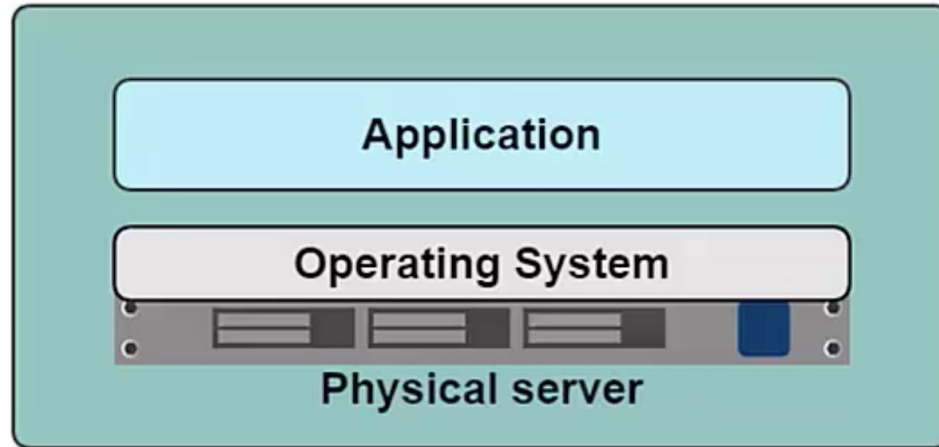


Docker

Quentin Hermiteau
quentinhermiteau.esgi@gmail.com

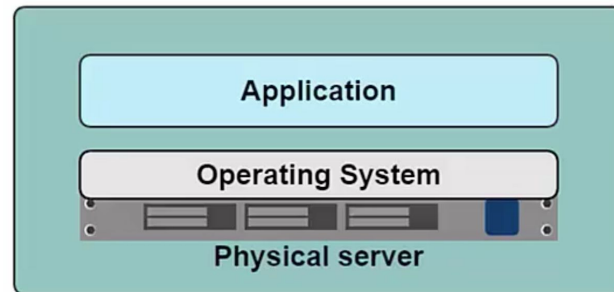
Comment on faisait avant?

Il y a longtemps, une seule application tournait sur un seul serveur physique.



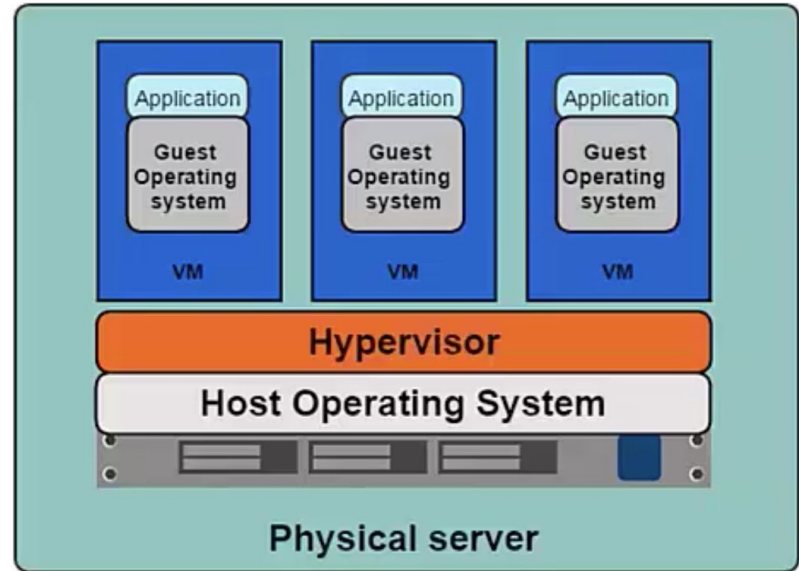
Problèmes rencontrés

- Déploiement lent
- Coût élevé
- Ressources perdues
- Difficulté à faire évoluer
- Difficulté à migrer
- Lien fort avec les fournisseurs



La virtualisation

- Un serveur physique contient plusieurs applications
- Chaque application tourne dans une machine virtuelle



Avantages des VM

- Meilleure gestion des ressources
 - Un serveur physique est divisé en plusieurs machines virtuelles
- Évolution facilitée
- VM dans le cloud
 - Souplesse et rapidité
 - Ne payer que ce que l'on consomme

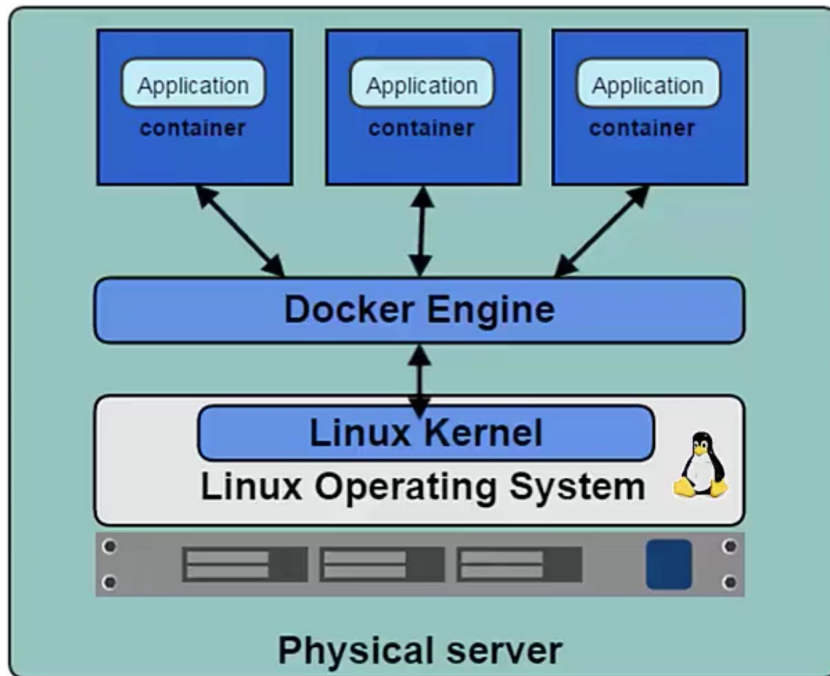
Limites des VM

- Chaque VM requiert pour ses besoins :
 - Allocation CPU
 - Espace disque
 - RAM
 - Un OS hôte complet
- Plus il y a de VM, plus il y a besoin de ressources
- L'OS hôte consomme des ressources
- Le portage des applications n'est pas garanti



Docker et le noyau Linux

- Docker Engine est le programme qui permet la construction, le déploiement et l'exécution du container
- Docker Engine utilise les namespaces du noyau Linux ainsi que les groupes de contrôle
- Les namespaces permet l'isolation des espaces de travail



Containers vs VMs

- Les containers sont beaucoup plus légers
- Pas besoin d'installer d'OS
- Moins de CPU, RAM et disque utilisés
- Il est possible de mettre beaucoup plus de containers que de VMs par serveur
- Extrême portabilité

Installation de docker

- Suivez les instructions sur <https://docs.docker.com/installation> pour installer la dernière version de docker sur votre machine
- Lancez un container depuis l'image hello-world pour tester votre installation :



```
docker run hello-world
```

Hello Docker

`docker run hello-world`

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
7050e35b49f5: Pull complete
Digest: sha256:6e8b6f026e0b9c419ea0fd02d3905dd0952ad1fee67543f525c73a0a790fefb
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/>

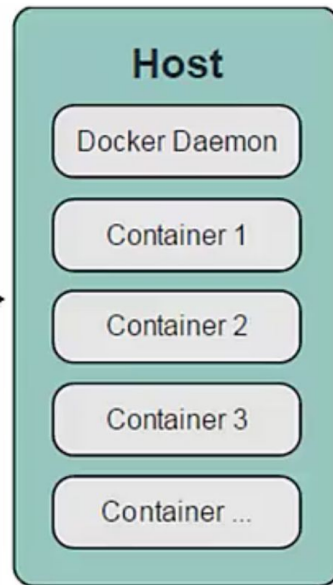
For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Client et Daemon Docker

- Architecture client/serveur
- Le client prend les données en entrée et les transmet au Daemon
- Celui-ci les construit, les exécute et les distribue aux containers
- Client et Daemon peuvent tourner ou non sur le même hôte

Client



Vérifier version Client et Daemon

docker version

Client:

Cloud integration: v1.0.29
Version: 20.10.22
API version: 1.41
Go version: go1.18.9
Git commit: 3a2c30b
Built: Thu Dec 15 22:28:41 2022
OS/Arch: darwin/arm64
Context: default
Experimental: true

Server: Docker Desktop 4.16.2 (95914)

Engine:

Version: 20.10.22
API version: 1.41 (minimum version 1.12)
Go version: go1.18.9
Git commit: 42c8b31
Built: Thu Dec 15 22:25:43 2022
OS/Arch: linux/arm64
Experimental: false

containerd:

Version: 1.6.14
GitCommit: 9ba4b250366a5ddde94bb7c9d1def331423aa323

runc:

Version: 1.1.4
GitCommit: v1.1.4-0-g5fd4c4d

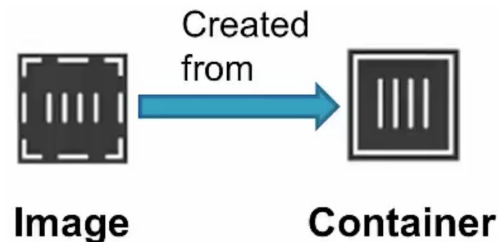
docker-init:

Version: 0.19.0
GitCommit: de40ad0

Image et container

- Images

- Schéma qui permet de créer des containers
- Créés par vous ou par d'autres utilisateurs de docker
- Stockées dans docker hub ou localement sur votre machine



- Containers

- “Entité” d’une application qui se suffit à elle-même
- Contient tout ce qui est nécessaire pour faire tourner l’application
- Basés sur une image

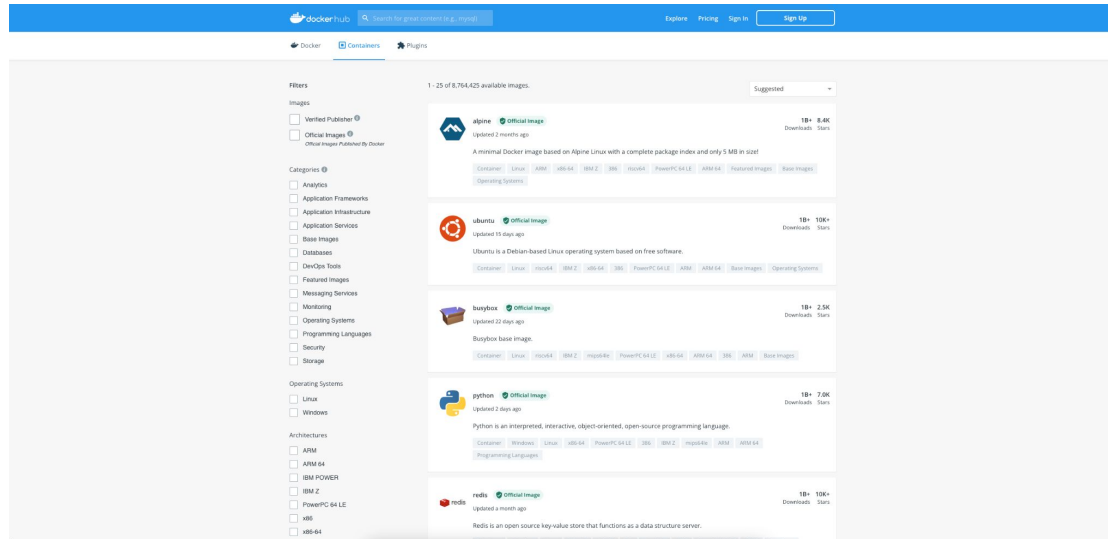
Avantages de docker

- Séparation des préoccupations
 - Les développeurs travaillent sur leurs applications
 - Les administrateurs système sur le déploiement
- Cycle de développement rapide
- Portabilité des applications
 - Développées dans un environnement, distribuées dans un autre
- Montée en charge
 - Il est très facile de gérer la montée en charge
- Plus d'applications peuvent tourner sur un serveur

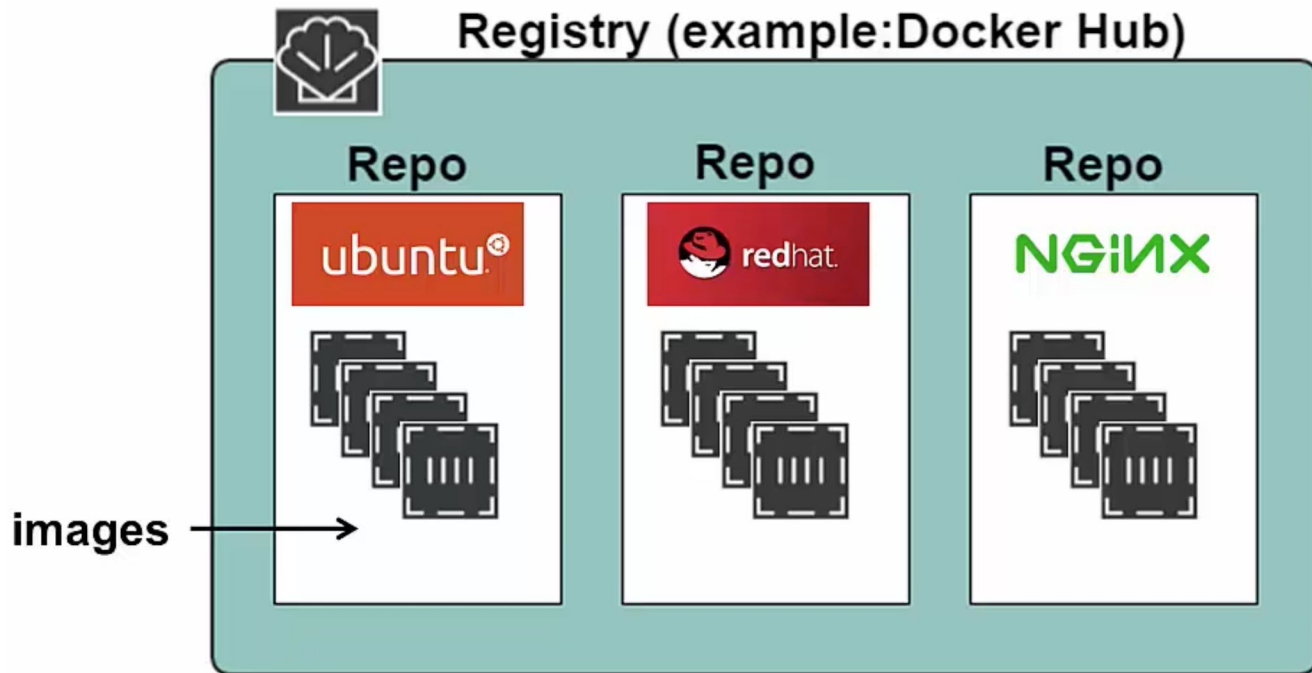
Introduction aux images docker

Docker hub

- Docker hub est un registry public qui contient un très grand nombre d'images pour tous les usages

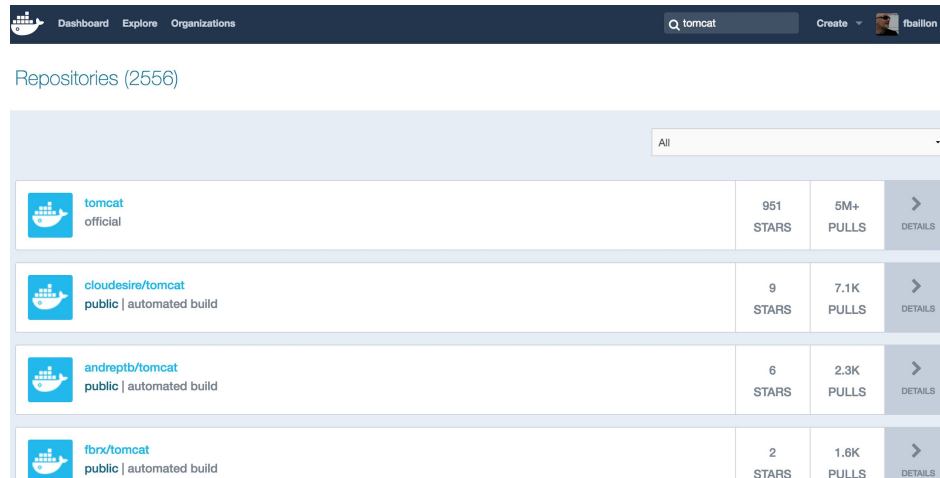


Registry et repo







Images sur docker hub

- Beaucoup d'images sont disponibles
- Les images sont dans différents repository



The screenshot shows the Docker Hub interface with a search bar containing 'tomcat'. Below the search bar, there is a list of repositories. The first repository is 'tomcat/official' with 951 stars and 5M+ pulls. The other three repositories are 'cloudesire/tomcat', 'andreptb/tomcat', and 'fbrx/tomcat', all marked as 'public | automated build'.

Repositories (2556)				
	tomcat official	951 STARS	5M+ PULLS	> DETAILS
	cloudesire/tomcat public automated build	9 STARS	7.1K PULLS	> DETAILS
	andreptb/tomcat public automated build	6 STARS	2.3K PULLS	> DETAILS
	fbrx/tomcat public automated build	2 STARS	1.6K PULLS	> DETAILS

Créez votre compte docker hub

- Créer votre compte si vous n'en avez pas encore
 - <https://hub.docker.com/account/signup/>
- Activez votre compte avec l'email reçu
- Naviguez sur le site dans les dépôts
- Cherchez des images de vos outils, langages, serveurs favoris...

Afficher les images locales

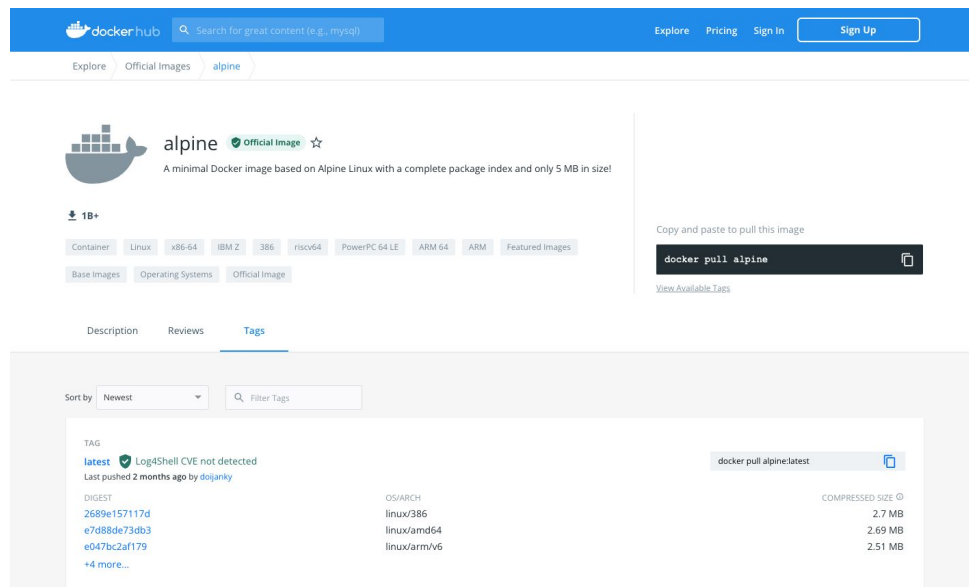
- `docker images`
- Chercher des images en local

`docker images`

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
alpine-ssh	latest	60228c9c409a	45 hours ago	17.4MB
alpine-user	latest	40cf2ffb5234	47 hours ago	7.46MB
node	alpine	309fc9aaf0d7	3 days ago	174MB
composer	latest	140ec3510943	10 days ago	191MB
php	fpm-alpine	f9461c1557f8	11 days ago	71.7MB
php	apache	088fba3a7330	11 days ago	430MB
nginx	alpine	c473535d7680	2 weeks ago	40.3MB
alpine-test	latest	3b616b37d0e7	2 weeks ago	7.46MB
alpine	3.17	d74e625d9115	2 weeks ago	7.46MB
alpine	latest	d74e625d9115	2 weeks ago	7.46MB
adminer	4.8.1	78090926dc14	2 weeks ago	243MB
phpmyadmin	5.2.0	cdc96bbdba0b	3 weeks ago	479MB
ubuntu	22.04	a6be1f66f70f	4 weeks ago	69.2MB
mariadb	10.9.3	cfe889c390f2	3 months ago	366MB
hello-world	latest	46331d942d63	11 months ago	9.14kB

Le tag sur les images

- Les images sont identifiées par des tags
- Une même image peut avoir plusieurs tags
- Le tag par défaut est `latest`
- Dans docker hub, il y a le détail de chaque tag



The screenshot shows the Docker Hub interface for the 'alpine' image. The page is titled 'alpine' and is marked as an 'Official Image'. It describes the image as 'A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size!'. The 'Tags' tab is selected, showing a list of tags. The 'latest' tag is highlighted, and its details are shown below, including the digest and the compressed size for different architectures.

Container Linux x86-64 IBM Z 386 riscv64 PowerPC 64 LE ARM 64 ARM Featured Images

Base Images Operating Systems Official Image

Description Reviews **Tags**

Sort by Newest Filter Tags

TAG	OS/ARCH	COMPRESSED SIZE
latest Log4Shell CVE not detected Last pushed 2 months ago by dojanjky		
DIGEST		
2689e157117d	linux/386	2.7 MB
e7d88de73db3	linux/amd64	2.69 MB
e047bc2af179	linux/arm/v6	2.51 MB
+4 more...		

docker pull alpine:latest

Commencer avec les containers

Créer un container

- Utiliser la commande `run`
- Syntaxe: `docker run [options] image [command] [args]`
- L'image est définie par `repository:tag`

```
docker run ubuntu:22.04 uname -a
```

```
Linux 7eee5913e3ee 5.15.49-linuxkit #1 SMP PREEMPT Tue Sep 13 07:51:32 UTC 2022 aarch64 aarch64 aarch64 GNU/Linux
```

```
docker run ubuntu:22.04 ps ax
```

PID	TTY	STAT	TIME	COMMAND
1	?	Rs	0:00	ps ax

Exécuter un container simple

- Dans votre terminal, tapez la commande suivante :
 - `docker run ubuntu:20.04 echo "Hello world"`
- Observez ce que vous obtenez
- Puis tapez :
 - `docker run ubuntu ps ax`
- Observez ce que vous obtenez
- Puis tapez :
 - `docker run ubuntu`
- Observez ce que vous obtenez
- Vous pouvez voir que la seconde commande est presque instantanée car maintenant l'image `ubuntu:latest` est locale

Container avec un terminal

- Utilisez les options `-i` et `-t` lors de l'exécution
 - `-i` demande la connexion à STDIN dans le container
 - `-t` demande un pseudo terminal
- Vous devez spécifier un processus de terminal
 - exemple de processus : `/bin/bash`

```
docker run -it ubuntu:22.04 bash
```

```
root@8ac0c3eb860b: /#
```

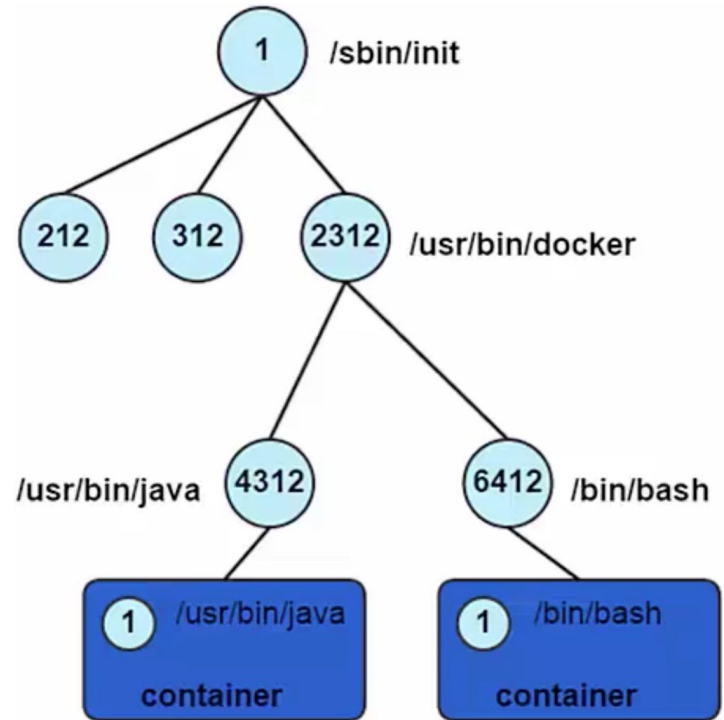


Terminal dans un container

- Créer un container ubuntu, et lancer le terminal:
 - `docker run -it alpine sh`
- Dans le container, créez un nouvel utilisateur
 - `adduser newuser`
- Puis ajoutez un fichier `README.md`
 - `cd /home/newuser`
 - `touch README.md`
 - `ls`
- Sortez du container, puis relancez la même commande pour créer un container, l'utilisateur n'est plus là
 - `ls`
 - Le fichier n'est plus dans le répertoire

Container processus

- Le container s'exécute tant que le processus que vous avez lancé avec `run` est actif
- Le numéro du PID de la commande lancée dans votre container porte toujours le numéro 1



ID d'un container

- Il est possible d'accéder à un container par son nom ou par son ID
- L'ID est long, la version courte est plus utile
- Le nom et l'ID peuvent être obtenus avec `docker ps`
- L'ID long est obtenu en inspectant le container

`docker ps`

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
e96f6913728a	ubuntu:22.04	"bash"	2 seconds ago	Up 1 second		stoic_blackwell

- Utiliser `docker ps` pour lister les containers actifs
- L'option `-a` permet de les lister tous, y compris ceux qui sont arrêtés

```
utilisateur@machine:~/ $ docker ps
```

CONTAINER ID	IMAGE	COMMAND	STATUS	NAMES
1b17ec69560f	docker:dind	"dockerd-entryp..."	Up 47 minutes	romantic_bhabha

Exécution en mode détaché

- Aussi appelé en tâche de fond ou detach
- Il faut utiliser l'option `-d`
- Pour observer la sortie, utilisez `docker logs [container ID]`

Lister vos containers

- Exécutez :
 - `docker run -d alpine ping -c 100 127.0.0.1`
- Afficher la liste de vos containers en utilisant
 - `docker ps`
- Vérifier que le container créé est actif
- Afficher la liste de tous vos containers en utilisant
 - `docker ps -a`
- Vérifier que l'ensemble des containers créés depuis le début est présente

Commandes principales

docker info

Affiche les informations sur l'installation de docker

- le nombre de containers (lancés, en pause, arrêtés)
- le nombre d'images
- la version
- la mémoire utilisée, le nombre de cpu
- l'utilisateur du hub

docker run

Créer un container depuis une image

- il existe de nombreux paramètres pour sa configuration, et qui peuvent être combinés
- `docker run [options] image[:tag|@digest] [command] [arg1, arg2, ...]`
 - **ex.:** `docker run -ti ipssi-sshd:2.0 /bin/sh`
 - `-d` lance en background
 - `-v` pour partager un dossier / fichier entre l'hôte et le container

docker ps

Liste et détaille les containers (par défaut ceux lancés)

- `-a` : liste tous les containers (même arrêtés)
- `-f` ou `--filter` pour filtrer par nom, label, exited, ...
- de nombreuses options existent pour afficher + ou - d'informations
- `-q` retourne seulement l'id (ex: sert pour concaténer le résultat à d'autres commandes linux)

docker logs

Récupère les logs d'un container

- `docker logs [options] container id`
- `docker logs [options] container name`

docker exec

Lance une commande dans un container démarré

- `ex. : docker exec -ti <container> /bin/sh`
 - lance un processus de shell
 - le retourne
 - nous laisse interagir dans un terminal dans le container
- cette commande n'est pas relancée si le container est relancé

docker start / stop

La commande `docker start` lance un ou plusieurs containers arrêtés

La commande `docker stop` arrête un ou plusieurs containers lancés

docker rm

Supprime un ou plusieurs containers

- peut être supprimé par l'id ou le nom du container
- un container lancé ne peut pas être supprimé
 - pour forcer la suppression, utiliser l'option `-f`

docker rmi

Supprime une ou plusieurs images

- aucun container ne doit utiliser l'image (sinon erreur)
- sinon passer le paramètre `-f` pour forcer (les containers utilisant l'image seront orphelins, mais pourront continuer à tourner ou à être démarrés)

docker pull

Télécharge les images du docker hub ou d'un registre privé

- est exécuté lorsque vous lancez `docker run` et que l'image n'est pas en local
- par défaut, le tag latest est utilisé. Il est possible de spécifier le tag en ajoutant `:tag` (ex. `esgi:2.0`)
- télécharge l'image du docker hub par défaut

docker login / logout

Pour pouvoir travailler avec le hub (répertoire public d'images), vous devez créer un compte sur `hub.docker.com`

- l'authentification est obligatoire (pour les repos privées également)
- les informations d'authentification sont nécessaires pour la commande `docker search`

docker commit

Cette commande permet de créer une nouvelle image à partir de modifications effectuées depuis un shell interactif depuis une autre image

- une des manières de créer une image (l'autre sera en utilisant un `Dockerfile`)
- par défaut, le container est arrêté durant le commit (`--pause=true`)
- vous pouvez fournir le nom de la nouvelle image et du repo après le nom/id du container à utiliser

docker push

Envoi (partage) une image de votre hôte vers le docker hub ou un repo privé.

L'image doit respecter une certaine nomenclature de nom et tag décrits dans la documentation :

<https://docs.docker.com/engine/reference/commandline/tag/>

docker search

Recherche une image sur le docker hub

- l'option `-f / --filter-value` permet d'appliquer un filtre sur le nombre d'étoiles, si l'image est officielle ou si elle correspond à des builds automatiques

docker update

Permet d'actualiser la configuration d'un ou plusieurs containers

- fonctionne même si le container est lancé

TP

Maintenant à vous de jouer

Dans un premier terminal (terminal 1 / dans container)

- récupérer `alpine`
- vérifier que l'image `alpine` est bien présente
- démarrer un container avec un shell de manière interactive avec l'image `alpine`
 - lister les processus en cours (2 processus)

TP

Dans un nouveau terminal (terminal 2 / sur host)

- lister les containers lancés
- ajouter un nouveau processus (`ping google.fr`) au container (attention il vous faut le nom du container)

Dans votre container (terminal 1 / dans container)

- lister de nouveau les processus en cours (3 processus)
- créer un fichier avec votre nom

TP

Dans le terminal 2 / sur host

- créer une nouvelle image à partir du container que vous avez modifié (vous avez ajouté un fichier dedans lors de la précédente étape) et nommer la `alpine-dm` (attention à votre nom d'utilisateur)
- se connecter au docker hub
- envoyer votre image sur le hub
 - vérifier sur hub.docker.com que votre image est présente

TP

Dans le terminal 2 / sur host

- arrêter et retirer votre container
- retirer l'image alpine et votre `alpine-dm` de la liste des images sur votre ordinateur
- démarrer un nouveau container `alpine-dm` depuis l'image du hub
 - lancer un shell pour entrer dans le container et vérifier que votre fichier est présent

TP 2

Autre TP : Créer un environnement Apache + PHP rapidement

- Sur votre ordinateur, créer un dossier avec un fichier `index.php` avec un `phpinfo()`

Utiliser l'image `php:apache` (tag `apache`)

- Partager le dossier entre votre ordi et le container (`/var/www/html`)
- Publier le port `80` du serveur web sur le port `8080` de votre ordi
- Donner le nom `alpine_apache` au container

Quelle commande docker et quels paramètres doivent être utilisés?

Création d'une image

Création d'une image

Une des premières méthodes pour créer une image est :

- de récupérer une image (`docker pull`)
- de démarrer un container (`docker run`)
- d'effectuer des modifications (installation de software, création de fichiers,)
- de commiter ces modifications pour créer une nouvelle image à partir de ce qui se trouve sur le container (`docker commit`)

Historique image

```
docker history [OPTIONS] image
```

IMAGE	CREATED	CREATED BY	SIZE
21e77433a11d	3 days ago	/bin/sh -c #(nop) CMD ["/bin/bash" "/start.s	0 B
3b0379305784	3 days ago	/bin/sh -c chmod 755 /start.sh	94 B
278b27bc3cf7	3 days ago	/bin/sh -c #(nop) ADD file:a84b2099c064d81b8f	94 B
6b8ce0e3cd14	3 days ago	/bin/sh -c #(nop) ADD file:5744380367ce40b106	1.607 kB
cf8d5979dcc7	3 days ago	/bin/sh -c adduser -D -u 1000 docker	5.158 kB
e494de88a292	3 days ago	/bin/sh -c curl -sS https://getcomposer.org/i	1.706 MB
e04904444ef7	3 days ago	/bin/sh -c ln -s /usr/bin/php7 /usr/bin/php	13 B
c771b94cafc6	3 days ago	/bin/sh -c apk --no-cache --update --reposito	103.1 MB
28b26f729781	4 days ago	/bin/sh -c apk update && apk upgrade && apk	7.599 MB
7d23b3ca3463	7 days ago	/bin/sh -c #(nop) ADD file:fd71807f3b22f7f51f	4.799 MB

Dockerfile

Docker peut créer des images automatiquement à partir d'instructions dans un fichier **Dockerfile**

- permet de spécifier un ensemble de commandes (ex.: installation de packages)
- `docker build` se sert de ce fichier pour automatiser le build de l'image

Dockerfile

La commande docker build permet de créer une image à partir d'un fichier dockerfile

- `docker build -t <user-dockerhub>/<image-name>:<tag> /path/`
- l'utilisateur du hub n'est pas obligatoire, vous pouvez seulement passer le nom de l'image
- le tag par défaut sera latest et n'est pas obligatoire
- **Attention, ne pas fournir un contexte (/path/) dans lequel il y a bcp de fichiers et de dossiers car il va tous les parcourir (pensez à faire un dossier différent pour chaque dockerfile)**

Dockerfile

Ce fichier d'instructions doit respecter des bonnes pratiques

- être le plus éphémère possible
- n'installer que le nécessaire
- un container = un processus
- limiter le nombre de couches
- trier les arguments multi-lignes par ordre alphanumérique

Trier les arguments multi-lignes

```
RUN apk --no-cache --update --repository=http://dl-4.alpinelinux.org/alpine/edge/community add \  
curl \  
nginx \  
php7 \  
php7-ctype \  
php7-curl \  
php7-fpm \  
php7-iconv \  
php7-intl \  
php7-json \  
php7-mbstring \  
php7-mcrypt \  
php7-opcache \  
php7-openssl \  
php7-pdo_mysql \  
php7-phar \  
php7-session \  
php7-xml \  
php7-xsl \  
supervisor && \  
chown -R nginx:www-data /var/lib/nginx && \  
ln -sf /dev/stdout /var/log/nginx/access.log && \  
ln -sf /dev/stderr /var/log/nginx/error.log
```

Dockerfile - FROM

FROM est la première instruction d'un Dockerfile

- toutes les images dérive d'une autre image ou d'une image de base (ex.: debian, alpine)

Exemples:

```
FROM alpine:3.17
```

```
FROM debian:wheezy
```

Dockerfile

Les 3 instructions `RUN`, `CMD` et `ENTRYPOINT` peuvent être définies selon une forme `exec` ou `shell`

Exemples:

- `RUN apt-get install php-fpm`
- `CMD echo "Hello world"`
- `ENTRYPOINT echo "Hello world"`

Dockerfile - RUN

Pour exécuter une commande (ou suite de commandes) dans le container, il faut utiliser l'instruction `RUN`

- `RUN /bin/bash -c "commande param1"`
- `RUN commande param1`

Dockerfile - RUN

```
Building alpine-php7-fpm-nginx
```

```
Step 1 : FROM alpine
```

```
----> 7d23b3ca3463
```

```
Step 2 : RUN apk update && apk upgrade && apk add ca-certificates bash && rm -rf /var/cache/apk/*
```

```
----> Using cache
```

```
----> 28b26f729781
```

1. Une image alpine existe déjà (et est disponible avec docker images)
2. La suite de commandes définie dans ce `RUN` a déjà été exécutée sur une base alpine et a déjà été mise en cache. Lors du build, Docker récupère l'image correspondante (`28b26f729781`) à cette instruction sur la base de l'image alpine et ne rejoue donc pas la commande.

Dockerfile - CMD

L'instruction `CMD` spécifie la commande qui sera exécutée lors du démarrage du container

- ne peut être utilisée qu'une seule fois dans le `Dockerfile`
- si plusieurs instructions `CMD` sont présentes, la dernière est utilisée
- si au lancement du container on précise une commande, celle-ci écrasera l'instruction `CMD`
- l'instruction `CMD` indique la commande qui sera exécutée par défaut si rien n'est précisée au lancement

Dockerfile - ENTRYPOINT

L'instruction `ENTRYPOINT` a le comportement que `CMD` mais:

- à l'inverse de `CMD`, les commandes ne sont pas écrasées par celle de la ligne de commande

Dockerfile - EXPOSE

L'instruction `EXPOSE` permet d'indiquer à docker que le container écoute sur les ports indiqués

- **attention : ne rend pas accessible par défaut les ports depuis l'hôte**
- l'option `-p` de la commande `docker run` doit être utilisé pour publier un range de port (`-p 35500:22`) ou `-P` pour publier tous les ports exposés
- un numéro de port exposé dans le container peut être un numéro de port différent accessible sur l'hôte (ex.: exposé sur le 22, accessible sur le 34500 depuis l'hôte)

Dockerfile - COPY

L'instruction `COPY` copie des fichiers ou des dossiers à l'intérieur du container

- les fichiers sources ne peuvent pas être en dehors du contexte de build (donc pas de `../`)
- il est possible d'utiliser des wilcards et matchings (`*`, `?`, `...`) pour les noms de source (ex.: `file*`)
- info : tout est créé avec l'id et gid 0
- info : la destination doit finir par un `/` si c'est un dossier, ou que plusieurs fichiers vont y être copiés

Dockerfile - Rappel

Rappel pour le TP

- `docker build -t <user-dockerhub>/<image-name>:tag /path/`
 - utiliser l'option `-t` si vous avez l'intention de partager votre image sur le hub docker
- `docker run [options] image` pour créer un container
 - `-v dossier_local:dossier_container` pour partager un dossier entre l'hôte et le container
 - `-p port_hote:port_container` pour publier des ports
 - `-e` pour spécifier des variables d'environnements nécessaires au container

TP - créer une image sshd

- créez un fichier **Dockerfile** dans un dossier de votre projet
- démarrez votre image d'alpine
- installez ssh
 - **info** : pour installer un package sur alpine `apk add --update --no-cache package_name1 package_name2 ...`
- générer les clés nécessaires à ssh
 - **commande** : `ssh-keygen -A`
- autoriser la connexion par mot de passe et changer le mot de passe root
 - **info** : `sed -i s/#PermitRootLogin.*/PermitRootLogin\ yes/
/etc/ssh/sshd_config && echo "root:password" | chpasswd`

TP - créer une image sshd

- exposez le port 22 pour pouvoir se connecter en ssh au serveur
- lancez le serveur sshd au démarrage du container
- buildez votre image
 - info : n'oubliez pas de lui donner un nom (afin d'éviter une image `<none>`), ce sera plus simple pour la lancer
- créez un container à partir de cette image et tenter de vous connecter à l'utilisateur root
 - info : n'oubliez pas de publier le port dans votre commande run pour pouvoir vous connecter sur le port exposé au sein du container (exemple 1222)

Docker-compose

Docker orchestration

- Orchestrer la distribution d'applications avec Docker
 - Docker Machine
 - Docker Swarm
 - Docker Compose
 - (Kubernetes - CNCF)

docker-compose.yml

- Le fichier docker-compose.yml peut contenir plusieurs services
 - un service peut se baser sur une image du hub (`image`) ou une image à construire à partir d'un `Dockerfile` (`build`)
 - spécification des dossiers partagés (`volumes`)
 - spécification des ports à publier (`ports`)
 - de la commande sur laquelle se lancera le container (`command`)
 - spécifier un nom de container (`container_name`)
 - linker un container à un autre (`depends_on`)
 -

docker-compose

- `docker-compose` permet de définir et lancer plusieurs services
- permet d'automatiser la commande `docker run` et les autres commandes `docker`
 - Ex: `docker-compose start`, `stop`, `logs`, `down`, ...
- `docker-compose.yml` - **version 3**

docker-compose.yml

```
version: "3.1"
services:
  redis:
    image: redis:alpine
    container_name: esgi-redis

  postgres:
    image: postgres:9.6-alpine
    container_name: esgi-postgres
    working_dir: /application
    volumes:
      - ./application
    environment:
      - POSTGRES_USER=root
      - POSTGRES_PASSWORD=root
      - POSTGRES_DB=mybase

  webserver:
    image: nginx:alpine
    container_name: esgi-webserver
    working_dir: /application
    volumes:
      - ./application
      - ./phpdocker/nginx/nginx.conf:/etc/nginx/conf.d/default.conf
    ports:
      - "10000:80"

  php-fpm:
    build: phpdocker/php-fpm
    container_name: esgi-php-fpm
    working_dir: /application
    volumes:
      - ./application
      - ./phpdocker/php-fpm/php-ini-overrides.ini:/etc/php/7.1/fpm/conf.d/99-overrides.ini
```

docker-compose.yml

- Docker compose fournit
 - commande **docker compose build** pour builder les images de services pour lesquelles il a été spécifié un dockerfile (donc pas nécessaire si tous les services utilisent une image du hub)
 - commande **docker compose up** pour démarrer tous les containers
 - l'option -d permet de tous les lancer en background et de rendre la main dans le terminal
 - commande **docker compose down** permet d'arrêter et de supprimer tous les containers créés par le up

TP 1

Créer votre propre `docker-compose.yml` et connectez-y 3 services

- mariadb (mariadb:10.9.3) -> volume
- phpmyadmin (phpmyadmin:5.2.0) -> connexion à mariadb
- adminer (adminer:4.8.1) -> connexion à mariadb

TP 2

Créer un nouveau docker-compose.yml avec:

- un service web nginx
- un service php-fpm (php:fpm)
- un service composer (<https://github.com/guzzle/guzzle>)
- un service npm (dayjs)

<https://jsonplaceholder.typicode.com/>

TP 3

Créer un nouveau docker-compose.yml avec:

- un service web nginx
- un service php-fpm
- un service postgres
- un service adminer

Objectif: Pouvoir se connecter à la bdd postgres depuis PHP (utiliser PDO)