Formation Docker : Manipulations pratiques avec JavaScript

El Hadji Gaye

Auteur	El Hadji Gaye
Pour	Formations
Date	06/11/2024
Objet	Formation Docker : Manipulations pratiques avec JavaScript.

I) Vocabulaire

Conteneurisation: En informatique, un conteneur est une structure de données, une classe, ou un type de données abstrait, dont les instances représentent des collections d'autres objets. Autrement dit, les conteneurs sont utilisés pour stocker des objets sous une forme organisée qui suit des règles d'accès spécifiques. On peut implémenter un conteneur de différentes façons, qui conduisent à des complexités en temps et en espace différentes. On choisira donc l'implémentation selon les besoins.

Un conteneur est une enveloppe virtuelle qui permet de distribuer une application avec tous les éléments dont elle a besoin pour fonctionner : fichiers source, environnement d'exécution, librairies, outils et fichiers. Ils sont assemblés en un ensemble cohérent et prêt à être déployé sur un serveur et son système d'exploitation (OS). Contrairement à la virtualisation de serveurs et à une machine virtuelle, le conteneur n'intègre pas de noyau, il s'appuie directement sur le noyau de l'ordinateur sur lequel il est déployé.

Virtualisation : La virtualisation consiste, en informatique, à exécuter sur une machine hôte, dans un environnement isolé, des systèmes d'exploitation — on parle alors de virtualisation système — ou des applications— on parle alors de virtualisation applicative. Ces ordinateurs virtuels sont appelés serveur privé virtuel (Virtual Private Server ou VPS) ou encore environnement virtuel (Virtual Environment ou VE).

II) Les commandes Docker à connaître

1. Commande docker ps

docker ps vous affiche toutes les instances de docker qui tournent actuellement sur votre environnement. Si vous ajoutez l'option -a, alors vous verrez mêmes les containers stoppés.

docker ps -a

2. *Commande docker images*

docker images est une commande qui vous montre les images que vous avez construites, et le -a vous montre les images intermédiaires.

docker images -a

3. *Commande docker network*

docker network ls est la commande docker qui liste les différents réseaux.

docker network ls

4. *Commande docker de runtime*

docker-compose up (-d) (--build)
docker-compose stop

La docker-compose est la plus simple car vous n'avez besoin que de 2 commandes : up et stop. stop est assez explicite et stop (mais ne supprime pas) vos conteneurs, mais up nécessite plus d'explications : cela va construire vos images si elles ne le sont pas déjà, et va démarrer vos dockers.

docker build (-t NAME) PATH /URL

Si vous voulez re-build vos images, utilisez l'option --build (vous pouvez aussi utiliser la commande docker-compose build pour uniquement construire des images). L'option -d, qui signifie "detach" fait tourner les conteneurs en tâche de fond.

Avec Docker, vous avez besoin d'une commande séparée pour construire votre image, où vous pouvez spécifier le nom de votre image et vous devez spécifier le PATH ou URL selon votre contexte (cela peut être un repo git).

docker run (-d) (-p hostPort :containerPort) (--name NAME)

run crée le conteneur en utilisant l'image que vous indiquez. Vous pouvez spécifier de nombreux paramètres. Nous vous recommandons d'ajouter un nom à votre conteneur et vous pourriez avoir besoin de spécifier quelques ports à exposer. Comme pour docker-compose, le -d lance le conteneur en tâche de fond.

docker start ID /NAME

docker stop ID /NAME

Le start and stop ne devraient pas être trop compliqués à comprendre, mais il faut noter que vous pouvez "start" uniquement des conteneurs qui sont déjà arrêtés, donc déjà build avec la commande run.

docker exec -it NAME /ID "sh" /"/bin/bash"

Cette commande vous permet de lancer un shell sur votre container. Je préfère utiliser "/bin/bash" mais votre conteneur peut ne pas avoir bash d'installé, et seulement "sh" qui est plus courant (surtout sur les alpines). Si vous avez des configurations spéciales dans votre conteneur, vous aurez peut-être besoin d'utiliser des arguments supplémentaires pour vous y connecter.

5. *Commandes docker de suppression*

Ces commandes permettent de supprimer vos conteneurs et vos images. Vous en aurez probablement besoin pour libérer de l'espace disque.

docker rm ID /NAME

docker-compose rm

Le docker rm supprime seulement un conteneur alors que docker-compose rm supprime tous les conteneurs démarrés avec une commande docker-compose.

docker rmi ID /NAME

Docker rmi supprime l'image que vous passez en paramètre et récursivement toutes les images intermédiaires utilisées pour la construire.

6. *Commandes docker de logs*

Les commandes suivantes sont utiles quand vous devez débugger certains de vos conteneurs (ou, plus souvent, l'application que vous déployez à l'intérieur).

docker logs ID /NAME (-f --tail NBLINE)

Cette commande affiche les logs du container passé en paramètre. Si vous utilisez l'option -f --tail NBLINE vous pouvez suivre en live le flux de vos logs (NBLINE est le nombre de lignes que vous souhaitez afficher). Gardez à l'esprit de choisir un nombre de lignes que vous serez capable de gérer, pour ne pas être dépassé par vos logs.

docker-compose logs (ID /NAME)

L'option (ID /NAME) avec docker-compose logs vous permet de voir les logs d'un conteneur uniquement, au lieu de voir tous les logs. L'astuce ici est que si vous n'utilisez pas l'option -d quand vous utilisez docker run ou docker-compose up vous verrez vos logs directement (mais vous aurez besoin d'arrêter le conteneur pour quitter la vue). Cela peut toujours être utile pour débugger des applications au démarrage.