

DevOps

Les conteneurs

Thomas Ropars

`thomas.ropars@univ-grenoble-alpes.fr`

2018

Agenda

X as a service

Virtualisation et conteneurs

Docker

Utiliser docker

Conteneurs et DevOps

Agenda

X as a service

Virtualisation et conteneurs

Docker

Utiliser docker

Conteneurs et DevOps

Avant les nuages

Chaque entreprise gère ses propres ressources informatiques.

Des coûts importants:

- Achat de nouveau matériel
- Moyens humains pour administrer ces ressources
- Taux d'utilisation faible:
 - En moyenne, moins de 10% des ressources utilisées
 - Nécessaire pour être capable d'absorber les pics de charge.

La révolution dans les nuages

Cloud Computing

Le cloud computing (l'informatique dans les nuages) est un modèle dans lequel un fournisseur de service fournit un accès à des ressources informatiques au travers d'une connexion Internet.

On parle aussi d'**Utility Computing**. (Idée d'un fournisseur de service – on paye pour ce qu'on consomme)

Le **Cloud Computing** fait aussi référence à l'infrastructure logicielle et matérielle mise en place pour fournir ce service.

Émergence à la fin des années 2000

- Amazon Elastic Compute Cloud en 2006
- Microsoft Azure en 2010

Avantages/Inconvénients

Avantages

- Économique
 - Mutualisation des coûts
 - Facturation des ressources réellement utilisées
- Qualité de service
 - Ressources à jour
 - Haute disponibilité
- Élasticité
 - Possibilité de “démarrer petit”
 - Quantité de ressources accessibles potentiellement infini

Inconvénients

- Confidentialité
- Déploiement d'applications sur infrastructure distante?

Les modèles de service

Software-as-a-service (SaaS)

- Accès à un logiciel au travers d'une connexion à un serveur distant
- Accès libre ou sur abonnement
- Accès au travers d'un client (ex: navigateur web)
- Exemples: Service mail, Editeur en ligne, Github

Les modèles de service

Platform-as-a-service (PaaS)

- Le client déploie son application sur l'infrastructure cloud
- Le fournisseur fournit les briques logicielles de base (OS, base de donnée, etc)
 - ▶ Le client peut configurer ces briques logicielles selon ses besoins
- Le fournisseur maintient la plateforme d'exécution
 - ▶ Serveur
 - ▶ Stockage
 - ▶ Réseau
- Le fournisseur peut fournir des services en plus
 - ▶ Persistance
 - ▶ Haute disponibilité
 - ▶ Sécurité

Les modèles de service

Infrastructure-as-a-service (IaaS)

- Le fournisseur donne accès à des ressources informatiques
 - ▶ Calcul, stockage, réseau
 - ▶ Accès sous forme de machines virtuelles
 - ▶ Possibilité de réserver des ressources à grain très fin (1 cœur de calcul)
- Le client installe sa propre pile logicielle sur les ressources obtenues
 - ▶ Possibilité de déployer son propre système d'exploitation

Agenda

X as a service

Virtualisation et conteneurs

Docker

Utiliser docker

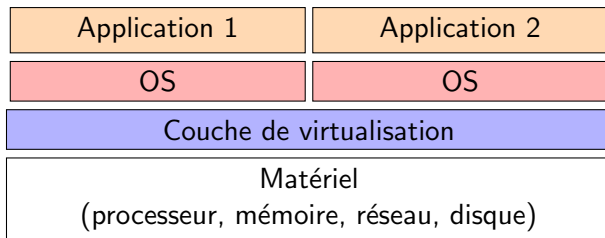
Conteneurs et DevOps

La virtualisation

Définition

Abstraire (virtualiser) le matériel à l'aide de couches logicielles

La virtualisation permet de donner l'illusion de plusieurs machines physiques.



La virtualisation est une des technologies clés sur laquelle se fonde le Cloud Computing

Un peu de vocabulaire

Une machine virtuelle (VM – Virtual Machine)

- Une mise en œuvre au niveau logiciel d'une machine réelle
- Permet d'exécuter un système d'exploitation complet

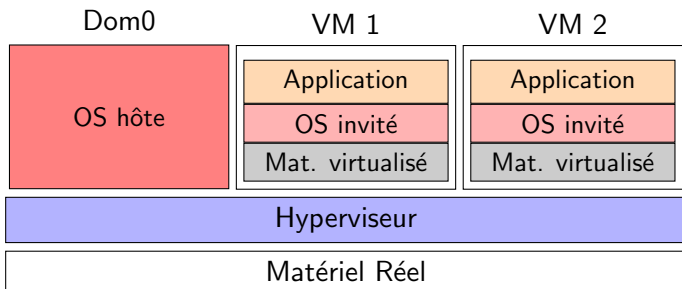
Un hyperviseur

- La brique logicielle qui orchestre l'exécution des machines virtuelles sur les machines physiques

Les techniques de virtualisation

Virtualisation de type I

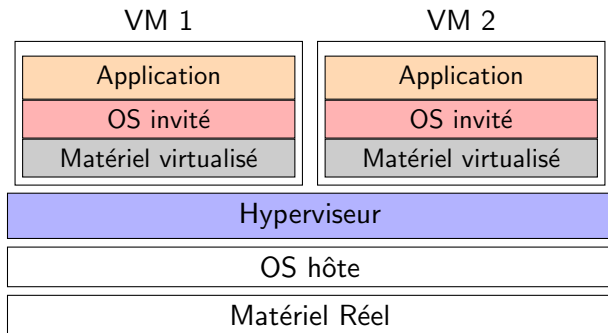
- L'hyperviseur de type I s'exécute directement au dessus du matériel
- L'hyperviseur est alors un noyau système spécialisé
- Exemple: Xen



Les techniques de virtualisation

Virtualisation de type II

- L'hyperviseur de type II est un logiciel qui s'exécute au dessus de l'OS hôte
- Exemple: VirtualBox, VMWare

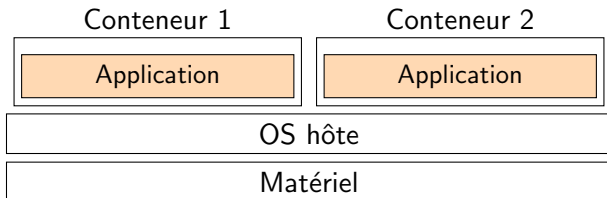


Intérêts de la virtualisation

- Portabilité sur différents matériels
- Isolation entre les utilisateurs
 - Sécurité
 - Performance
- Manipulations d'images de VMs
 - L'état complet de la VM sauvegardée dans un fichier (OS, applications, données)
 - Migration
 - Clonage
 - Configurer une seule fois pour de multiples exécutions (templates)

Les conteneurs

- Alternative à la virtualisation
- Environnement d'exécution isolé au dessus du système d'exploitation
 - ▶ Les conteneurs partagent le même noyau et une grande partie des services de l'OS hôte
 - ▶ *Virtualisation* de l'environnement d'exécution (espace de nommage, système de fichier isolé, quotas, etc)
 - ▶ Exemple: Linux LXC



Les avantages des conteneurs

- Meilleures performances
 - ▶ Accès direct au matériel
- Démarrage beaucoup plus rapide
 - ▶ Pas besoin de démarrer un système complet
- Images plus légères
 - ▶ Ne contient que les informations en lien avec l'application
 - ▶ Moins coûteux en terme d'espace de stockage
 - ▶ Plus rapide à transférer
- **Les machines virtuelles offrent de meilleures garanties en terme d'isolation et de sécurité**

Des infos en plus

Technologies de conteneurs (liste non exhaustive)

- Docker
- CoreOS Rocket (rkt)
- Linux LXC/LXD

Exemple d'utilisation

- Toutes les applications chez Google s'exécutent dans des conteneurs
- Google démarre plus de 2 milliards de conteneurs chaque semaine.

Agenda

X as a service

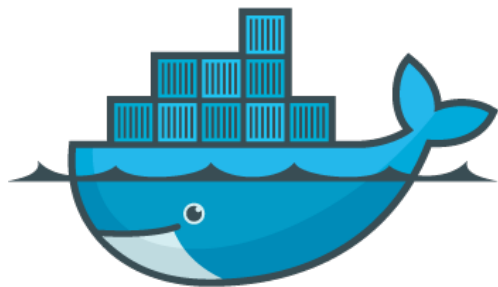
Virtualisation et conteneurs

Docker

Utiliser docker

Conteneurs et DevOps

Docker



docker

Analogie



Manipulation simplifiée d'un ensemble d'objets (ou d'applications)
grâce à une interface standardisée.

The Challenge

Slides by B. Golub

Multiplicity of Stacks



Static website

nginx 1.5 + modsecurity + openssl + bootstrap 2



Background workers

Python 3.0 + celery + pyredis + libcurl + ffmpeg + libopencl + nodejs + phantomjs



User DB

postgresql + pgv8 + v8



Queue

Redis + redis-sentinel



Analytics DB

hadoop + hive + thrift + OpenJDK



Web frontend

Ruby + Rails + sass + Unicorn



API endpoint

Python 2.7 + Flask + pyredis + celery + psycopg + postgresql-client



Multiplicity of hardware environments



Development VM



QA server

Customer Data Center



Public Cloud

Disaster recovery

Production Servers



Production Cluster



Contributor's laptop







Do services and apps interact appropriately?

Can I migrate smoothly and quickly?

The Matrix from Hell

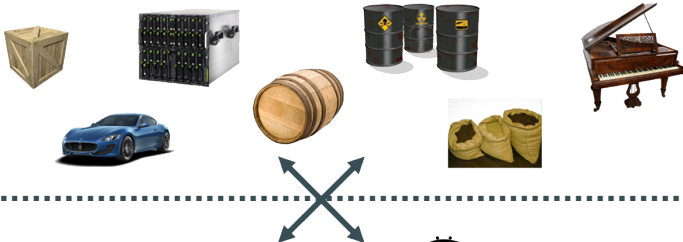
Slides by B. Golub

	Static website	?	?	?	?	?	?	?
	Web frontend	?	?	?	?	?	?	?
	Background workers	?	?	?	?	?	?	?
	User DB	?	?	?	?	?	?	?
	Analytics DB	?	?	?	?	?	?	?
	Queue	?	?	?	?	?	?	?
		Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
								

Cargo Transport Pre-1960

Slides by B. Golub

Multiplicity of Goods



Do I worry about
how goods interact
(e.g. coffee beans
next to spices)

Multiplicity of
methods for
transporting/storing



Can I transport quickly
and smoothly
(e.g. from boat to train
to truck)

Also a Matrix from Hell

Slides by B. Golub

	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
	?	?	?	?	?	?	?
							

Solution: Intermodal Shipping Container

Slides by B. Golub

Multiplicity of Goods



A standard container that is loaded with virtually any goods, and stays sealed until it reaches final delivery.



...in between, can be loaded and unloaded, stacked, transported efficiently over long distances, and transferred from one mode of transport to another

Multiplicity of methods for transporting/storing

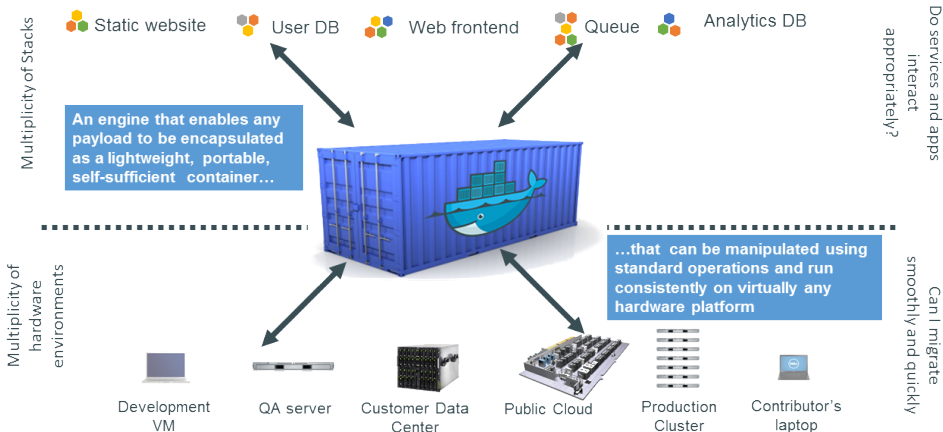


Do I worry about how goods interact (e.g. coffee beans next to spices)

Can I transport quickly and smoothly (e.g. from boat to train to truck)

Docker is a Container System for Code

Slides by B. Golub



Docker Eliminates the Matrix from Hell

Slides by B. Golub



Static website							
Web frontend							
Background workers							
User DB							
Analytics DB							
Queue							
	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers



Qu'est ce que Docker?

- Enrichissement de Linux LXC
 - ▶ Migration transparente entre machines
 - ▶ Docker développe maintenant sa propre solution (libcontainer)
- Docker engine
 - ▶ Un environnement d'exécution et un ensemble de services pour manipuler des conteneurs docker
 - ▶ [Une application client-serveur](#)
- Un *daemon* docker (le serveur)
 - ▶ Processus persistant qui gère les conteneurs sur une machine
- Un *client* docker
 - ▶ Interface à la ligne de commande pour communiquer avec un daemon dockeur
- Un registre d'images docker
 - ▶ Bibliothèque d'images disponibles

Les images Docker

Les images de machines virtuelles

- Sauvegarde de l'état de la VM (Mémoire, disques virtuels, etc) à un moment donné
- La VM redémarre dans l'état qui a été sauvegardé

Les images Docker

- Une copie d'une partie d'un système de fichier
- Pas de notion d'état¹

¹Parmi les évolutions récentes, possibilité de sauvegarder l'état (*checkpoint*) un conteneur

Les images Docker

UnionFS

- Un système de fichier pour Linux
- Crée un système de fichier cohérent à partir de fichiers/répertoires appartenant à des systèmes de fichiers différents

Un ensemble de couches

- Une image est composée d'un ensemble de couches ("layers")
- UnionFS est utilisé pour combiner ces couches

Les couches

Images de base

- Toute image est définie à partir d'une image de base
- Exemples: ubuntu:latest, ubuntu:14.04, opensuse:latest

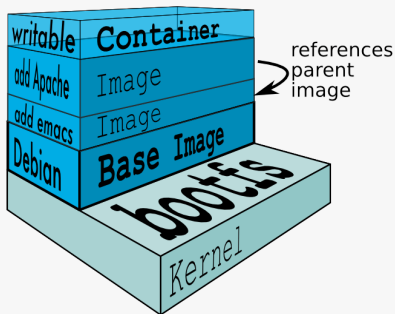
Relation avec le système d'exploitation hôte

- Une image n'inclut que les bibliothèques et services du système d'exploitation mentionné
- Le conteneur utilise le noyau du système hôte

Les couches

Les autres couches

- Une image docker est accessible seulement en lecture
- Modifier une image crée une nouvelle couche
- Toutes les modifications sont incluses dans cette nouvelle couche
- La nouvelle image inclut seulement les modifications (correspondant à la nouvelle couche)
 - Explique la faible empreinte de chaque image



Registre Docker

Principe

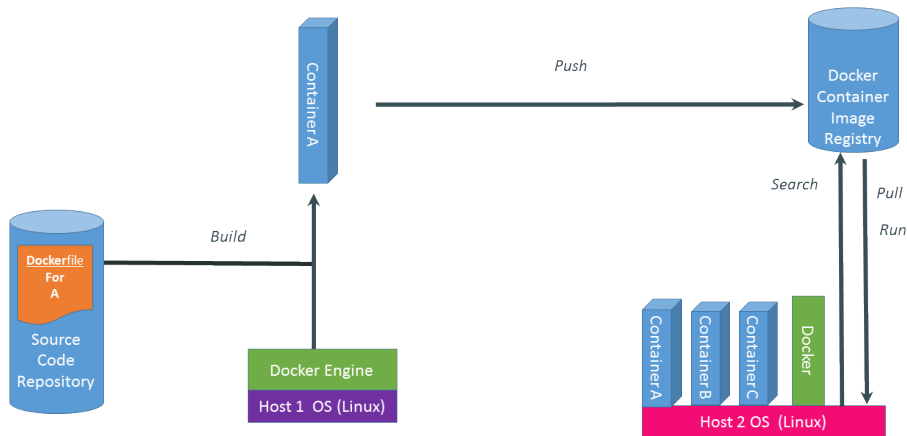
- Un serveur stockant des images docker
- Possibilité de récupérer des images depuis ce serveur (*pull*)
- Possibilité de publier de nouvelles images (*push*)

Docker Hub

- Dépôt publique d'images Docker

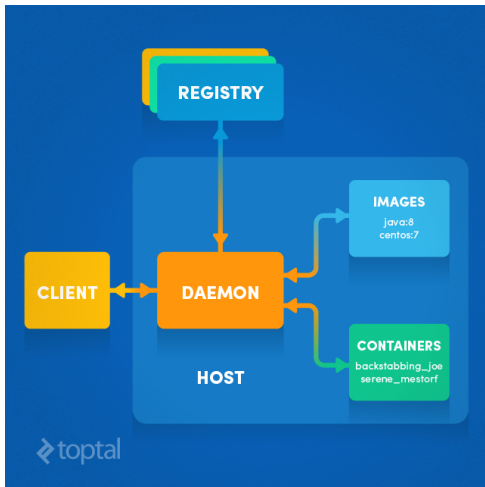
Basics of a Docker System

Slides by B. Golub



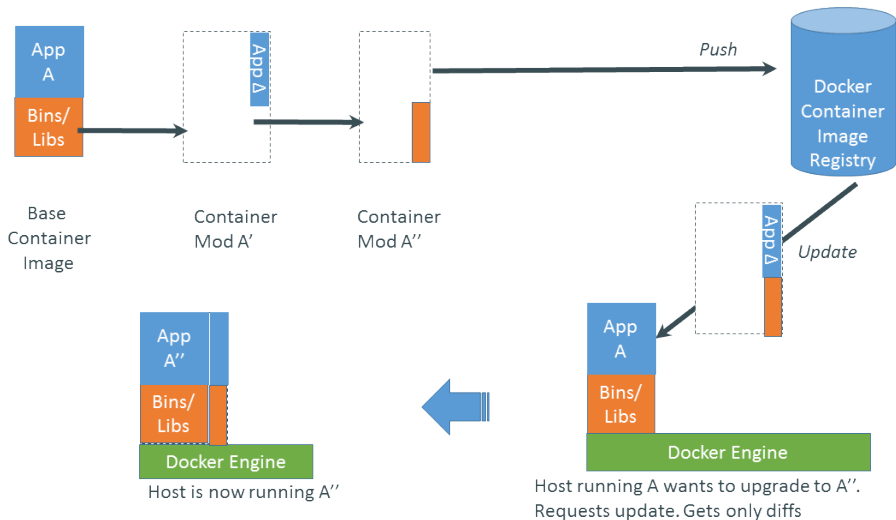
Architecture Docker

www.toptal.com/devops/getting-started-with-docker-simplifying-devops



Changes and Updates

Slides by B. Golub



Agenda

X as a service

Virtualisation et conteneurs

Docker

Utiliser docker

Conteneurs et DevOps

Warning

Docker est un outil puissant. Il existe un grand nombre de commandes. Pour un inventaire détaillé:

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

Projet évoluant rapidement

- Projet récent (première version en mars 2013)
- Projet open source avec de nombreux contributeurs (plus de 1700)
- <https://github.com/moby/moby>
 - ▶ Moby: Open source project created by Docker to work on work on container technologies.

Premiers pas

L'outil à la ligne de commande pour exécuter des commandes Docker:

```
$ docker
```

`docker run hello-world`

- `docker`: Nous voulons exécuter une commande docker
- `run`: Commande pour créer et exécuter un conteneur docker
- `hello-world`: Nom de l'image à charger dans le conteneur

Premiers pas

```
$ docker run hello-world
```

Que va-t-il se passer?

- L'image à charger est `hello-world:latest`
 - ▶ L'image identifiée avec le tag `latest`
- Vérification: est ce que l'image est présente localement?
- Sinon, télécharger l'image depuis Docker Hub
- Charger l'image dans le conteneur et exécuter

Dockerfile

Fichier contenant une suite d'instructions pour créer une image Docker (commande `docker build`).

Éléments de syntaxe

- **FROM**: Définit l'image à partir de laquelle la nouvelle image est créée
- **LABEL**: Associe des meta-données à la nouvelle image (par exemple, l'auteur de l'image)
- **RUN**: Définit une commande exécutée dans la couche au dessus de l'image courante
- **CMD**: Définit la commande exécutée au démarrage du conteneur

Dockerfile

Éléments de syntaxe

- **EXPOSE**: Informe docker que le conteneur va écouter sur le port réseau défini
- **COPY**: Copier un fichier/répertoire depuis le contexte de construction de l'image vers la nouvelle couche
 - ▶ La destination peut être un chemin absolu ou un chemin relatif depuis **WORKDIR**

Dockerfile

Notion de contexte

- Lors de la construction d'une nouvelle image, un contexte de construction de l'image est défini (par défaut le répertoire dans lequel on exécute la commande pour créer l'image)
- Tous les fichiers appartenant au contexte sont envoyés au daemon docker
- Seuls les fichiers appartenant au contexte peuvent être copiés vers la nouvelle image
- Bonne pratique: Créer un répertoire contenant le Dockerfile et seulement les fichiers dont vous avez besoin dans le contexte

Dockerfile

Exemple (source: [docker.com](https://docs.docker.com/engine/tutorials/dockerfiles/))

```
FROM docker/whalesay:latest
RUN apt-get -y update && apt-get install -y fortunes
CMD /usr/games/fortune -a | cowsay
```

- Récupère l'image whalesay dans le namespace docker depuis docker hub
 - ▶ le namespace correspond au nom de l'utilisateur sur docker hub
- Installe le logiciel fortunes
- Définit la commande à exécuter au sein du conteneur

Construire une image

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

- **docker build**: Crée une image à partir d'un fichier Dockerfile
 - ▶ `docker build -t docker-whale .`
 - ▶ Crée l'image `docker-whale` avec le contexte correspondant au répertoire courant (le `."` est nécessaire)
 - ▶ Par défaut, le Dockerfile est cherché à la racine du contexte
 - ▶ Option `-f` pour changer le chemin
 - ▶ Dans tous les cas le fichier doit se trouver dans le contexte

Construire une image

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

- **docker build**: Crée une image à partir d'un fichier Dockerfile
 - ▶ `docker build -t docker-whale .`
 - ▶ Crée l'image `docker-whale` avec le contexte correspondant au répertoire courant (le `."` est nécessaire)
 - ▶ Par défaut, le Dockerfile est cherché à la racine du contexte
 - ▶ Option `-f` pour changer le chemin
 - ▶ Dans tous les cas le fichier doit se trouver dans le contexte
- A propos du contexte:
 - ▶ Tous les fichiers du contexte sont envoyés vers le daemon docker
 - Ne pas donner `"/` comme contexte !!
 - ▶ Possibilité de définir un fichier `.dockerignore` pour exclure des fichiers du contexte à copier
- Build à partir d'une URL:
 - ▶ `docker build github.com/creack/docker-firefox`
 - ▶ Clone le dépôt et utilise le clone comme context
 - ▶ Un Dockerfile doit se trouver à la racine du dépôt

Manipuler les images

Quelques commandes

- `docker image COMMAND`
- `docker image ls`
 - ▶ liste des images existantes localement
 - ▶ Option `-a` permet d'afficher toutes les images locales
 - ▶ `docker images ubuntu`: liste des images nommées ubuntu
 - ▶ Les images `<none>: <none>` sont des images intermédiaires
 - Image "parent" d'une image locale
- `docker image rm`: Supprime une image

Dangling images

- Une image intermédiaire qui n'est plus référencée par aucune image
- `docker image prune`: supprime les *dangling* images

Démarrer une commande dans un nouveau conteneur

- **docker run**: Crée et démarre un conteneur
 - ▶ `docker run --name whale-test docker-whale`
 - ▶ Exécute la commande spécifiée par CMD
 - ▶ Plusieurs options (notamment pour limiter les ressources utilisées)
 - ▶ Option **-p** (port host:port container): associe un port de la machine locale avec un port du conteneur
`docker run -d -p 80:5000 training/webapp`
 - ▶ Option **-d**: démarre le conteneur en arrière plan
 - ▶ Options **-it**: Crée une session interactive avec le conteneur et ouvre un pseudo-terminal
`docker run --name test -it debian`

Manipuler des conteneurs

Commande parent

- `docker container COMMAND`

Quelques commandes

- `create`: Crée un conteneur
- `start`: Démarre un conteneur arrêté
- `stop/restart`: Arrête/redémarre un conteneur en cours d'exécution
- `rm/prune`: Supprime un conteneur arrêté
- `prune`: supprime tous les conteneurs arrêtés

Manipuler des conteneurs

Commande parent

- `docker container COMMAND`

Informations sur les conteneurs

- `ls`: Montre les conteneurs en cours d'exécution
- `logs`: Obtenir les logs d'un conteneur
- `stats`: Obtenir les informations sur la consommation de ressources d'un conteneur
- `top`: affiche la liste des processus du conteneur

Interactions avec un registre

https://docs.docker.com/engine/getstarted/step_six/

Publier une image sur Docker Hub

- Tagger l'image à publier:

```
docker tag 7d9495d03763  
mydockeraccount/docker-whale:latest
```

- ▶ "mydockeraccount": Mon login sur docker hub
- ▶ "7d9495d03763": L'identifiant de l'image

- Se connecter à docker hub:

```
docker login
```

- Publier l'image:

```
docker push mydockeraccount/docker-whale
```

- L'image peut maintenant être récupérée par d'autres:

```
docker pull mydockeraccount/docker-whale
```

Gérer les données dans Docker

<https://docs.docker.com/storage/>

On peut stocker les données manipulées par une application directement dans le conteneur.

Cependant, cela a des inconvénients:

Gérer les données dans Docker

<https://docs.docker.com/storage/>

On peut stocker les données manipulées par une application directement dans le conteneur.

Cependant, cela a des inconvénients:

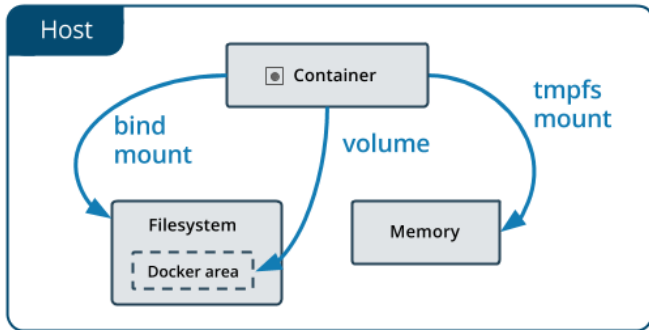
- Les données disparaissent avec le conteneur
- Les données sont difficilement accessibles depuis l'extérieur du conteneur
- C'est peu efficace en terme de performance (coût de UnionFS)

Gérer les données dans Docker

Les alternatives pour le stockage des données

- **Les volumes:** Stockage géré par Docker dans le système de fichier hôte
 - ▶ Partager des données entre plusieurs conteneurs
 - ▶ Stocker des données sur un support distant (ex: dans un cloud) en utilisant un *driver*
- **Les *bind mounts*:** Stockage à n'importe quel endroit du système de fichier hôte
 - ▶ Partager des fichiers de configuration avec le système hôte
 - ▶ Utiliser du code source présent sur la machine hôte
- **Les montages *tmpfs*:** Stockage en mémoire
 - ▶ Plus efficace qu'un volume lorsque les données n'ont pas besoin de persistance

Gérer les données dans Docker



Les volumes

<https://docs.docker.com/storage/volumes/>

Creation d'un volume

```
docker volume create my-vol
```

Utilisation d'un volume

- Option `--mount` pour la commande `docker run`
- Utilisation du volume `my-vol` avec comme répertoire destination au sein du conteneur `/app`.
 - ▶ `docker run -d --name devtest \`
`--mount source=my-vol,target=/app debian`

Commentaires:

- Le conteneur n'a pas besoin d'être créé à l'avance
- Si le répertoire destination contient déjà des données, elles sont copiées dans le conteneur
- L'option `readonly` peut être utiliser pour empêcher les données d'un conteneur d'être modifiées

Les bind mounts

- Option `--mount` pour la commande `docker run` avec l'option `type=bind`

```
$ docker run -d -it \  
  --name devtest \  
  --mount type=bind,source="$(pwd)"/code,target=/app \  
  debian:latex
```

Commentaires

- Si le répertoire `/app` existe déjà au sein du conteneur, son contenu est remplacé celui du répertoire de l'hôte qui est *bindé*.

Docker compose

<https://docs.docker.com/compose/>

Docker compose permet de définir et d'exécuter des applications multi-conteneurs.

Un fichier `docker-compose.yml` décrit une composition. Il permet de définir:

- Les services (conteneurs) à démarrer
- Les services peuvent être construits à partir d'une image existante ou d'un Dockerfile
- Les liens (links) entre les services
- La gestion des ports
- La gestion des volumes de données
- etc

La commande `docker-compose up` démarre la composition

Exemple de fichier docker-compose.yml

```
version: '3'
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - ./code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

Autre exemple de fichier docker-compose.yml

```
version: "3"
services:
  web:
    # replace username/repo:tag with your name and image details
    image: username/repo:tag
    deploy:
      replicas: 5
      resources:
        limits:
          cpus: "0.1"
          memory: 50M
      restart_policy:
        condition: on-failure
    ports:
      - "80:80"
    networks:
      - webnet
networks:
  webnet:
```

Dans un environnement distribué

Services permettant de déployer des applications multi-conteneurs sur plusieurs machines:

- Docker swarm
- Kubernetes (Google)

Services fournis (Kubernetes)

- Déploiement automatique des conteneurs
- Placement en fonction des besoins de ressources
- Monitoring et redémarrage automatique de conteneurs
- Équilibrage de charge entre instances d'un service
- etc

Agenda

X as a service

Virtualisation et conteneurs

Docker

Utiliser docker

Conteneurs et DevOps

Conteneurs et DevOps

L'émergence des technologies de conteneurs contribuent à l'essor de l'approche DevOps:

- Le même environnement d'exécution peut être utilisé pour le développement et la production
 - ▶ Réduit le temps nécessaire au passage en production
 - ▶ Les "opérateurs" peuvent donner un retour aux développeurs en terme d'environnement d'exécution
- Déployer une nouvelle version d'un logiciel est peu coûteux (livraison en continu)
- Générer un environnement de test devient très simple (intégration continue)

Microservices

Principe

- Remplacer les grosses applications monolithiques difficiles à maintenir
- Construire une application comme un ensemble de services utilisant un protocole simple pour communiquer (ex: API RESTful)
- Les services:
 - ▶ résolvent un problème particulier
 - ▶ peuvent être déployés de manière indépendante

Les architectures fondées sur des microservices tendent à se généraliser.

Conteneurs, DevOps et Microservices

Synergie entre ces 3 *approches*:

- Les conteneurs sont utiles à l'approche DevOps (voir précédemment)
- Les technologies de conteneurs permettent de déployer des architectures microservices plus simplement et efficacement
- Les architectures microservices simplifient la mise en place de l'approche DevOps
 - ▶ Développement indépendant de chaque service
 - ▶ Déploiement indépendant de chaque service

Références

Quelques liens:

- <https://docs.docker.com/engine/understanding-docker/>
- <https://github.com/wsargent/docker-cheat-sheet>

Autres sources de la présentation:

- Présentation de Ben Golub (CEO Docker)