Construction d'images

Une image est une suite de couches empilées les unes au dessus des autres. La façon dont on construit une image se base sur ce principe :

- un conteneur est lancé basé sur une image de base (par exemple, **debian**) ;
- une commande est exécuté (par exemple, **apt install apache2**) ;
- l'état du conteneur est capturé et sauvegardé dans une nouvelle image (**debian avec apache2**).

Commit

La façon la plus simple est d'utiliser la commande commit. Elle permet de capturer l'état d'un conteneur et d'en créer une image.

- Lancer un conteneur basé sur l'image **alpine** en mode interactif (`-it`) ;
- Dans le conteneur, installer le programme **vim** (le système d'exploitation Alpine utilise **apk** comme gestionnaire de paquet) ;
- Quitter le conteneur et sauvegarder son état dans une image via la commande `docker commit <container-name>` ;
- Quel est le nom de l'image générée ?
- Trouver les options de la commande `docker commit` pour changer le nom de cette image.

Inspecter les changements

Il est possible d'inspecter les changements entre un conteneur et son image via la commande `docker diff <container-name>`.

- Exécuter cette commande sur le conteneur précédent ; Quels fichiers ont étés changés ?

Dockerfile

La construction des images est plus rapide, simple et reproductibles en se basant sur un fichier Dockerfile, qui liste les différentes étapes de génération de l'image.

- Créer un dossier "**first-img**", et y créer un fichier **Dockerfile** contenant le contenu suivant :

```
# Dockerfile content
# Base image
FROM alpine
```

```
# Execute command
RUN apk update
RUN apk add nginx
```

- Dans ce même dossier, exécuter la commande suivante permettant le lancement de la construction de l'image :

```
cd first-img
docker build -t first-img .
```

- Expliquer en quelques mots l'affichage
- À votre avis, que signifie l'option `-t` ?

Insertion de fichiers

Après avoir installé les dépendances logicielles de l'application, il est temps de copier l'application elle-même. Il y a deux commandes permettant de faire cela : COPY et ADD.

- Chercher dans la documentation Docker les différences entre ces deux instructions **COPY** et **ADD**.
- Écrire un fichier HTML simple affichant "**Hello World!**" dans le dossier "**first-img/index.html**" ;
- Ajouter une ligne au **Dockerfile** précédant copiant ce fichier dans le répertoire utilisé par **nginx** (https://hub.docker.com/_/nginx) ;
- Construire l'image et lancer un conteneur nommé **my_hello_world** basé sur cette image (le conteneur **doit exposer le port 80**) ;
- En accédant à "**localhost**" depuis un navigateur (ou en utilisant curl depuis vagrant), la page est-elle disponible ?

Configuration d'un serveur web

Les conteneurs sont des versions minimalistes de vrais systèmes d'exploitation, et ne contiennent la plupart du temps pas de gestionnaires de processus (**systemd, initd**).

- En utilisant le conteneur précédant, **lister les processus** en cours d'exécution via la commande `ps` (ne pas oublier **d'ouvrir le port**) ;
    - Quels sont les processus en cours ?
- Dans le conteneur, démarrer **nginx** manuellement via la commande `nginx` ;
    - Quelle est la sortie de la commande ?
    - Corriger le problème via un ajout d'une ligne dans le Dockerfile et reconstruire l'image ;

- Réitérer l'étape précédente avec la nouvelle image. Le site est-il fonctionnel via **http://localhost** ? Pourquoi ?
    - Corriger le problème en modifiant le fichier **/etc/nginx/conf.d/default.conf**.

Commande de démarrage

Une fois que la création de l'image est fonctionnelle, il faudrait que nginx démarre automatiquement à la création du conteneur (pour le moment, la commande par défaut est celle définie par l'image **alpine**, qui est de lancer un shell).
Pour cela, il existe l'instruction Dockerfile **CMD**. Son rôle est de définir quelle commande est lancée au démarrage du conteneur si aucune n'est passée en paramètre.

Par défaut, la commande `nginx` démarre en tâche de fond. Il faut utiliser le paramètre `-g` pour définir une option globale pour qu'elle reste active :

```
/usr/sbin/nginx -g "daemon off;"
```

- Ajouter l'instruction **CMD** dans votre **Dockerfile**, reconstruire l'image et lancer un conteneur l'utilisant ;
- Le site est-il accessible ?
  Il est possible d'utiliser `docker logs <container-name>` pour récupérer la sortie d'un conteneur depuis l'hôte) ;

Améliorations

Sources HTML dans un volume configurable

Nous voudrions considérer les pages web comme des données et les stocker dans un volume. De plus, nous voudrions que l'emplacement de ces données dans le conteneur soit configuré dynamiquement.

Il n'est donc pas possible de le faire dans le **Dockerfile** étant donné que cette configuration doit être faite au moment du lancement du conteneur (dynamicité).

La solution généralement utilisée est de créer un script de démarrage lisant des **variables d'environnements**. Ces variables d'environnement sont créées via le paramètre `-e` (il est possible de spécifier plusieurs variables d'environnement en spécifiant plusieurs fois le paramètre).

- Créer un fichier **startup.sh** ayant des droits d'exécution (utiliser la commande `chmod`) contenant la commande de démarrage **nginx** utilisée précédemment ;
- Remplacer la commande **CMD** dans le **Dockerfile** par l'exécution de ce script ;

- Dans **startup.sh**, lire la variable d'environnement **NGINX_WEB_ROOT** et remplacer l'emplacement "**root**" des données dans le fichier de configuration nginx (**/etc/nginx/conf.d/default.conf**) par la valeur de la variable d'environnement. On utilisera sed pour faire ce remplacement (`sed -i "s~\(\s*root\s\).*~\1 $vol;~" /etc/nginx/conf.d/default.conf`) ;
- Sur l'hôte, dans le dossier "**first_img**", créer le dossier "**html-src**" et y copier le (ou les) fichier(s) HTML déjà créé(s).
- Recréer l'image et lancer le conteneur en spécifiant la variable d'environnement :

```
docker run -ti -p 80:80 -v $(pwd)/html-src/:/data/ -e
NGINX_WEB_ROOT=/data/ <img-name>
```

- Modifier le contenu du fichier HTML sur l'hôte pour vérifier que le montage est correctement configuré.

Logs nginx sur la sortie standard (stdout)

Les « Best Practices » préconisent de ne lancer qu'un seul processus dans un conteneur (dans notre cas, nginx).
Il est également conseillé de rediriger les logs applicatifs vers les sorties standards (*STDOUT* et *STDERR*).

nginx possède deux types de logs : les accès et les erreurs.
Les logs d'accès contiennent toutes les requêtes réalisées par les navigateurs. On peut y retrouver notamment le code de retour HTTP, la taille de la page ou encore l'adresse IP source du navigateur.
Les logs d'erreurs contiennent les détails des erreurs rencontrées par le serveur (erreur côté serveur (code 500), page non trouvée (code 404), …).

Ces deux logs sont configurables via les paramètres **access_log** et **error_log** (https://docs.nginx.com/nginx/admin-guide/monitoring/logging/).

nginx supporte nativement la redirection vers la sortie standard pour les erreurs, mais c'est un ajout relativement récent et la redirection vers STDOUT n'est pas possible pour les logs d'accès. Nous allons préconiser une autre solution.

- Créer deux **liens symboliques** dans le **Dockerfile** :
  - **/var/log/nginx/access.log** vers **/dev/stdout** ;
  - **/var/log/nginx/error.log** vers **/dev/stderr** ;
- Modifier la définition du site (**default.conf**) et utiliser ces deux chemins (**/var/log/nginx/…**) ;
- Recréer l'image pour tester.