TD docker web server

> Executer un server web à partir d'un container docker construit sur **Alpine** et **Nginx**

Dans cet exercice vous allez:

- créer un container avec la distribution **Alpine**
- installer **nginx** et **wget**
- downloader une page html
- lancer le serveur web dans le container
- servir la page en local sur localhost:8080

> Pour que ce tuto soit efficace, je vous conseille.
>
> - d'utiliser exclusivement le **terminal**
> - de ne pas copier coller les commandes, mais de les écrire

Tout au long du tuto

- demandez-vous le pourquoi de chaque opération
- pensez à verifier le résultat de chaque commande

Dans la suite, il faut noter que :

1. Le shell d'Alpine est `/bin/sh` et non `/bin/bash`
2. le package manager d'Alpine est `apk` au lieu de `apt-get`

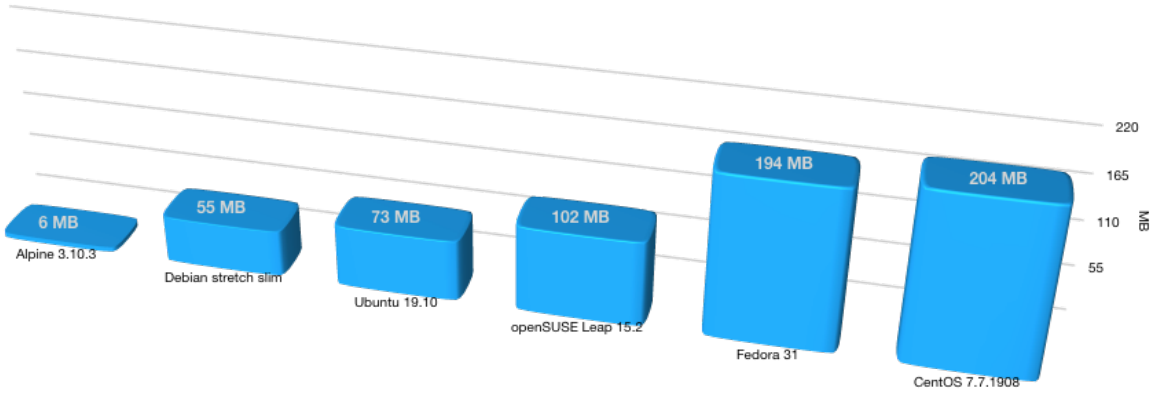
Pourquoi Alpine et pas Ubuntu ou Debian ?

Alpine est une distribution bien plus petite que Ubuntu ou Debian. 7Mb au lieu 80Mb ou 120Mb. La taille de l'image finale sera donc réduite, les transferts plus rapides.

Mais Alpine ne contient qu'un minimum de programmes.

La distribution est donc ainsi plus sécurisé car elle offre une surface de vulnérabilités réduite.

- Alpine est idéal pour les micro-services, les applications simples
- Ubuntu/Debian sont plus adaptées aux applications complexes.



Création et accès au container

1. récupérer (`pull`) l'image `alpine:latest`

```Bash
docker pull alpine:latest
```

1. Lancer un container sur cette image en mode interactif en accédant à un terminal

utiliser les flags suivants

- `-it` pour permettre l'accès a un terminal interactif
- `-name` pour lui donner le nom `nginx_container`

Alpine utilise `/bin/sh` et non `/bin/bash`

```Bash
docker run -it --name nginx_container alpine:latest /bin/sh
```

Une fois dans la session shell, répondez à ces quelques questions

- qui êtes vous ?
- allez dans `/home` . Quel sont les utilisateurs du système ?
- allez dans `/root` et listez les fichiers.
- qu'est ce qui est different par rapport à la distribution Ubuntu ?
- par exemple est ce que `ll` est définie comme alias pour le user root ?

setup du container

On va installer Nginx, le server web.

Install de Nginx

Mise à jour du package manager:

```bash
apk update
```

puis installation de Nginx

```bash
apk add nginx
```

Install de wget

```bash
apk add wget
```

La page html

On va servir une page html statique: `index.html`

Créez le repertoire suivant :

```bash
mkdir /var/www/html
cd /var/www/html
```

Puis récupérez une page web. Par exemple, la page Linux de wikipedia: https://fr.wikipedia.org/wiki/Linux. Vous pouvez choisir la page qui vous plait.

On veut que le fichier html soit nommée `index.html`. On peut spécifier le nom du fichier cible dans `wget`. Pour trouver le flag qui permet de specifier le nom du fichier cible:

```bash
wget --help
```

Ce qui donne:

```bash
wget -O index.html https://fr.wikipedia.org/wiki/Linux
```

Affichez ensuite les 10 premieres lignes du fichier index.html avec `head -n 10 index.html`

Il faut aussi configurer Nginx

Une bonne pratique est de sauvegarder d'abord la configuration par défaut en la copiant dans un fichier backup:

```bash
cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.backup
```

Maintenant vous pouvez réécrire la configuration de Nginx en dumpant directement le string de config dans le fichier `nginx.conf`. La commande est `echo -e "<la string>" > <le fichier>`. Notez le `>` entre la string et le fichier.

```bash
echo -e "daemon off;\n\nevents {\n worker_connections 1024;\n}\n\nhttp {
```

Démarrez le server nginx

Ce n'est pas nécessaire pour le bon fonctionnement du container mais cela permet de vérifier que tout fonctionne comme prévu.

```bash
nginx
```

Profitez en pour lister les processus liés à nginx avec `ps` aux et `grep`

```bash
ps -ef | grep nginx
```

Enfin, sortez du container en l'arrêtant avec `exit`

L'image est configurée avec les bonnes librairies.

On va maintenant la cloner / copier et la renommer.

Docker commit

> `docker commit` crée une nouvelle image à partir d'un container.

La commande suit:

```bash
docker commit <nom_du_container> <nom_de_image>
```

C'est utile pour capturer les changements fait à l'intérieur du container (installation, configuration, etc...) et les sauvegarder dans une nouvelle image.

Donc on crée la nouvelle image `my_nginx_alpine:v1` à partir du container appelé `nginx_container` avec :

```Bash
docker commit nginx_container my_nginx_alpine:v1
```

Notez qu'on tag l'image avec `v1`

Vérification que la nouvelle image existe bien en listant les images avec :

```Bash
docker images
```

Lancez le container

Run un nouveau container

- en mode détaché `-d`
- en associant (mapping) le port interne 80 du container au port externe 8080 sur le host : `-p 8080:80` .
- en spécifiant le nom du container avec `-name`

ici on appelle le container `my_nginx_server`

> Les correspondances entre host et container sont toujours dans le sens : `host:container` .

```Bash
docker run -d -p 8080:80 --name my_nginx_server my_nginx_alpine:v1 ngin
```

Dans cette commande on a donc :

- `-d` : mode détaché qui fait tourner le container en background
- `-p 8080:80` : la mapping du port externe vers le port interne : 8080 vers 80
- `--name my_nginx_server` : le nom du container
- `my_nginx_alpine:v1` : le nom de l'image
- enfin `nginx` : la commande envoyée au container pour démarrer le server

On peut vérifier que tout fonctionne

En allant sur http://localhost:8080. La page affichée doit correspondre à celle que vous avez récupérée.