Virtualisation de l'OS et conteneurs

Rappel de la problématique

Comment partager un serveur ?

- Utilisateurs indépendants
- Toute combinaison d'applications
- Utilisation à la demande en self-service
  - Déploiement rapide
  - Sans intervention d'un administrateur
- Difficulté
  - Toutes les applications voient
    - Les mêmes fichiers
    - Les mêmes identifiants réseaux
    - Les mêmes processus
  - Inadaptation des interfaces standardard des OS
    - Manque de flexibilité
    - Nécessite de définir une politique centralisée
    - Configuration de l'application spécifique à chaque déploiement

La virtualisation du matériel

Multiplexage du matériel

- Chaque utilisateur accède à son instance du matériel (VM) qui lui est dédiée
  - Interface matérielle équivalente à une machine physique
  - Déploiement d'un OS adapté à son application
- Installation de l'application dans un environnement maîtrisé
  - Installation des bibliothèques, supports exécutifs
  - Pas de conflits sur les chemins, ports, …

La virtualisation du matériel

Avantages

- Interface matérielle relativement simple, stable
  - Facilite l'écriture d'un hyperviseur sécurisé
- Offre aux utilisateurs le contrôle complet de leur environnement logiciel
  - Y compris le système d'exploitation
  - Même flexibilité qu'une machine physique dédiée
- Isolation complète des VMs
  - Multiplexage bas-niveau
  - L'hyperviseur ne gère que du matériel virtuel
  - Indépendance complète à ce qui est exécuté dans chaque VM

La virtualisation du matériel

Suivant les situations: avantages = inconvénients

- Nécessité de gérer un système d'exploitation complet
  - Besoin réel: déployer une application de façon reproductible
  - Concepts d'administration système à maitriser
  - Comlexité d'un déploiement d'OS
  - Lourdeur d'un transfert d'image de VM
- Coût de l'isolation apportée par la virtualisation du matériel
  - Perte de performance
    - Interruptions logicielles
    - MMU imbriquée
    - Multiples niveaux d'ordonnancement indépendants
  - Faible densité d'applications
    - Un OS complet par application
    - Mémoire consommée par chaque OS
    - Pas de partage de cache de fichier etc.
  - Temps de lancement d'un OS complet

La virtualisation du système d'exploitation

Virtualisation des interfaces d'un OS

- Equivalent à un multiplexage de l'OS
  - Plusieurs OS virtuels indépendants
- Un même OS gère simultanément plusieurs instances de son interface
  - Utilisables de façon indépendante, sans conflit par plusieurs applications
- Notion de namespace
  - Permet à un même identifiant de ressource de désigner différentes choses pour différentes applications

La virtualisation du système d'exploitation

Exemple: virtualisation du système de fichiers

- Sans virtualisation de l'OS
  - Attribuer à chaque application des répertoires pour leurs executables, données ..
  - Configurer les droits d'accès nécessaires
  - Configurer les applications pour utiliser ces chemins
- Avec virtualisation de l'OS
  - Chaque application a une vue différente du système de fichiers
    - Mise en place pour un groupe de processus
  - Toutes les applications peuvent être configurées dans des chemins identiques
    - Un même chemin de fichier pointe vers des données différentes

La virtualisation du système d'exploitation

Autres exemples

- Noms d'utilisateurs et permissions
  - Plusieurs utilisateurs root
  - Privilèges limités à une partie des ressources
- Identifiants de processus
  - Plusieurs processus ayant le même PID
  - Visibilité restreinte aux processus du même espace de PID
- Identifiants réseaux
  - Adresses IP, ports et noms d'interfaces indépendants
  - Plusieurs processus peuvent écouter sur 0.0.0.0:80

La virtualisation du système d'exploitation

Virtualisation "à la carte"

- Exemple:
  - Deux processus peuvent partager les mêmes identifiants réseaux …
  - Mais pas les mêmes identifiants de fichiers

Le noyau est partagé par tous les processus

- Pas d'augmentation de la consommation mémoire
- Partage des caches
- Pas de temps d'intialisation au lancement d'une application

La gestion des namespaces par Linux

Linux gère 6 namespaces différents

- **MOUNT**: (CLONE_NEWNS, depuis Linux 2.4, 2002)
  - Points de montages vus par un groupe de processus
  - Appels systèmes *mount(), unmount()*
- **UTS**: (CLONE_NEWUTS, depuis Linux 2.6.19)
  - UNIX Time-sharing System
  - Vision du hostname et domainname
  - Appels sytèmes *uname(), sethostname() setdomainname()*
- **IPC**: (CLONE_NEWIPC, depuis Linux 2.6.19)
  - Vision des objets de communcation inter-processus
  - Identifiants hors système de fichiers
  - Voir commandes *ipcmk ipcs* ..

La gestion des namespaces par Linux

Linux gère 6 namespaces différents

- **PID**: (CLONE_NEWPID, depuis Linux 2.6.24)
  - Vision des identifiants de processus
  - Deux processus peuvent avoir le même PID dans deux NS différents
    - Possibilité d'avoir plusieurs PID 1
  - Correspondance avec un PID dans le namespace parent
  - Interaction uniquement avec des processus de son namespace (et ses fils)
    - Appel système *kill()*
- **NET**: (CLONE_NEWNET, depuis Linux 2.6.24-29)
  - Vision des ressources réseaux
  - Adresses IP, numéro de ports, tables de routage etc.
- **USER**: (CLONE_NEWUSER, depuis Linux 2.6.23-3.8, 2013)
  - Vision des identifiants d'utilisateur et de groupes
  - Un même UID peut correspondre à deux utilisateurs différents dans deux NS
  - Correspondance avec un UID dans le namespace parent
  - Privilèges limités à un namespaces (et ses fils)

La gestion des namespaces par Linux

Linux gère 6 namespaces différents

- Un travail de longue haleine
  - Plus de 10 ans pour développer l'ensemble des namespaces
  - Bien définir la sémantique de ces nouvelles API
  - Nombreuses problématiques de stabilité, sécurité …
  - Nouvelles surfaces d'attaque
    - Interfaces précédemment réservées aux administrateurs

La gestion des namespaces par Linux

Création de namespaces

- Tout processus est dans une instance de chacun des 6 namespaces
- Création d'un nouveau namespace pour un processus fils lors d'un *fork*
  - Appel système *clone()*
  - Par défaut, un processus fils hérite du namespace de son père
  - Utilisation des flags CLONE_* en argument de clone()
    - Place le fils dans une nouvelle instance d'un ou plusieurs namespaces
- Création d'un nouveau namespace pour le processus courant
  - Appel système *unshare()*
  - Utilisation des mêmes flags CLONE_*
  - Commande système **unshare**
- Rejoindre un namespace existant
  - Appel système *setns()*
  - Flags CLONE_* et descripteur d'un fichier dans /proc/[pid]/ns
  - Commande système **nsenter**

La gestion des namespaces par Linux

Création de namespaces

- Par défaut un namespace est détruit lorqu'il ne contient plus de processus
  - Toutes les ressources associées sont détruites
  - Ex: interfaces réseau
- Les namespaces d'un processus sont visibles à l'aide de

```
$ ls -l /proc/self/ns
lrwxrwxrwx 1 root root 0 Dec  9 22:31 ipc -> 'ipc:[4026531839]'
lrwxrwxrwx 1 root root 0 Dec  9 22:31 mnt -> 'mnt:[4026531840]'
lrwxrwxrwx 1 root root 0 Dec  9 22:31 net -> 'net:[4026531992]'
lrwxrwxrwx 1 root root 0 Dec  9 22:31 pid -> 'pid:[4026531836]'
lrwxrwxrwx 1 root root 0 Dec  9 22:31 user -> 'user:[4026531837]'
lrwxrwxrwx 1 root root 0 Dec  9 22:31 uts -> 'uts:[4026531838]'
```

- Savoir si processus sont dans un même namespace
  - Leur fichier /proc/[pid]/[type ns] pointe sur le même inode
- Commande système **lsns**
  - Liste les namespaces de chaque processus

La gestion des namespaces par Linux

Namespaces UTS

- Lancement d'un processus dans un nouveau namespace
  - Nécessité d'être **root** pour appeler **unshare**
    - Sauf si on créee un nouveau user namespace avec **- -user**
  - Par défaut **unshare** lance un shell
    - Possibilité de lancer toute commande

```
$ unshare --uts
$ hostname -f
vm0.pcocc
$ hostname container
$ hostname -f
container
```

- Pour le reste du sytème le hostname n'a pas changé

```
$ hostname -f
vm0.pcocc
```

La gestion des namespaces par Linux

Namespaces réseau

- À la création, un namespace réseau n'a qu'une interface loopback

```
$ unshare --net
$ ip a
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

La gestion des namespaces par Linux

Namespaces réseau

- Une paire veth est souvent utilisée pour établir une connexion entre deux namespaces

```
$ ip link add name host_side type veth peer name ns_side
$ btctl addbr br0
$ ip link set br0 up
$ ip addr add br0 172.16.0.1/24
$ brctl addif br0 host_side
$ ip link set host_side up
$ ip link set ns_side netns 1866
```

- Le veth est alors visible dans le nouveau namespace réseau

```
$ ip link set ns_side name eth0 up
$ ip addr add 172.16.0.2/24 dev eth0
```

La gestion des namespaces par Linux

Namespaces utilisateur

- Création d'un nouveau namespace utilisateur
  - Peut être réalisé sans droits particuliers

```
user@vm0 $ unshare --user --map-root-user
root@vm0 $ brctl addbr br0
add bridge failed: Operation not permitted
```

- Pas de d'accès privilégiés à des namespaces créés par un autre user namespace !

```
user@vm0 $ unshare --user --map-root-user --net
root@vm0 $ brctl addbr br0 # OK
```

La gestion des namespaces par Linux

Namespaces utilisateur

- Correspondance entre UIDs dans les namespaces parent et fils

```
$ cat /proc/self/uid_map
0       1000          1
```

- Possibilités de définir plusieurs lignes
- Chacune contient trois valeurs:
  - UID de départ dans le namespace fils
  - UID de départ dans le namespace parent
  - Nombre d'UIDs consécutifs
- Cas d'un utilisateur non-privilégié dans le namespace parent
  - Uniquement possible d'assigner son UID à un unique UID du namespace fils

La gestion des namespaces par Linux

Namespaces utilisateur

- Plus de droits possibles grâce la commande *setuid* **newuidmap**
  - Configurée par un administrateur dans /etc/subuid
  - Autorise à utiliser une plage d'UID parent

```
$ cat /etc/subuid
diakhate:100000:65536
$ newuidmap [pid] 0 1000 1 1 100000 65536
$ cat /proc/[pid]/uid_map
0       1000          1
1       100000        65536
```

Les conteneurs: notion aux contours variables

Isolation (*contain*)

- Groupes de processus isolés les uns des autres et du reste de l'OS
  - Différents mécanismes et niveaux d'isolation
- Interfaces de l'OS
  - Namespaces (et capabilities)
    - Vision d'un OS partiellement ou totalement indépendant
    - Prévention des interactions avec les objets hors du conteneurs
    - Elevation de privilège restreinte au conteneur
- Ressources matérielles disponibles
  - Control Groups
    - CPU: sous-ensemble des CPUs, QoS …
    - Mémoire: quantité, zones NUMA …
    - Périphériques: restrictions des périphériques utilisables …
    - I/O: QoS, quotas …

Les conteneurs: notion aux contours variables

Format de transport interoperable (*conteneur*)

Les conteneurs: notion aux contours variables

Format de transport interoperable (*conteneur*)

- 90% of all cargo now shipped in a standard container
- Order of magnitude reduction in cost and time to load and unload ships
- Massive reduction in losses due to theft or damage
- Huge reduction in freight cost as percent of final goods (from >25% to <3%)

→ massive globalization

- 5000 ships deliver 200M containers per year

docker

Les conteneurs: notion aux contours variables

Format de transport interoperable (*conteneur*)

- Transport facile d'une application d'un serveur à lautre
  - Fonctionne à l'identique sur:
    - Un laptop de développement
    - De mutliples serveurs en production
- Format contenant l'ensemble des données nécessaires
  - Fichers de l'application (exécutables, …)
  - Methode de lancement de l'application
  - Établissement de correspondances entre ressources virtuelles et hôtes
    - Ex: Identifiant de port dans un conteneur -> identifiant port hôte

Les conteneurs: notion aux contours variables

Premières formes de conteneurs: isolation

- Objectif: machines virtuelles plus légères
  - Plus rapide à démarrer
  - Plus dense
  - Un système complet par conteneur
    - init/systemd
    - Démons traditionnels (SSH)
- Exemples:
  - OpenVZ (2005)
    - Nécessite un noyau Linux patché
    - Adopté par des fournisseurs d'hébergement en ligne
    - Moins coûteux que des machines virtuelles
  - LXC (2008)
    - Basé sur les namespaces introduits progressivement dans Linux
    - Adoption relativement faible initialement
    - Support des namespaces incomplet
    - Gain insuffisant par rapport aux VMs

Les conteneurs: notion aux contours variables

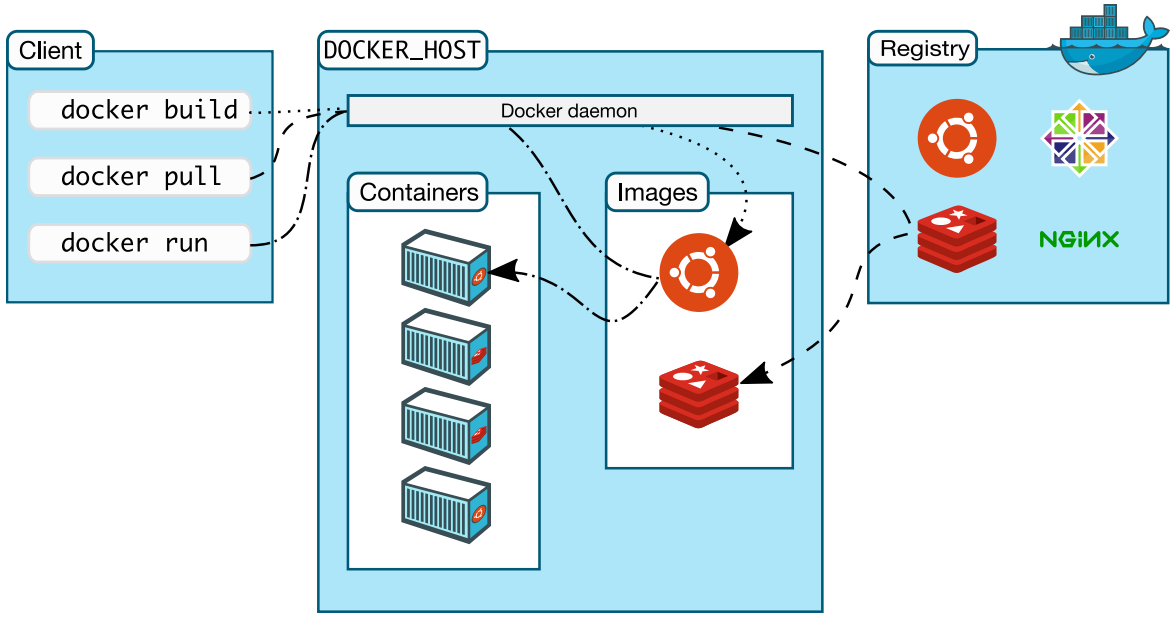
Docker (2013): conteneurs transportables

- Empaquete une unique application (souvent un unique processus)
  - Inclus toutes les dépendances
- Recette de construction de conteneurs
  - Génération reproductible
  - Peut être associée au code source de l’application
- Base de registre de conteneurs
  - Push/Pull
  - Recherche d’applications containerisée
- Execution à l’identique sur tout type de machine

Architecture de Docker

Une application client/serveur

- Écrite en langage go
- La CLI (docker run/build …) communique avec un démon docker
  - API REST (socket Unix local ou TCP)
- Utilisation reservée à un utilisateur privilégié
  - L'accès à ces commandes équivaut à être root sur le machine hôte



Images Docker

Données et configuration permettant de créer un conteneur

- Données: système de fichiers root du conteneur
- Configuration:
  - Méta-données: Auteur, labels, date de création, etc.
  - Commande à exécuter pour lancer le conteneur
  - Variables d’environnement à positionner
  - Ports à rendre accessibles
  - etc.

Images Docker

Empilement de couches (layers)

- Chaque couche ajoute/modifie/supprime données ou configuration
- Partage possible de couches entre plusieurs images
- Permet de créer des couches communes réutilisables
- Spécialisation progressive
- Découple la gestion de différents aspects
  - OS de base
  - Dépendances communes à une classe d’application
  - Application
  - Configuration de l’application

Images Docker

Exemples de couches pour une application Web Java

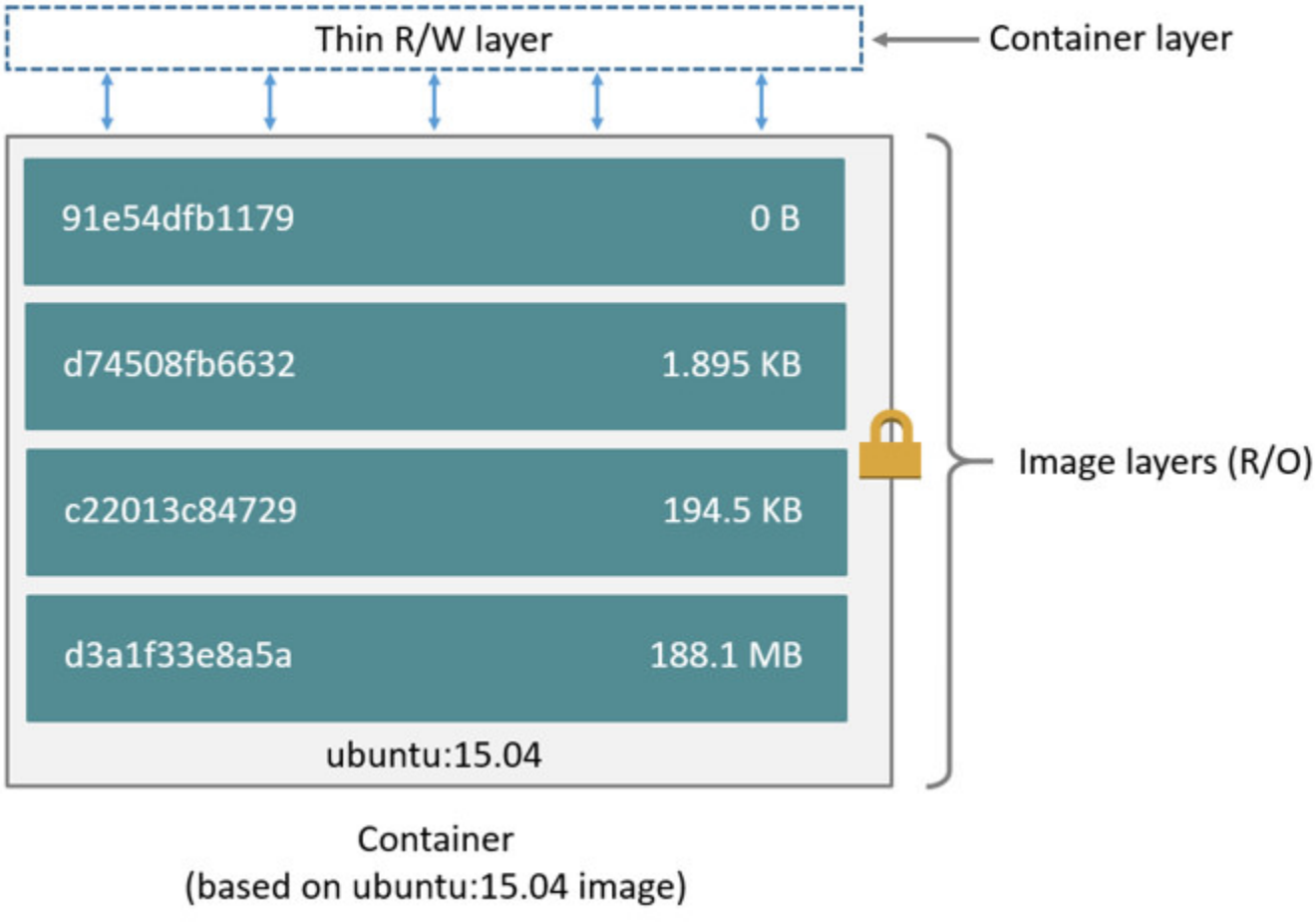
- OS de base (ex: Ubuntu)
- Personnalisation de la distribution par l’entreprise
- Runtime Java
- Tomcat
- Dépendances de l’application
- Code et données de l’application
- Configuration de l’application

Conteneurs et images

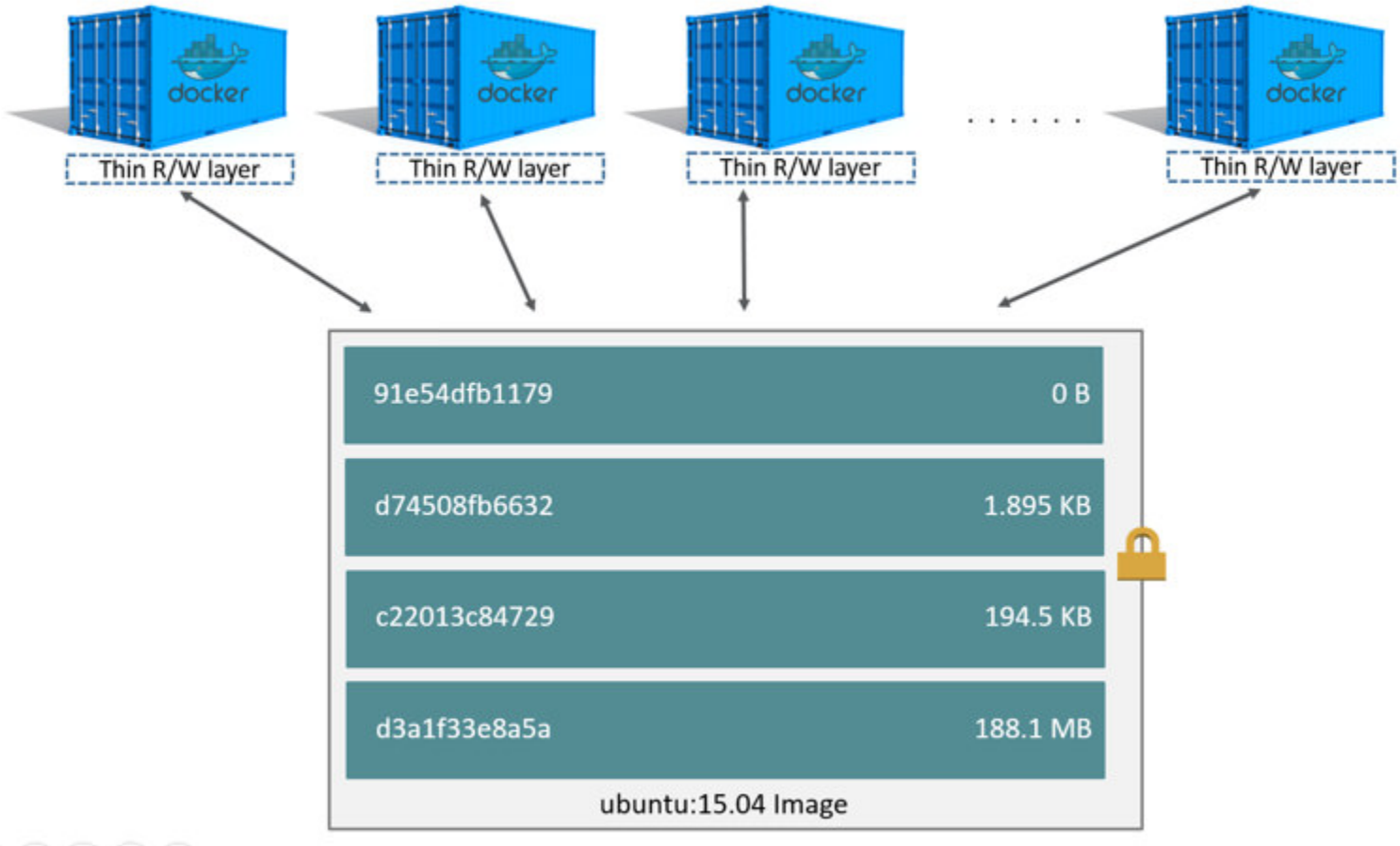
Deux concepts distincts

- Une image est un modèle
- Elle permet d'instancier un nombre illimité de conteneurs
- Chaque conteneur correspond à un ensemble de processus partageant
  - Espaces de nommages
    - Système de fichier construit à partir de l'image
- Fonctionnement en mode copy-on-write
  - Évite de copier l'image complète pour lancer un conteneur
  - Couches de l'image utilisées en lecture seule
  - Ajout d'une couche supplémentaire modifiable

Conteneurs et images



Conteneurs et images



Lancement de conteneurs

Syntaxe ligne de commande

- Commande **docker run**
  - Instancie un conteneur à partir d'une image
    - Namespaces, rootfs, couche modifiable, isolation …
  - Exécute une commande dans cet environnement
    - Commande par défaut spécifiée dans les meta-données
    - Commande passée en ligne de commande
  - Arrêt du conteneur quand la commande se termine

Lancement de conteneurs

Exemple

```
$ docker run -i -t ubuntu cat /etc/os-release
Unable to find image 'ubuntu:latest' locally
latest: Pulling from library/ubuntu
32802c0cfa4d: Pull complete
da1315cffa03: Pull complete
fa83472a3562: Pull complete
f85999a86bef: Pull complete
Digest: sha256:6d0e0c26489e33f5a6f0020edface2727db9489744ecc9b4f50c7fa671f23c49
Status: Downloaded newer image for ubuntu:latest
NAME="Ubuntu"
VERSION="18.04.1 LTS (Bionic Beaver)"
ID=ubuntu
[...]
$
```

- -i: Redirige stdin dans le conteneur
- -t: Alloue un pseudo-terminal
- ubuntu: Nom de l'image de conteneur

Lancement de conteneurs

Récupération de l'image

- Initialement l'image *ubuntu* n'est pas stockée localement
  - Récupération dans une base de registre
  - Stockage local pour les futures utilisations
- *ubuntu* fait référence à:
  - l'image ubuntu
  - dans la bibliothèque d'image officielle (library)
  - dans la base de registre DockerHub (*docker.io*)
  - avec le version *latest*
- Equivalent à: *docker.io/library/ubuntu:latest*
- La bibliothèque DockerHub contient
  - Des images boites à outil tel que busybox
  - Des images de distributions Linux de base
  - De nombreux composants standards tel que httpd, nginx, mysql, redis …
  - Visible avec un navigateur sur https://hub.docker.com/
  - En ligne de commande avec **docker search**

Lancement de conteneurs

Exemples

- La commande **docker pull** permet de récupérer manuellement une image

```
$ docker pull centos
$ docker pull docker.io/library/fedora:27
$ docker pull nvcr.io/hpc/namd:2.13b2-singlenode
```

- Lister les images stockées localement

```
$ docker image ls
REPOSITORY          TAG                 IMAGE ID      CREATED       SIZE
centos              latest              1e1148e4cc2c  3 days ago    202MB
ubuntu              latest              93fd78260bd1  2 weeks ago   86.2MB
nvcr.io/hpc/namd    2.13b2-singlenode   5375a283a442  4 weeks ago   366MB
fedora              27                  7a2e85963474  3 months ago  236MB
```

- Une image locale est référençable par son identifiant

```
$ docker run -it 93fd
```

Lancement de conteneurs

Gestion des conteneurs instanciés

- La commande **docker run** est une commande cliente de dockerd
- Les processus du conteneur sont exécutés par le démon docker
- Redirection des E/S standard vers le client

```
root      8663  S    09:16   0:00  bash
root     12212  Sl+  11:06   0:00   \_ docker run -ti ubuntu
[..]
root      8469  Ssl  09:08   0:48 /usr/bin/dockerd -H unix://
root      8470  Ssl  09:08   0:27 /usr/bin/containerd
root     12230  Sl   11:06   0:00  \_ containerd-shim -namespace moby -wo [...]
root     12247  Ss+  11:06   0:00      \_ /bin/bash
```

- Lancement du conteneur en arrière plan: **docker run -d**
  - Sorties standard collectées par dockerd
  - **^P^X**: se détacher d'un conteneur lancé en interactif
    - Alternative: tuer le client docker

Lancement de conteneurs

Gestion des conteneurs instanciés

- Lister les conteneurs

```
$ docker ps -a # Inclure les conteneurs terminés
CONTAINER ID  IMAGE          COMMAND      STATUS         PORTS  NAMES
72a936f72eae  jpetazzo/clock "/bin/sh     Up 39 seconds         zen_archimedes
75ad68b88272  ubuntu         "/bin/bash"  Exited (0)            clever_nightingale
05a16ca917ac  httpd          "httpd-for   Exited (0)            priceless_montalcini
```

```
$ docker ps -l # Dernier conteneur lancé
$ docker ps -ql # ID du dernier conteneur lancé
```

- Un conteneur peut être référencé par son ID ou son nom
  - Nom par défaut ([adjectif]_[hacker ou scientifique])
  - Spécifiable à la création du conteneur avec **- -name**

Lancement de conteneurs

Gestion des conteneurs instanciés

- S’attacher à un conteneur en arrière plan

```
$ docker attach zen_archimedes
```

- Afficher les logs d’un conteneur

```
$ docker logs 72a936f72eae
```

  - Stockés par dockerd via différents drivers
    - json-file (par défaut), journald, gelf (intégration à logstash) …
    - Seuls json-file et journald permettent l’utilisation de **docker logs**
  - Configurable globalement ou par conteneur

```
$ docker run --log-driver=json-file --log-opt max-size=10m --log-opt max-file=3 [...]
$ docker run --log-driver=gelf --log-opt=gelf-address=udp://elasticsearch:12201 [...]
```

Lancement de conteneurs

Gestion des conteneurs instanciés

- Se connecter à un conteneur
  - Pas besoin de démon SSH
  - Lancement d'un processus dans les mêmes namespaces que le conteneur

```
$ docker exec -ti [container id] /bin/bash
```

- Equivalent à la commande *nsenter*

```
$ nsenter -a -t [container process pid] /bin/bash
```

- **nsenter** permet de choisir les namespaces rejoints
  - Utile pour exécuter une commande inexistante dans le conteneur
- A réserver à des fins de mise au point / debug
  - Lorsque les logs et métriques collectées ne donnent pas suffisament d'information
  - Automatiser la construction et déploiement de conteneurs

Lancement de conteneurs

Gestion des conteneurs instanciés

- Par défaut un conteneur s'arrête à la terminaison de la commande exécutée
- Tuer un conteneur:

```
$ docker stop 72a936f72eae   # SIGTERM puis SIGKILL
$ docker kill zen_archimedes # SIGKILL
```

- Relancer un conteneur arrêté:

```
$ docker start [-a] 72a936f72eae
```

  - Relance la commande exécutée à la création du conteneur
  - La couche de stockage modifiable est réutilisée
  - Les nouvelles logs sont écrites à la suite des précédentes

Lancement de conteneurs

Gestion de la couche modifiable

- Lister les différences contenues dans la couche modifiable

```
$ docker diff [container id]
/var
C /var/lib
C /var/lib/apt
C /var/lib/apt/lists
A /var/lib/apt/lists/security.ubuntu.com_ubuntu_dists_bionic-security_InRelease
[...]
```

Lancement de conteneurs

Gestion de la couche modifiable

- Sauvegarder la couche modifiable pour créer une nouvelle image
  - Pas la manière recommandée de créer des images

```
$ docker commit 0d3652f5973e diakhate/myimage:v2
```

- Possibilité de pousser ses propres conteneurs sur DockerHub
  - Règle de nommage: [login]/[container_name]:[tag]

```
$ docker login
$ docker push diakhate/myimage:v2
```

- Note sur les tags de version (ici v2)
  - Pas de sémantique particulière, y compris le tag *latest*
    - Simple tag par défaut à la création/selection d'une image
  - Une image peut avoir plusieurs tag

```
$ docker tag myimage:v3 diakhate/myimage:latest
```

Lancement de conteneurs

Gestion de la couche modifiable

- Nettoyer les données inutilisées ou en cache
  - Par défaut des données s'accumulent à chaque lancement de conteneur

```
$ docker container rm [container id]
$ docker container prune # Conteneurs arrêtés
```

```
$ docker image rm [image id]
$ docker image prune # Images inutilisées
```

```
$ docker system prune # Nettoyage complet
```

Implémentation des couches d'images sous Linux

Le système de fichiers **overlay**

- Capable de combiner les données de plusieurs répertoires indépendants
  - N répertoires read-only (lowerdir)
  - 1 répertoire modifiable (upperdir)
  - 1 répertoire temporaire (workdir, doit être dans le même FS que upperdir)
- Utilisation par docker

```
$ mount -t overlay overlay  -o \
      lowerdir=/var/lib/docker/overlay2/[layer id0]/diff:\
               /var/lib/docker/overlay2/[layer id1]/diff:\
               /var/lib/docker/overlay2/[layer id2]/diff:\
      upperdir=/var/lib/docker/overlay2/[layer id3]/diff,\
      workdir=/var/lib/docker/overlay2/[layer id3]/work \
     /var/lib/docker/overlay2/[layer id3]/merged
```

Implémentation des couches d'images sous Linux

Le système de fichiers **overlay**

- Fonctionnement
  - Ouverture d'un fichier en lecture
    - Parcours en profondeur des couches d'image jusqu'à trouver le fichier
  - Ouverture d'un fichier en écriture
    - Si le fichier est dans la couche modifiable, ouverture de ce fichier
    - Sinon copie du fichier depuis la couche précédente le contenant
      - *copy up*: potentiellement coûteux
    - Si aucune couche ne le contient, création dans la couche modifiable
  - Suppression d'un fichier
    - Création d'un fichier spécial dans la couche modifiable
    - Masque les fichiers des couches suivantes

Lancement de conteneurs

Gestion du stockage persistent

- Éviter d'écrire dans la couche modifiable
  - Performances sub-obptimales à cause du copy-on-write
  - Séparer l'application de ses données persistentes
  - Recréer le conteneur sans perdre les données
    - Conteneur jetable
    - Mise à jour en relançant une nouvelle version de l'image
    - Exemple: conteneur de base de données
  - Ne pas stocker de secrets dans une image
    - Éviter une fuite accidentelle

Lancement de conteneurs

Volumes

- Espace persistent attaché à un ou plusieurs conteneurs
  - Permet de partager un dossier entre plusieurs conteneurs
    - Simultanés ou successifs (mise à jour)
  - Offre les performances natives du FS sous-jacent
    - Pas de copy-on-write
  - Stockage géré par docker
    - Plugins permettant de gérer divers systèmes de stockage
- Création d'un volume

```
$ docker volume create dbvolume
$ docker volume ls
$ docker run  -v dbvolume:/var/libmysql --name mysql57 mysql:5.7
$ docker run  --volume-from mysql57 --name mysql80 mysql:8.0
```

Lancement de conteneurs

Réseau

- Docker gère différents plugin réseaux.
- Les plugins de base incluent:
  - null
  - bridge (par défaut)
  - host
  - container
  - macvlan
- L'option **- - net** permet de selectionner un plugin au lancement

Lancement de conteneurs

Réseau *null*

- Pas de réseau disponible dans le conteneur
- Seule l'interface lo est présente
  - Interface propre au conteneur
  - Ne permet de communiquer qu'avec les autres processus du conteneur
- Impossibilité de communiquer avec l'extérieur
- Permet d'isoler un conteneur pour raison de sécurité par exemple

Lancement de conteneurs

Réseau *bridge*

- Le conteneur reçoit une interface **lo** et **eth0**
  - **eth0** est implémentée par une paire **veth**
  - L'autre côté de la paire est connectée à un bridge géré par docker
    - Par défaut docker0
  - Son IP est attribuée dans un subnet privé interne au bridge
  - Le traffic est routé via du NAT
    - Règle iptables MASQUERADE en sortie (~équivalent SNAT)
    - Règle iptables DNAT en entrée
  - Le conteneur peut mettre en place ses propres configuration réseau
    - Routes
    - Regles IPTables
    - etc.

Lancement de conteneurs

Réseau *bridge*

- Création de réseaux additionnels

```
$ docker network create mynet
```

- Lister les différents réseaux

```
$ docker network list
NETWORK ID          NAME                DRIVER              SCOPE
2c98279bf33c        mynet               bridge              local
[...]
```

- Chaque réseau bridge correspond à un périphérique bridge hôte

```
$ ip a
[...]
br-2c98279bf33c: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
link/ether 02:42:9a:87:0e:82 brd ff:ff:ff:ff:ff:ff
inet 172.18.0.1/16 brd 172.18.255.255 scope global br-2c98279bf33c
```

Lancement de conteneurs

Réseau *bridge*

- Exposition de port
  - Les IP alloués au conteneur sont privées
  - Accessible uniquement depuis l'hôte
    - Interface bridge docker0
  - Des ports peuvent être redirigés de l'hôte vers un conteneur
  - Accessible depuis l'exterieur via l'IP hôte

```
$ docker run -d  -p 8080:80 httpd
$ docker ps
CONTAINER ID  IMAGE  COMMAND         STATUS       PORTS                   NAMES
8ac00c87cca4  httpd  "httpd-fore... Up 4 seconds 0.0.0.0:8080->80/tcp  cranky_darwin
$ curl localhost:8080
<html><body><h1>It works!</h1></body></html>
```

Lancement de conteneurs

Réseau *bridge*

- Défintion de noms DNS dynamiques
  - Au sein d'un réseau
  - Basés sur le nom du conteneur

```
$ docker run -d --net mynet --name web httpd
```

- Les autres conteneurs du réseau pourront accéder via les adresses
  - web
  - web.mynet
- Permet de découvrir l'adresse des services dynamiquement
  - Un service par conteneur
- Utilisation d'une adresse différente du nom du conteneur avec l'option **- -net-alias**

Lancement de conteneurs

Réseau *host*

- Aucune isolation réseau n’est appliquée au conteneur
  - Accès direct aux interfaces réseau de l’hôte
  - Peut s’attacher à n’importe quel port réseau
  - Performances natives
    - Pas de traversée de veth, bridge etc.
    - Pas de traduction d’adresse via IPTables
  - La configuration des interfaces reste maîtrisée par l’hôte

Lancement de conteneurs

Réseau *container*

- Le conteneur partage le réseau d'un autre conteneur
  - Même namespace réseau
  - Mêmes interfaces
    - Communication possible à travers l'interface **lo**
  - Partage les interfaces, routes, règles IPtables etc.

Construction d'images

Dockerfiles

- Recette de construction d'image
- Suite d'instructions indiquant
  - Comment construire l'image
  - Comment lancer un conteneur à partir de l'image
    - Commande à exécuter
    - Ports à exposer
    - Volumes à monter
- Un Dockerfile est associé à un contexte
  - Répertoire contenant le Dockerfile
  - Peut contenir des fichiers nécessaire à la construction

Construction d'images

Exemple de Dockerfile

- Fichier Dockerfile simple

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y cowsay
CMD ["/usr/games/cowsay", "Salut",  "!"]
```

- FROM: image à utiliser pour commencer la construction
- RUN: commande (non-interactive) exécutée pour la construction
- CMD: commande par défaut lancée à l'exécution du conteneur

Construction d'images

- Dans le répertoire contenant (uniquement) le Dockerfile

```
$ docker build -t cowsay .
Sending build context to Docker daemon  2.048kB
Step 1/4 : FROM ubuntu
---> 93fd78260bd1
Step 2/4 : RUN apt-get update
---> Running in 74bfed125b00
[...] # Sorties de la commande apt-get update
Removing intermediate container 74bfed125b00
---> ffb71bf8b10a
[...] # Sorties de la commande apt-get install -y cowsay
---> Running in 9fed24d532fb
Removing intermediate container 9fed24d532fb
---> f7af17acd5c1
Step 4/4 : CMD ['/usr/games/cowsay', 'Salut',  '!']
---> Running in f978d29bf5a6
Removing intermediate container f978d29bf5a6
---> b65d477ae004
Successfully built b65d477ae004
Successfully tagged cowsay:latest
```

Construction d'images

Étapes de la construction

```
Sending build context to Docker daemon  2.048kB
```

- Le contexte de construction
  - Répertoire passé en argument à docker build
  - Le répertoire complet est envoyé au démon docker
  - Permet de lancer une construction à distance
  - Ne pas y stocker des fichiers inutiles

Construction d'images

Étapes de la construction

```
Step 2/4 : RUN apt-get update
 ---> Running in 74bfed125b00
[...] # Sorties de la commande apt-get update
Removing intermediate container 74bfed125b00
 ---> ffb71bf8b10a
```

- Un conteneur (74bfed125b00) est créé à partir de l'image de base
- La commande apt-get update y est exécutée
- Le conteneur est sauvegardé dans l'image ffb71bf8b10a
- Le conteneur temporaire (74bfed125b00) est supprimé
- L'image ffb71bf8b10a sera utilisée pour l'étape suivante
- Il peut être utile de limiter le nombre de couches
  - Notamment images déployées en production sur plusieurs hôtes
  - Regrouper plusieurs lignes RUN avec des '&&'

Construction d'images

Étapes de la construction

- Visualisation avec la commande **docker history**

```
$ docker history cowsay
IMAGE               CREATED             CREATED BY
3b85d6a31aa3        24 minutes ago      /bin/sh -c #(nop)  CMD ["/usr/games/cowsay" …
f7af17acd5c1        27 minutes ago      |0 /bin/sh -c apt-get install -y cowsay
ffb71bf8b10a        27 minutes ago      |0 /bin/sh -c apt-get update
93fd78260bd1        2 weeks ago         /bin/sh -c #(nop)  CMD ["/bin/bash"]
<missing>           2 weeks ago         /bin/sh -c mkdir -p /run/systemd && echo 'do…
<missing>           2 weeks ago         /bin/sh -c rm -rf /var/lib/apt/lists/*
<missing>           2 weeks ago         /bin/sh -c set -xe   && echo '#!/bin/sh' > /…
<missing>           2 weeks ago         /bin/sh -c #(nop) ADD file:39e5bc157a8be63bb…
```

Construction d'images

Étapes de la construction

- Mise en cache de chaque image intermédiaire
  - Si on relance la même construction le résultat est instantanné
  - Cache basé sur les chaînes de caractère du Dockerfile
  - Les commandes suivantes sont différentes pour Docker

```
RUN apt-get install httpd nginx
```

```
RUN apt-get install nginx httpd
```

- Il peut parfois être utile de s'affranchir du cache

```
RUN apt-get update
```

- Les paquets ne sont jamais mis à jour après la premiere exécution
  - **docker build –no-cache**

Construction d'images

Résultat

```
$ docker run -ti cowsay

 _________
< Salut ! >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
$ docker run -ti cowsay /usr/games/cowsay Bye !

 _______
< Bye ! >
 -------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Construction d'images

La directive ENTRYPOINT

- Définit une commande par défaut (comme **CMD**)
- Différence: traitement des arguments passés au lancement du conteneur
  - Complète la commande **ENTRYPOINT** au lieu de la remplacer
- **CMD** peut être utilisé simultanément
  - Correspond aux arguments par défaut
- Exemple

```
FROM ubuntu
RUN apt-get update
RUN apt-get install -y cowsay
ENTRYPOINT ["/usr/games/cowsay"]
CMD ["Salut !"]
```

Construction d'images

Résultat

```
docker run -ti cowsay-ep

 _________
< Salut ! >
 ---------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
$ docker run -ti cowsay-ep Cool !

 ________
< Cool ! >
 --------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Construction d'images

La directive EXPOSE

```
EXPOSE 80
EXPOSE 53/udp
```

- Indique les ports à rendre accessible depuis l'extérieur
- Tous les ports sont privés par défaut
- La directive EXPOSE ne fait que donner une information
- Lancement du conteneur avec

```
$ docker run -P -it container
```

- Alloue automatiquement des ports hôte pour chaque port exposé
- **docker ps** ou **docker inspect** permettent de connaître les ports attribués

```
$ docker inspect 2e66 --format \
             '{{(index (index .NetworkSettings.Ports "80/tcp") 0).HostPort}}'
32775
```

Construction d'images

Les directives COPY et ADD

```
COPY . /src
```

- Copie depuis le répertoire contexte vers un répertoire cible du conteneur
  - Compatible avec la mise en cache des couches
  - Verifie si le fichier a changé
- Empêche toute copie hors du répertoire (via ..)

```
ADD [url] ./src
ADD ./data.tar ./src
```

- Similaire à **COPY**
  - Capable de récupérer des fichiers distants
    - Pas de mise en cache possible
  - Décompresser des archives

Construction d'images

La directive VOLUME

```
VOLUME /var/lib/mysql
```

- Crée automatiquement un volume (nommé aléatoirement) au lancement du conteneur

```
$ docker volume list
DRIVER              VOLUME NAME
local               88e7d02e4f3688db2eccb02081f8affaca10a0bf82c16f8f3d504bd2b29c3946
local               52214d63b2487141350a425136cc6b63e296dda75aab17376533561286cbfe88
local               fc71fd0362dfd7b006ff8f223676a26fba175157c2cbf02ad7aa6833ec045ef9
```

- L'utilisateur peut toujours spécifier un volume spécifique avec **-v**

```
$ docker run -ti -v myvolume:/toto  cowsay-ep /bin/bash
```

Construction d'images

La directive ENV

```
ENV HTTP_PROXY http://webproxy.mycompany.com:3128
ENV WEBAPP_PORT 8080
```

- Positionne des variables d'environnement à l'exécution de commandes dans le conteneur
- Possibilité de les surcharger au lancement (ou d'en définir d'autres)

```
docker run -e WEBAPP_PORT=8000
```

Construction d'images

Construction multi-étapes

- Compilation à l'intérieur d'un conteneur
- Permet de compiler son application de façon reproductible
- Inclusion des dépendances de compilation au sein du conteneur
- Générer une image de conteneur pour l'exécution
  - Contenant uniquement les dépendances d'exécution
- Supprimer des fichiers/paquets ne sert à rien
  - Crée une nouvelle couche qui masque les fichiers
  - La couche précédente les contient toujours
- Utilisation de:
  - **FROM image AS**: nommage d'une étape intermédiaire
  - **COPY –from=[image]**: copie de fichier d'une image à lautre

Construction d'images

Construction multi-étapes

- Exemple:

```
FROM ubuntu AS compiler
RUN apt-get update
RUN apt-get install -y build-essential
ADD appsrc.tar /
RUN make -C app
FROM ubuntu
COPY --from=compiler /app/bin/app.exe /app.exe
CMD /app.exe
```

Composition de conteneurs

Docker compose

- Outil externe à docker (ancienemment nommé fig)
  - Ecrit en python
- Lancement reproductible de plusieurs conteneurs formant une application
- Workflow
  - Inclure un ficher docker-compose.yml dans son code source
  - Cloner le dépot de code
  - Démarrer l'application multi-conteneurs

```
$ docker compose up
```

Exemple: Wordpress

```
version: '3.3'
services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress
   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress
volumes:
    db_data:
```

Composition de conteneurs

Syntaxe Compose

- Sections
  - **version**: indique la version de format de fichier Compose
    - Les versions plus récentes supportent plus de fonctionnaliés
  - **services**: images de conteneurs à exécuter
    - **image**: tag d'une image locale ou d'un d'épot
    - **build**: chemin vers un Dockerfile
    - Options de lancement du conteneur (ports, volumes, variables, …)
  - **networks**: optionnel, par défaut utilisation d'un réseau privé à chaque déploiement
  - **volumes**: optionnel, définit des volumes utilisés par les conteneurs
    - Compose réutilise les mêmes volumes lorsque l'on relance l'application

Composition de conteneurs

Quelques commandes utiles

- Choisir un nom de projet unique (par defaut: nom du répertoire)

```
$ docker-compose -p myproject_dev up
```

- Construire les images au lancement de la pile de conteneurs

```
$ docker-compose up --build
```

- Lister les conteneurs de la pile

```
$ docker-compose ps
```

Composition de conteneurs

Quelques commandes utiles

- Tuer les conteneurs de la pile

```
$ docker-compose kill
```

- Supprimer les conteneurs

```
$ docker-compose rm
```

- Tout nettoyer

```
$ docker-compose down -v # Y compris les volumes
```

Merci de votre attention !

Questions ?