DOCKER: Comment créer un plus petit conteneur d'une image

Table of Contents

Cet article montre comment on peut réduire la taille d'une image de quelques pour cent de celle d'origine.

Prenons l'exemple d'une plateforme composée de plusieurs conteneurs (NGiNX, HAProxy, Registrator et Consul). Ces conteneurs s'exécutent sur chacun des nœuds du cluster CoreOS et, au démarrage du cluster, plus de 600 Mo sont téléchargés par les 3 nœuds du cluster. Ceci prend beaucoup de temps.

```
cargonauts/consul-http-router latest 7b9a6e858751 7 days ago 153 MB
cargonauts/progrium-consul latest 32253bc8752d 7 weeks ago 60.75 MB
progrium/registrator latest 6084f839101b 4 months ago 13.75 MB
```

La taille des images est non seulement préjudiciable au temps de démarrage de la plate-forme, mais elle augmente également la surface d'attaque du conteneur. Avec 153 Mo d'utilitaires dans le routeur consul-http basé sur NGiNX, le conteneur contient de nombreux éléments qu'un attaquant peut utiliser une fois qu'il est entré. Alors que nous envisagions de faire fonctionner ce routeur dans une zone démilitarisée, nous avons voulu minimiser le nombre d'outils disponibles pour un pirate potentiel.

Alors, lorsque l'image est trop grosse, on peut essayer de la dépouiller avec l'utilitaire **strip-docker** pour rendre les conteneurs plus rapides et plus sûrs en même temps!

Trouver les fichiers nécessaires

En utilisant l'utilitaire dpkg, on peut lister tous les fichiers installés par NGiNX.

```
docker run nginx dpkg -L nginx
…
/.
/usr
/usr/sbin
/usr/sbin/nginx
/usr/share
/usr/share/doc
/usr/share/doc/nginx
```

```
…
/etc/init.d/nginx
```

Localiser les bibliothèques partagées dépendantes

On a donc la liste des fichiers dans le package, mais il faut aussi les bibliothèques partagées référencées par l'exécutable. Heureusement, ils peuvent être récupérés à l'aide de l'utilitaire ldd.

```
docker run nginx ldd /usr/sbin/nginx
…
linux-vdso.so.1 (0x00007fff561d6000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0
(0x00007fd8f17cf000)
libcrypt.so.1 => /lib/x86_64-linux-gnu/libcrypt.so.1 (0x00007fd8f1598000)
libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fd8f1329000)
libssl.so.1.0.0 => /usr/lib/x86_64-linux-gnu/libssl.so.1.0.0
(0x00007fd8f10c9000)
libcrypto.so.1.0.0 => /usr/lib/x86_64-linux-gnu/libcrypto.so.1.0.0
(0x00007fd8f0cce000)
libz.so.1 => /lib/x86_64-linux-gnu/libz.so.1 (0x00007fd8f0ab2000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fd8f0709000)
/lib64/ld-linux-x86-64.so.2 (0x00007fd8f19f0000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fd8f0505000)
```

Suivre et inclure des liens symboliques

Maintenant qu'on a l'exécutable et les bibliothèques partagées référencées, il s'avère que ldd nomme normalement le lien symbolique et non le nom de fichier de la bibliothèque partagée.

```
docker run nginx ls -l /lib/x86_64-linux-gnu/libcrypt.so.1
…
lrwxrwxrwx 1 root root 16 Apr 15 00:01 /lib/x86_64-linux-gnu/libcrypt.so.1
-> libcrypt-2.19.so
```

En résolvant les liens symboliques et en incluant à la fois le lien et le fichier, on est prêts à exporter l'essentiel depuis le conteneur!

Résoudre les dépendances

Mais après avoir copié tous les fichiers essentiels sur une image scratch, NGiNX n'a pas démarré. Il est apparu que NGiNX essayait de résoudre l'identifiant utilisateur «nginx» sans succès.

```
docker run -P –entrypoint/usr/sbin/nginx stripped-nginx -g "démon
désactivé;"
```

```
…
29/06/2015 21:29:08 [émerg] 1 # 1: échec de getpwnam ("nginx") (2: aucun
fichier ou répertoire de ce type) dans /etc/nginx/nginx.conf:2
nginx: [émerg] getpwnam ("nginx") a échoué (2: aucun fichier ou répertoire
de ce type) dans /etc/nginx/nginx.conf:2
```

Il s'est avéré que les bibliothèques partagées pour le service de commutation de noms lisant/etc/passwd et/etc/group sont chargées à l'exécution et non référencées dans les bibliothèques partagées. En ajoutant ces bibliothèques partagées ((/ lib/*/libnss *) au conteneur, NGiNX a fonctionné!

Exemple de strip-docker-image

Alors maintenant, l'utilitaire strip-docker-image est là pour vous!

```
strip-docker-image -i image-name
-t target-image-name
[-p package]
[-f file]
[-x expose-port]
[-v]
```

Les options sont expliquées ci-dessous:

- **-i**: nom-image à déshabiller
- **-t target-image-name**: le nom de l'image dépilée
- **-p package**: package à inclure à partir de l'image, plusieurs -p autorisés.
- **-f file**: fichier à inclure à partir de l'image, plusieurs -f sont autorisés.
- **-x**: port à exposer.
- **-v**: mode verbeux.

L'exemple suivant crée une nouvelle image nginx, nommée stripped-nginx, basée sur l'image officielle de Docker:

```
strip-docker-image -i nginx -t stripped-nginx \
-x 80 \
-p nginx \
-f /etc/passwd \
-f /etc/group \
-f '/lib/*/libnss*' \
-f /bin/ls \
-f /bin/cat \
-f /bin/sh \
-f /bin/mkdir \
-f /bin/ps \
-f /var/run \
-f /var/log/nginx \
-f /var/cache/nginx
```

Outre le paquet nginx, nous ajoutons les fichiers /etc/passwd, /etc/group et les bibliothèques partagées /lib/*/libnss *. Les répertoires /var/run, /var/log/nginx et /var/cache/nginx sont requis pour que NGiNX puisse fonctionner. De plus, nous avons ajouté/bin/sh et quelques utilitaires pratiques, juste pour pouvoir fouiner un peu.

L'image dépouillée a maintenant été réduite à un incroyable total de 5,4% des 132,8 Mo d'origine, à seulement 7,3 Mo, et est toujours pleinement opérationnelle!

```
docker images | grep nginx
…
stripped-nginx latest d61912afaf16 21 seconds ago 7.297 MB
nginx 1.9.2 319d2015d149 12 days ago 132.8 MB
```

Et cela fonctionne!

```
ID=$(docker run -P -d –entrypoint /usr/sbin/nginx stripped-nginx -g "daemon
off;")
docker run –link $ID:stripped cargonauts/toolbox-networking curl -s -D –
http://stripped
…
HTTP/1.1 200 OK
```

Conclusion

Il est possible d'utiliser les images officielles gérées et distribuées par Docker et de les réduire à l'essentiel, prêtes à l'emploi! Il accélère les temps de chargement et réduit la surface d'attaque de ce conteneur spécifique.