



Docker - Création d'une image



Création d'une image

Une des premières méthodes pour créer une image est :

- de récupérer une image (**pull**)
- de démarrer un container (**run**)
- d'effectuer des modifications (installation de software, création de fichiers,)
- de commiter ces modifications pour créer une nouvelle image à partir de ce qui se trouve sur le container (**commit**)



Historique image

docker history [OPTIONS] image

IMAGE	CREATED	CREATED BY	SIZE
21e77433a11d	3 days ago	/bin/sh -c #(nop) CMD ["/bin/bash" "/start.s	0 B
3b0379305784	3 days ago	/bin/sh -c chmod 755 /start.sh	94 B
278b27bc3cf7	3 days ago	/bin/sh -c #(nop) ADD file:a84b2099c064d81b8f	94 B
6b8ce0e3cd14	3 days ago	/bin/sh -c #(nop) ADD file:5744380367ce40b106	1.607 kB
cf8d5979dcc7	3 days ago	/bin/sh -c adduser -D -u 1000 docker	5.158 kB
e494de88a292	3 days ago	/bin/sh -c curl -sS https://getcomposer.org/i	1.706 MB
e04904444ef7	3 days ago	/bin/sh -c ln -s /usr/bin/php7 /usr/bin/php	13 B
c771b94cafc6	3 days ago	/bin/sh -c apk --no-cache --update --reposito	103.1 MB
28b26f729781	4 days ago	/bin/sh -c apk update && apk upgrade && apk	7.599 MB
7d23b3ca3463	7 days ago	/bin/sh -c #(nop) ADD file:fd71807f3b22f7f51f	4.799 MB

Dockerfile

Docker peut créer des images automatiquement à partir d'instructions dans un fichier **Dockerfile**

- permet de spécifier un ensemble de commandes (ex.: installation de packages)
- **docker build** se sert de ce fichier pour automatiser le build de l'image

Dockerfile

La commande docker build permet de créer une image à partir d'un fichier dockerfile

- `docker build -t <user-dockerhub>/<image-name>:tag /path/`
- l'utilisateur du hub n'est pas obligatoire, vous pouvez seulement passer le nom de l'image
- le tag par défaut sera latest et n'est pas obligatoire
- Attention, ne pas fournir un path dans lequel il y a bcp de fichiers et de dossiers car il va tous les parcourir (Pensez à faire un dossier différent pour chaque dockerfile)

Dockerfile

Ce fichier d'instructions doit respecter des bonnes pratiques

- être le plus éphémère possible
- n'installer que le nécessaire
- un container = un processus
- limiter le nombre de couches
- trier les arguments multi-lignes par ordre alphanumérique

Trier les arguments multi-lignes

```
RUN apk --no-cache --update --repository=http://dl-4.alpinelinux.org/alpine/edge/community add \  
curl \  
nginx \  
php7 \  
php7-ctype \  
php7-curl \  
php7-fpm \  
php7-iconv \  
php7-intl \  
php7-json \  
php7-mbstring \  
php7-mcrypt \  
php7-opcache \  
php7-openssl \  
php7-pdo_mysql \  
php7-phar \  
php7-session \  
php7-xml \  
php7-xsl \  
supervisor && \  
chown -R nginx:www-data /var/lib/nginx && \  
ln -sf /dev/stdout /var/log/nginx/access.log && \  
ln -sf /dev/stderr /var/log/nginx/error.log
```



Dockerfile - FROM

FROM est la première instruction d'un Dockerfile

- toutes les images dérive d'une autre image ou d'une image de base (ex.: debian, alpine)

FROM alpine

FROM debian:wheezy



Dockerfile - MAINTAINER

L'instruction **MAINTAINER** permet de spécifier le nom et le contact du responsable de l'image

MAINTAINER Cyrille Grandval <cgrandval@darkmira.com>

MAINTAINER Esgi



Dockerfile

Les 3 instructions **RUN**, **CMD** et **ENTRYPOINT** peuvent être définies selon une forme exec ou shell

Shell form

- RUN apt-get install php-fpm
- CMD echo "Hello world"
- ENTRYPOINT echo "Hello world"



Dockerfile

Exec form

- RUN ["apt-get", "install", "php-fpm"]
- CMD ["/bin/echo", "Hello world"]
- ENTRYPOINT ["/bin/echo", "Hello world"]



Dockerfile - RUN

Pour exécuter une commande (ou suite de commandes) dans le container, il faut utiliser l'instruction **RUN**

- **exec form**

- RUN ["executable", "param1", "param2", ...]
- RUN ["/bin/bash", "/start.sh"]
- permet d'exécuter une commande même si aucun shell n'est installé sur le container

- **shell form**

- RUN /bin/bash -c "commande param1"
- RUN commande param1



Dockerfile - RUN

```
Building alpine-php7-fpm-nginx
```

```
Step 1 : FROM alpine
```

```
----> 7d23b3ca3463
```

```
Step 2 : RUN apk update && apk upgrade &&
```

```
----> Using cache
```

```
----> 28b26f729781
```

```
apk add ca-certificates bash &&
```

```
rm -rf /var/cache/apk/*
```

1. Une image alpine existe déjà (et est disponible avec docker images)
2. La suite de commandes définie dans ce RUN a déjà été exécutée sur une base alpine et a déjà été mise en cache. Lors du build, Docker récupère l'image correspondante (**28b26f729781**) à cette instruction sur la base de l'image alpine et ne rejoue donc pas la commande.

```
<none>  
<none>
```

```
<none>  
<none>
```

```
bc77c77d0430  
28b26f729781
```

```
27 hours ago  
24 hours ago
```

```
11.5 MB  
12.4 MB
```

Dockerfile - CMD

L'instruction **CMD** spécifie la commande qui sera exécutée lors du démarrage de l'image

- ne peut être utilisée qu'une seule fois dans le **Dockerfile**
- si plusieurs **CMD** sont présentes, le dernier est utilisé
- la commande et les paramètres peuvent être surchargés depuis la ligne de commande
- 3 types d'utilisations
 - exec form : `CMD ["executable", "param1", "param2", ...]`
 - shell form : `RUN commande param1 param2`
 - `CMD ["param1", "param2", ...]` par l'instruction **ENTRYPOINT**



Dockerfile - ENTRYPOINT

L'instruction **ENTRYPOINT** comporte les mêmes 2 formes (exec et shell) que **RUN** et **CMD**

- un seul entrypoint doit être défini (seul le dernier sera utilisé)
- configure un container qui sera lancé comme un exécutable
- à l'inverse de **CMD**, les commandes ne sont pas écrasées par celle de la ligne de commande



Dockerfile - ENTRYPOINT

	No ENTRYPOINT	ENTRYPOINT exec_entry p1_entry	ENTRYPOINT ["exec_entry", "p1_entry"]
No CMD	<i>error, not allowed</i>	/bin/sh -c exec_entry p1_entry	exec_entry p1_entry
CMD ["exec_cmd", "p1_cmd"]	exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry exec_cmd p1_cmd	exec_entry p1_entry exec_cmd p1_cmd
CMD ["p1_cmd", "p2_cmd"]	p1_cmd p2_cmd	/bin/sh -c exec_entry p1_entry p1_cmd p2_cmd	exec_entry p1_entry p1_cmd p2_cmd
CMD exec_cmd p1_cmd	/bin/sh -c exec_cmd p1_cmd	/bin/sh -c exec_entry p1_entry /bin/sh - c exec_cmd p1_cmd	exec_entry p1_entry /bin/sh -c exec_cmd p1_cmd

Dockerfile - EXPOSE

L'instruction **EXPOSE** permet d'indiquer à docker que le container écoute sur les ports indiqués

- **attention : ne rend pas accessible par défaut les ports depuis l'hôte**
- l'option **-p** de la commande **docker run** doit être utilisé pour publier un range de port (**-p 35500:22**) ou **-P** pour publier tous les ports exposés
- un numéro de port exposé dans le container peut être un numéro de port différent accessible sur l'hôte (ex.: exposé sur le 22, accessible sur le 34500 depuis l'hôte)

Dockerfile - COPY

L'instruction COPY copie des fichiers ou des dossiers à l'intérieur du container

- les fichiers sources ne peuvent pas être en dehors du dossier de build (donc pas de ../)
- il est possible d'utiliser des wilcards et matchings (*, ?, ...) pour les noms de source (ex.: file*)
- info : tout est créé avec l'id et gid 0
- info : la destination doit finir par un / si c'est un dossier, ou que plusieurs fichiers vont y être copiés

Dockerfile - Rappel

Rappel pour le TP

- **docker build -t <user-dockerhub>/<image-name>:tag /path/**
 - utiliser l'option -t si vous avez l'intention de partager votre image sur le hub docker
- **docker run [options] image** pour créer un container
 - -v dossier_local:dossier_container pour partager un dossier entre l'hôte et le container
 - -p port_hote:port_container pour publier des ports
 - -e pour spécifier des variables d'environnements nécessaires au container

TP - créer une image sshd

- créer un fichier Dockerfile en partant d'un alpine
- définir un mainteneur
- installer ssh
 - info : pour installer un package sur alpine **apk add package_name1 package_name2 ...**
 - info : pour mettre à jour le gestionnaire de package de alpine, vous pouvez utiliser l'option **--update** avec apk add
- générer les clés nécessaires à ssh
 - commande : **ssh-keygen -A**
- autoriser la connexion par mot de passe et changer le mot de passe root
 - info : **sed -i s/#PermitRootLogin.*/PermitRootLogin\ yes/ /etc/ssh/sshd_config && echo "root:esgi" | chpasswd**

TP - créer une image sshd

- exposer le port 22 pour pouvoir se connecter en ssh au serveur
- lancer le serveur sshd au démarrage du container
- builder votre image
 - info : n'oubliez pas de lui donner un nom (afin d'éviter une image <none>), ce sera plus simple pour la lancer
- Démarrer votre image et tenter de vous connecter à l'utilisateur root
 - info : n'oubliez pas de publier le port dans votre commande run pour pouvoir vous connecter sur le port exposé au sein du container