Chapitre 4
Création et gestion d'images Docker

1. Création manuelle d'une nouvelle image

1.1 Objectif

Dans le chapitre précédent, nous avons étudié le fonctionnement de base de Docker en utilisant des images préparées à l'avance pour nos exemples : l'image `hello-world` pour les premiers tests, l'image `ubuntu` pour les approches interactives par la ligne de commande, et enfin l'image `nginx` pour les tests impliquant un processus serveur. Ces images ont été récupérées en ligne sur le registre Docker Hub.

Bien que cette façon de fonctionner puisse suffire à des mises en œuvre extrêmement simples, un moment survient où il devient nécessaire de créer ses propres images pour évoluer vers plus de complexité. C'est ce type de manipulation que nous allons illustrer dans le présent chapitre.

1.2 Approche

L'approche naïve pour créer sa propre image est d'utiliser la commande `commit` entrevu dans le chapitre précédent pour persister l'état d'un conteneur sous forme d'une nouvelle image, après avoir ajouté à celui-ci les modifications nécessaires. Pour cela, nous pourrions lancer par exemple un conteneur sur une image `ubuntu` comme base et installer un produit en ligne de commande dans le conteneur.

Ensuite, le conteneur obtenu serait sauvegardé sous forme d'une image qu'il serait possible d'instancier ensuite autant de fois que souhaité. Bref, nous aurions créé une image avec un processus intégré selon nos besoins.

1.3 Difficultés

Dans les précédentes éditions du présent ouvrage, cette méthode était montrée en détail, de façon à faire apparaître toutes les difficultés liées :

- Taille de l'image, car les modifications, même d'effacement, s'empilent dans les couches et les effets d'une installation manuelle, avec des étapes de compilation et des productions de fichiers temporaires, peuvent être importants.
- Difficulté de changer le processus à démarrer lors du lancement d'un conteneur, car par défaut c'est celui de l'image de base qui se lance.
- Besoin d'une réelle expertise sur le paramétrage de l'application, de façon à l'installer de manière optimale et surtout suffisamment souple pour qu'un lancement de conteneur permette de faire varier certains des paramètres.
- Problématique de sécurité à gérer, en particulier lié au fait d'utiliser un utilisateur avec des droits élevés dans l'image.
- Complexité de gestion du cycle de vie si l'image de base est faite pour fonctionner en mode interactif et que nous souhaitons mettre en œuvre un serveur, fait pour fonctionner en mode démon, en arrière-plan.

1.4 Conclusion

La mise en œuvre d'une image propre par cette méthode est extrêmement complexe. De plus, dans les nouvelles versions des images de base, plus restreintes pour des raisons d'efficacité, certaines dépendances nécessaires au déploiement d'une application serveur doivent elles-mêmes être installées au préalable, y compris des fonctionnalités de niveau système qui nécessitent pour elles également une compétence développée.

Comme on l'aura compris, il est fortement recommandé de faire réaliser cette opération de construction d'une image par une personne experte de l'application ciblée, de façon à obtenir une image propre. Or, cette mise en œuvre propre existe certainement déjà, sous la forme d'une image Docker officielle. De plus, les personnes en charge de cette image nous offrent la "recette utilisée" sous la forme d'un fichier contenant toutes les commandes nécessaires à une installation parfaite. Pourquoi faire autrement ?

2. Utilisation d'un Dockerfile

2.1 Intérêt des fichiers Dockerfile

La "recette" dont nous venons de parler se présente sous la forme d'un fichier texte nommé `Dockerfile`, utilisant une grammaire particulière à Docker. Ce fichier contient toutes les opérations nécessaires à la préparation d'une image Docker. Ainsi, au lieu de construire une image par des opérations manuelles dans un conteneur suivie par une commande `commit` (voire plusieurs si l'on procède en étapes) comme nous l'avons évoqué ci-dessus, nous allons pouvoir compiler une image depuis une description textuelle de ces opérations.

Voici par exemple le contenu du fichier `Dockerfile` correspondant à l'image officielle MongoDB en ligne 3.6 (un système de gestion de base de données NoSQL) telle qu'elle peut être retrouvée sur le registre Docker Hub à la date d'écriture de ce livre (version 3.6.19 de MongoDB, disponible sur https://registry.hub.docker.com/_/mongo/) :

```
FROM ubuntu:xenial

# add our user and group first to make sure their IDs get assigned
consistently, regardless of whatever dependencies get added
RUN groupadd -r mongodb && useradd -r -g mongodb mongodb

RUN set -eux; \
    apt-get update; \
    apt-get install -y --no-install-recommends \
           ca-certificates \
           jq \
           numactl \
    ; \
    if ! command -v ps > /dev/null; then \
           apt-get install -y --no-install-recommends procps; \
    fi; \
    rm -rf /var/lib/apt/lists/*

# grab gosu for easy step-down from root
(https://github.com/tianon/gosu/releases)
ENV GOSU_VERSION 1.12
# grab "js-yaml" for parsing mongod's YAML config files
(https://github.com/nodeca/js-yaml/releases)
ENV JSYAML_VERSION 3.13.1

RUN set -ex; \
    \
    savedAptMark="$(apt-mark showmanual)"; \
```

```
    apt-get update; \
    apt-get install -y --no-install-recommends \
           wget \
    ; \
    if ! command -v gpg > /dev/null; then \
           apt-get install -y --no-install-recommends gnupg dirmngr; \
           savedAptMark="$savedAptMark gnupg dirmngr"; \
    elif gpg --version | grep -q '^gpg (GnuPG) 1\.'; then \
# "This package provides support for HKPS keyservers." (GnuPG 1.x only)
           apt-get install -y --no-install-recommends gnupg-curl; \
    fi; \
    rm -rf /var/lib/apt/lists/*; \
    \
    dpkgArch="$(dpkg --print-architecture | awk -F- '{ print $NF }')"; \
    wget -O /usr/local/bin/gosu
"https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-
$dpkgArch"; \
    wget -O /usr/local/bin/gosu.asc
"https://github.com/tianon/gosu/releases/download/$GOSU_VERSION/gosu-
$dpkgArch.asc"; \
    export GNUPGHOME="$(mktemp -d)"; \
    gpg --batch --keyserver hkps://keys.openpgp.org --recv-keys
B42F6819007F00F88E364FD4036A9C25BF357DD4; \
    gpg --batch --verify /usr/local/bin/gosu.asc /usr/local/bin/gosu; \
    command -v gpgconf && gpgconf --kill all || :; \
    rm -r "$GNUPGHOME" /usr/local/bin/gosu.asc; \
    \
    wget -O /js-yaml.js "https://github.com/nodeca/js-yaml/raw/$
{JSYAML_VERSION}/dist/js-yaml.js"; \
# TODO some sort of download verification here
    \
    apt-mark auto '.*' > /dev/null; \
    apt-mark manual $savedAptMark > /dev/null; \
    apt-get purge -y --auto-remove -o
APT::AutoRemove::RecommendsImportant=false; \
    \
# smoke test
    chmod +x /usr/local/bin/gosu; \
    gosu --version; \
    gosu nobody true

RUN mkdir /docker-entrypoint-initdb.d

ENV GPG_KEYS 2930ADAE8CAF5059EE73BB4B58712A2291FA4AD5
RUN set -ex; \
    export GNUPGHOME="$(mktemp -d)"; \
    for key in $GPG_KEYS; do \
```

```
            gpg --batch --keyserver ha.pool.sks-keyservers.net --
recv-keys "$key"; \
    done; \
    gpg --batch --export $GPG_KEYS > /etc/apt/trusted.gpg.d/mongodb.gpg; \
    command -v gpgconf && gpgconf --kill all || :; \
    rm -r "$GNUPGHOME"; \
    apt-key list

# Allow build-time overrides (eg. to build image with MongoDB
Enterprise version)
# Options for MONGO_PACKAGE: mongodb-org OR mongodb-enterprise
# Options for MONGO_REPO: repo.mongodb.org OR repo.mongodb.com
# Example: docker build --build-arg MONGO_PACKAGE=mongodb-
enterprise --build-arg MONGO_REPO=repo.mongodb.com .
ARG MONGO_PACKAGE=mongodb-org
ARG MONGO_REPO=repo.mongodb.org
ENV MONGO_PACKAGE=${MONGO_PACKAGE} MONGO_REPO=${MONGO_REPO}

ENV MONGO_MAJOR 3.6
ENV MONGO_VERSION 3.6.19
# bashbrew-architectures:amd64 arm64v8
RUN echo "deb http://$MONGO_REPO/apt/ubuntu
xenial/${MONGO_PACKAGE%-unstable}/$MONGO_MAJOR multiverse" | tee
"/etc/apt/sources.list.d/${MONGO_PACKAGE%-unstable}.list"

RUN set -x \
# installing "mongodb-enterprise" pulls in "tzdata" which prompts for input
    && export DEBIAN_FRONTEND=noninteractive \
    && apt-get update \
    && apt-get install -y \
           ${MONGO_PACKAGE}=$MONGO_VERSION \
           ${MONGO_PACKAGE}-server=$MONGO_VERSION \
           ${MONGO_PACKAGE}-shell=$MONGO_VERSION \
           ${MONGO_PACKAGE}-mongos=$MONGO_VERSION \
           ${MONGO_PACKAGE}-tools=$MONGO_VERSION \
    && rm -rf /var/lib/apt/lists/* \
    && rm -rf /var/lib/mongodb \
    && mv /etc/mongod.conf /etc/mongod.conf.orig

RUN mkdir -p /data/db /data/configdb \
    && chown -R mongodb:mongodb /data/db /data/configdb
VOLUME /data/db /data/configdb

COPY docker-entrypoint.sh /usr/local/bin/
ENTRYPOINT ["docker-entrypoint.sh"]
```

```
EXPOSE 27017
CMD ["mongod"]
```

Les lignes passées en gras correspondent aux actions auxquelles un administrateur sans connaissance experte de MongoDB aurait certainement pensé, ce qui montre, par différence, la richesse additionnelle du fichier descriptif de l'image officielle. Les opérations les plus simples mises en œuvre dans ce fichier de génération d'image sont les suivantes :

– La mise en place d'un utilisateur et d'un groupe dédiés.
– L'installation des dépendances.
– Un commentaire de rappel comme quoi une vérification du téléchargement devrait être réalisée (par très rassurant de voir que la sécurité est laissée pour plus tard dans une image officielle d'une version récente de la base de données NoSQL la plus utilisée dans le monde…).
– L'installation proprement dite de l'application serveur.
– La définition des volumes pour les données ainsi que pour la configuration.
– L'utilisation d'un `ENTRYPOINT` dédié, qui permet de spécifier le script à lancer lors du démarrage du conteneur (nous reviendrons plus longuement sur cette commande par la suite).
– L'exposition du port sur lequel le processus reçoit les commandes (ce qui nécessite de connaître le fonctionnement de ce processus).
– La mise en œuvre d'un processus par défaut.

Encore une fois, il aurait été possible de trouver toutes ces manipulations par une recherche dans la documentation ou sur les forums. Mais la présence d'un `Dockerfile` produit par des experts de MongoDB, tout en nous laissant valider les commandes, est un énorme avantage de Docker.

Remarque

Un regard critique reste nécessaire sur les fichiers `Dockerfile`, sous peine de ne pas être plus en sécurité que dans le cas de la simple instanciation d'une image de provenance inconnue. En particulier, tout appel à des URL externes (par `curl`, typiquement) devrait être soigneusement vérifié. C'est le sens du `TODO` relevé dans le fichier, et qui est un manque de sécurité qui devrait être vérifié sur une mise en production, au minimum en vérifiant que le domaine associé est sûr et que le contenu sur lequel l'URL pointe ne comporte pas de problème de sécurité.

2.2 Utilisation d'un fichier Dockerfile

Bien que, dans le cas d'une image officielle, le fichier `Dockerfile` serve surtout au producteur pour réaliser l'image et montrer publiquement la façon dont celle-ci a été réalisée (le lien entre les deux étant assuré en particulier par le fait que c'est Docker Hub qui s'occupe de compiler les images), il est tout à fait possible d'utiliser soi-même ce fichier pour créer une image.

- Sur la machine hôte, créez un répertoire nommé par exemple `mongogb`.
- Placez-vous dans ce répertoire.
- Recopiez les deux fichiers que vous retrouverez sur https://github.com/docker-library/mongo/tree/master/3.6

Remarque

*Cette URL est celle contenant le fichier `Dockerfile`, qu'on peut retrouver directement depuis la page du registre Docker Hub concernant l'image officielle de MongoDB. Le plus simple pour recopier les fichiers est de lancer une commande `wget` sur les adresses qui sont obtenues en cliquant sur le bouton **Raw** dans la page d'affichage par GitHub des contenus des fichiers.*

- Si nécessaire, affectez les droits de lecture sur les deux fichiers et les droits d'exécution sur `docker-entrypoint.sh`.

Le second fichier (en plus de `Dockerfile`) est le fichier `docker-entrypoint.sh` auquel fait référence le `Dockerfile` (commande `ENTRYPOINT`, que nous allons expliquer plus bas). Il est nécessaire pour que le conteneur lance le serveur MongoDB sous l'utilisateur dédié au lieu de `root`. Il utilise pour cela l'utilitaire `gosu` qui avait été précédemment installé par une commande `RUN` du `Dockerfile`.

La partie du contenu de ce fichier correspondant à cette commande est la suivante :

```
#!/bin/bash
set -Eeuo pipefail

(...)

    exec gosu mongodb "$BASH_SOURCE" "$@"

(...)

exec "$@"
```

- Lancez la commande de compilation de l'image.