Docker

INITIATION

Sommaire

- Introduction
- La plateforme Docker
- Les containers avec Docker
- Les images Docker
- Registry
- Stockage
- Docker Compose
- Network
- Sécurité
- Gestion des logs
- Mise en pratique
- Swarm
- kubernetes

Introduction

Introduction

- De nombreux logiciels
  - Packagés dans des images
  - Disponibles sur dockerhub
  - Utilisables immédiatement

Introduction

- Exemples d'utilisation
  - docker container run -it python
  - docker container run -it ruby
  - docker container run -it -p 27017:27017 mongo

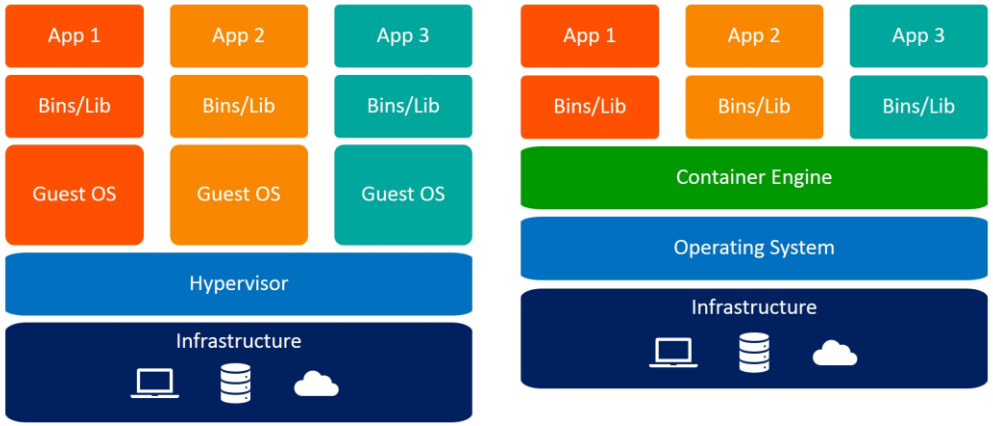
Les containers linux

- Un processus !
- Isolé des autres processus
- Avec sa propre vision du système sur lequel il tourne (namespace)
- Limité dans des ressources qu'il peut utiliser (Control Groups)
- Partage le Kernel de la machine hôte avec les autres containers

Les containers linux : Namespaces

- Technologie linux pour isoler un processus
- Les namespaces limitent ce qu'un container peut voir
- Différents namespaces :
  - Pid : isolation de l'espace de processus
  - Net : donne une stack réseau privée
  - Mount : system de fichiers privée
  - Uts : nom du host
  - Ipc : isole les coms interprocessus
  - User :mapping des uid/gid entre l'hôte et les containers

Les containers linux : VMs / Containers

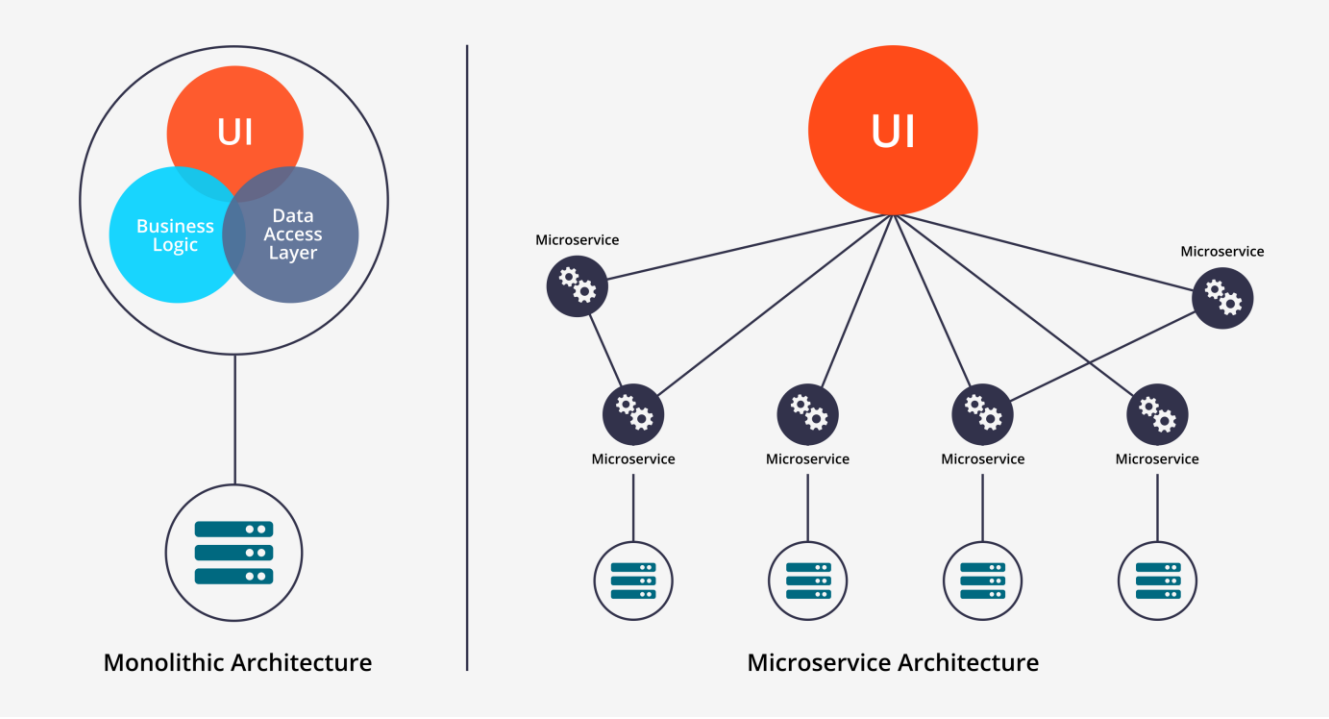


Virtual Machines

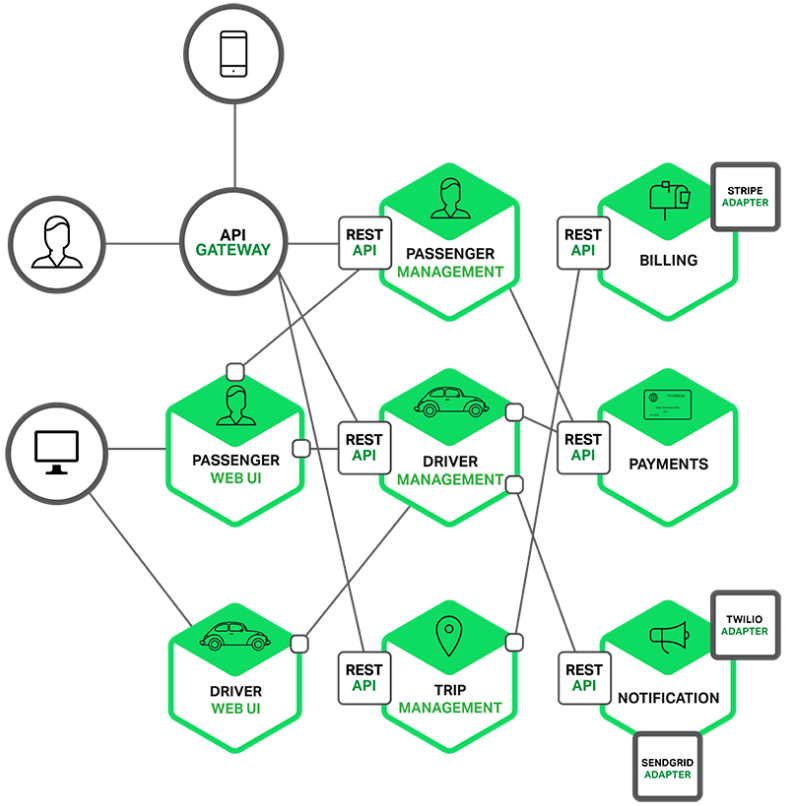
Containers

- VMs
  - Nécessite un hyperviseur
  - Chaque VM a son OS
  - Overhead RAM / CPU
- Container
  - Processus
  - Partage le Kernel de la machine hôte

Architecture microservice



Architecture microservice



Architecture microservice

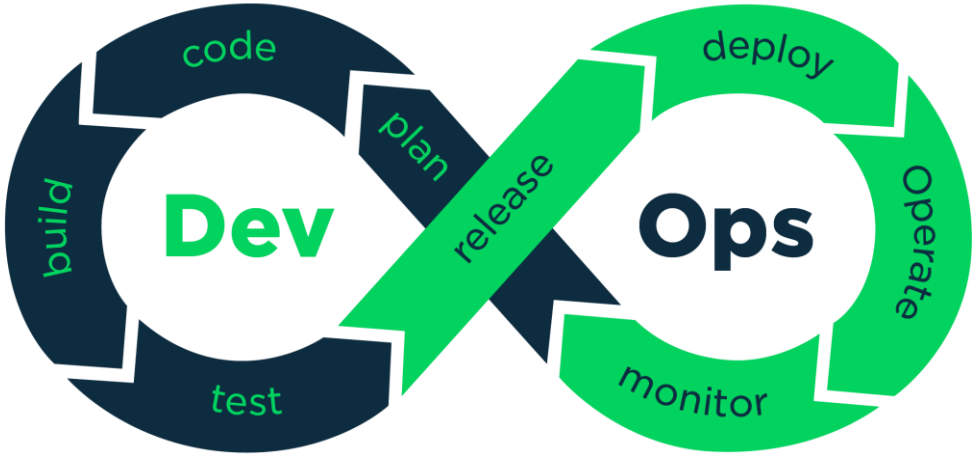
- Pros :
  - Découpage de l'application en processus (services) indépendants
  - Chacun a sa propre responsabilité métier
  - Équipe dédiée pour chaque service
  - Plus de liberté de choix dans le langage
  - Mise à jour et scaling horizontal
  - Containers très adapté pour les microservices

Architecture microservice

- Cons :
  - Nécessite des interfaces bien définies
  - Focus sur les tests d'integration
  - Déplace la compléxité dans l'orchestration de l'application globale

Devops

- Un objectif : minimiser le temps de livraison d'une fonctionnalité
- Déploiements réguliers
- Mise en place de tests
- Amélioration continue
- Automatisation des processus
  - Provisioning / configuration
  - CI / CD
  - …



La plateforme Docker

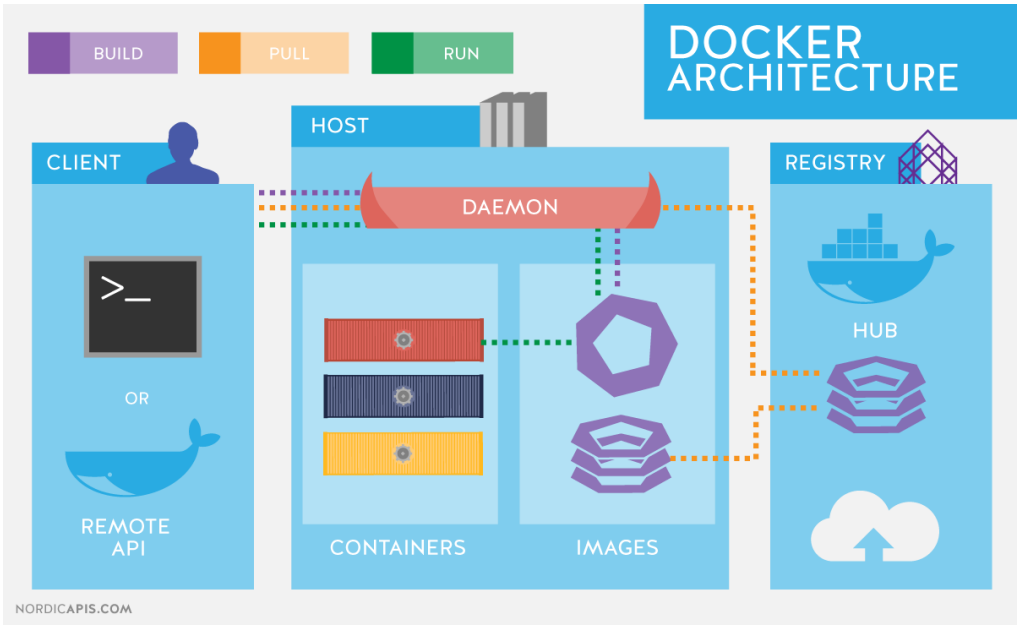
La plateforme Docker

Présentation

- Une plateforme de référence pour la construction, la distribution et le déploiement d'applications dans des containers
- Agnostique du langage *(indépendant du langage)* et de la stack applicative
- Orchestration intégrée
- Assure la scalabilité de l'application
- Accélère la mise en place de pipeline de déploiement automatiques etla fréquence des releases

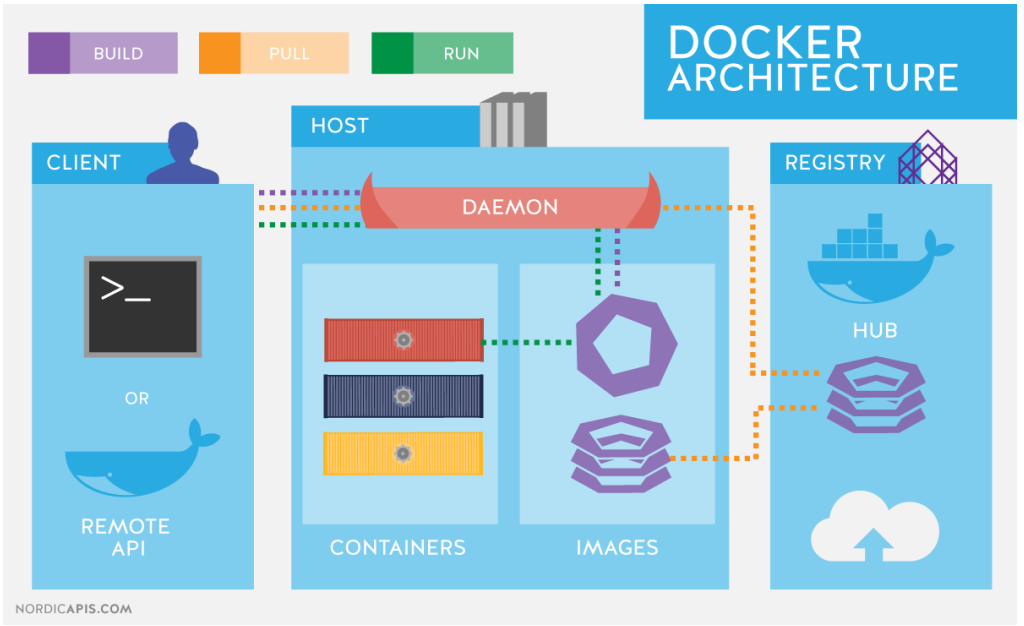
Architecture client / serveur

- Le client docker communique avec le deamon dockerd via une API REST
- **Côté serveur**
  - Processus dockerd
    - Gestion des images, networks, volumes, clusters..
    - Délègue la gestion des containers à containerd
  - Expose une API
  - Écoute sur la socket unix /var/run/docker.sock par défaut
  - Peut être configuré pour écouter sur une socket tcp



Architecture client / serveur

- Le client docker communique avec le deamon dockerd via une API REST
- **Côté client**
  - Client docker
  - Binaire développé en Go
  - Installé en même temps que dockerd
  - Communique avec le daemon local par défaut via /var/run/docker.sock
  - Peut être configuré pour communiquer avec un daemon distant (modification de DOCKER_HOST)



Les concepts essentiels

- Docker facilite l'utilisation des **containeurs** linux
- Un container est lancé à partir d'une image
- Une image contient toute une application et toutes ses dépendances
- Un fichier Dockerfile est utilisé pour la création d'une image
- Une image est distribuée via un registry (dockerhub par defaut)

Docker : installation

- https://docs.docker.com/get-docker/
- https://multipass.run/
  - $ multipass launch -n docker
  - $ multipass info docker
  - $ multipass list
  - $ multipass shell docker
  - ubuntu@docker:~$ curl -sSL https://get.docker.com | sh
  - ubuntu@docker:~$ sudo usermod -aG docker ubuntu
  - ubuntu@docker:~$ docker version

Les containers avec Docker

Sommaire

- Création d'un container
- Mode intéractif
- Foreground vs background
- Publication d'un port
- Bind-mount
- Limitation des ressources
- Des options utiles
- Les commandes de base
- Des alias utiles

Création d'un container

- Un container ⇔ un processus
- Lancement d'un processus dans un context d'isolation
- Création à partir d'une image
  - Système de fichier / package complet d'une application
  - Disponible dans un registry (ex: dockerhub ou en local)
- docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG...]
- https://docs.docker.com/engine/reference/run/

Création d'un container

- Hello world
  - docker container run hello-world
- Mode interactif
  - Utilisé pour lancer un shell dans un container
  - *-t* alloue un pseudo TTY, *-i* garde STDIN ouvert
  - docker container run -it **alpine**

Création d'un container

- Foreground vs background
  - Container lancé en Foreground par défaut
  - Option *-d* pour le lancer en background
  - L'identifiant du container est retourné

```
ubuntu@docker:~$ docker container run -itd alpine
2ec5a721b4ae22c845be72bdd00dc73226951bc5321652bed8c44aaf95914a5d
```

Création d'un container

- Publication d'un port
  - Permet de rendre le processus d'un container accessible depuis l'extérieur
  - Port d'écoute du processus mappé sur la machine hote
  - Allocation statique : -p HOST_PORT:CONTAINER_PORT
  - Allocation dynamique de l'ensemble des ports : -p
  - Conflit si plusieurs containers utilisent le même port de la machine hôte
  - Ex : docker container run -d -p 8080:80 nginx
- En mode admin
  - https://download.sysinternals.com/files/PSTools.zip
  - & $env:USERPROFILE\Downloads\PSTools\PsExec.exe -s $env:VBOX_MSI_INSTALL_PATH\VBoxManage.exe controlvm "**docker**" natpf1 "myservice,tcp,,8080,,8080"

Bind-Mount

- Répertoire ou fichier de la machine hôte monté dans un container
- À la création d'un container avec l'option -v ou --mount
  - docker container run -v HOST_PATH:CONATAINER_PATH
  - docker container run --mount type=bind,src=HOST_PATH,dst=CONTAINER_PATH...
- Cas d'usage
  - En développement : montage du code source dans un container

Limitation des ressources

- Pas de limite par defaut : RAM, CPU, I/O
- Nécessite d'imposer des limites pour ne pas impacter les autres processus
  - https://www.thorsten-hans.com/docker-container-cpu-limits-explained
    - # 20 seconds limit of 1 CPU
    - docker run -d --rm --cpus 1 progrium/stress -c 8 -t 20s

Des options utiles

- # Spécification
- docker container run -d -name debug alpine sleep 10000

- # Suppression du container quand il est stoppé
- docker container run --rm --name debug alpine sleep 10000

- # redémarage automatique
- docker container run --name api --restar=on-failure lucy/api

Les commandes de base

run	Création d'un container
ls	Liste des conatiner
inspect	Détail d'un container
logs	Visualisations des logs
exec	Lancement d'un processus dans un container existant
stop	Arrêt d'un processus
rm	Suppression d'un container

Les commandes de base

run	Création d'un container
ls	Liste des container
inspect	Détail d'un container
logs	Visualisations des logs
exec	Lancement d'un processus dans un container existant
stop	Arrêt d'un processus
rm	Suppression d'un container

[Exercice] Les commandes de bases

Exercice 1 : Hello from Alpine

- Le but de ce premier exercice est de lancer des containers basés sur l'image **alpine**

1. Lancez un container basé sur **alpine** en lui fournissant la command **echo hello**
2. Quelles sont les étapes effectuées par le docker daemon ?
3. Lancez un container basé sur alpine sans lui spécifier de commande. Qu'observez-vous ?

[Exercice] Les commandes de bases

Exercice 2 : shell intéractif

Le but de cet exercice est lancer des containers en mode intéractif

1. Lancez un container basé sur alpine en mode interactif sans lui spécifier de commande
2. Que s'est-il passé ?
3. Quelle est la commande par défaut d'un container basé sur alpine ?
4. Naviguez dans le système de fichiers
5. Utilisez le gestionnaire de package d'alpine (apk) pour ajouter un package

[Exercice] Les commandes de bases

Exercice 2 : shell intéractif

Le but de cet exercice est lancer des containers en mode intéractif

1. Lancez un container basé sur alpine en mode interactif sans lui spécifier de commande
2. Que s'est-il passé ?
3. Quelle est la commande par défaut d'un container basé sur alpine ?
4. Naviguez dans le système de fichiers
5. Utilisez le gestionnaire de package d'alpine (apk) pour ajouter un package

[Exercice] Les commandes de bases

Exercice 3 : foreground / background

Le but de cet exercice est de créer des containers en foreground et en background

1. Lancez un container basé sur alpine en lui spécifiant la commande ping 8.8.8.8
2. Arrêter le container avec CTRL-C

- Le container est t-il toujours en cours d'exécution ?
- Note: vous pouvez utiliser la commande docker ps que nous détaillerons dans l'une des prochaines lectures), et qui permet de lister les containers qui tournent sur la machine.

[Exercice] Les commandes de bases

Exercice 3 : foreground / background (suite)

3. Lancez un container en mode interactif en lui spécifiant la commande ping 8.8.8.8
4. Arrêter le container avec CTRL-P CTRL-Q

Le container est t-il toujours en cours d'exécution ?

5. Lancez un container en background, toujours en lui spécifiant la commande ping 8.8.8.8

Le container est t-il toujours en cours d'exécution ?

[Exercice] Les commandes de bases

Exercice 4 : publication de port

Le but de cet exercice est de créer un container en exposant un port sur la machine hôte

1. Lancez un container basé sur nginx et publiez le port 80 du container sur le port 8080 de l'hôte
2. Vérifiez depuis votre navigateur que la page par défaut de nginx est servie sur http://localhost:8080
3. Lancez un second container en publiant le même port

Qu'observez-vous ?

[Exercice] Les commandes de bases

Exercice 5 : liste des containers

Le but de cet exercice est de montrer les différentes options pour lister les containers du système

1. Listez les containers en cours d'exécution

Est ce que tous les containers que vous avez créés sont listés ?

2. Utilisez l'option -a pour voir également les containers qui ont été stoppés
3. Utilisez l'option -q pour ne lister que les IDs des containers (en cours d'exécution ou stoppés)

[Exercice] Les commandes de bases

Exercice 6 : inspection d'un container

Le but de cet exercice est l'inspection d'un container

1. Lancez, en background, un nouveau container basé sur nginx:1.14 en publiant le port 80 du container sur le port 3000 de la machine host.

Notez l'identifiant du container retourné par la commande précédente.

2. Inspectez le container en utilisant son identifiant
3. En utilisant le format Go template, récupérez le nom et l'IP du container
4. Manipuler les Go template pour récupérer d'autres information

[Exercice] Les commandes de bases

Exercice 7 : exec dans un container

Le but de cet exercice est de montrer comment lancer un processus dans un container existant

1. Lancez un container en background, basé sur l'image **alpine**. Spécifiez la commande ping 8.8.8.8 et le nom ping avec l'option **–name**
2. Observez les logs du container en utilisant l'ID retourné par la commande précédente ou bien le nom du container

Quittez la commande de logs avec CTRL-C

3. Lancez un shell sh, en mode interactif, dans le container précédent
4. Listez les processus du container

Qu'observez vous par rapport aux identifiants des processus ?

[Exercice] Les commandes de bases

- **Exercice 8 : cleanup**

Le but de cet exercice est de stopper et de supprimer les containers existants

1. Listez tous les containers (actifs et inactifs)
2. Stoppez tous les containers encore actifs en fournissant la liste des IDs à la commande stop
3. Vérifiez qu'il n'y a plus de containers actifs
4. Listez les containers arrêtés
5. Supprimez tous les containers
6. Vérifiez qu'il n'y a plus de containers

Les images docker

- Définition
- Contenu d'une image
- Dockerfile
- Création d'une image
- Contexte de build
- Les commandes de base

Définition

- Un template pour instancier des containers
- Contient une application et l'ensemble des ses dépendances
- Portable sur n'importe quel environnement ou tourne Docker
- Composée d'une ou de plusieurs layers
  - Chacune contient un système de fichiers et des méta data
- Distribuée via un **Registry** (ex: Docker Hub)

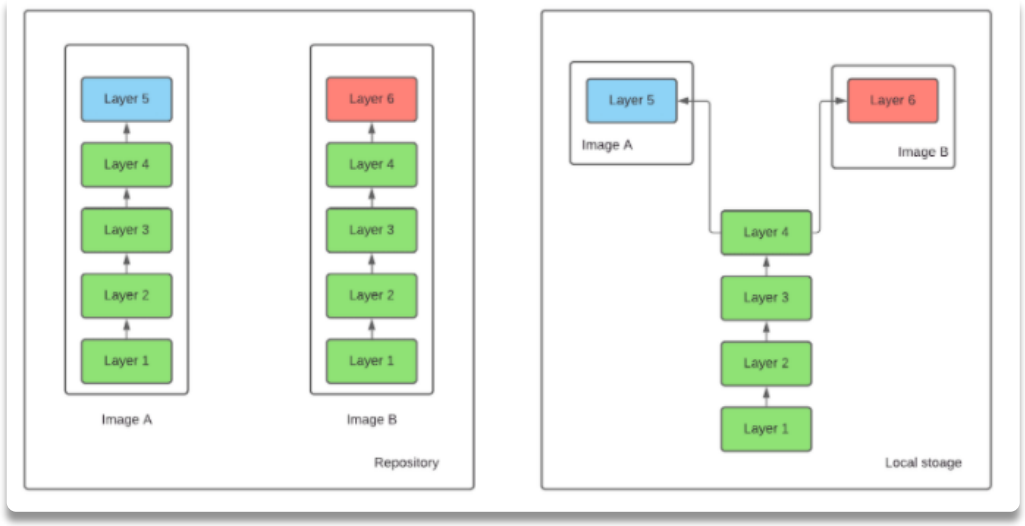
Contenu d'une image

- une image Docker est une collection ordonnée de changements d'un système de fichier et des paramètres d'exécution correspondant à son utilisation à l'exécution.
- Un ensemble atomique de changements sur le système de fichier est une couche. Une image Docker peut être vue comme une pile de couches dont chaque couche dépend de tous les précédents.
- Chaque couche représente donc un ensemble de changement que l'on fait au système de ficher de base

```
❯ docker pull postgres
Using default tag: latest
latest: Pulling from library/postgres
d121f8d1c412: Already exists
9f045f1653de: Pull complete
fa0c0f0a5534: Pull complete
54e26c2eb3f1: Pull complete
cede939c738e: Pull complete
69f99b2ba105: Pull complete
218ae2bec541: Pull complete
70a48a74e7cf: Pull complete
c0159b3d9418: Pull complete
353f31fdef75: Pull complete
03d73272c393: Pull complete
8f89a54571bf: Pull complete
4885714928b5: Pull complete
3060b8f258ec: Pull complete
Digest: sha256:0171a93d62342d2ab2461069609175674d2a1809a1ad7ce9ba141e2c09db1156
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
```

Contenu d'une image

- Chaque couche représente donc un ensemble de changement que l'on fait au système de ficher de base.
- Ces changements sont commis par les instructions présentes dans le Dockerfile de l'image. Lorsque l'on récupère une image Docker depuis un répertoire, l'image Docker se télécharge couche par couche.
- Cette décomposition en couche des images Docker permet de partager les différents couches entre celles-ci et d'économiser de l'espace de stockage et en volume de téléchargement



Dockerfile

- Fichier texte
- Série d'instructions pour construire le système de fichier d'une image
- Flow standard
  - Ajout d'une image de base
  - Ajout des dépendances
  - Ajout du code applicatif
  - Définition de la commande à lancer
- Build de l'image

Dockerfile : instructions principales

FROM	Image de base
ENV	Définition de variables d'environnement
RUN	Exécution d'une image, construction du filesystem de l'image
COPY / ADD	Copie de ressources depuis la machine hôte vers le container
EXPOSE	Expose un port de l'application
HEALTHCHECK	Vérifie l'état de santé de l'application
VOLUME	Définition d'un volume pour la gestion des données
WORKDIR	Définition du répertoire de travail
USER	Utilisateur auquel appartient le processus du container
ENTRYPOINT	Définie la commande exécutée au lancement du container
CMD	

Dockerfile : FROM

- Définit l'image de base
- Différentes possibilités
  - Image d'un OS (Alpine, CentOS, Ubuntu, …)
  - Serveur applicatif
  - Environnement d'exécution
- scratch (image particulière)
  - Pour construire une image de base
  - Pour construire une image minimale

Dockerfile : ENV

- Définition de variables d'environnement
- Valeur utilisée dans les instructions suivantes du build
- Dans l'environnement des containers créés à partir de l'image

```
FROM nginx:1.14.0

ENV path /usr/share/nginx/html/


WORKDIR ${path}

COPY . $path
```

Dockerfile : COPY / ADD

- Copie des fichiers et répertoires dans le système de fichiers de l'image
- Engendre la création d'une nouvelle layer (une partie de son système de fichier)
- Otpion --*chown* attribuer des droits lors de la copie
- ADD
  - Permet de spécifier une URL pour la source
  - Unpack un fichier tar.gz
- Privilégier l'utilisation de COPY (maitrise de la copie)

Dockerfile : RUN

- Exécute une commande dans une nouvelle layer
- 2 formats
  - Shell : lancé dans le un shell avec '/bin/sh –c' par défaut
    - RUN apt-get update -y && apt-get install vim
  - Exec : non lancé dans un shell
    - RUN [« /bin/bash« , "-c", "echo hello"]

Dockerfile : EXPOSE

- Information des ports utilisés par l'application
- Peut être modifié au lancement du container
  - Option -p CONTAINER_PORT
  - Option -p HOST_PORT:CONTAINER_PORT
  - Option -P

Dockerfile : VOLUME

- Découple les données du cycle de vie d'un container
- Gestion des données en dehors de l'union file-system
- Par défaut création d'un répertoire sur la machine hôte
- Initialisation du volume avec des données présentes dans l'image
  - `VOLUME /data/db /data/configdb`

Dockerfile : USER

- Username ou uid / gid utilisés pour lancer le processus du container
- Intérêt : ne pas lancer le processus du container en root
  - Comportement par défaut
  - Root dans le container ⇔ root sur la machine hôte
- Utilisé par les instructuion RUN, CMD, ENTRYPOINT qui suivent
- Utilisateur parfois changé dynamiquement au lancement du container

Dockerfile : ENTRYPOINT / CMD

- Définition de la commande exécutée dans le container
- ENTRYPOINT : binaire de l'application
- CMD : arguments par défaut
- Concaténation de ENTRYPOINT et CMD
- 2 formats possibles
  - Shell, ex : /bin/ping localhost
  - Exec, ex: ["ping", "localhost"]

Création d'images

- À partir d'un Dockerfile
  - *docker image build [OPTIONS] PATH | URL | -*
  - des options courantes
    - -f: spécifie le fichier à utiliser pour la construction (Dockerfile par défaut)
    - --tag / -t : spécifie le nom de l'image ([registry/]user/repository:tag)
    - --label: ajout de métadonnées à l'image

Création d'images

```
FROM php:8-apache
COPY . /var/www/html/
```

```
docker image build –t phpreq:1.0 .
```

```
docker container run -p 8888:80 phpreq:1.0
```

Création d'une image à partir d'un container

- **Enoncé**

1. Lancez une container basé sur une image *alpine:3.8*, en mode interactif, et en lui donnant le nom *c1*
2. Lancez la commande *curl google.com*

- Qu'observez-vous ?

3. Installez *curl* à l'aide du gestionnaire de package *apk*
4. Quittez le container avec CTRL-P CTRL-Q (pour ne pas killer le processus de PID 1)
5. Créez une image, nommée *curly*, à partir du container *c1*

- Utilisez pour cela la commande *commit* (*docker commit --help* pour voir le fonctionnement de cette commande)

6. Lancez un shell interactif dans un container basée sur l'image *curly* et vérifiez que *curl* est présent

Création d'une image à partir d'un Dockerfile

- **Enoncé**

1. Développez un serveur HTTP qui expose le endpoint */ping* sur le port 80 et répond par PONG.
2. Créez le fichier Dockerfile qui servira à construire l'image de l'application. Ce fichier devra décrire les actions suivantes
   - image de base: alpine:3.10
   - installation du runtime du langage choisi
   - installation des dépendances de l'application
   - copie du code applicatif
   - exposition du port d'écoute de l'application
   - spécification de la commande à exécuter pour lancer le serveur
3. Construire l'image en la taguant *pong:v1.0*
4. Lancez un container basé sur cette image en publiant le port 80 sur le port 8080 de la machine hôte
5. Tester l'application
6. Supprimez le container