

# Conteneurs



# Conteneurs



## Les conteneurs

- ▶ Technologie popularisée par Docker ( < 10 ans)

## Qu'est-ce qu'un conteneur ?

- ▶ Un conteneur permet d'embarquer
  - Une application et
  - Ses dépendances
- ▶ Puis de l'exécuter dans un environnement
  - « virtualisé » et
  - « isolé »
- ▶ Basé sur des technologies linux comme
  - chroot, lxc, cgroups

## Historique

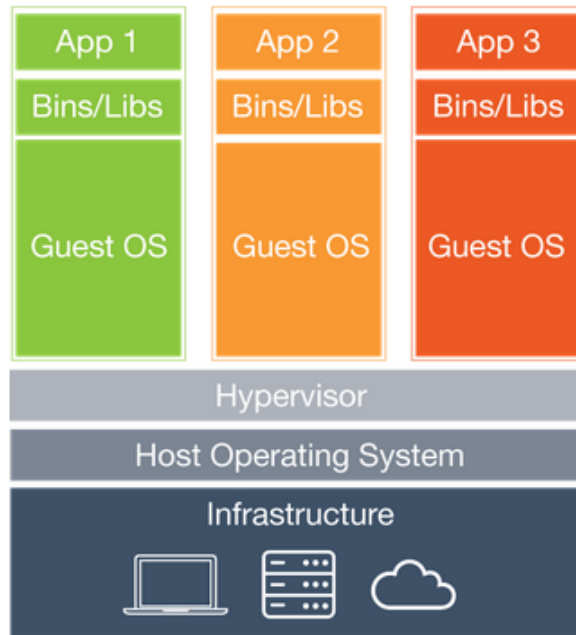
- ▶ 2010/2011
  - ➔ Création dotCloud par Solomon Hykes and Sebastien Pahl
  - ➔ Développement de docker comme projet interne de dotCloud par
    - Solomon Hykes (2010/2011) avec
    - Andrea Luzzardi et Francois-Xavier Bourlet
- ▶ 2013
  - ➔ Mars 2013 – 1<sup>ère</sup> distribution sous forme de projet open-source
  - ➔ Mai 2013 – docker registry (repository public pour images docker)
  - ➔ Septembre 2013 – alliance avec Red Hat (OpenShift)
  - ➔ Octobre 2013 – dotCloud ➔ Docker Inc

## Historique

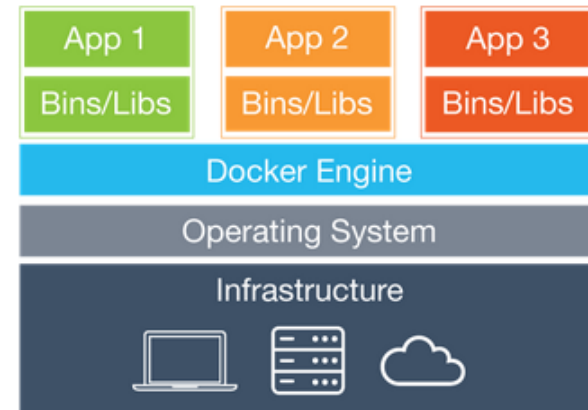
- ▶ 2015
  - ➔ Juin 2015 – création de OCI (Open Container Initiative)
- ▶ 2016
  - ➔ Docker donne containerd à CNCF (Cloud Native Computing Foundation)
- ▶ 2017
  - ➔ Mars 2017 – création de Docker Enterprise
  - ➔ Avril 2017 – Moby Project and LinuxKit (projets open sources)
- ▶ 2019
  - ➔ Novembre 2019 : Mirantis achète Docker Enterprise
  - ➔ Docker se recentre sur Docker Desktop et les outils pour développeurs

# Docker

## Container vs Machine virtuelle



Virtual Machines



Containers

## Technologies

- ▶ Virtualisation légère (chroot / process isolation / etc.)
  - ➔ OS-level virtualization : pas nouveau
    - Chroot (1979)
    - FreeBSD jail (2000)
    - OpenVz (1999)
    - Linux-Vserver (2003)
    - LXC (LinuX Container / cgroups) (2008)
- ▶ Système de fichiers en oignon (images)
  - ➔ Pas nouveau : utilisé déjà dans les live-cd
  - ➔ Image Docker : séquence de couches
    - chaque couche est « immutable » (non modifiable)
    - partage de couches entre images
    - extensible : on peut étendre des images en ajoutant des couches
- ▶ Docker registry
  - ➔ Dépôt d'images sur internet
    - Gratuit pour les projets open-source
    - ➔ Beaucoup d'images disponibles...

## Fonctionnement

- ▶ Quand on crée un conteneur et qu'on le démarre
  - Le conteneur est créé à partir d'une image
  - Une couche est ajoutée (gestion des modifications du système de fichiers)
  - Il est mis dans réseau à part (par défaut nattu vers l'extérieur)
  - La commande de lancement du conteneur est exécutée
  
- ▶ Le tout se fait dans un environnement isolé de l'hôte local
  - Système de fichier isolé
  - Réseau séparé
  - Processus du conteneurs isolés de ceux de l'hôte



## Usage

- ▶ Développement: Contrôle de l'environnement de dev
  - ➔ Gestion des versions des logiciels utilisés
  - ➔ Besoin d'un environnement spécifique(clang, node 14, java17, ...)
    - ➔ utilisation d'un conteneur
      - Pas d'installation à faire
      - Ni de désinstallation
      - Pas de conflit ou d'interaction possible avec les autres logiciels installés
  - ➔ Automatisation des build
- ▶ Test : automatisation et reproductibilité des tests
  - ➔ Contrôle fin des environnements de test
  - ➔ Chaque test peut se faire dans un nouveau conteneur
    - C'est facile de créer et supprimer des conteneurs
  - ➔ Facile de tester dans différents environnements
- ▶ Déploiement
  - ➔ On peut utiliser les conteneurs créés dans les phases de test/dev
    - Environnement de production est proche de l'environnement de test
    - K8s : orchestration des conteneurs





## Commandes

### ► Exécution d'un conteneur

`docker run [paramètres] <image> [commande]`

#### → Exemples :

`docker run hello-world`

`docker run alpine ls -al`

`docker run -it debian bash`

`docker run -d -p 8080:80 httpd`

`docker run -d --rm --name tomee -p 8080:8080 tomee:11-jre-8.0.0-M3-plume`

`docker run -d --name bd -e POSTGRES_PASSWORD=password -d postgres`

#### → Paramètres :

- `-i` ou `--interactive` : attacher stdin – permet d'entrer des commandes
- `-t` ou `--tty` : alloue un pseudo tty
- `-t` ou `--detach` : exécution en arrière plan
- `--name` : attribut un nom au conteneur
- `-p` ou `--publish` : port forwarding
- `-v` ou `--volume` : montage de volume
- `--rm` : supprime le conteneur après son arrêt
- `--net` ou `--network` : associe à un réseau spécifique



## commandes

### ► Images

- `docker images` : liste les images locales
- `docker pull <image>` : téléchargement d'une image d'un dépôt
  - `docker pull debian:stable`
  - `docker pull nginx:latest`
- `docker push <image>` : envoi d'une image vers un dépôt (registry)
  - Dépôt par défaut : Docker Hub
- `docker rmi <images>` : suppression d'images
- `docker image rm <images>` : suppression d'images
- `docker tag <image:tag> <image>` : association d'un tag à une image

## commandes

### ► Conteneurs

#### → Lancement/Creation/Arrêt d'un conteneur

- docker run [paramètres] <image> [commande]
- docker container create [paramètres] <image> [commande]
- docker start <conteneur> : démarrage conteneur existant
- docker stop <conteneur> > : arrêt conteneur
- docker kill <conteneur> > : arrêt conteneur

#### → Exécution d'une commande sur un conteneur existant

- docker exec [paramètres] <image> [commande]

#### → Gestion des conteneurs

- docker ps : liste les conteneurs
- docker stats : info sur les conteneurs en cours d'exécution
- docker container rm <conteneur> : suppression d'un conteneur
- docker logs <conteneur> : affichage des logs d'un conteneur



## Commandes

### ► Réseau

#### → docker network

- docker network create ... : création nouveau réseau
- docker network ls : lister les réseaux docker
- docker network inspect <network> : inspection réseau
- docker network rm <network> : suppression réseau

### ► Systèmes de fichiers

#### → docker volume

- docker volume create... : création de volume
- docker volume ls : lister les volumes existants
- docker volume inspect <volume> : inspection de volume
- docker volume rm <volumes> : suppression des volumes

### ► Divers

#### → docker version : version docker

#### → docker login/logout : connexion/déconnexion registry docker

#### → docker build : construction d'images

# Docker

## Création d'images

- ▶ A partir d'un conteneur existant
  - ➔ `docker container commit...` : création d'une image à partir d'un conteneur
- ▶ A l'aide d'un dockerfile
  - ➔ Le dockerfile décrit toutes les étapes de la construction de l'image
    - Image de base
    - Commandes à exécuter pour construire le conteneur
      - Installation de paquets logiciels
      - Copie de fichiers
      - Déclaration de volumes
      - Ports à exporter
      - Commande de lancement du conteneur
      - Etc..
  - ➔ Chaque étape correspond à une couche de l'image
  - ➔ Commande utilisée pour la création
    - `docker build`

# Dockerfile

## Exemple de dockerfile

```
FROM gcc
WORKDIR /src/
COPY . /src/
RUN make hello
ENTRYPOINT [ "/src/hello" ]
CMD ["World"]
```

## Build

```
docker build --tag stalb/hello .
docker push stalb/hello
```

- ▶ Taille d l'image > 390 mo
  - On conserve un peu trop de trucs
    - Artéfacts de compilation
    - Image de base (gcc) contient des éléments inutile pour l'exécution de pg

# Dockerfile

## Multi-stage

```
FROM gcc as build
WORKDIR /src/
COPY . /src/
RUN make hello
```

```
FROM debian:stable-slim as slim
COPY --from=build /src/hello /bin/hello
ENTRYPOINT [ "/bin/hello" ]
CMD ["World"]
```

## Build

```
docker build --tag stalb/hello:slim .
docker push stalb/hello:slim
```

- Taille de l'image < 30 mo

# Dockerfile

## Multi-stage - en utilisant une image alpine

```
FROM alpine:3 as build
RUN apk update && apk add build-base
WORKDIR /src/
COPY . /src/
RUN make hello
```

```
FROM alpine:3
COPY --from=build /src/hello /bin/hello
ENTRYPOINT [ "/bin/hello" ]
CMD ["World"]
```

## Build

```
docker build --tag stalb/hello:alpine .
docker push stalb/hello:alpine
```

Taille de l'image < 3 mo



# Dockerfile

## Multi-stage - en gardant uniquement l'exécutable

```
FROM alpine:3 as build
RUN apk update && apk add build-base
WORKDIR /src/
COPY . /src/
RUN gcc -o hello -static hello.c
```

```
FROM scratch
COPY --from=build /src/hello /hello
ENTRYPOINT [ "/hello" ]
CMD ["World"]
```

## Build

```
docker build --tag stalb/hello:extra-small .
docker push stalb/hello:extra-small
```

Taille de l'image < 50 ko

# Orchestration

# Orchestration de conteneurs



## 1 conteneur c'est bien, plein c'est mieux

- ▶ Approche microservice
  - 1 fonctionnalité == 1 service
- ▶ Bonnes pratiques pour les containers :
  - 1 conteneur == 1 processus
  - Plutôt que d'avoir 1 gros conteneur avec tout dedans (BD, backend, etc..)
  - Décomposition en plusieurs conteneurs
    - 1 service == 1 conteneur
      - Base de données
      - Backend
      - Etc...
- ▶ Pb : si une application = plusieurs conteneurs
  - il faut pouvoir gérer des groupes de conteneurs
- ▶ Solution : orchestration de conteneurs

# Orchestration de conteneurs

## Orchestration de conteneurs :

- ▶ Kubernetes (K8s) – standard de fait
- ▶ Swarm – orchestrateur de Docker ( mais Swarm est mort !)
- ▶ Docker-compose (gestion de groupes de conteneurs sur 1 machine)

## K8s

- ▶ Développé au départ par Google (2014)
- ▶ Donné à la Cloud Native Computing Foundation (CNCF) en 2015
- ▶ Disponible en standard sur la plupart des plateformes cloud
  - GKE (Google)
  - AKS (Microsoft)
  - EKS (Amazon)
  - OVH
  - Scaleway

# Orchestration de conteneurs



## Docker compose :

- ▶ 1 seule machine
- ▶ Possibilités de gestion limitées
- ▶ Assez simple à utiliser
- ▶ Approche déclarative pour définir
  - ➔ Liste de conteneurs à démarrer simultanément
  - ➔ Configuration des conteneurs
    - Images à utiliser
    - Variables d'environnement
    - Volumes
    - Réseaux
    - Etc...

# Orchestration de conteneurs

## Docker compose : exemple de fichier docker-compose.yml

```
version: "3.7"
```

```
services:
```

```
  bd:
```

```
    image: postgres:12
```

```
    environment:
```

- "POSTGRES\_USER=admin"
- "POSTGRES\_PASSWORD=admin"

```
  volumes:
```

- "bd\_data:/var/lib/postgresql/data"

```
  networks:
```

```
    bdnet:
```

```
      ipv4_address: 172.22.0.10
```

```
  payara:
```

```
    image: payara/server-full
```

```
    ports:
```

- "8080:8080"

```
    networks:
```

```
      bdnet:
```

```
        ipv4_address: 172.22.0.40
```

```
    depends_on:
```

- bd

```
volumes:
```

```
  bd_data:
```

```
networks:
```

```
  bdnet:
```

```
    ipam:
```

```
      driver: default
```

```
      config:
```

- subnet: 172.22.0.0/24

# Containers et mvt dev/ops

# Docker

## Intérêt pour le développeur

- ▶ Contrôle de l'environnement de dev
- ▶ Automatisation des build
- ▶ Test : automatisation et reproductibilité des test
  - Contrôle fin des environnements de test
  - Chaque test peut se faire dans un nouveau conteneur
    - C'est facile de créer et supprimer des conteneurs
  - Facile des tester dans différents environnements
- ▶ Déploiement plus facile
  - Environnement de production est proche de l'environnement de test
  - K8s : orchestration des conteneurs



# Docker

## Intérêt pour le déploiement

- ▶ Test : automatisation et reproductibilité des tests
  - ➔ Contrôle fin des environnements de test
  - ➔ Facile des tester dans différents environnements
- ▶ Déploiement plus facile
  - ➔ Environnement de production est proche de l'environnement de test
  - ➔ K8s : orchestration des conteneurs
    - Supervision
    - Passage à l'échelle
    - Interopérabilité

## Problèmes potentiels

- ▶ Complexité K8s
- ▶ Nouveauté des environnements
  - ➔ Maîtrise
  - ➔ Sécurité

# Docker

## Sécurité

### ► Avantages

#### → Surface d'attaque plus faible

- Peu de services par container
- Isolation réseau
- Etc.

#### → Maj des images plus facile

- Build automatique avec les dockerfiles

#### → Système de fichier immutable

- On ne peut pas modifier facilement les couches d'une images
- ➔ En cas de compromission, on peut repartir de l'image de base

### ► Problèmes

#### → Nouveauté des environnements / Complexité K8s

#### → Isolation moindre que dans le cas de machines virtuelles

#### → Contrôle des images utilisées par les dev

#### → Gestion des risques de fuite de « secrets » dans images

#### → Le démon docker (et les containers) s'exécutent avec les droits « root »

# Docker

## Sécurité

### ► Gestion des images

- Il est possible d'héberger ses propres images dans son propre dépôt
- Il est possible de signer les images
  - Garantie la non modification des images utilisées.
- Il y a des outils d'analyse des images
  - Contrôles des couches (et des failles de sécurités connues)
  - Contrôle de l'absence de fuite de « secrets »
- Définir des bonnes pratique pour les dev (et les ops)
  - Choix et gestion des images
  - Construction des images

# Docker

## Outils alternatifs

- ▶ Podman / Buildah (Redhat)
  - Podman : exécution des containers
  - Buildah : construction des images
  - Fonctionnent dans l'espace utilisateur
- ▶ Rkt (CoreOS)
- ▶ Kaniko (Google)
  - Construction d'images
- ▶ Kata Containers (fondation OpenStack)
  - Exécution de containers dans des machines virtuelles « légères »