

DOCKER

DOCKERFILE



INTRODUCTION

- Comme vu jusqu'à présent, la préparation d'une image "*from scratch*" peut être un processus long
- Il peut être utile de souhaiter réutiliser une image "*préparée*" pour ne pas avoir à refaire
 - Les mises à jour
 - Les installations d'outils/utilitaires
 - Le paramétrage des volumes et/ou networks

INTRODUCTION

Nous avons deux options :

- Créer une image à partir d'un conteneur modifié (docker commit)
- Créer une image à partir d'un script (docker file)

DOCKER COMMIT

PRINCIPE

- On utilise une image comme vu jusqu'à présent
- On prépare notre image avec les paramètres souhaités
- Une fois la préparation faite, on réalise un commit

DOCKER COMMIT

COMMANDE

```
docker commit [OPTIONS] CONTAINER <REPOSITORY[:TAG]>>
```

La commande va créer une nouvelle image qui sera disponible dans la liste des images

DOCKER COMMIT

DÉMONSTRATION

Création d'une image ubuntu custom

demo

DOCKERFILE

PRINCIPE

- Un dockerfile est un script qui permet de construire des images.
- Une image est un modèle permettant de construire des conteneurs

- Nous allons pouvoir utiliser une image de base (ubuntu, alpine ...)
- Y ajouter autant de layers que souhaités
- Définir les options lors de la création du conteneur (volumes, networks ...)

- Toutes les options de création sont passées au travers d'instructions
- Il existe beaucoup d'instructions que l'on peut retrouver sur la [doc officielle](#)

DOCKERFILE

INSTRUCTIONS COURANTES

- 1 **FROM** # Définit l'image de base (parent)
- 2 **LABEL** # Ajoute des métadatas
- 3 **ARG** # Variables utilisables uniquement par le dockerfile
- 4 **ENV** # Variables utilisables par le dockerfile ET les conteneurs créés
- 5 **RUN** # Exécute une commande lors de la création de l'image
- 6 **COPY** # Permet de copier des fichiers
- 7 **ADD** # Idem COPY, mais permet de décompresser les archives
- 8 **ENTRYPOINT** # définit la commande de démarrage du conteneur
- 9 **CMD** # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
- 10 **WORKDIR** # Définit le répertoire de travail utilisé au lancement par défaut
- 11 **EXPOSE** # Définit le port exposé
- 12 **VOLUME** # Définit le volume du conteneur
- 13 **USER** # Désigne l'utilisateur par défaut (root si non spécifié)

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l'image de base (parent)
2 LABEL # Ajoute des métadonnées
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l'image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l'utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l'image de base (parent)
2 LABEL # Ajoute des métadonnées
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l'image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l'utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l image de base (parent)
2 LABEL # Ajoute des métadatas
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l image de base (parent)
2 LABEL # Ajoute des métadatas
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l image de base (parent)
2 LABEL # Ajoute des métadatas
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de tableau
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l'image de base (parent)
2 LABEL # Ajoute des métadonnées
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l'image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l'utilisateur par défaut (root si non spécifié)
```


DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l'image de base (parent)
2 LABEL # Ajoute des métadonnées
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l'image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l'utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l image de base (parent)
2 LABEL # Ajoute des métadatas
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de tableau
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l image de base (parent)
2 LABEL # Ajoute des métadatas
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de tableau
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l'image de base (parent)
2 LABEL # Ajoute des métadonnées
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l'image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l'utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l'image de base (parent)
2 LABEL # Ajoute des métadonnées
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l'image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de liste
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l'utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

INSTRUCTIONS COURANTES

```
1 FROM # Définit l image de base (parent)
2 LABEL # Ajoute des métadatas
3 ARG # Variables utilisables uniquement par le dockerfile
4 ENV # Variables utilisables par le dockerfile ET les conteneurs créés
5 RUN # Exécute une commande lors de la création de l image
6 COPY # Permet de copier des fichiers
7 ADD # Idem COPY, mais permet de décompresser les archives
8 ENTRYPOINT # définit la commande de démarrage du conteneur
9 CMD # Spécifie les arguments envoyés aux ENTRYPOINT sous forme de tableau
10 WORKDIR # Définit le répertoire de travail utilisé au lancement par défaut
11 EXPOSE # Définit le port exposé
12 VOLUME # Définit le volume du conteneur
13 USER # Désigne l utilisateur par défaut (root si non spécifié)
```

DOCKERFILE

.DOCKERIGNORE

- Comme pour git avec le fichier .gitignore
- Permet de définir les fichiers à ignorer lors d'une COPY / ADD
- Utile lorsqu'on a une app à copier et qu'on veut ignorer les node_modules

DOCKERFILE

EN ÉTAPES

1. On constitue notre dockerfile dans un répertoire dédié,
2. Optionnel : on ajoute notre .dockerignore
3. Si nécessaire, on ajoute les fichiers que l'on souhaite injecter dans le répertoire
4. On exécute la commande build

BEST PRACTICES

- On utilise une image de base la plus légère possible
- Chaque instruction ajoute un layer à l'image
- Compter 1 layer par instruction

LA COMMANDE BUILD

```
docker build [OPTIONS] <PATH> | URL | -
```

Options communes :

- **-t** : ajoute un tag (nom) à l'image
- **-q** : mode quiet (ne prompt pas les étapes)
- La doc [ici](#) !

NB : L'argument 'PATH' est obligatoire !
Si le dockerfile se trouve dans le même répertoire, on
indique juste '.'

DÉMONSTRATION

Une app nodejs basique

DOCKERFILE

DOCKERHUB

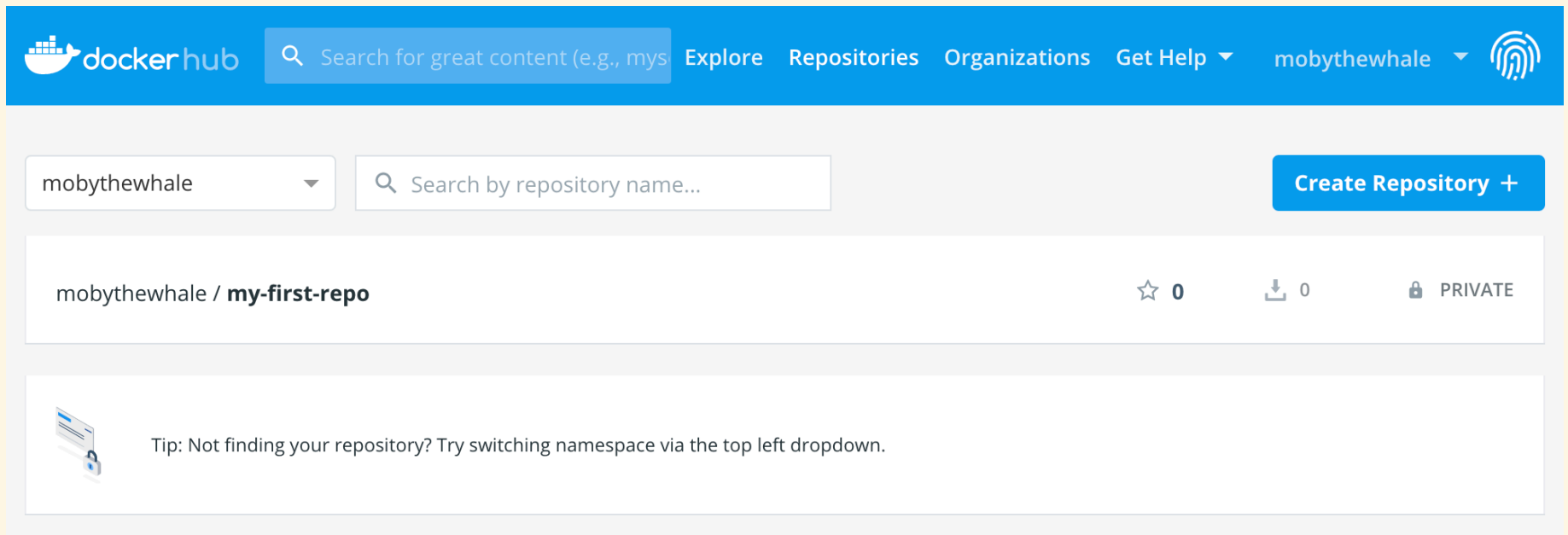


ENVOYER UNE IMAGE EN LIGNE

- Comme github qui permet d'envoyer du code en ligne,
- DockerHub permet d'envoyer ses images en ligne
- Chaque utilisateur peut créer un 'repository' pour envoyer ses images.

La doc [ici](#) !

Il est très facile de créer un repository sur dockerhub



The screenshot shows the Docker Hub web interface. At the top is a blue navigation bar with the Docker Hub logo, a search bar, and links for Explore, Repositories, Organizations, Get Help, and a user profile for 'mobythewhale'. Below the navigation bar, there is a section for creating a repository. It features a dropdown menu with 'mobythewhale' selected, a search bar with the placeholder 'Search by repository name...', and a prominent blue button labeled 'Create Repository +'. Below this, a card displays 'mobythewhale / my-first-repo' with 0 stars, 0 downloads, and a lock icon indicating it is 'PRIVATE'. At the bottom, a tip box with a document icon states: 'Tip: Not finding your repository? Try switching namespace via the top left dropdown.'

CONTRAINTES DES NAMESPACES

- Le nom du repository peut contenir entre 2 et 255 caractères,
- Minuscules, chiffres, - et _ uniquement,
- La description ne peut pas dépasser 100 caractères,
- Peut être lié à un github/bitbucket

ENVOYER SON IMAGE SUR LA TOILE

Une fois l'image créée, on peut facilement l'envoyer sur le repository :

```
docker push <hub-user>/<repo-name>:<tag>
```

IMAGE EXISTANTE

S'il s'agit d'une image existante, il faut mettre à jour son tag

```
docker tag <nomImage>:<tagImage> <userName>/<repo>:<tag>  
docker push <userName>/<repo>:<tag>
```

RÉCUPÉRER L'IMAGE

On pourra facilement récupérer l'image avec :

```
docker pull <userName>/<repo>:<tag>
```

VOIR LES LAYERS

Une fois l'image en ligne, on peut consulter les différentes couches de l'image



mobythewhale / my-first-repo : latest

DIGEST: sha256:e4355b66995c96b4b468159fc5c7e3540fcef961189ca13fee877798649f531a

SIZE	ARCHITECTURE	OS	PUSHED
2.66 MB	amd64	linux	10 minutes ago

IMAGE HISTORY ?

1	ADD file ... in /	2.66 MB
2	CMD ["/bin/sh"]	0 B

Command

```
ADD file:fe1f09249227e2da2089afb4d07e16cbf832eeb804120074acd2b8192876cd28 in /
```

DOCKERFILE

BONNES PRATIQUES : LA PHILOSOPHIE MICRO-SERVICES



LES CONTENEURS ÉPHÉMÈRES

- Un conteneur n'a pas pour objectif de perdurer
- Il sera remplacé par un autre, basé sur la même image
- On va donc utiliser des volumes pour conserver les datas

LE CONTEXTE

- On appelle "*contexte*" le répertoire à partir duquel on build l'image
- Ce répertoire doit contenir tout ce qui va servir à notre app
- Docker sera incapable de récupérer des éléments en dehors du contexte !
- On y insérera un `.dockerignore`

WATCH LES LAYERS

- Faire toujours en sorte d'avoir un minimum de layers
- Les commandes RUN, COPY et ADD créent des couches

RUN : LES ARGUMENTS MULTI-LIGNES

- Il est conseillé de passer les arguments RUN en mode multi-lignes, comme suit :

```
RUN apt-get update && apt-get install -y \  
    bzip2 \  
    cvs \  
    git \  
    mercurial \  
    subversion
```

- Il est recommandé de les trier par ordre alphabétique pour une meilleure relecture

BEST PRACTICES - BAD LAYERING

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y \
    curl \
    wget \
    vim
RUN rm -rf /var/cache/apk/*

VOLUME /config /data

EXPOSE 80 443

CMD ["--datadir=/config", "--nolaunch"]

ENTRYPOINT ["/usr/bin/env", "python2", "/sickrage/SickBeard.py"]
```

BEST PRACTICES - GOOD LAYERING

```
FROM ubuntu:latest
RUN apt-get update && apt-get install -y \
    curl \
    wget \
    vim \
    && rm -rf /var/cache/apk/*

VOLUME /config /data

EXPOSE 80 443

CMD ["--datadir=/config", "--nolaunch"]

ENTRYPOINT ["/usr/bin/env", "python2", "/sickrage/SickBeard.py"]
```

VIDER LE CACHE APT

- Penser à combiner `apt update` et `apt upgrade` avec `&&`
- Procéder ainsi évite les problèmes de cache et versions de paquets

ADD OU COPY ?

- On préférera COPY plutôt que ADD
- ADD possède plus d'options, et permet d'extraire des archives ou de copier depuis une URL
- On ne l'utilisera que lorsque nécessaire

OPTIMISATION

- On peut faire appel au concept de multi-stage build
- On peut intégrer le build d'une image dans un docker-compose (voir prochain chapitre)

OPTIMISATION - MULTI-STAGE BUILD

- On peut utiliser plusieurs FROM dans un Dockerfile.
- L'avantage ici est de pouvoir utiliser une image de base très légère pour le build, et une image plus lourde pour l'exécution.
- On peut ainsi réduire la taille de l'image finale.

- Exemple de multi-stage build :

```
FROM golang:1.12-alpine AS builder
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN apk --no-cache add git && go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=builder /go/src/github.com/alexellis/href-counter/app .
CMD [ "./app" ]
```


- L'exemple ci-dessus va créer une image de base `golang:1.12-alpine` qui va servir à compiler le code.
- Une fois le code compilé, on va créer une image `alpine:latest` qui va servir à l'exécution.
- En faisant appel à `COPY --from=builder`, on va copier le binaire compilé dans l'image d'exécution (image finale).
- En gros, on va créer une image de base, puis une image finale qui va être plus légère.

À VOUS DE JOUER

Réalisez l'exercice 1

À VOUS DE JOUER

Réalisez l'exercice 2

La suite !