R4.08 - Virtualisation

1 - Virtualisation

Terminologie :

- Hôte (Host) = la machine physique qui héberge les Invités
- Invité (Guest) = la machine logique (VM) hébergée sur l'Hôte
- Emulation = reproduction exacte d'un système dans un autre système (un ordinateur dans un ordinateur)

Virtualisation :

- 1 ordinateur physique (poste de travail, serveur)
- Une ou plusieurs VMs
- Un OS à installer par VM
- Allocation exclusive de RAM et CPU par VM
- Allocation exclusive d'une partie du disque hôte par VM
- Possibilité de répertoires partagés entre l'hôte et chacune des VMs.

2 - Conteneurisation

Comparable à la virtualisation :

- Une “sorte” de virtualisation mais en beaucoup plus léger.
- Une machine hôte (physique) qui accueille des conteneurs (sortes de “machines logiques”)
- Cloisonnement : environnement d’exécution séparé de l’hôte et des autres conteneurs

Des différences fondamentales :

- Pas d'émulation du matériel
- Pas d'OS (kernel) sur les invités (guests)
- Un cloisonnement par Name Spaces (Process, Mount, IPC, User et Network)
- Les conteneurs ne voient pas les processus de l'hôte
- L'hôte voit les processus des conteneurs !

Un conteneur = un processus du point de vue de l'hôte.

Avantages :

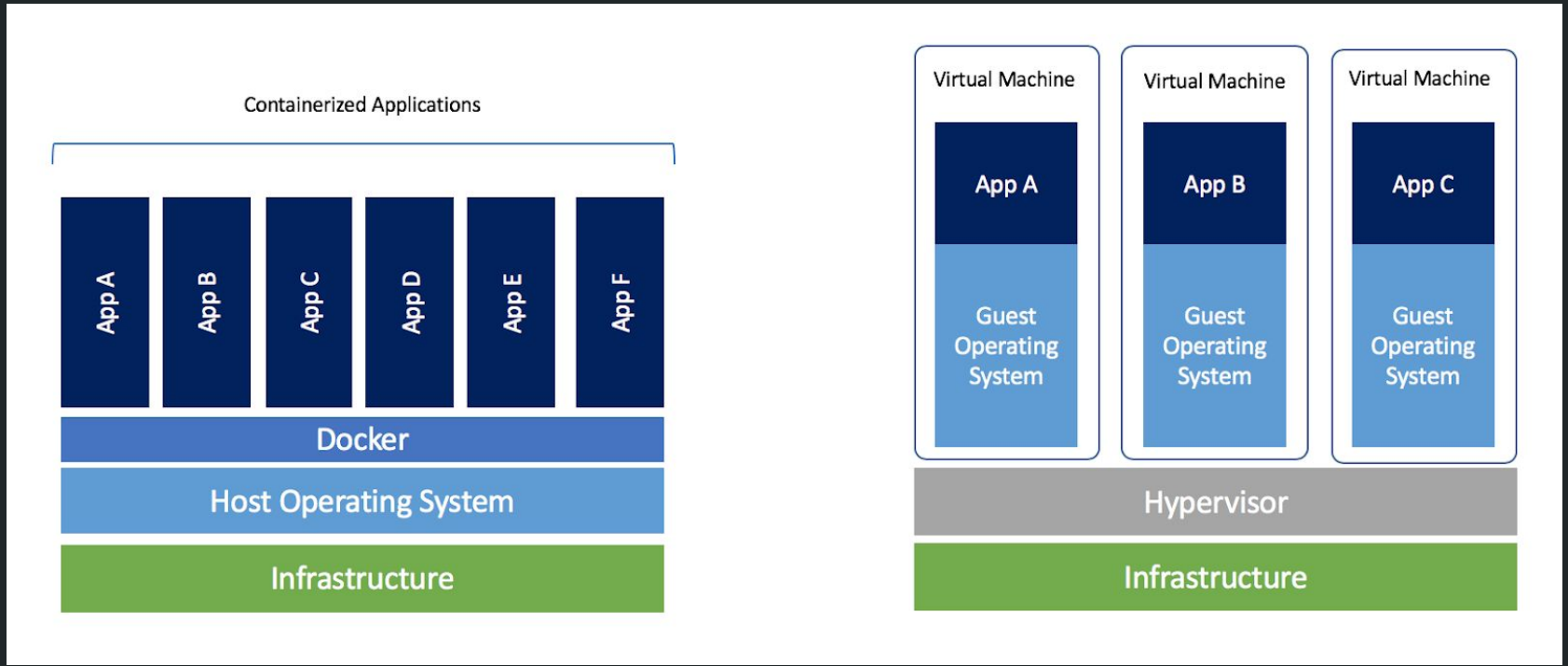
- Une archi extrêmement légère
- Pas de surcharge CPU (car un conteneur ~= le processus qui tourne dans le conteneur)
- Une consommation de ressources maîtrisée (pas d'allocation réservée en CPU, RAM, Disque)
- Un démarrage très rapide (quelques secondes)
- Pas de pollution de l'hôte (pas d'install. de packages)
- Elimination des incompatibilités (plusieurs versions d'un même package, exemple : les VM Java)

Inconvénients :

- Le même OS (kernel) pour tous les conteneurs = pas possible d'avoir des versions de noyau différentes
- Une gestion des droits plus complexe
- Nécessité d'avoir des droits admin pour certaines actions ou certaines configurations
- Pour Docker : uniquement Linux, mais des solutions existent pour Windows et macOS

3 - Docker

Un dessin vaut 1000 mots...



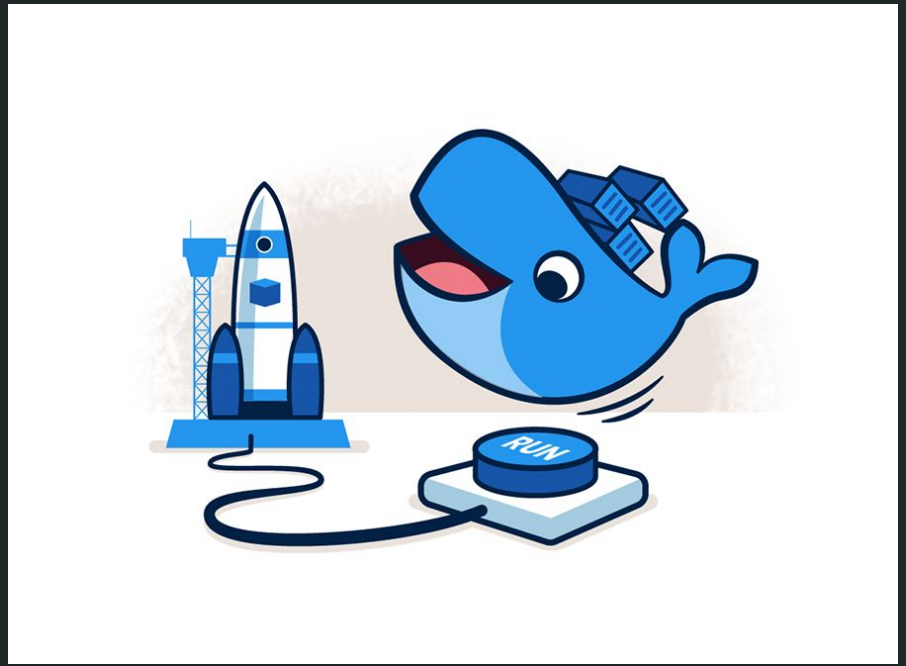
En standard, un conteneur c'est :

- Un unique processus (celui qui tourne dans le conteneur). Ou parfois plusieurs, par fork().
- Un noyau partagé avec l'hôte
- Une arborescence privée
- Un réseau privé local (un VPN local à l'hôte)
- Une vision cloisonnée des processus (pas de vue sur les processus du système) + remappage des PIDs
- Un remappage des Users (root dans conteneur n'est pas root dans l'hôte)

En option, un conteneur peut :

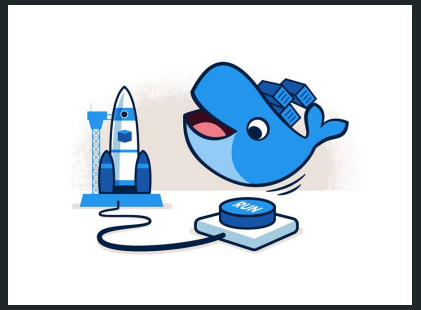
- Partager un bout de l'arbo de son FS avec son hôte
- Avoir un VPN partagé avec d'autres conteneurs explicitement ciblés
- Partager la même pile TCP/IP que l'hôte
- Avoir root dans conteneur = root hôte

Testons un peu tout ça...



Terminal 1 :

```
docker image pull r408_test1
docker container run --rm r408_test1 60
```



Terminal 2 :

```
ps -edf
```

Localisez le processus go, quel est son propriétaire ? quel est son PID ? Est-ce pareil que ce que le conteneur prétend (Terminal 1) ?

4 - Terminologie

Image

- Un modèle de départ pour créer l'arbo de fichiers des conteneurs.
- Construction par couches (Layers)
- Layers ~= Calques (papier ou à la "Photoshop")
- Un calque modifie la pile de calques du dessous
- Tous les calques sont en lecture seule
- En terminologie POO, on pourrait dire qu'une image est un peu comme une classe

Une image :

- Se base sur une autre image (ou image “scratch”)
- Apporte ses adaptations (par l’ajout de layers)
- Peut servir de base à d’autres images
- Layers en RO = immuables -> pas de risque de casser des images qui l’utilisent

Images publiques : Docker Hub (hub.docker.com)
Images privées : Hub privé ou machine perso/Serveur privé

Couche (Layer)

- Une étape dans la construction d'une image = 1 ligne du Dockerfile (vu plus tard)
- Généralement un layer = modification (ajout, changement, suppression) dans le File System
- C'est aussi une sorte d'image "intermédiaire"

Conteneur (container)

- Une instanciation d'une image. En POO, on pourrait dire qu'un conteneur est un peu comme un objet
- Utilise les layers de l'image (RO)
- Ajoute un layer supplémentaire (RW) = layer de travail du conteneur (contient les créa, modifs des objets issus des layers RO du FS -> Copy On Write)
- Tous les layers (RO) sont partagés entre images et conteneurs.

Un conteneur est volatile :

- Par défaut les données sont dans le conteneur
- Suppression conteneur = données perdues

Pour persister les données : les volumes (voir plus loin)

Hub (Dépôt)

- Images officielles : produites par Docker Inc.
- Images vérifiées : produites par les éditeurs (“Verified publishers”).
- Images publiques : tout le monde peut publier des images. Attention à la sécurité
- Dépôt officiel : hub.docker.com
- Dépôts tiers : publics ou privés (“Registries”)
- IUT : docker.iutlan.etu.univ-rennes1.fr

Par image :

- Historique de versions = tags
- Tag “latest” = Dernière version
- Conservation des versions
- Possibilité de cibler une version spécifique = garantie de stabilité pour l’utilisateur
- Accès aux sources de construction des images (voir Dockerfile)
- Souvent lié à un dépôt Gitlab/Github

Compte gratuit :

- 200 “images pulls” / jour
- Dépôt public illimité

Comptes payants :

- Limites beaucoup plus larges, voire illimitées
- Pour les entreprises avec gros besoins

Volumes

Données d'un conteneur = stockage dans conteneur
Conteneur supprimé = perte du stockage
Persistance = volumes
Hôte et Conteneur lisent et écrivent les mêmes fichiers

2 types :

- Volumes mappés
- Volumes managés

Volumes mappés (bind mounts) :

- Pratique pour partager des données présentes sur le FS de l'hôte (ex : un dossier html)
- Forte dépendance du File System hôte (droits etc)
- Performance moins bonnes

Volumes managés (managed volumes) :

- Possible à migrer entre plateformes (compatibilité)
- Meilleures performances

Networks

Par défaut un conteneur :

- voit son hôte , voit les autres conteneurs configurés aussi par défaut, accède au monde extérieur (Internet)
- n'accepte pas de connexion de l'extérieur (ports ouverts uniquement dans le conteneur)

Conteneur peut être attaché à un ou plusieurs réseaux :

- Ca change le mode par défaut (plus d'accès à l'hôte, plus d'accès aux conteneurs configurés par défaut)
- Échanges entre conteneurs attachés au(x) mêmes réseau(x)
- Attachement possible à un réseau à chaud (en live)

5 - Commande docker

Manipulation de l'environnement Docker.

Types :

- image
- container
- network
- volume
- divers autres

Syntaxe : docker <type> <commande>

Aide : docker --help

Aide sur un type donné :

docker <type> --help

Exemples :

- docker image --help
- docker container --help

Aide sur une commande donnée :

docker <type> <commande> --help

Exemples :

- docker image pull --help
- docker container run --help

run

Instancie un conteneur à partir d'une image et démarre son exécution (i.e. lance le processus du conteneur)

Syntaxe : docker container run <image>[:tag]

Exemples :
docker container run hello-world
docker container run sae4-php php script.php

Syntaxe : docker container run <image>[:tag]

Si tag pas spécifié => latest

Utiliser latest (par défaut) = pas toujours une bonne idée. Privilégier une version (un tag) connue pour fonctionner comme attendu.

Cycle de vie

Types de conteneurs :

- Sans fin : ne s'arrêtent pas seuls (ex : serveur Web)
- Avec fin : traitement fini, fin du processus (ex : un convertisseur d'images)

Quand le processus dans un conteneur s'arrête, le conteneur s'arrête aussi. Rappel : un conteneur = un processus (celui qui tourne "dans le conteneur")

Conteneur démarré = ajouté dans une liste

Conteneur (processus) terminé = gardé dans la liste

Liste = possibilité de faire “revivre” un conteneur

Chaque docker container run => nouveau conteneur => ajouté dans la liste => Liste grandit !

Solution, l’option --rm : docker run --rm etc.

ps

Listing des conteneurs actifs, arrêtés et terminés.

Syntaxe : docker container ps [-a]

Sans -a : seulement les conteneurs actifs (processus en cours d'exécution). Avec -a : tous (a = all)

Syntaxe : docker container ps [-a]

Sans -a : seulement les conteneurs actifs (processus en cours d'exécution). Avec -a : tous (a = all)

NB :
docker container ps == docker container ls

ID

Tous les objets (conteneurs, images, networks, volumes etc.) dans Docker ont un ID unique.

Exemple : docker container ps

Les commandes créant des objets affichent ID créé : docker container run hello-world -> Affiche ID du conteneur créé

ID = très longue valeur hexadécimale (64 cars)

OK pour utiliser un début d'ID tant que pas d'ambiguïté

docker container ps = généralement les 12 premiers cars. Possibilité d'afficher plus si besoin. Proba conflit sur 12 cars ~= nulle.

start/stop

Sur conteneur actif :

- Arrêt du conteneur (SIGTERM) :
  docker container stop <ID>

Sur conteneur arrêté :

- Redémarrage du conteneur :
  docker container start <ID>

kill

Sur conteneur actif, arrêt du conteneur (SIGKILL) :

docker container kill <ID>

kill vs stop = SIGKILL vs SIGTERM => KILL garanti, TERM ça dépend...

start/kill : données conservées dans le conteneur

rm

Sur conteneur arrêté, supprime le conteneur :

docker container rm <ID>

Supprime aussi les données du conteneur mais pas les volumes.

Detached

Un conteneur sans fin (daemon, style serveur Web) = par défaut est attaché au Terminal => Problème.

Pour détacher (laisser tourner en background) :

docker container run -d <image>[:tag]

Exemple : docker container run -d nginx

exec

Sur conteneur actif, exécute une commande DANS le conteneur :

docker container exec <ID> <commande>

Exemple :
docker container run -d nginx
docker container exec <ID> ls /

exec (interactif)

Pour commandes interactive, besoin d'attacher un TTY

docker container exec -t -i <ID> <commande>

Exemple :
docker container run -d nginx
docker container exec -ti <ID> bash

Ports (mappage)

Mappage port hôte <-> port conteneur

Syntaxe :
docker container run -p port_hote:port_conteneur <image>

Exemple : docker container run -p 9999:80 nginx
Navigateur : http://localhost:9999

Bind mount (volume mappé)

Mappage dossier hôte <-> dossier conteneur
Syntaxe :
docker container run -v
dossier_hote:dossier_conteneur <image>

Exemple :
docker container run -v $(pwd):/usr/share/nginx/html -p 8888:80
-d nginx
Navigateur : http://localhost:8888

https://commons.wikimedia.org/wiki/File:Thats_all_folks.svg