Virtualisation - R4.A.08

Isolateurs - Conteneurs

Cours 2

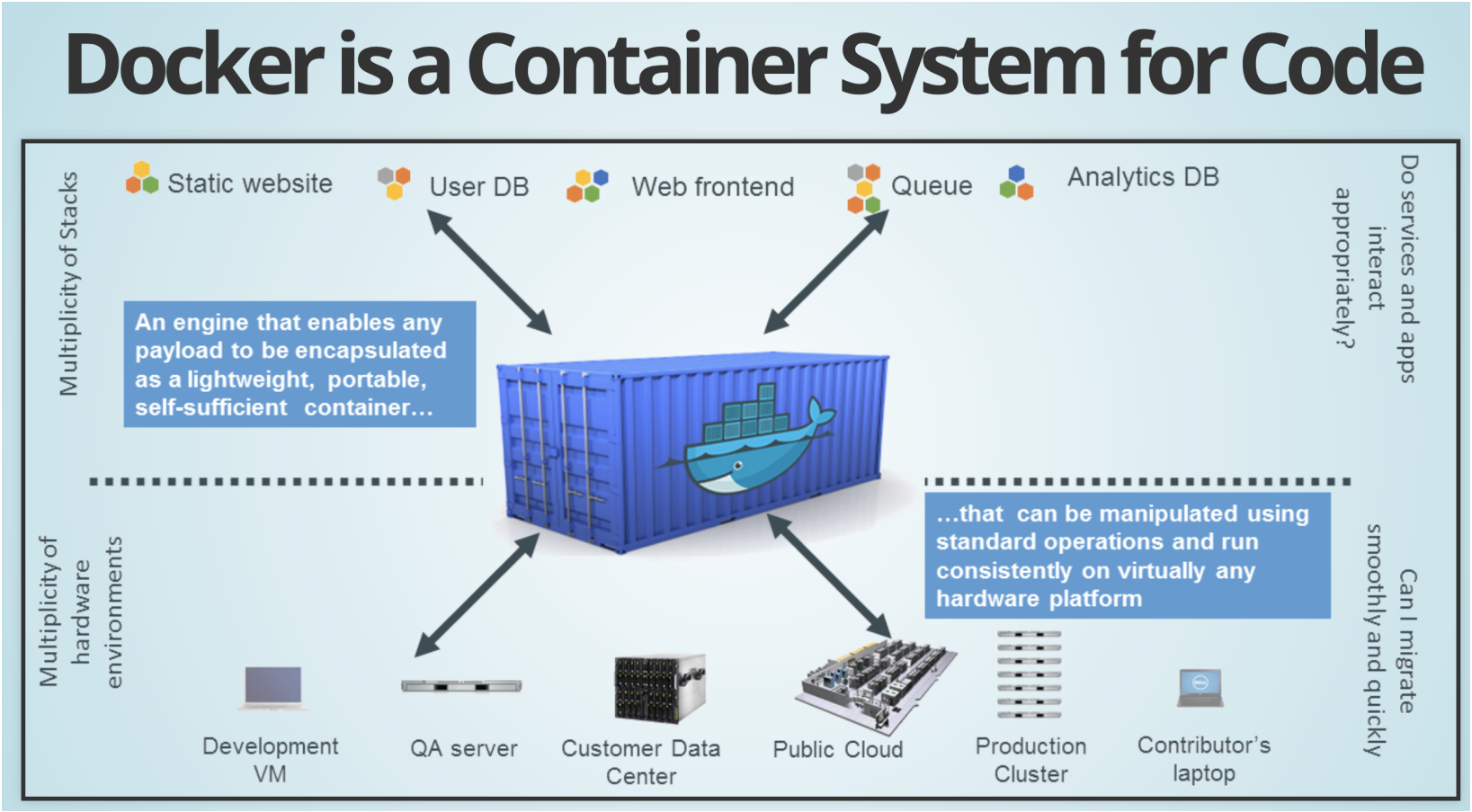
Michel Salomon & David Martinet

IUT Nord Franche-Comté
Département d'informatique

basé sur des supports de Guillaume Urvoy-Keller et Hadrien Pelissier

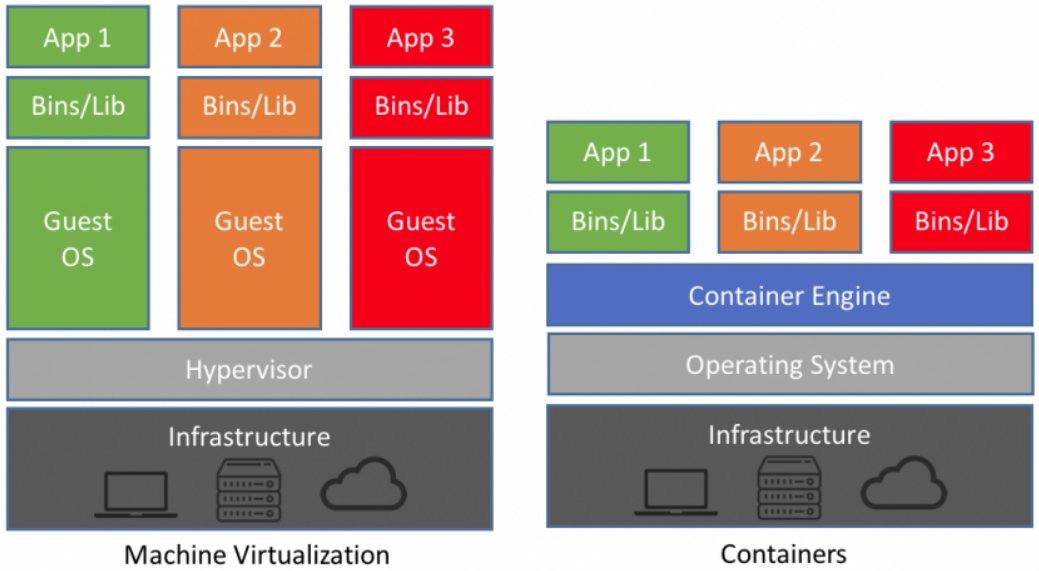
Les promesses de la conteneurisation

- Isolation → les installations (bibliothèques) de différentes appli. ne créent pas des problèmes de configuration mutuels
- Portabilité → déploiement possible sur des archi. variées



Comparaison entre VM et Conteneur - 1

Un même objectif ⇒ isoler des programmes dans des “contextes”



- VM → abstraction complète d’une machine (proc., mémoire, etc.)
- Conteneur → un découpage dans Linux pour séparer des ressources

Utilisation possible d’un système de quotas pour l’accès aux ressources matérielles (accès disque, carte réseau, proc.)

Comparaison entre VM et Conteneur - 2

Machines virtuelles

- Pros
  - OS complet
  - Isolation “parfaite” d’une VM par rapport à une autre
- Cons
  - Trop “lourdes” pour être multipliées librement
  - Faible efficacité pour isoler chaque application

Conteneurs

- Pros
  - Perf. proches du natif
  - Moins complexes et plus standards que les divers hyperviseurs
- Cons
  - Fortement lié au kernel hôte
  - Sécurité

Les conteneurs offrent une meilleur flexibilité que les VMs

- Passage à une architecture microservices
- La scalabilité nécessaire pour les besoins des services cloud

Comparaison entre VM et Conteneur - 3

Une machine virtuelle

- Comporte un système d'exploitation invité complet
- Inclut l'application, les bibliothèques ou binaires
- Nécessite plus de ressources

Un conteneur

- Partage le noyau (*kernel*) du système d'exploitation hôte avec d'autres conteneurs
- Inclut l'application et ses dépendances
- Est exécuté en tant que processus isolé dans l'espace utilisateur sur le système hôte (pas le cas d'un conteneur Hyper-V $\rightarrow$ un conteneur dans une VM spécifique)

Qu'est-ce qu'un conteneur ? - 1

Un groupe de processus associé à un ensemble de permissions

Genèse du concept de conteneur

Un vieux concept d'isolation des processus
permis par la philosophie "tout est fichier"

- `chroot` → changer de racine - dans les OS Unix depuis 1979

  "Comme tout est fichier, en changeant la racine d'un processus, c'est comme le faire changer de système."

- `jail` → intro. par FreeBSD en 2002 pour compléter `chroot`
  - `chroot` isolait un processus qu'au niveau du système de fichiers
  - `jail` permet une isolation réelle (et sécurisée) des processus
- **`namespaces`** → limiter ce qu'un processus peut voir de l'OS
  - Implémentent le concept de `jail` dans Linux
- **`cgroups`** → limiter les ressources matérielles d'un processus

Qu'est-ce qu'un conteneur ? - 2

Un conteneur c'est

- **un groupe de processus**
- associé à un **ensemble de permissions sur le système**

1 conteneur = 1 groupe de processus Linux

- des `namespaces` → séparation entre ces groupes
- des `cgroups` → quotas en ressources matérielles

Des `LXC` (*LinuX Containers*) à `Docker`

- Projet LXC démarré en 2008
  - Rassembler les `cgroups`, le `chroot` et les `namespaces`
- Docker
  - Basé initialement sur LXC qui a été abandonné ultérieurement
  - Standardisation ; Robustesse ; Interface d'utilisation simple
  - Cloud pour diffuser les images - `Docker Hub`

Bloquer un système hôte depuis un conteneur est possible

Comment ?

- Utilisation d'une *fork bomb* dans un conteneur non privilégié
- Résultat → blocage de Docker, voire du système hôte

Solution

- Limiter la création de processus via une option du noyau
- Exemple

```
docker run -it --ulimit nproc=3 --name fork-bomb bash
```

Attention

- **L'isolation des conteneurs n'est pas absolue**, mais
- avec un bon paramétrage elle est **robuste, mature et testée**

Pourquoi utiliser Docker ?

Pensé à l'origine pour faire des **conteneurs applicatifs**

- **Isoler** les modules applicatifs
- Gérer les **dépendances** en les embarquants dans le conteneur
- Se baser sur l'**immutabilité**
  - Config. conteneur pas faite pour être modifiée après la création
- Avoir un **cycle de vie court**
  - Logique DevOps → "bétail *vs.* animal de compagnie"

Modifie la "logistique" applicative

- **Uniformisation**
  - Face aux divers langages de prog., config. et briques logicielles
- **Installation sans accroc** et **automatisation**
- Simplifie l'**intégration**, la **livraison**, le **déploiement continus**
- Rapproche **monde du développement** et des **opérations**

Positionnement de Docker sur le marché

Docker est la techno. ultra-dominante de conteneurisation

- Simplicité et standardisation
  - Format ouvert standardisé → `Open Container Initiative`
- Logique du conteneur fonctionnelle, bonne doc. et écosystème

Des alternatives existent

- `LXC` existe toujours et fonctionne bien en liaison avec `LXD`
  - Positionnement différent → conteneurs système
- `Mesos` permet de gérer un cluster en se passant de Docker
  - Propose quand même un support des conteneurs OCI (Docker)
- `Podman` permet un un mode *rootless* et *daemonless*
  - Même syntaxe que Docker, fait fonctionner des conteneurs OCI
- `systemd-nspawn` exécute des conteneurs à l'image de `LXC`

Terminologie Docker - 1

Quelques termes

- **Image conteneur (image container)**
  - Package de toutes les dépendances et informations nécessaires pour créer un conteneur
- **Dockerfile**
  - Fichier texte contenant des instructions pour la création d'une image Docker
- **Création (build)**
  - Action de créer une image conteneur sur la base des info. et du contexte fournis par le fichier Dockerfile associé, plus des fichiers supplémentaires dans le dossier où l'image est créée
- **Conteneur (container)**
  - Instance d'une image Docker
- **Volume**
  - Système de fichiers accessible en écriture par le conteneur
  - **Les images sont UNIQUEMENT en lecture seule**

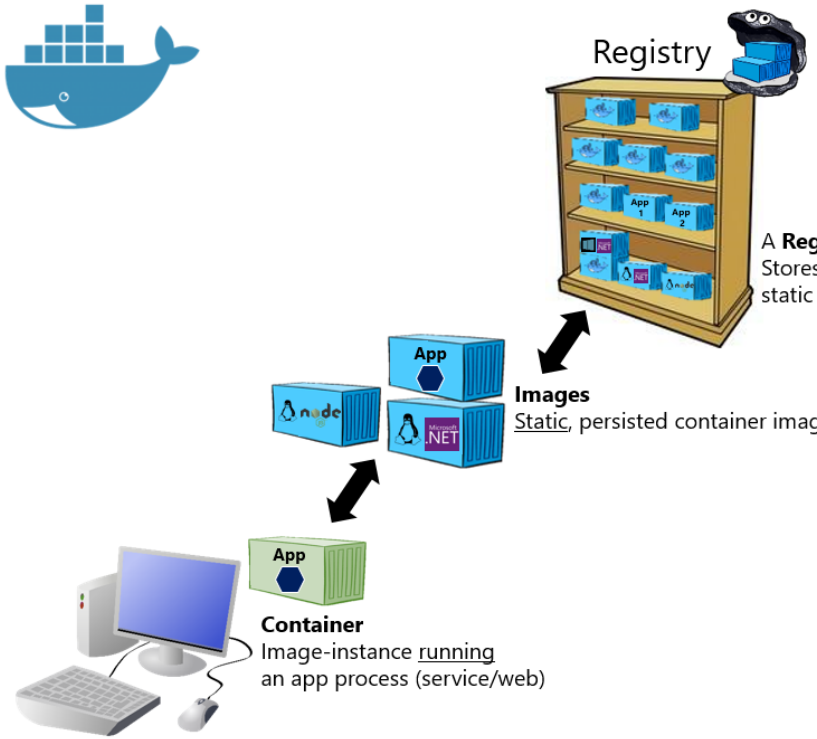
Terminologie Docker - 2

Quelques termes

- **Balise (tag)**
  - Marque ou étiquette que l'on peut appliquer aux images
- **Dépot (repository)**
  - Collection d'images Docker associées, identifiées par une balise qui indique la version de chaque image
- **Registre (registry)**
  - Service qui fournit l'accès aux dépôts. Le registre par défaut utilisé pour la plupart des images publiques est Docker Hub
- **Docker Hub**
  - Registre public dans lequel on peut charger et manipuler des images Docker. Fournit un hébergement d'images, des registres publics ou privés, ...

Terminologie Docker - 3

Basic taxonomy in Docker



Hosted Docker Registry

Docker Trusted Registry on-prem.

On-premises
('n' private organizations)

Docker Hub Registry

Docker Trusted Registry on-cloud

Azure Container Registry

AWS Container Registry

Google Container Registry

Quay Registry

Other Cloud

Public Cloud
(specific vendors)

Namespaces (espaces de noms)

Un concept informatique pour parler simplement de...

- groupes séparés auxquels on donne un nom, d'ensembles de choses sur lesquelles on colle une étiquette
- on parle aussi de **contexte**

Généralités

- Apparus en 2002 dans le noyau 2.4.19
- Réellement utilisés en 2013 dans le noyau 3.8
- **Limite ce qu'un processus peut voir du système**
- 6 namespaces → `PID`, `NET`work, `MOUNT`, `USER`, `UTS` et `IPC`
- Chaque processus est dans un namespace de chaque type
- **Les namespaces sont natifs dans Linux**
  - Même si il n'y a pas de conteneur de créé ou Docker installé,
  - chaque processus créé est associé à 6 namespaces (par défaut)

Namespace PID

PID - Rôle

fournit l'isolation pour l'allocation des identifiants de processus (PIDs), la liste des processus et de leurs détails

Principes

- Les processus d'un namespace "parent" voient les processus dans les namespaces "enfants", mais avec des numéros de PID différents
- Un processus voit les processus de son namespace PID
- Un processus est aussi visible dans le namespace par défaut (il a un numéro dans les 2, mais différents)
- Chaque namespace PID a sa propre numérotation à partir de 1
- Si le PID 1 disparaît, le namespace est détruit

Namespace NET

NET - Rôle

isole le contrôleur de l'interface réseau (physique ou virtuel), les règles de pare-feu iptables, les tables de routage, etc.

Principes

- Le processus ne voit que la pile réseau de son namespace
  - Ses interfaces (différentes de l'hôte)
  - Sa table de routage séparée
  - Ses règles iptables
  - Ses sockets

Namespace MOUNT

MOUNT - Rôle

permet de créer différents modèles de systèmes de fichiers,
ou de créer certains points de montage en lecture-seule

Principes

- Un namespace MOUNT
  - Dispose de son propre `rootfs` (concept proche d'un `chroot`)
  - Peut disposer de son propre `/proc`, `/sys`
  - Peut aussi avoir ses montages "privés"
  - Son `/tmp` (par utilisateur, par service)

Namespaces USER, UTS et IPC

Rôles

- USER → isole l'allocation des IDs user / group (UIDs/GIDs)
- UTS → détermine le nom de l'hôte (*get / set hostname*)
- IPC → permet à un groupe de processus d'avoir des mécanismes de communication qui leur sont propres

Principes

- Un namespace USER mappe UID/GID vers différents utilisateurs de l'hôte
- Un namespace IPC permet à un groupe de processus d'avoir
  - des sémaphores
  - des files de messages
  - de la mémoire partagée

  sans risque de conflit avec d'autres groupes d'autres namespaces

Manipulation des namespaces

Manipulations

- Création à l'aide des commandes `clone` ou `unshare`

  `man namespaces`, `man clone`, `man unshare`

- Matérialisés par des pseudo-fichiers dans `/proc/$PID/ns`

Exemple

```
root@hyperion:/proc/843/ns# ls -l
total 0
lrwxrwxrwx 1 root root 0  7 mars  18:12 cgroup -> 'cgroup:[4026531835]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 net -> 'net:[4026532008]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 pid_for_children -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 time -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 time_for_children -> 'time:[4026531834]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0  7 mars  18:12 uts -> 'uts:[4026531838]'
root@hyperion:/proc/843/ns#
```

Control groups - 1

Une façon de...

- tagger les demandes processeur et les appels systèmes
- pour les grouper et les isoler

Généralités

- Apparus en 2008 dans le noyau 2.6.24
- **Limite les ressources matérielles d'un processus**
  - processeur, mémoire, réseau et entrées / sorties

Namespaces *versus* cgroups

- **Namespaces** $\rightarrow$ **permettent une séparation logique des conteneurs**
- **Cgroups** $\rightarrow$ **permettent de spécifier les ressources phys. consommables**

Control groups - 2

Principes

- Une hiérarchie par ressource (ou groupe de ressources)
  - Hiérachie processeur, mémoire
- Groupes matérialisés par des pseudos-fichiers
  - Généralement montés dans `/sys/fs/group`
- Processus de PID 1 placé à la racine de chaque hiérarchie
- Nouveaux processus démarrés dans le groupe de leur parent

Comment placer un processus dans un cgroup ?

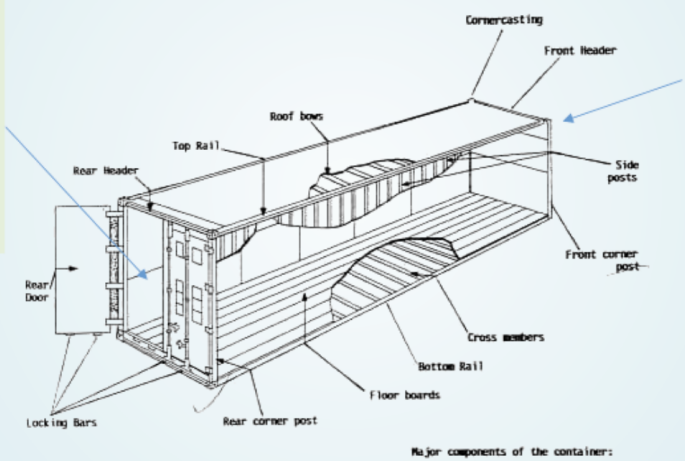
On écrit le numéro du processus dans un fichier spécial

Fin de la rivalité “development *vs.* operations”

- Équipe de développement ⇒ focalisée sur l’“intérieur”
  - Packaging de l’application → les fichiers nécessaires à son exécution (code, runtime, outils système, biblio. et paramètres)
- Équipe d’exploitation ⇒ focalisée sur l’“extérieur”
  - Gestion de conteneurs → s’occupe de l’env. autour des conteneurs (réseau, logs, monitoring, configuration dynamique)

Why it Works: Separation of Concerns

- **Dan the Developer**
  - Worries about what’s “inside” the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same



- **Oscar the Ops Guy**
  - Worries about what’s “outside” the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

Bénéfices mutuels attendus

Pour les développeurs en charge d'une application

- Un conteneur pouvant s'exécuter sur différentes infrastructures
- Moins de problèmes
  - Dépendances / paquets manquant(e)s, soucis de versions
  - Compatibilité des plateformes d'exécution
- Cohérence de l'environnement de développement
  - Les développeurs utilisent le même OS, les mêmes biblio., langages quel que soit le système d'exploitation hôte

Pour les opérateurs (admin. sys.) en charge de son déploiement

- Déploiement plus efficace, rapide, fiable et reproductible
- Élimination des incohérences entre les environnements de développement, de test, de production
- Amélioration des perf. et de la portabilité, réduction des coûts

Au cœur de Docker

- Utilisation des conteneurs Linux et des facilités du noyau
  - Namespaces / control groups → isolation / contrôle ressources
  - AuFS → union mount évitant la duplication de fichiers
- Couche de virtu. Linux entre Windows / MacOS et runtime Docker