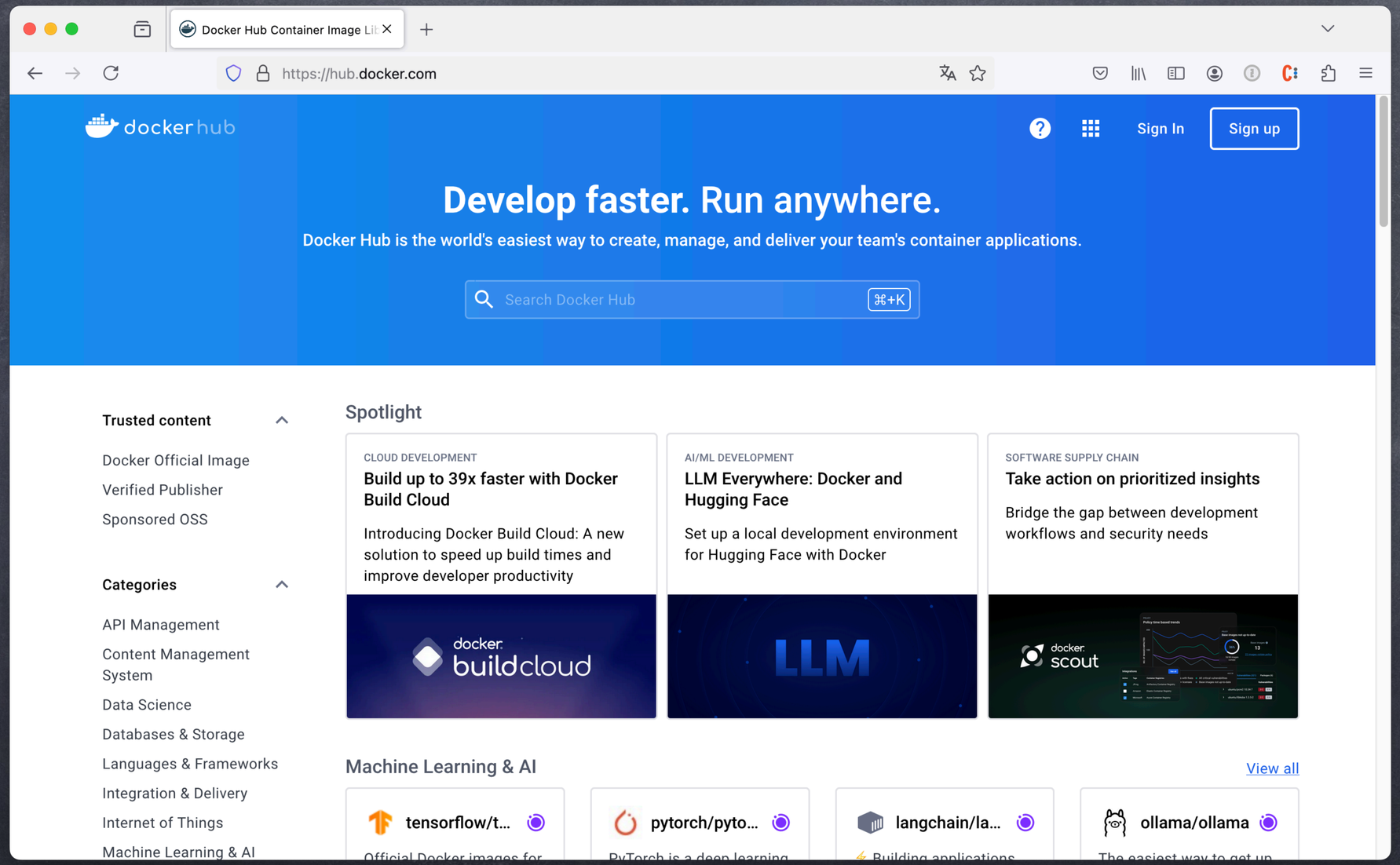
Visualisez les images et les containers dans Docker Desktop

Exécuter à nouveau la commande et vérifier l'effet dans Docker Desktop

Constats

- ✓ La commande docker container run créée un conteneur à partir d'une image
  - ✓ Chaque exécution de docker container run génère un nouveau conteneur
- ✓ Le paramètre --rm supprime le conteneur après son exécution
- ✓ Si l'image demandée n'est pas sur le disque dur, elle est téléchargée et mise en cache pour ne pas avoir à la télécharger à nouveau
- ✓ On peut visualiser les images et les conteneurs dans Docker Desktop ou bien dans un terminal

Où trouver des images Docker



Quelques astuces

Exécuter une commande
au lancement du conteneur

```
docker container run --rm bash echo hello
```

Exécuter en mode interactif

```
docker container run --rm -it bash
```

taper exit pour quitter le conteneur

Télécharge une image
sans créer de conteneur

```
docker image pull alpine
```

Lister les conteneurs

Liste les conteneurs actifs

```
docker container ps
```

```
docker container list
```

Liste tous les conteneurs

```
docker container ps|list -a
```

Le nom ou l'id du conteneur servira à faire référence à celui-ci pour le supprimer, le relancer ou encore exécuter un commande dans le conteneur

Concernant l'id il suffit de taper les premiers caractères tant qu'ils sont uniques sur l'ensemble des conteneurs sur la machine hôte

seb — -zsh — 146×11

```
seb@mo-ghra-leictreonach-1 ~ % docker container list -a
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS                      PORTS                    NAMES
08115821ca38   nginx     "/docker-entrypoint.…"   21 hours ago   Up 21 hours                 0.0.0.0:8000->80/tcp     recursing_dhawan
c82dc275b12f   nginx     "/docker-entrypoint.…"   21 hours ago   Exited (137) 21 hours ago                            trusting_galileo
4011dc762221   bash      "docker-entrypoint.s…"   25 hours ago   Exited (0) 25 hours ago                              goofy_borg
4ce623fcf984   bash      "docker-entrypoint.s…"   25 hours ago   Exited (0) 24 hours ago                              affectionate_lamarr
c6fbe0c50c9d   nginx     "/docker-entrypoint.…"   41 hours ago   Exited (0) 41 hours ago                              admiring_poitras
546e54a50f03   nginx     "/docker-entrypoint.…"   41 hours ago   Exited (0) 41 hours ago                              recursing_dirac
bbf0dbccfde6   nginx     "/docker-entrypoint.…"   42 hours ago   Exited (0) 41 hours ago                              funny_ardinghelli
279ef821e26a   bash      "docker-entrypoint.s…"   2 days ago     Exited (0) 2 days ago                                lucid_nightingale
seb@mo-ghra-leictreonach-1 ~ %
```

Nommer un conteneur


```
docker container run --name dockerShell bash
```

Le nom ou l'id du conteneur servira à faire référence à celui-ci pour le supprimer, le relancer ou encore exécuter une commande dans le conteneur

Concernant l'id il suffit de taper les premiers caractères tant qu'ils sont uniques sur l'ensemble des conteneurs sur la machine hôte

seb — -zsh — 148×11

```
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS                      PORTS                    NAMES
c8d5ef23c756   bash      "docker-entrypoint.s…"   7 seconds ago    Exited (0) 6 seconds ago                             dockerShell
08115821ca38   nginx     "/docker-entrypoint.…"   21 hours ago     Up 21 hours                 0.0.0.0:8000->80/tcp     recursing_dhawan
c82dc275b12f   nginx     "/docker-entrypoint.…"   21 hours ago     Exited (137) 21 hours ago                            trusting_galileo
4011dc762221   bash      "docker-entrypoint.s…"   25 hours ago     Exited (0) 25 hours ago                              goofy_borg
4ce623fcf984   bash      "docker-entrypoint.s…"   25 hours ago     Exited (0) 25 hours ago                              affectionate_lamarr
c6fbe0c50c9d   nginx     "/docker-entrypoint.…"   42 hours ago     Exited (0) 41 hours ago                              admiring_poitras
546e54a50f03   nginx     "/docker-entrypoint.…"   42 hours ago     Exited (0) 42 hours ago                              recursing_dirac
bbf0dbccfde6   nginx     "/docker-entrypoint.…"   42 hours ago     Exited (0) 42 hours ago                              funny_ardinghelli
279ef821e26a   bash      "docker-entrypoint.s…"   2 days ago       Exited (0) 2 days ago                                lucid_nightingale
seb@mo-ghra-leictreonach-1 ~ %
```

Utiliser une version particulière


```
docker run --rm bash:5.0 echo hello
```

bash - Official Image | Docke

hub.docker.com/_/bash

bash Docker Official Image · 100M+ · 634

Bash is the GNU Project's Bourne Again SHell

DEVELOPER TOOLS

```
docker pull bash
```
Copy

Overview Tags

Quick reference

- **Maintained by**:
  Tianon (of the Docker Community), with Chet's support (from Bash upstream)
- **Where to get help**:
  the Docker Community Slack, Server Fault, Unix & Linux, or Stack Overflow

Supported tags and respective `Dockerfile` links

- `devel-20240628`, `devel`, `devel-20240628-alpine3.20`, `devel-alpine3.20`
- `5.3-alpha`, `5.3-rc`, `rc`, `5.3-alpha-alpine3.20`, `5.3-rc-alpine3.20`, `rc-alpine3.20`
- `5.2.26`, `5.2`, `5`, `latest`, `5.2.26-alpine3.20`, `5.2-alpine3.20`, `5-alpine3.20`, `alpine3.20`
- `5.1.16`, `5.1`, `5.1.16-alpine3.20`, `5.1-alpine3.20`
- `5.0.18`, `5.0`, `5.0.18-alpine3.20`, `5.0-alpine3.20`
- `4.4.23`, `4.4`, `4`, `4.4.23-alpine3.20`, `4.4-alpine3.20`, `4-alpine3.20`
- `4.3.48`, `4.3`, `4.3.48-alpine3.20`, `4.3-alpine3.20`
- `4.2.53`, `4.2`, `4.2.53-alpine3.20`, `4.2-alpine3.20`
- `4.1.17`, `4.1`, `4.1.17-alpine3.20`, `4.1-alpine3.20`
- `4.0.44`, `4.0`, `4.0.44-alpine3.20`, `4.0-alpine3.20`
- `3.2.57`, `3.2`, `3`, `3.2.57-alpine3.20`, `3.2-alpine3.20`, `3-alpine3.20`
- `3.1.23`, `3.1`, `3.1.23-alpine3.20`, `3.1-alpine3.20`

Recent Tags

devel-alpine3.20 devel-20240628-alpine3.20 devel-20240628 devel devel-20240621-alpine3.20 devel-20240621 latest alpine3.20 5.2.26-alpine3.20 5.2.26

About Official Images

Docker Official Images are a curated set of Docker open source and drop-in solution repositories.

Why Official Images?

These images have clear documentation, promote best practices, and are designed for the most common use cases.

Le tag indique la version particulière de l'image utilisée pour construire le conteneur

Si on omet le tag alors c'est latest qui est utilisé

Obtenir de l'aide

docker commande --help

docker container --help

docker container run --help

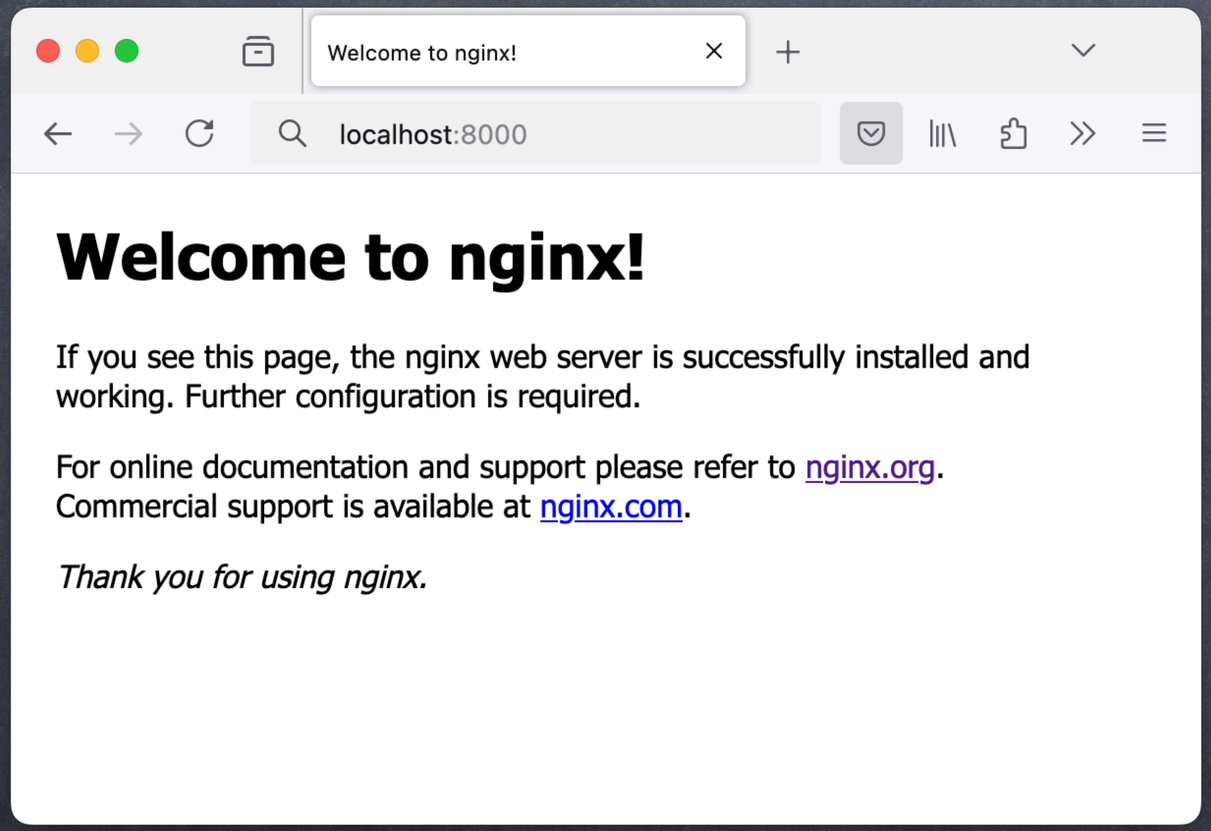
docker image --help

seb — -zsh — 81×29

```
seb@mo-ghra-leictreonach-1 ~ % docker container run --help

Usage:  docker container run [OPTIONS] IMAGE [COMMAND] [ARG...]

Create and run a new container from an image

Aliases:
  docker container run, docker run

Options:
      --add-host list                    Add a custom host-to-IP mapping
                                         (host:ip)
      --annotation map                   Add an annotation to the
                                         container (passed through to the
                                         OCI runtime) (default map[])
  -a, --attach list                      Attach to STDIN, STDOUT or STDERR
      --blkio-weight uint16              Block IO (relative weight),
                                         between 10 and 1000, or 0 to
                                         disable (default 0)
      --blkio-weight-device list         Block IO weight (relative device
                                         weight) (default [])
      --cap-add list                     Add Linux capabilities
      --cap-drop list                    Drop Linux capabilities
      --cgroup-parent string             Optional parent cgroup for the
                                         container
      --cgroupns string                  Cgroup namespace to use
                                         (host|private)
                                         'host':    Run the container in
```

Ouvrir un port


```
docker container run -p 8000:80 nginx
```



Ici le port 80 du conteneur est mis en correspondance avec le port 8000 sur la machine hôte

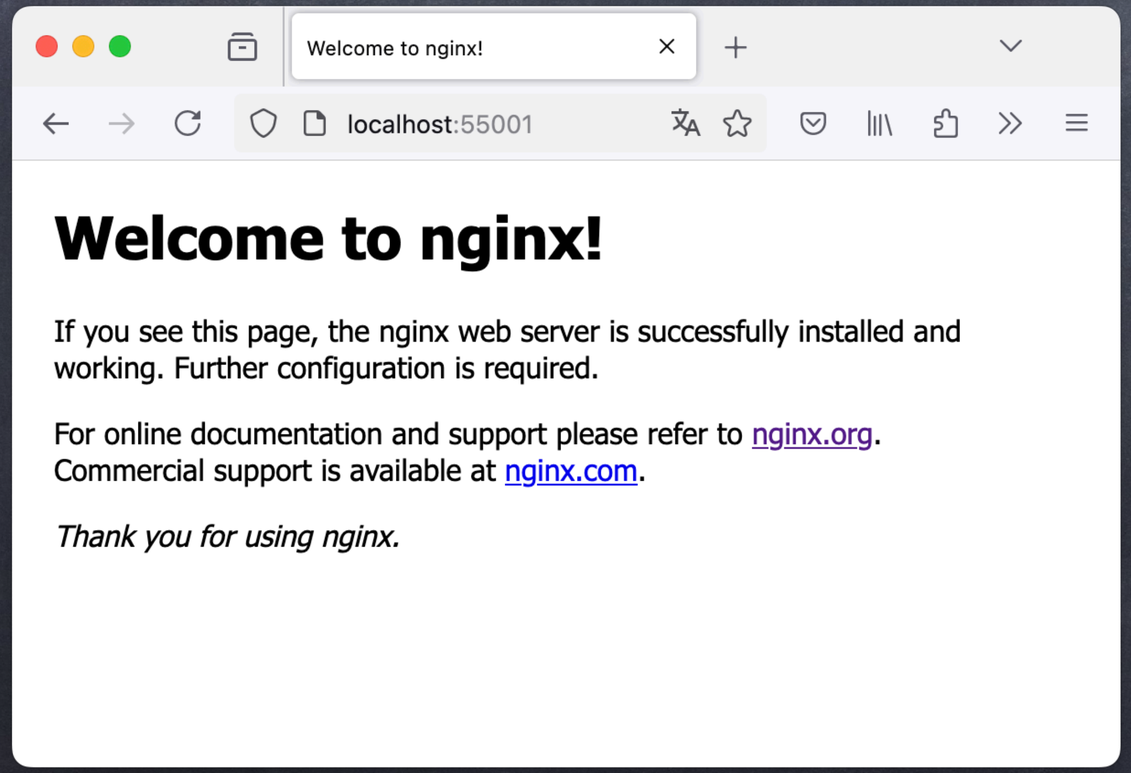
Le paramètre -d permet d'exécuter le conteneur en mode détaché et donc de laisser disponible le terminal

Ouvrir un port automatique


```
docker container run -P -d nginx
```

```
docker container ps
```

```
seb — -zsh — 122×5
seb@mo-ghra-leictreonach-1 ~ % docker container ps
CONTAINER ID   IMAGE     COMMAND                  CREATED         STATUS          PORTS                    NAMES
c6fbe0c50c9d   nginx     "/docker-entrypoint.…"   3 minutes ago   Up 3 minutes    0.0.0.0:55001->80/tcp    admiring_poitras
seb@mo-ghra-leictreonach-1 ~ %
```



Le port est ici automatiquement attribué, on peut connaitre ce port avec la commande docker container ps

Pour stopper un conteneur lancé en mode détaché on utilise la commande docker container stop suivie de l'id du conteneur ou simplement des premiers caractères de celle-ci

```
docker container stop c6fbe0c50c9d
```

Stopper ou relancer un conteneur


```
docker container stop <nom ou id>
```

```
docker container start <nom ou id>
```

Il est possible de lancer ou stopper plusieurs conteneur avec une seule commande, pour cela il suffit de taper la liste des noms ou id en séparant chaque élément avec un espace

```
docker container stop 08 eb da
```

seb — -zsh — 123×6

```
seb@mo-ghra-leictreonach-1 ~ % docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                    NAMES
eb61eb057485   nginx     "/docker-entrypoint.…"   12 seconds ago   Up 11 seconds   0.0.0.0:55002->80/tcp    crazy_lamarr
da7b48207b4c   nginx     "/docker-entrypoint.…"   14 seconds ago   Up 13 seconds   0.0.0.0:55001->80/tcp    busy_mclaren
08115821ca38   nginx     "/docker-entrypoint.…"   25 hours ago     Up 25 hours     0.0.0.0:8000->80/tcp     recursing_dhawan
seb@mo-ghra-leictreonach-1 ~ %
```

Supprimer un conteneur


```
docker container rm <nom ou id>
```

On peut aussi supprimer
tous les conteneurs inactifs

```
docker container prune
```

Passer une variable d'environnement


```
docker container run -P -d -e MYSQL_ROOT_PASSWORD=123 --name database mysql
```

exec lance une commande dans un conteneur qui tourne en mode détaché

Pour tester on ouvre une session terminal dans le contenu

```
docker container exec -it database sh
```

On lance un client mysql

```
mysql --user=root --password=123
```

On exécute des commandes SQL

```
SHOW DATABASES;
```

La persistance

Par défaut, tous les fichiers créés à l'intérieur d'un conteneur sont stockés sur une couche de conteneur accessible en écriture. Cela signifie que :

- ✓ Les données ne persistent pas lorsque le conteneur n'existe plus, et il peut être difficile d'extraire les données du conteneur si un autre processus en a besoin.
- ✓ La couche inscriptible d'un conteneur est étroitement liée à la machine hôte sur laquelle le conteneur s'exécute. Il n'est pas facile de déplacer les données ailleurs.
- ✓ L'écriture dans la couche inscriptible d'un conteneur nécessite un pilote de stockage pour gérer le système de fichiers. Le pilote de stockage fournit un système de fichiers d'union, en utilisant le noyau Linux. Cette abstraction supplémentaire réduit les performances par rapport à l'utilisation de volumes de données, qui écrivent directement dans le système de fichiers hôte.

Un petit test


```
docker container run -it bash
```

```
touch test.txt
```

```
echo "un test" > test.txt
```

```
ls
```

```
exit
```

```
docker container run -it bash
```

```
ls
```

que constatez-vous ?

```
exit
```

```
docker container start -ai [id du conteneur]
```

```
ls
```

que constatez-vous ?

On peut enregistrer des fichiers dans un conteneur

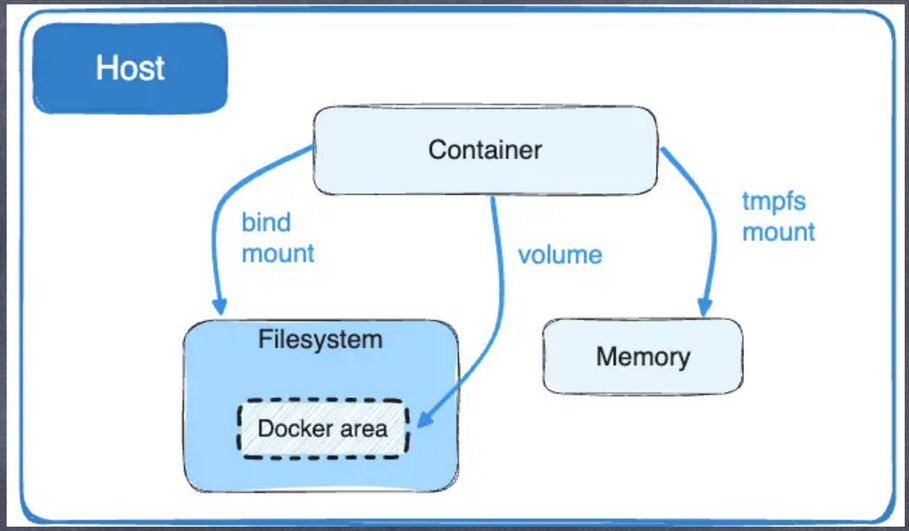
Docker container run génère un nouveau conteneur à chaque exécution

Problèmes

- ✓ Je veux travailler sur les fichiers en dehors du conteneur
- ✓ Je veux partager les mêmes fichiers entre plusieurs conteneur
- ✓ Je veux gérer les fichiers avec Git
- ✓ Je veux faire un backup des fichiers

Deux solutions

- ✓ bind mount
- ✓ volumes

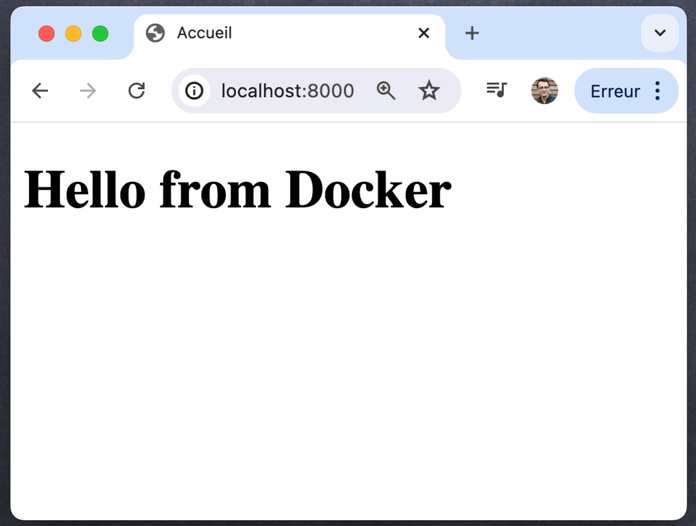


Bind mount


```
docker run -d -v ./public:/usr/share/nginx/html -p 8000:80 nginx
```

le dossier dans le conteneur

le dossier sur la machine hôte



Fait correspondre un dossier sur la machine hôte avec un dossier dans le conteneur

Montage en lecture seule


```
docker run -d -v ./public:/usr/share/nginx/html:ro -p 8000:80 nginx
```

Ici le conteneur n'a pas les droits en écriture sur le dossier monté dans son système de fichier

Cette option sécurise le dossier sur la machine hôte

Syntaxe verbeuse


```
docker run -d --mount type=bind source=./public target=/usr/share/nginx/html
readonly -p 8000:80 nginx
```

Plus explicite mais la ligne de commande est bien plus longue

A votre tour

Créer un conteneur basé sur bash qui affiche le contenu d'un fichier message.txt dans le dossier /app-data du conteneur (cat /app-data/message.txt) et établir une liaison entre ce dossier et un dossier sur la machine hôte dans lequel se trouvera un fichier message.txt

Quand utiliser bind mount

- ✓ Quand on veut centraliser les fichiers dans un seul dossier de projet
- ✓ Quand on veut modifier la configuration du conteneur depuis l'extérieur
- ✓ Quand on veut versionner les fichiers utilisés par le conteneur
- ✓ Quand le conteneur et un logiciel sur la machine hôte doivent travailler sur les mêmes fichiers

Quand ne pas utiliser bind mount

- ✓ Quand on veut partager des informations entre plusieurs conteneurs
- ✓ Quand on veut obtenir un conteneur indépendant et isolé pour le déploiement

Les volumes

Les volumes sont des espaces de stockage gérés par Docker et sont plus performants que le simple bind mount mais on ne peut pas facilement y accéder depuis la machine hôte

Utilisation des volumes

Créer un volume

```
docker volume create mysql-db
```

Utiliser un volume

```
docker run -d -v mysql-db:/var/lib/mysql -P --name database mysql
```

Lister les volumes

```
docker volume ls
```

Supprimer un volume

```
docker volume rm <nom du volume>
```

Supprimer tous les volumes inutilisés

```
docker volume prune
```

Copie des données d'un conteneur sur le disque dur


```
docker container cp /backup/db database:/var/lib/mysql
```

le conteneur

le dossier à copier

le dossier cible sur l'hôte

seb — -zsh — 144×5

```
                    is attached
seb@mo-ghra-leictreonach-1 ~ % docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED          STATUS          PORTS                                                   NAMES
1f47e3edca68   mysql     "docker-entrypoint.s…"   43 minutes ago   Up 43 minutes   0.0.0.0:55006->3306/tcp, 0.0.0.0:55005->33060/tcp   database
seb@mo-ghra-leictreonach-1 ~ %
```

Petit débat

- ✓ Lister des cas d'utilisation pour bind mount et les volumes
- ✓ Comment choisir l'un ou l'autre

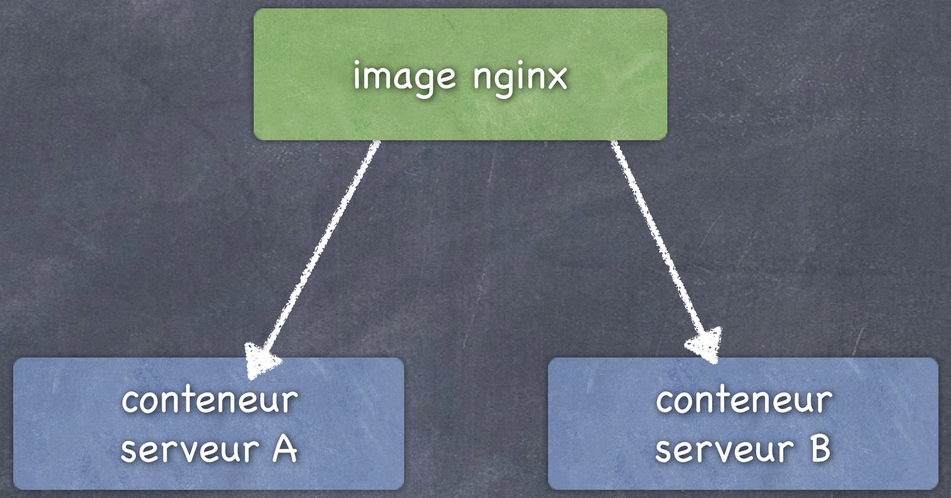
Les images

Objectifs

- ✓ Gérer les images
- ✓ Optimiser les images
- ✓ Créer nos propres images
- ✓ Publier une image sur le Docker Hub

Rappel

- ✓ L'image est un modèle sur lequel Docker se base pour instancier un conteneur
- ✓ Le Docker Hub est un dépôt en ligne pour de telles images
- ✓ Les images sont téléchargées et stockées sur le disque dur de la machine hôte



Quelques commandes utiles

Commande	Description
`docker image ls`	Affiche la liste des images
`docker image rm <nom ou id>`	Supprime une image
`docker image prune`	Supprime les images inutilisées
`docker image pull <nom ou id>`	Télécharge une image sans instancier de conteneur

Anatomie d'une image

docker image history bash

seb — -zsh — 111×13

```
seb@mo-ghra-leictreonach-1 ~ % docker image history bash
IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
722acc000ce5   6 weeks ago   CMD ["bash"]                                    0B        buildkit.dockerfile.v0
<missing>      6 weeks ago   ENTRYPOINT ["docker-entrypoint.sh"]             0B        buildkit.dockerfile.v0
<missing>      6 weeks ago   COPY docker-entrypoint.sh /usr/local/bin/ # …   212B      buildkit.dockerfile.v0
<missing>      6 weeks ago   RUN /bin/sh -c set -eux;   apk add --no-cach…   8.75MB    buildkit.dockerfile.v0
<missing>      6 weeks ago   ENV _BASH_LATEST_PATCH=26                       0B        buildkit.dockerfile.v0
<missing>      6 weeks ago   ENV _BASH_BASELINE_PATCH=21                     0B        buildkit.dockerfile.v0
<missing>      6 weeks ago   ENV _BASH_BASELINE=5.2.21                       0B        buildkit.dockerfile.v0
<missing>      6 weeks ago   ENV _BASH_VERSION=5.2.26                        0B        buildkit.dockerfile.v0
<missing>      6 weeks ago   /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      6 weeks ago   /bin/sh -c #(nop) ADD file:033626ac44156f3d5…   8.83MB
seb@mo-ghra-leictreonach-1 ~ %
```

Une image est composée de couches successives

Chaque couche représente une instruction

Ajouter Git à une image alpine


```
docker container run -it --name alpine-container alpine
```

```
apk add git
```

Installation de Git

```
git --version
```

Test de l'installation

```
exit
```

sortie du conteneur

Transformation du conteneur en image


```
docker image commit alpine-container alpine-with-git:1.0
```

- alpine-container : nom ou id du conteneur
- alpine-with-git : nom de l'image a créer
- 1.0 : tag de l'image a créer

Instancier un conteneur basé sur cette nouvelle image et tester la disponibilité de Git.

```
docker container run -it alpine-with-git:1.0
```

Comparer les historiques des deux images

```
docker image history alpine-with-git:1.0
docker image history alpine
```

Pause réflexion

Que pensez-vous de cette méthode pour créer des images ?

Une autre façon de créer des images

- ✓ Les étapes de la création sont scriptées dans un fichier Dockerfile (attention au D majuscule)
- ✓ Chaque étape est convertie en couche dans la nouvelle image

Un Dockerfile


```
FROM alpine:latest

RUN apk add git

RUN apk add vim
```

L'image de départ

Les instructions supplémentaires

```
docker image build -t alpine-with-git:2.0 .
```

nom et tag
de la nouvelle image

chemin du contexte où le fichier
Dockerfile se trouve.
Ici le . représente le dossier en cours

Quelle différence par rapport à un simple commit ?

docker image history alpine-with-git:2.0

```
seb — -zsh — 121×8
seb@mo-ghra-leictreonach-1 ~ %  docker image history alpine-with-git:2.0

IMAGE          CREATED              CREATED BY                                      SIZE      COMMENT
adbd170e2b85   About a minute ago   RUN /bin/sh -c apk add vim # buildkit           29.8MB    buildkit.dockerfile.v0
<missing>      About a minute ago   RUN /bin/sh -c apk add git # buildkit           15.6MB    buildkit.dockerfile.v0
<missing>      2 weeks ago          /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      2 weeks ago          /bin/sh -c #(nop) ADD file:033626ac44156f3d5…   8.83MB
seb@mo-ghra-leictreonach-1 ~ %
```

docker image history alpine-with-git:1.0

```
seb — -zsh — 96×6

IMAGE          CREATED       CREATED BY                                      SIZE      COMMENT
250bf497ed4a   3 hours ago   /bin/sh                                         15.6MB
092561eea88f   2 weeks ago   /bin/sh -c #(nop)  CMD ["/bin/sh"]              0B
<missing>      2 weeks ago   /bin/sh -c #(nop) ADD file:033626ac44156f3d5…   8.83MB
seb@mo-ghra-leictreonach-1 ~ %
```

Dockerfile vs commit

	Dockerfile	Commit
Objectif	Construire une image à partir d'un script	Sauvegarder un conteneur actif dans une image
Reproductibilité	Importante	Faible
Version control	Oui	Non
Cas d'usage	Automatiser la création des images	Faire un cliché de l'état d'un conteneur lors d'une phase de prototypage ou d'expérimentation
Inconvénients	Demande d'apprendre une nouvelle syntaxe	Pas d'historique des modifications

Quelques instructions du Dockerfile

Instruction	Description
FROM	Définit l'image de départ
RUN	Exécute une instruction
ENV	Définit une variable d'environnement
WORKDIR	Définit le dossier de travail qui sera utilisé par défaut pour les instructions suivantes
COPY	Copie un fichier ou un dossier
ADD	Comme COPY, mais la source peut être un url
CMD	Définit la commande exécutée lors du lancement d'un conteneur

Dockeriser une application

app.js

```
const process = require('process');

// Gestion du timer
let count = 0;
const timer = setInterval(()=> {
    count++;
    console.clear();
    console.log(count);
}, 1000);

// Arrêt du timer sur l'appui d'une touche
process.stdin.setRawMode(true);

process.stdin.on('data', ()=> {
    console.log('terminé');
    process.exit();
});
```

Dockerfile

```
FROM node:alpine
COPY . .
CMD node app
```

Créer une image et instancier un conteneur pour tester cette application

Ici on ne fait que copier les fichiers de l'application

Une application avec des dépendances

app.js

```javascript
const express = require('express');
const randomColor = require('randomcolor');

const app = express();

app.get('/', (req, res) => {
    const result = {
        color: randomColor({ format: 'hex' })
    };

    res.status(200).send(JSON.stringify(result));
});

app.listen(3000);
```

package.json

```json
{
  "name": "random-color-app",
  "version": "1.0.0",
  "description": "get a random color",
  "main": "app.js",
  "scripts": {
    "start": "node app"
  },
  "license": "ISC",
  "dependencies": {
    "express": "^4.19.2",
    "randomcolor": "^0.6.2"
  }
}
```

Le Dockerfile

Dockerfile

```
FROM node:alpine

EXPOSE 3000

COPY . .

RUN npm install

CMD node start
```

Le port exposé, cette instruction n'est qu'indicative

Copie des fichiers

Installation des dépendances

Lancement de l'application

.dockerignore

```
node_modules
.idea
.git
```

Liste les fichiers qui ne doivent pas être copiés dans l'image

Le Dockerfile optimisé

Dockerfile

```
FROM node:alpine

WORKDIR /app
EXPOSE 3000

COPY package.json .
RUN npm install

COPY . .
CMD "node start"
```

Définition d'un dossier de travail dans lequel les instruction seront exécutées

Copie de package.json et téléchargement des dépendances

Copie du reste du code de l'application

.dockerignore

```
node_modules
.idea
.git
```

Liste les fichiers qui ne doivent pas être copiés dans l'image

Comment Docker construit les images ?

- ✓ Une image est constituée de couches (voir docker history)
- ✓ Chaque instruction du Dockerfile génère une nouvelle image qui sera utilisée comme base pour l'instruction suivante et apparaitra comme une couche
- ✓ Chacune des étapes est mise en cache pour accélérer une exécution future

Comment limiter le nombre de couches ?


```
FROM alpine:latest

RUN apk add git vim
```

La commande apk add admet une liste d'arguments

```
FROM alpine:latest

RUN apk add git \
vim
```

On peut également écrire l'instruction sur plusieurs lignes

```
FROM alpine:latest

RUN apk update && \
    apk add git vim && \
    apk cache clean
```

&& permet de combiner des instructions différentes

ENTRYPOINT ou CMD

- ✓ Les deux exécutent une instruction au lancement d'un conteneur
- ✓ S'il y a plus d'un ENTRYPOINT ou CMD dans une Dockerfile seul le dernier est pris en compte
- ✓ CMD peut être écrasé par la commande docker run
- ✓ ENTRYPOINT est toujours exécuté

ENTRYPOINT ou CMD exemple 1

Dockerfile

```
FROM bash:latest
CMD echo "hello"
```

```
docker image build -t hello-cmd:latest .
```

```
docker container run hello-cmd
```

La commande CMD de l'image est utilisée

```
docker container run hello-cmd echo "bonjour"
```

La ligne de commande écrase CMD

ENTRYPOINT ou CMD exemple 2

Dockerfile

```
FROM bash:latest
ENTRYPOINT echo "hello"
```

```
docker image build -t hello-entrypoint:latest .
```

```
docker container run hello-entrypoint
```

La commande ENTRYPOINT de l'image est utilisée

```
docker container run hello-entrypoint echo "bonjour"
```

La ligne de commande est ignorée

ENTRYPOINT ou CMD exemple 3

Dockerfile

```
FROM bash:latest
ENTRYPOINT echo
CMD "world"
```

```
docker image build -t hello-entrypoint:latest .
```

```
docker container run hello-entrypoint
```

CMD est utilisé comme argument de ENTRYPOINT

```
docker container run hello-entrypoint
"bonjour"
```

La ligne de commande est utilisée comme argument

Les variables d'environnement

Dockerfile

```
FROM mysql
ENV MYSQL_ROOT_PASSWORD 123
```

```
docker image build -t mysql-image .
```

```
docker container run -d --name mysqlDB mysql-image
```

Lancement du conteneur en mode déconnecté

```
docker container exec mysqlDB sh
```

Lancement d'un shell dans le conteneur

```
mysql --user=root --password=123
```

Ouverture d'un client MySQL

Récap sur les variables d'environnement

- ✓ La variable d'environnement est disponible pendant l'exécution du conteneur
- ✓ Une variable définie dans un Dockerfile peut être écrasée par un argument -e ou --env de docker container run

Une image utilitaire

Dockerfile

```
FROM alpine
WORKDIR /app
RUN apk add nodejs npm
```

```
docker image build -t npm-util .
```

```
docker container run -d --name node-util -v ./:/app npm-util
```

```
docker container exec -it  node-util npm init
```

Tester l'initialisation d'un projet

Cette image expose un outil de dev (ici npm) et peut nous éviter de l'installer sur la machine hôte. Cette technique est très utile quand d'un projet à l'autre les outils demandent des versions ou des paramètres différents

Une image utilitaire autre variante

Dockerfile

```
FROM alpine
WORKDIR /app
RUN apk add nodejs npm
ENTRYPOINT npm
```

```
docker image build -t npm-util2 .
```

```
docker container run --rm -v ./:/app npm-util2 init
```

Ici on créé un nouveau conteneur à chaque fois que l'on veut utiliser une commande npm. Grâce ua paramètre --rm ce conteneur est détruit automatiquement après utilisation.

Image PHP

Dockerfile

```
docker image build -t php .
```

```
FROM php:8.0-apache

# Dépendances PHP
RUN apt-get update && \
    apt-get install -y libpng-dev libjpeg-dev libfreetype6-dev && \
    docker-php-ext-configure gd --with-jpeg --with-freetype && \
    docker-php-ext-install gd mysqli pdo_mysql
```

Installation de bibliothèques indispensables pour l'extension GD

Configuration des extensions

Installation des extensions

Exercices

- ✓ Créer un site web simple (une ou deux pages, quelques images du css) et le dockeriser dans un conteneur incluant le serveur web (nginx ou apache)
- ✓ Faire de même pour une application javascript construite avec Webpack

Les réseaux

Les types de réseaux

- ✓ Bridge (défaut)
- ✓ Host
- ✓ Overlay (pour swarm et Kubernetes)
- ✓ None

Réseau bridge par défaut

création des conteneurs

```
docker run -dit --name conteneur1  alpine
```

```
docker run -dit --name conteneur2  alpine
```

Obtention de l'adresse IP

```
docker inspect --format='{{.NetworkSettings.IPAddress}}' conteneur2
```

Ouverture d'une session shell

```
docker exec -it conteneur1 sh
```

ping sur conteneur 2

```
ping <ip de conteneur2>
```

Réseau nommé

création du réseau perso

```
docker network create --driver bridge mon_reseau
```

création des conteneurs

```
docker run -dit --name conteneur1 --network mon_reseau  alpine
```

```
docker run -dit --name conteneur2 --network mon_reseau  alpine
```

Ouverture d'une session shell

```
docker exec -it conteneur1 sh
```

ping sur conteneur 2

```
ping conteneur2
```

le réseau nommé inclus un DNS
qui fait correspondre
les noms des conteneurs
avec leur adresse IP

Avantages du réseau bridge nommé

- ✓ Communication interne via le DNS intégré
- ✓ Isolation du réseau bridge par défaut

Réseau host

création dun conteneur

```
docker run -d --network host --name nginx_test nginx
```

le conteneur n'a pas d'IP

```
docker inspect nginx_test
```

test du serveur web

```
curl http://localhost:80
```

```
http://localhost
```

Le conteneur utilisant le réseau host partage directement le réseau de l'hôte, pas besoin de publier les ports

Comparaison avec bridge

- ✓ Aucune isolation
- ✓ Conflits de ports possibles
- ✓ Moins de latence réseau
- ✓ Aucune configuration
- ✓ Uniquement sur Linux

Compose

Objectifs

- ✓ Créer des stacks multi conteneur avec Compose
- ✓ Simplifier la configuration des conteneurs et la création des images

Principes

- ✓ Un seul fichier yaml pour décrire la configuration d'un ensemble de conteneurs
- ✓ Une seule commande console pour les lancer tous

Anatomie d'un fichier Yaml

une sous clef
(l'indexation est importante)

une clef

```
services:
    frontend:
        image: my-vue-app
    backend:
        image: my-springboot-app
    db:
        image: postgres
```

une clef et une valeur

docker-compose.yml

Un exemple avec un seul service

le nom du service

le nom du conteneur

correspondance des volumes

```
services:
  webServer:
    image: nginx
    container_name: nginx_server
    ports:
      - '8000:80'
    volumes:
      - './www:/usr/share/nginx/html'
```

l'image instanciée

correspondance des ports

```
docker compose up -d
```

Deux services nginx et PHP


```yaml
services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "8080:80"
    volumes:
      - ./www:/var/www/html
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
    depends_on:
      - php

  php:
    build:
      context: ./php
    container_name: php
    volumes:
      - ./www:/var/www/html
```

```dockerfile
FROM php:8.2-fpm
RUN docker-php-ext-install pdo_mysql
```

./php/Dockerfile

```nginx
server {
    listen 80;
    server_name localhost;

    root /var/www/html;
    index index.php index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME /var/www/html$fastcgi_script_name;
    }

    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
}
```

Le fichier default.conf


```
server {
    listen 80;
    server_name localhost;

    root /var/www/html;
    index index.php index.html;

    location / {
        try_files $uri $uri/ =404;
    }

    location ~ \.php$ {
        include fastcgi_params;
        fastcgi_pass php:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME /var/www/html$fastcgi_script_name;
    }

    error_log  /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
}
```

Mariadb avec un volume géré par Docker


```
services:
    #serveur de base de donnees
    myDatabase:
       image: 'mariadb:10.3'
       container_name: myDatabase
       restart: always
       environment:
          - MYSQL_USER: user
          - MYSQL_PASSWORD: 123
          - MYSQL_DATABASE: my_database
          - MYSQL_ROOT_PASSWORD: 456
       ports:
           - '3306:3306'
       volumes:
           - mariadb_data:/var/lib/mysql/
volumes:
    mariadb_data:
          driver: local
```

Intégration multi service


```yaml
version: '3.8'

services:
  nginx:
    image: nginx:latest
    container_name: nginx
    ports:
      - "8080:80"
    volumes:
      - ./nginx.conf:/etc/nginx/nginx.conf:ro
      - ./app:/var/www/html
    depends_on:
      - php
    networks:
      - shared_network

  php:
    image: php:8.1-fpm
    container_name: php
    volumes:
      - ./app:/var/www/html
    networks:
      - shared_network

  mariadb:
    image: mariadb:latest
    container_name: mariadb
    environment:
      MYSQL_ROOT_PASSWORD: rootpassword
      MYSQL_DATABASE: mydatabase
      MYSQL_USER: myuser
      MYSQL_PASSWORD: mypassword
    volumes:
      - mariadb_data:/var/lib/mysql
    networks:
      - shared_network

  phpmyadmin:
    image: phpmyadmin/phpmyadmin:latest
    container_name: phpmyadmin
    environment:
      PMA_HOST: mariadb
      PMA_USER: myuser
      PMA_PASSWORD: mypassword
    ports:
      - "8081:80"
    depends_on:
      - mariadb
    networks:
      - shared_network

volumes:
  mariadb_data:

networks:
  shared_network:
    driver: bridge
```