Solutions de virtualisation et de conteneurisation

Frédéric Lassabe

FEMTO-ST - DISC Department - OMNI Team

Conteneurisation/Isolation

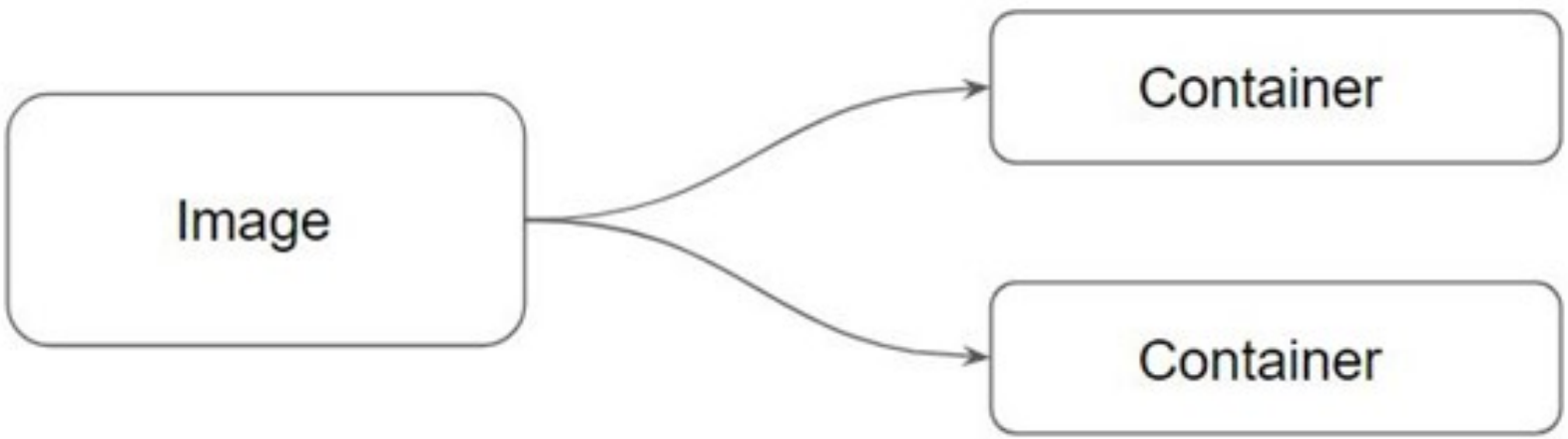
- Exécution d'un processus dans un environnement fermé
- Possibilité de faire tourner plusieurs fois la même application dans un mode multi-instance
- Séparation des ressources de l'hôte
- Pas de virtualisation
- Très performant car peu d'overhead
- Exemples :
  - Proxmox : conteneurs LXC
  - Linux-VServer (isolation des processus en espace utilisateur)
  - BSD Jail (isolation en espace utilisateur)
  - LXC (usage des Cgroups du noyaux Linux)
  - docker
  - podman
- Supports basés sur F. K. Tossou (UTBM/UL)

Docker

- Outil logiciel système ouvert pour faciliter
  - Création
  - Déploiement
  - Exécution d'applications conteneurisées
- Une fois construit
  - Exécuté n'importe où
- Fonctionne sur la majorité des OS PC
  - GNU/Linux
  - Mac
  - Windows
- Instructions d'installation sur le site Docker

Conteneur docker vs Image docker

- Image
  - Fichier avec les dépendances et configurations
  - Pour exécuter une application
- Conteneur
  - Image docker en cours d'exécution
  - Une image peut s'exécuter dans plusieurs conteneurs en même temps
- Relation image/conteneur analogue à classe/objet (instance) en POO.



Une image docker

- Un conteneur docker se base sur une image
- Image en lecture seule
  - Intégrité des données
  - Conservation des dépendances
  - Des configurations
- Un conteneur modifié (erreur ou malveillance) peut être relancé
- Repart sur une instance "propre"

La Création d'image docker

- Soit utiliser des images publiques
  - Notamment Docker hub
  - `https://hub.docker.com`
- Soit en créer soi-même
  - Écriture d'un dockerfile
    - Instructions permettant de créer l'image
    - Une commande pour exécuter ces instructions
  - Sauvegarde d'un conteneur
  - Devient une image pour le reproduire

Le Dockerfile

Un Dockerfile est un document texte qui contient toutes les commandes qu'un utilisateur peut appeler en ligne de commande et qui permet à docker de créer une image

La syntaxe des instructions dans le Dockerfile se présente comme suit :

```
1 # Commentaire
2
3 INSTRUCTION Arguments
```

Le Dockerfile

- Fichier dockerfile par défaut : `Dockerfile`
- Commande pour générer une image : `docker build`

```
docker build .
```

- Pour utiliser un fichier différent
- Option -f, par exemple :

```
docker build -f mon_dockerfile .
```

- Les deux exemples : le fichier est dans "."
- Nommer l'image générée : option -t

```
docker build -t mon_image .
```

Le Dockerfile

- Plusieurs instructions peuvent être présentes dans un dockerfile
- Liste exhaustive dans la doc :
  - `https://docs.docker.com/engine/reference/builder/`
- Nous en verrons quelques unes
- Ordre des instructions
  - Soit défini par la syntaxe dockerfile
    - FROM : première entrée du fichier
    - ENTRYPOINT et CMD : dernière entrée du fichier
  - Soit en fonction de votre environnement

Le Dockerfile

FROM FROM spécifie l'image de base à partir de laquelle la nouvelle image sera créée. Un dockerfile doit obligatoirement commencer par une instruction `FROM`

RUN exécute toutes les commandes dans un nouveau calque au-dessus de l'image actuelle et valide les résultats. L'image validée résultante sera utilisée pour l'étape suivante dans le Dockerfile

CMD Une seule CMD par fichier ! Il s'agit de la commande par défaut que le conteneur exécutera au démarrage

L'instruction CMD

- Définit la commande principale de l'image
- On peut l'écraser en lançant un conteneur avec une commande
- Dans le dockerfile, on utilise la forme exec :

```
CMD ["commande", "param1", "param2"]
```

- Où :

  commande est la commande principale

  param1 est le premier argument

  param2 est le second argument

L'instruction ENTRYPOINT

- Définit la commande principale de l'image
- Ne permet pas le remplacement comme avec CMD
- Dans le dockerfile, on utilise la forme exec :

```
ENTRYPOINT ["commande", "param1", "param2"]
```

- Où :

  commande est la commande principale

  param1 est le premier argument

  param2 est le second argument

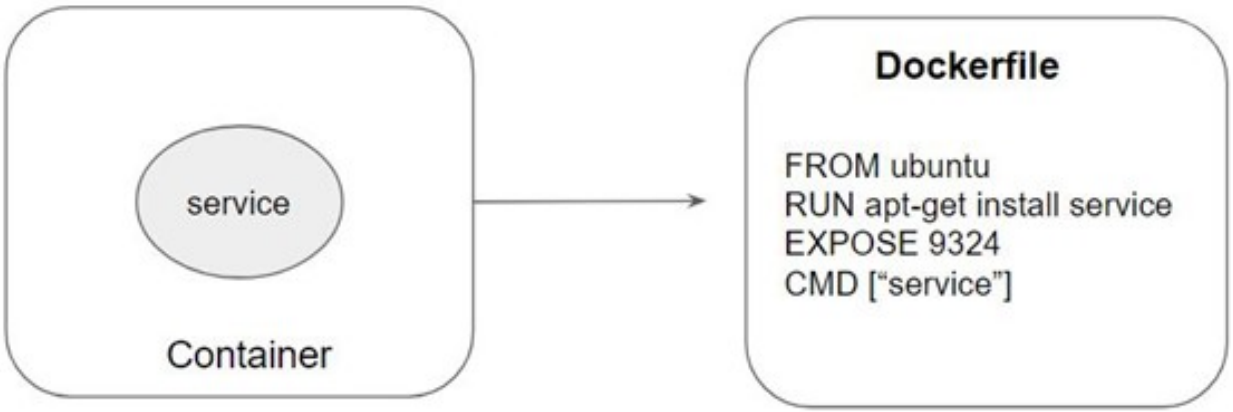
Exemple de ENTRYPOINT

```
1  FROM ubuntu
2
3  ENTRYPOINT ["top", "-b"]
4
5  CMD ["-c"]
```

- Image construite à partir de Ubuntu
- Au démarrage, `top -b -c` sera exécutée
  - `top -b` vient de ENTRYPOINT, non remplaçable
  - `-c` vient de CMD et peut être écrasée

L'instruction EXPOSE

- Informe docker d'un port en écoute sur le conteneur
- La publication n'est pas réelle (le port est ouvert par un service)
- Peut être vu comme une documentation
- Utile lors du déploiement



Instructions COPY et ADD

- Deux instructions avec objectifs similaires (ajout de fichiers)
- COPY
  - COPY prend deux paramètres (la source et la destination, comme cp)
  - Copie d'une source (la machine qui a lancé `docker build`)
  - Vers une destination (dans l'arborescence de l'image)
- ADD
  - Fait pareil
  - Mais la source peut être une URL ou une archive
  - Déconseillé pour les URL (préférer curl ou wget)

Exemple avec ADD

- Préférer ceci :

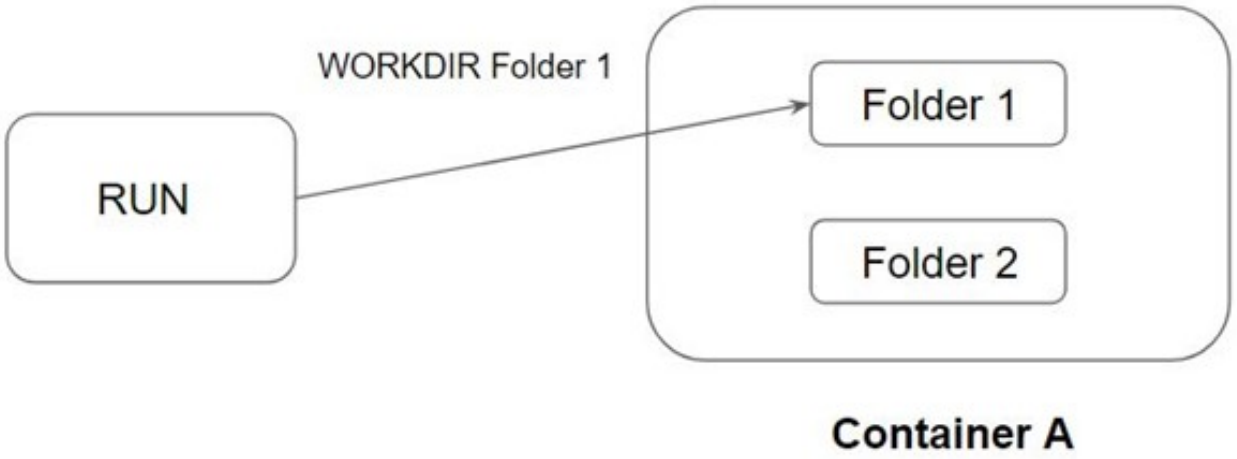
```
RUN mkdir -p /usr/dest \
    && curl -SL http://exple.com/file.tar.xz \
    | tar -xJC /usr/dest \
    && make -C /usr/dest all
```

- Plutôt que ceci :

```
ADD http://exple.com/file.tar.xz /usr/dest/
RUN tar -xJf /usr/dest/file.tar.xz -C /usr/dest
RUN make -C /usr/dest all
```

L'instruction WORKDIR

- Définit le répertoire de travail
- Peut être utilisée plusieurs fois
- Équivalente à `cd` en shell
- S'applique aux commandes qui la suivent dans le dockerfile :
  - RUN
  - CMD, ENTRYPOINT
  - COPY, ADD



Exemple avec WORKDIR

```
WORKDIR /a
WORKDIR b
WORKDIR c
RUN pwd
```

Le résultat de l'instruction `RUN pwd` sera le suivant : `/a/b/c`

L'instruction ENV

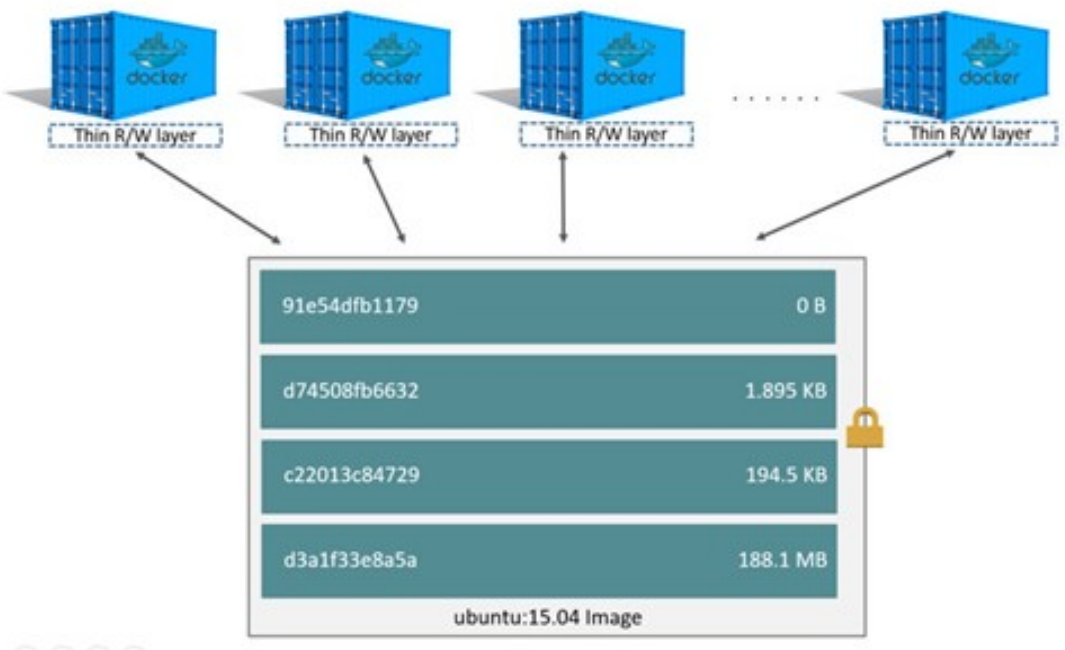
- Définit une variable d'environnement
- `ENV KEY value` définit
  - La variable d'environnement `KEY`
  - à la valeur *value*
- Exemple :

```
ENV NGINX 1.2
RUN curl -SL http://example.com/web-$NGINX.tar.xz
RUN tar -xzvf web-$NGINX.tar.xz
```

- On peut le faire à l'exécution avec les options (écrase éventuellement celles du dockerfile)
  - `-e`
  - `-env`
  - `-env-file`

Les couches d'une image Docker

La principale différence entre un conteneur et une image est la couche inscriptible supérieure



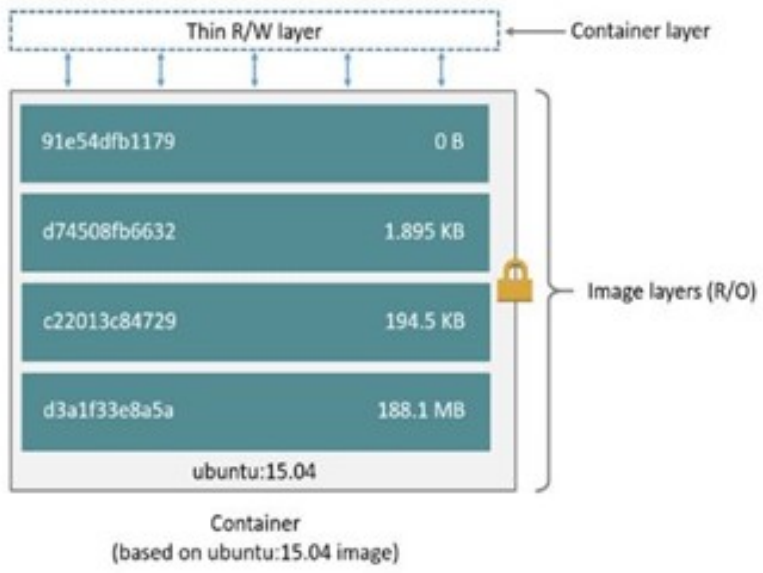
Toutes les écritures dans le conteneur qui ajoutent de nouvelles données ou modifient des données existantes sont stockées dans cette couche inscriptible.

Les couches d'une image Docker

Une image Docker est constituée d'une série de calques/couches

Voici un exemple :

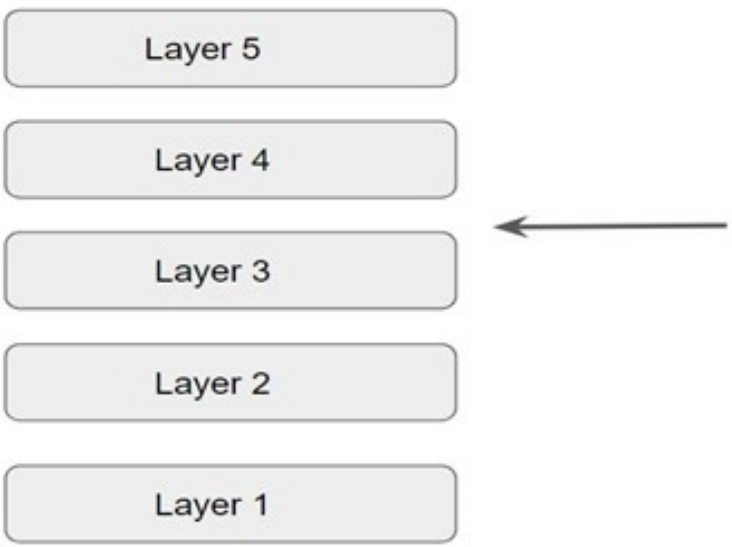
```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```



Les couches d'une image Docker

Chaque couche représente une instruction dans le Dockerfile de l'image

Voici un exemple :



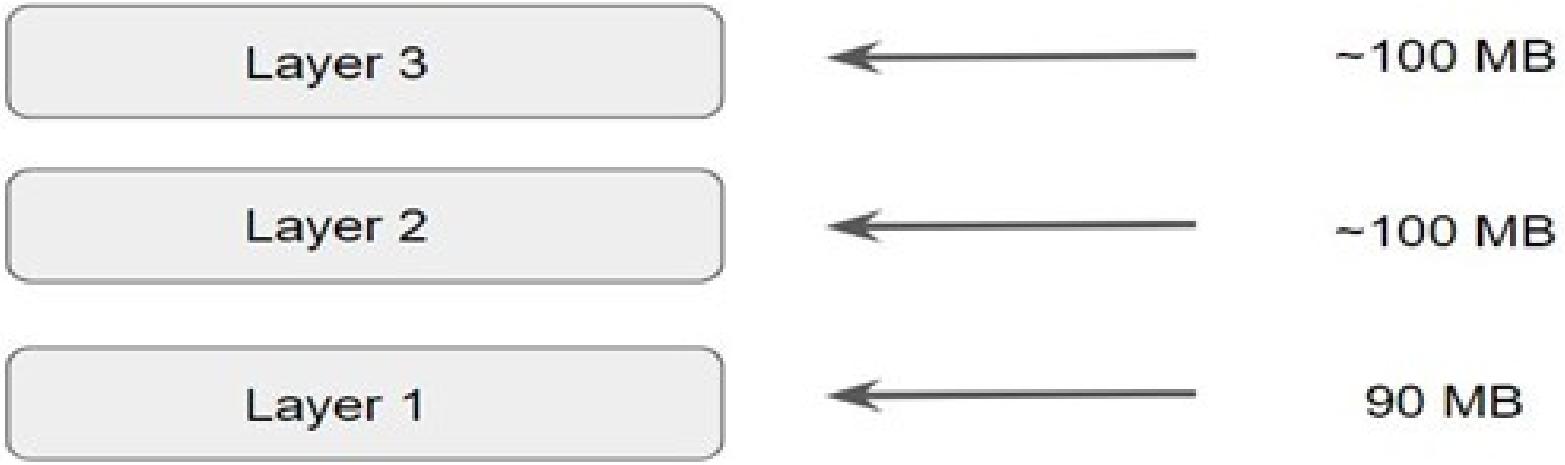

```
FROM ubuntu
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
RUN rm -f /root/file1.txt

RUN rm -f /root/file2.txt
```

Les couches d'une image Docker

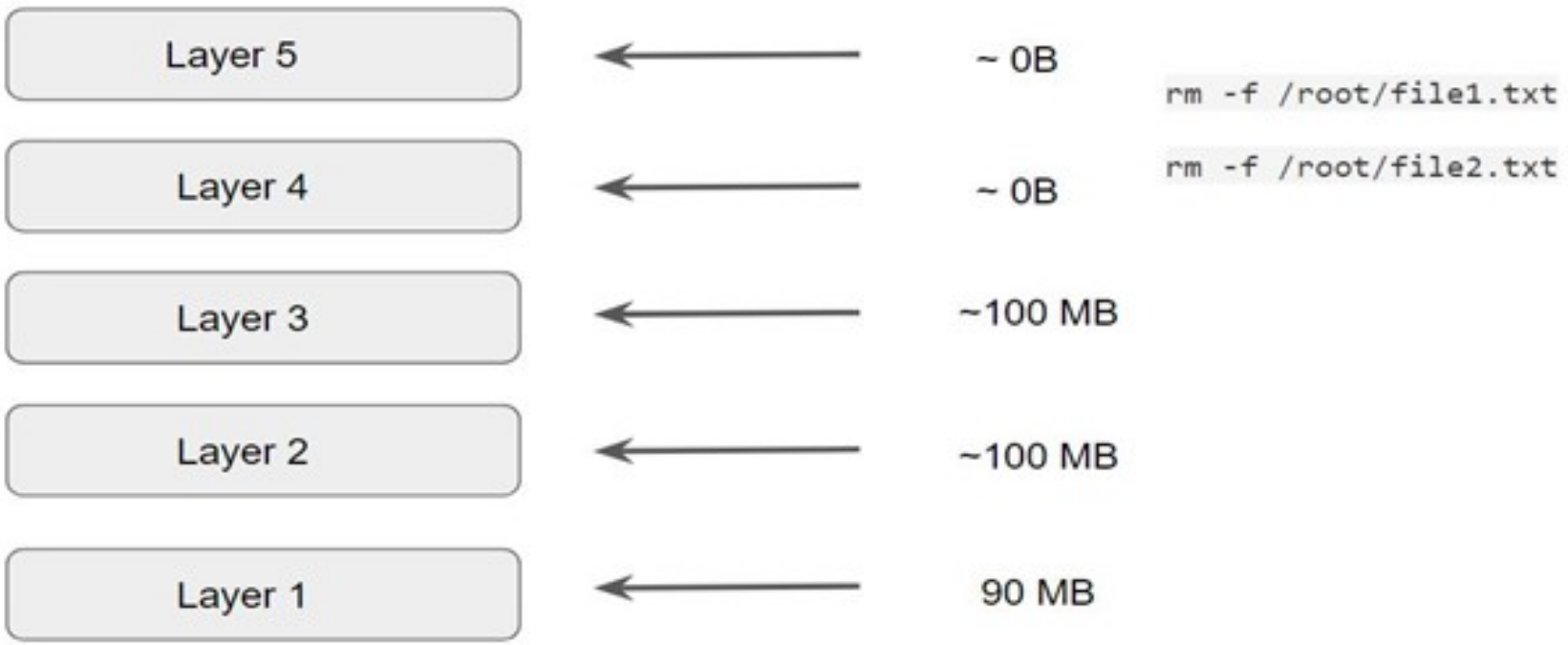
En analysant les trois premières instructions, nous pouvons en déduire la taille des couches

```
FROM ubuntu
RUN dd if=/dev/zero of=/root/file1.txt bs=1M count=100
RUN dd if=/dev/zero of=/root/file2.txt bs=1M count=100
```



Les couches d'une image Docker

Ensuite en analysant les deux dernières instructions, nous déduirons la taille des deux dernières couches



Les couches d'une image Docker

- Finalement, l'image fera 290 MB environ
- Pourtant les fichiers ont été supprimés
- Comportement à prendre en compte dans l'écriture des dockerfiles
  - Éviter de perdre de la place inutilement
- Voir les couches d'une image :

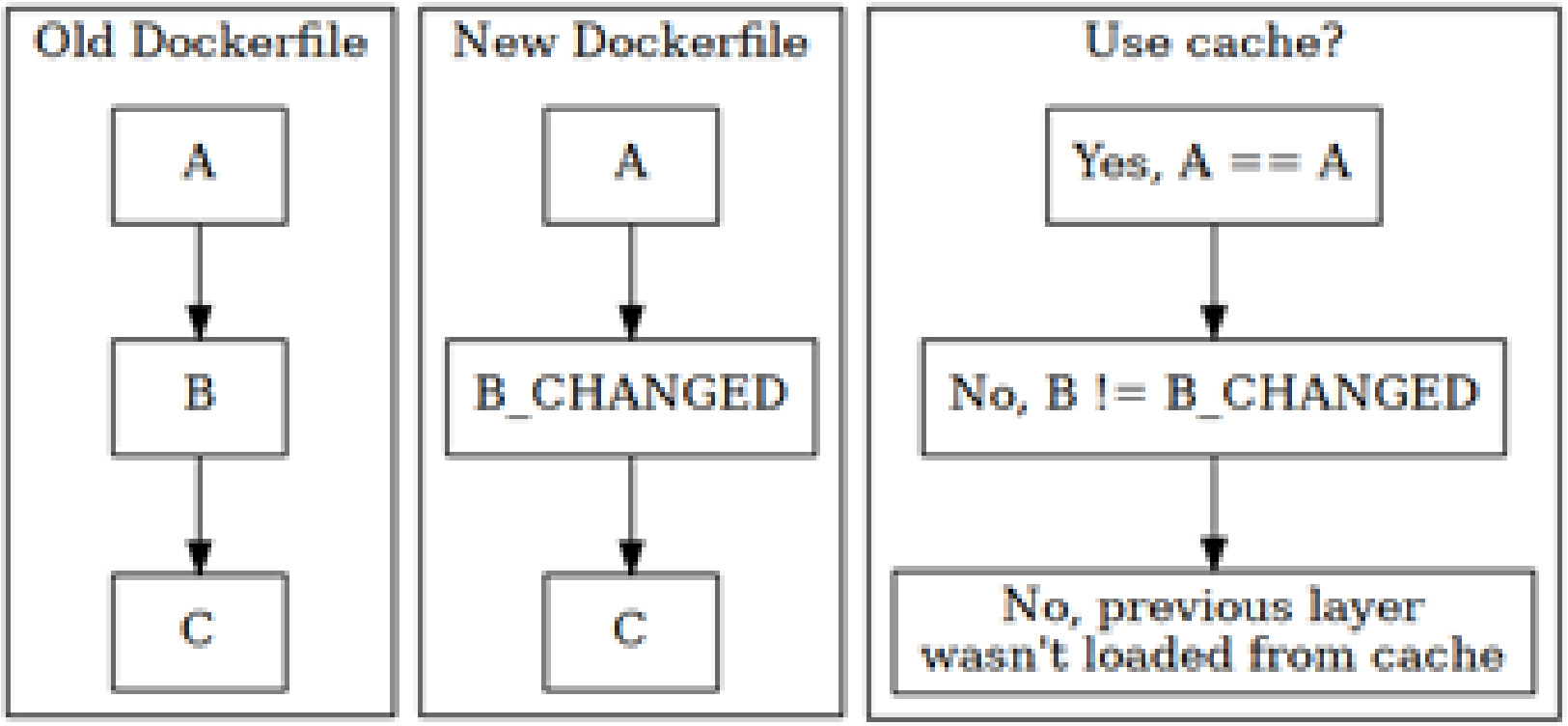
```
docker image history <nom_image>
```

Pourquoi ce système de couches ?

- Permettre à docker de réutiliser les couches existantes
- Deux dockerfiles avec des instructions exactement dans le même ordre :
  - réutilisation des couches d'une image pour l'autre
  - Gain d'espace (déduplication)

Les couches d'une image Docker

- Remarque :
  - Après des instructions différentes
  - Les couches ne sont plus réutilisables
  - Car la base diffère



Aplatissement d'une image Docker

- Généralement, plus il y a de calques
- Plus l'image est grosse
- On peut atteindre plusieurs GB, voire plusieurs dizaines de GB
- L'aplatissement de l'image peut réduire la taille globale
- Comment aplatir une image ?
  - Exporter l'image avec `docker export`
  - Importer cette image comme une nouvelle avec `docker import`
- L'aplatissement restitue une image avec un seul calque
- Le processus consiste à
  - Exporter le résultat des couches dans un .tar
  - Importer ce dernier comme une nouvelle image

Mise en œuvre de l'aplatissement

- Étapes à suivre :
  1. Créer un conteneur basé sur l'image à aplatir
  2. Exporter le conteneur vers un fichier tar
  3. Importer le fichier tar
- Exemple :

```
docker container run \
        --name nomconteneur nomimage
docker export --output=nomconteneur.tar \
        nomconteneur
cat nomconteneur.tar | docker import \
        - nouvelleimage:imported
```

Suppression d'images inutilisées

- Des images peuvent devenir inutiles
  - Temporaires
  - Tests
  - Versions obsolètes
- Occupent inutilement de l'espace
- Deux façons de supprimer les images
  - `docker prune` : supprime toute image sans balise ni référencée par un conteneur
  - `docker rmi <image>` : supprime une image par son nom ou son ID

Inspection d'une image docker

- Chaque images docker contient de nombreuses informations
- Par exemple :
  - La date de création
  - Les variables d'environnement
  - L'architecture
  - Le SE
  - La Taille
- Inspecter une image (nommée `nom_image`) avec la commande :

```
docker image inspect <nom_image>
```

Instruction HEALTHCHECK

- Permet la surveillance de l'état de l'application
- Quand docker démarre un conteneur, il surveille le processus
- Le processus se termine : le conteneur se ferme
- Vérification basique
- Pas de détails sur la fermeture
  - Fin normale ?
  - Crash ?
  - Demande de fermeture ?

Exemple de HEALTHCHECK

```
HEALTHCHECK --interval=5s CMD ping -c 1 172.17.0.2
```

Les options :

–interval=DUREE par défaut la valeur de ce paramètre est 30 secondes

–timeout=DUREE par défaut la valeur de ce paramètre est 30 secondes

–start-period=DUREE par défaut la valeur de ce paramètre est 60 secondes

–retries=N par défaut la valeur de ce paramètre est 3

Le Docker commit

- Seconde façon de créer une image
- En se basant sur un conteneur en exécution
- Permet de figer des modifications du conteneur
  - En fait une nouvelle image
  - Contenant les modifications
- Mise en pause pendant le commit

Le Docker commit

- Repose sur la commande `docker container commit`
- Syntaxe requise :

```
docker container commit CONTAINER-ID myimage01
```

- Option `--change`
  - Permet d'appliquer des instructions dockerfile à l'image créée
  - Instructions prises en compte (doc à lire, compte dans le cours) :
    - CMD et ENTRYPOINT
    - ENV
    - EXPOSE
    - WORKDIR
    - LABEL
    - ONBUILD
    - USER
    - VOLUME

Partage d'une image

- Une image est créée localement
- La partager est généralement nécessaire
- On peut par exemple l'envoyer sur un support partagé
- Sauvegarde d'images dans un .tar :

```
docker image save myapp -o myapp.tar
```

Partage d'une image

- À destination, charger l'image ou les images :

```
docker image load -i myapp.tar
```

Le Registre Docker

- Un registre est une application serveur
- Stocke et distribue des images docker
- Il en existe des publics (docker hub par exemple)
- Des registres de confiance
- Des registres privés

Le Registre Docker

- Envoi vers un registre :
  1. Authentification du client Docker auprès du registre Docker. Il faut déjà avoir créé son compte sur le docker hub. Ensuite s'authentifier dans le terminal avec la commande `docker login`
  2. Etiquetage de l'image Docker avec le référentiel de registre et l'étiquette d'image optionnelle

```
docker tag busybox 5066/mydemo:v1
```

  3. Envoyer l'image avec la commande `docker push`

```
docker push 5066/mydemo:v1
```

Filtrer les images

- La recherche dans un registre peut être complexe
- Exemple de Docker Hub et des milliers d'images
  - Certaines officielles
  - D'autres non
- Recherche dans un registre avec `docker search`
- Voir `docker search --help`

Exemples de filtres

`docker search nginx` Rechercher une image Nginx

`docker search nginx –limit 5` Rechercher une image Nginx avec un maximum de 5 résultats

`docker search –filter is-official=true nginx` Afficher juste les images officielles

Etiquetter une image Docker

- Les étiquettes donnent des informations d'images
  - Version
  - Variante
- Alias d'un ID (une suite hexa peu pratique, par ex. : `8f5487c8b942`)

```
[root@docker-demo ~]# docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
alpine              latest              caf27325b298        15 hours ago        5.53MB
<none>              <none>              7c116aacaae3        17 hours ago        172MB
```

- Étiquette (ou "nom") avec -t

```
1 docker build -t femto/predictops .
```

Questions

Merci de votre attention

Questions ?