Les conteneurs

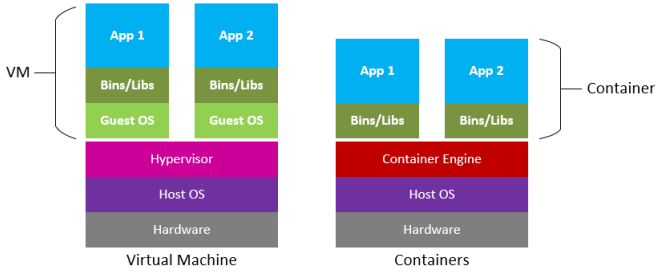
In this section the dynamic representation of a quadrotor unmanned aerial vehicle (UAV) is presented.

Une machine virtuelle:

Lorsque on utilise une machine virtuelle (VM), on fait ce qu'on appelle de la **virtualisation lourde**. En effet, on recrée un système complet dans le système hôte, pour qu'il ait ses propres ressources.
L'isolation avec le système hôte est donc totale ; cependant, cela apporte plusieurs contraintes :
⇒ une machine virtuelle prend du temps à démarrer ;
⇒ une machine virtuelle réserve les ressources (CPU/RAM) sur le système hôte.
Mais cette solution présente aussi de nombreux avantages :
⇒ une machine virtuelle est totalement isolée du système hôte;
⇒ les ressources attribuées à une machine virtuelle lui sont totalement réservées ; ⇒ vous pouvez installer différents OS (Linux, Windows, BSD, etc.).

Conteneur:

Un conteneur Linux est un processus ou un ensemble de processus isolés du reste du système, tout en étant légers. Le conteneur permet de faire de la **virtualisation légère**, c'est-à-dire qu'il ne virtualise pas les ressources, il ne crée qu'une isolation des processus. Le conteneur partage donc les ressources avec le système hôte.
Les conteneurs partagent entre eux le kernel Linux ;
Avantages des conteneurs
⇒ Ne réservez que les ressources nécessaires
⇒ Démarrez rapidement vos conteneurs
Les conteneurs n'ayant pas besoin d'une virtualisation des ressources mais seulement d'une isolation ⇒Donnez plus d'autonomie à vos développeurs



Pourquoi utiliser des conteneurs ?
Les conteneurs permettent de réduire les coûts, d'augmenter la densité de l'infrastructure, tout en améliorant le cycle de déploiement.
Les conteneurs étant capables de démarrer très rapidement, ils sont souvent utilisés en production pour ajouter des ressources disponibles et ainsi répondre à des besoins de mise à l'échelle, ou de scalabilité. Mais ils répondent aussi à des besoins de préproduction ; en étant légers et rapides au démarrage, il permettent de créer des environnements dynamiques et ainsi de répondre à des besoins métier.

Docker

Dans la vision Docker, un conteneur ne doit faire tourner qu'un seul processus.
Grâce à Docker, vous n'aurez plus de problème de différence d'environnement, et votre code marchera partout !
Avantages:
⇒ La création d'environnements locaux.
⇒ Les conteneurs Docker apportent aussi les notions de **stateless** et d'**immutabilité**.
Si nous prenons le cas d'une base de donnée MySQL, celle-ci est **stateful** car elle stocke un état. Ainsi, si vous éteignez et rallumez votre base de données, vous la retrouverez dans le même état de fonctionnement.
Stateless est donc l'inverse : l'application ne stocke pas d'état.
Un conteneur est immuable
L'immutabilité d'un conteneur est aussi importante. Un conteneur ne doit pas stocker de données qui doivent être pérennes, car il les perdra (à moins que vous les ayez pérennisées).
⇒ démarrage d'un conteneur avec un **docker run** ;
⇒ utilisation des arguments **-d et -p** lors du démarrage d'un conteneur ;
⇒ récupération d'une image depuis une registry avec la commande **docker pull** ;
⇒ nettoyage du système avec docker system prune

Créez Dockerfile

Chaque instruction que nous allons donner dans notre Dockerfile va créer une nouvelle layer correspondant à chaque étape de la construction de l'image.
Les instructions dans un Dockerfile

```
FROM debian:9
```

Définir dans le Dockerfile l'image utiliser comme base, grâce à l'instruction **FROM**.
L'instruction FROM n'est utilisable qu'une seule fois dans un Dockerfile.
Ensuite, on utilise l'instruction **RUN** pour exécuter une commande dans le conteneur.

```
RUN apt-get update -yq \
&& apt-get install curl gnupg -yq \
&& curl -sL ***ur*** | bash \
&& apt-get install nodejs -yq \
&& apt-get clean -y
```

ur https://deb.nodesource.com/setup_10.x

> Limitez au maximum le nombre d'instructions RUN, afin de limiter le nombre de layers créées, et donc de réduire la taille de notre image Docker.

L'instruction **ADD** permet de copier ou de télécharger des fichiers dans l'image.

```
ADD . /app/
```

WORKDIR permet de modifier le répertoire courant. La commande est équivalente à une commande cd en ligne de commande. L'ensemble des commandes qui suivront seront toutes exécutées depuis le répertoire défini.

```
WORKDIR /app
```

L'instruction **EXPOSE** permet d'indiquer le port sur lequel votre application écoute.
L'instruction **VOLUME** permet d'indiquer quel répertoire vous voulez partager avec votre host.
Nous allons conclure par l'instruction qui doit toujours être présente, et la placer en dernière ligne pour plus de compréhension : **CMD**. Celle-ci permet à notre conteneur de savoir quelle commande il doit exécuter lors de son démarrage.
docker build, Docker va créer un conteneur pour chaque instruction, et le résultat sera sauvegardé dans une layer. Le résultat final étant un ensemble de layers qui construisent une image Docker complète.

Si une layer ne change pas entre deux builds, Docker ne la reconstruira pas.

Résumé

Pour créer une image Docker, on utilise les instructions suivantes :

- **FROM** qui vous permet de définir l'image source ;
- **RUN** qui vous permet d'exécuter des commandes dans votre conteneur ;
- **ADD** qui vous permet d'ajouter des fichiers dans votre conteneur ;
- **WORKDIR** qui vous permet de définir votre répertoire de travail ;
- **EXPOSE** qui permet de définir les ports d'écoute par défaut ;
- **VOLUME** qui permet de définir les volumes utilisables ;
- **CMD** qui permet de définir la commande par défaut lors de l'exécution de vos conteneurs Docker.

⇒ docker push qui vous permet d'envoyer vos images locales sur une registry ;
⇒ docker search qui vous permet de rechercher une image sur votre registry.

Docker Compose

> Docker Compose va vous permettre d'orchestrer vos conteneurs, et ainsi de simplifier vos déploiements sur de multiples environnements. Docker Compose est un outil écrit en Python qui permet de décrire, dans un fichier YAML, plusieurs conteneurs comme un ensemble de services.

Les commandes principales pour utiliser une stack Docker Compose. Voici les commandes les plus importantes :

- docker-compose up -d vous permettra de démarrer l'ensemble des conteneurs en arrière-plan ;
- docker-compose ps vous permettra de voir le status de l'ensemble de votre stack ;
- docker-compose logs -f –tail 5 vous permettra d'afficher les logs de votre stack ;
- docker-compose stop vous permettra d'arrêter l'ensemble des services d'une stack ;
- docker-compose down vous permettra de détruire l'ensemble des ressources d'une stack ;
- docker-compose config vous permettra de valider la syntaxe de votre fichier docker-compose.yml.