Construction d'image avec Dockerfile

Dockerfile et layers

Les images Docker fonctionnent par couches successives d'instructions, **les layers**.

Un **Dockerfile** est un fichier texte qui décrit les différentes couches d'une image Docker.

Chaque couche permet d'ajouter des actions à l'image, par exemple :

- Installation de dépendances
- Configuration d'un environnement
- Copie de fichiers depuis la machine hôte
- Définition de la commande lancée au démarrage du conteneur
- Etc.

> Le dockerfile est comparable à une recette de cuisine : chaque instruction du Dockerfile représente une étape dans la préparation de l'environnement logiciel souhaité.

Son utilisation offre certains avantages :

- **Automatisation** : chaque build est reproductible et scripté.
- **Traçabilité** : le fichier peut être versionné avec Git.
- **Portabilité** : un Dockerfile peut être utilisé sur n'importe quelle machine.
- **Modularité** : il permet de construire des images à partir d'autres images, facilitant la réutilisation.

Ci-dessous un exemple de Dockerfile :

```
FROM debian:trixie # point de départ : une image existante
RUN apt-get update -y # commande à éxécuter
RUN apt-get install fastfetch -y # une autre commande, pour une nouvelle couche
```

```
ENTRYPOINT ["fastfetch"] # point d'entrée au lancement du conteneur
```

Les instructions d'un Dockerfile

> N'hésitez pas à consulter la documentation de Docker à ce sujet : Dockerfile reference.

`FROM` - Définir l'image de base

```
FROM ubuntu:24.04
```

C'est le point de départ de l'image : une autre image de base officielle ou personnalisée.

`LABEL` - Ajouter des métadonnées

```
LABEL Maintainer="Thibaud FRICHET"
LABEL Description="Description de l'image"
```

Les métadonnées de l'image s'affichent avec `docker image inspect`.

`WORKDIR` - Définir le répertoire de travail

```
WORKDIR /var/www/html
```

Les instructions `RUN`, `CMD` et `ENTRYPOINT` s’exécutent dans le répertoire de travail.

`COPY` - Copier des fichiers locaux dans l'image

```
# copie fichier.txt à partir du répertoire courant de la machine hôte
# dans le répertoire courant de l'image (défini avec WORKDIR)
COPY fichier.txt ./

# copie le répertoire sources depuis la machine hôte vers le répertoire /var/www de l'image
COPY /home/thibaud/projet/sources/ /var/www/
```

`RUN` - Exécuter des commandes pendant la construction

```
# installation et lancement de Redis
RUN apt-get update -y
RUN apt-get install redis-server -y
RUN service redis-server start
```

Les commandes `RUN` s’exécutent pendant la construction de l'image, pas au démarrage d'un conteneur.

> Les commandes doivent se terminer sans saisie utilisateur.

`EXPOSE` - Indiquer les ports utilisés par l'application

```
EXPOSE 80
EXPOSE 443
```

Ce sont les ports d'écoute du conteneur, ils peuvent être redirigés avec `docker run -p 90:80` à l'exécution.

`CMD` - Définir les commandes par défaut à exécuter

L'instruction CMD accepte un tableau en paramètre.

```
CMD ["date", "echo Bonjour"] # affiche la date courante, puis "Bonjour"
```

Contrairement à `RUN`, l'instruction `CMD` est exécutée au démarrage du conteneur, pas pendant la construction.

`ENTRYPOINT` - Définir le point d'entrée du conteneur

```
ENTRYPOINT ["service nginx start"] # lance le serveur web nginx
```

`CMD` et `ENTRYPOINT` définissent les commandes exécutées au démarrage d'un conteneur, mais avec un comportement différent :

- `CMD` indique la commande avec des arguments par défaut qui peuvent être remplacés lors de l’exécution avec docker run.
- `ENTRYPOINT` fixe le point d’entrée du conteneur, rendant son comportement plus rigide.

Lorsqu’ils sont utilisés ensemble, `ENTRYPOINT` définit le programme à exécuter et `CMD` les arguments par défaut.

> Dans la majorité des cas, vous pouvez utiliser l'un ou l'autre uniquement.

Bonnes pratiques

- Utilisez des images légères. Exemples : `python:3.10-slim`, `nginx-stable-alpine`, etc.

- Utilisez des versions fixes dans vos dépendances. Évitez les tags `-latest` par exemple.

- Minimisez le nombre de couches, par exemple avec plusieurs commandes sur une ligne `RUN` et `&&`.

  ```
  RUN apt-get update -y && apt-get install redis-server
  ```

- Utilisez `.dockerignore` pour exclure les fichiers inutiles, de la même manière qu'un `.gitignore`.
  Le contenu du répertoire courant de la machine hôte est envoyé à Docker lors de la construction d'une image. Docker appelle cela le `context`.

- Respectez la convention de nommage d'un dockerfile :
  - Par défaut : `Dockerfile`
  - Avec un préfixe : `dev.Dockerfile`, `prod.Dockerfile`, `app-v3.Dockerfile`

La construction de l'image

La construction d'une image docker s'effectue avec `docker build`.

Le paramètre `-t`, `--tag` indique le nom de l'image et sa version. Le paramètre `-f`, `--file` le nom du fichier dockerfile, par défaut `Dockerfile`.

Un répertoire de contexte doit être indiqué, par exemple le répertoire courant avec `.`.

```
docker build -t mon_appli:1.0.0 -f monappli.Dockerfile .
```

```
debian@debian:~/docker_build_exemple$ cat Dockerfile
FROM debian:trixie
RUN apt-get update -y && apt-get install fastfetch -y
ENTRYPOINT ["fastfetch"]
debian@debian:~/docker_build_exemple$ docker build -t mon_appli:1.0.0 .
[+] Building 10.0s (6/6) FINISHED
 => [internal] load build definition from Dockerfile
 => => transferring dockerfile: 135B
 => [internal] load metadata for docker.io/library/debian:trixie
 => [internal] load .dockerignore
 => => transferring context: 2B
 => [1/2] FROM docker.io/library/debian:trixie@sha256:6d87375016340817ac2391e670971725a9981cfc24e221c47734681ed0f6c0f5
 => [2/2] RUN apt-get update -y && apt-get install fastfetch -y
 => exporting to image
 => => exporting layers
 => => writing image sha256:080ef57bfee7674fd7793b71e26d6c42f8bdcc2463e585a087e7bf921d7bf585
 => => naming to docker.io/library/mon_appli:1.0.0
```

```
debian@debian:~/docker_build_exemple$ docker image ls
REPOSITORY                TAG          IMAGE ID       CREATED              SIZE
mon_appli                 1.0.0        080ef57bfee7   About a minute ago   144MB
portainer/portainer-ce    lts          79b66b8e24d3   5 days ago           186MB
debian                    bookworm     3d2058890b96   3 weeks ago          117MB
debian                    latest       047bd8d81940   3 weeks ago          120MB
hello-world               latest       1b44b5a3e06a   3 weeks ago          10.1kB
php                       8.4-apache   27a77f7fe238   3 weeks ago          511MB
ubuntu                    24.04        e0f16e6366fe   4 weeks ago          78.1MB
ubuntu                    latest       e0f16e6366fe   4 weeks ago          78.1MB
busybox                   latest       0ed463b26dae   11 months ago        4.43MB
strm/helloworld-http      latest       29acbf1818de   7 years ago          158MB
debian@debian:~/docker_build_exemple$ docker run -ti mon_appli:1.0.0
        _,met$$$$$gg.          root@4f1071111007
     ,g$$$$$$$$$$$$$$$P.       ----------------
   ,g$$P""       """Y$$.".     OS: Debian GNU/Linux 13 (trixie) x86_64
  ,$$P'              `$$$.     Host: KVM/QEMU Standard PC (i440FX + PIIX, 1996) (pc-i440fx-2.2)
',$$P       ,ggs.     `$$b:   Kernel: Linux 6.12.41+deb13-amd64
`d$$'     ,$P"'   .    $$$    Uptime: 1 day, 30 mins
 $$P      d$'     ,    $$P    Packages: 80 (dpkg)
 $$:      $$.   -    ,d$$'    Shell: bash 5.2.37
 $$;      Y$b._   _,d$P'      Display (Virtual-1): 1024x768
 Y$$.    `.`"Y$$$$P"'         Terminal: xterm
 `$$b      "-.__              Terminal Font: fixed (8.0pt)
  `Y$$b                       CPU: AMD Ryzen Embedded V1500B (4) @ 2.20 GHz
   `Y$$.                      GPU: Unknown Device 1111 (VGA compatible)
     `$$b.                    Memory: 354.06 MiB / 966.56 MiB (37%)
       `Y$$b.                 Swap: 0 B / 2.00 GiB (0%)
         `"Y$b._              Disk (/): 2.86 GiB / 95.90 GiB (3%) - overlay
             `"""             Local IP (eth0): 172.17.0.3/16
                              Locale: C

                              
debian@debian:~/docker_build_exemple$
```

> Par la suite, une image Docker peut être exportée dans un fichier ou uploadée sur un registre privé par exemple.

Revision #21
Created 16 July 2025 08:48:50 by Thibaud FRICHET
Updated 5 September 2025 08:57:05 by Thibaud FRICHET