E6 - Compétence

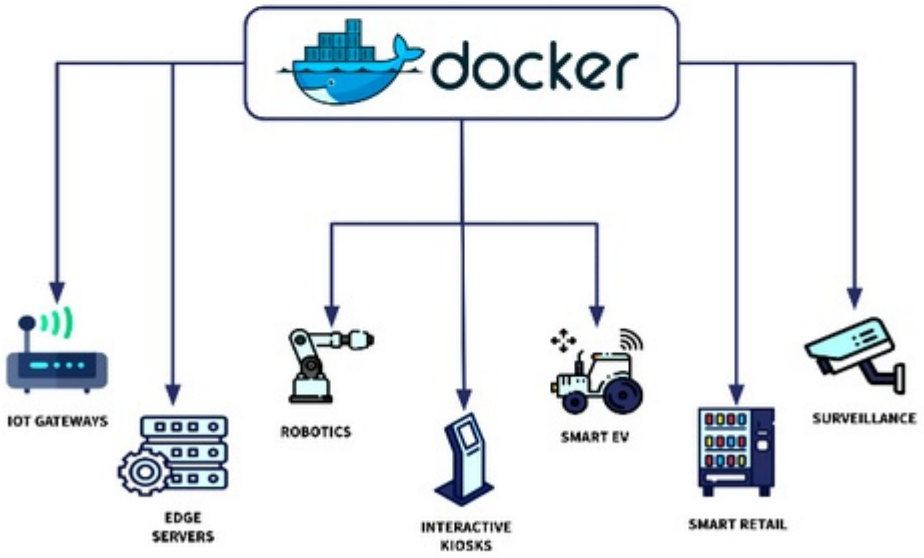
Docker

Introduction

Docker est un logiciel libre qui permet de facilement utiliser des applications qui son normalement nom utilisable sur votre machine. Ceci est faisable grâce à un systeme permetant de mettre l'application en question dans un boîte appeler conteneur. Elle même permet de reproduire l'environnement adéquat au bon fonctionnement de l'application peut importe la plateforme initial utilisé.

Docker n'utilise pas de technologie t'elle que la virtualisation, il utilise la conteneurisation qui est plus légère et qui s'appuie sur la machine hôte pour son fonctionnement. Cette technologie permet une meilleur portabilité et flexibilité d'exécution des applications sur une majeur partie des differentes plateforme utiliser à ce jour.
Ex: machine local, cloud privé ou bien service public.

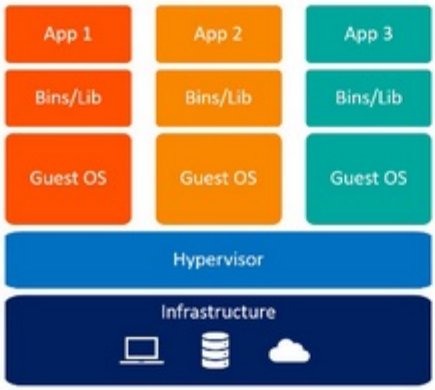
La technologie de conteneurisation utiliser par Docker, permet d'étendre les systèmes distribués de sorte à pouvoir s'exécuter de façon autonome depuis une seule machine ou instance par noeud. Ce systeme permet de déployés les ressources au fur et à mesure de leur disponibilitée à l'image des PaaS (plateforme en tant que service).



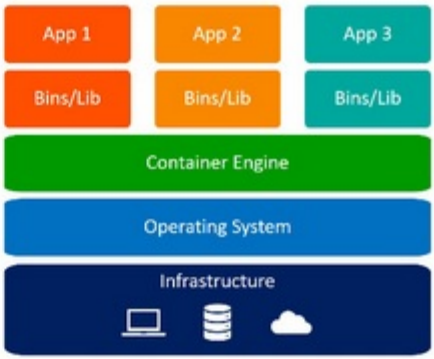
Présentation du concept

Docker la création de conteneurs exécutable de façon isole du reste de la machine hôte. Il est construit à partir de certain outil du noyau Linux, ce qui rend la fonctionnaliter des conteneurs Docker completement different des machines virtuelles qui elle utiliser un système d'exploitation séparé.

L'utilisation de Docker soit des conteneurs, pour la création et utilsation d'application simplifie le déploiment et d'exécution autonome de celle-ci sur des machines physique ou differente machine isolée.



Virtual Machines



Containers

La notion de conteneur

Un conteneur a le meme objectif qu'une machine virtuel. Il permet d'heberger des services sur un même serveur physique tout en els isolants les uns des autres.
Un conteneur est cependant moins figé qu'une machine virtuelle en termes de taille de disque et de ressources allouées.

Contrairement à la virtualisation qui consiste à exécuter de nombreux systèmes d'exploitation sur un seul et même système, les containers se partagent le même noyau de système d'exploitation et isolent les processus de l'application du reste du système.

Le serveur web, la base de données, une application peuvent être exécutée de façon indépendante dans leur conteneur dédié, contenant uniquement les dépendances nécessaires. Chaque conteneur est relié par des réseaux virtuels. Il est possible de monter des volumes de disque de la machine hôte dans un conteneur.

Image Docker et DockerFiles

Votre système d'exploitation est majoritairement composé de 2 choses :
un système de fichiers, et des processus.
Une image Docker représente le système de fichiers, sans les processus.
Elle contient tout ce que vous avez décidé d'y installer
(Java, une base de donnée, un script que vous allez lancer, etc...),
mais est dans un état inerte. Les images sont créées à partir de fichiers de configuration, nommés "Dockerfile", qui décrivent exactement ce qui doit être installé sur le système.

Un conteneur est l'exécution d'une image :
il possède la copie du système de fichiers de l'image, ainsi que la capacité de lancer des processus. En gros, c'est un OS, avec lequel vous pouvez interagir. Dans ce conteneur, vous allez donc pouvoir interagir avec les applications installées dans l'image, exécuter des scripts, faire tourner un serveur, etc.

Pour faire l'analogie avec le monde Java (ou le monde des langages objets en général) :

le "Dockerfile" est votre fichier source (vous décrivez ce que vous voulez)
l'image est le fichier compilé (gardé au chaud en attendant d'être utilisé)
le container est une instance de votre classe (vous pouvez changer ses propriétés, et appeler des méthodes)

Un "Dockerfile" peut être inclus dans d'autres "Dockerfile", et être à la base de plusieurs images différentes. Par exemple, si tous vos projets utilisent MySQL comme SGDB, vous pouvez créer un "Dockerfile" qui installe MySQL, et ensuite créer un autre "Dockerfile" pour chacun de vos projets. Si vous mettez à jour votre "Dockerfile" MySQL, vos images projets pourront êtres mises à jour en même temps. Vous pouvez aussi utiliser une même image pour créer plusieurs conteneurs différents, mais avec les mêmes propriétés (pensez aux instances de classes).

La liste des commandes utilisable dans le Dockerfile

- FROM permet de définir depuis quelle base votre image va être créée.
- MAINTAINER permet de définir l'auteur de l'image, elle s'écrit de la manière suivante Nom <email>.
- RUN permet de lancer une commande, mais aura aussi pour effet de créer une image intermédiaire.
- ADD permet de copier un fichier depuis la machine hôte ou depuis une URL.
- EXPOSE permet d'exposer un port du container vers l'extérieur.
- CMD détermine la commande qui sera exécutée lorsque le container démarrera.
- ENTRYPOINT permet d'ajouter une commande qui sera exécutée par défaut, et ce, même si on choisit d'exécuter une commande différente de la commande standard.
- WORKDIR permet de définir le dossier de travail pour toutes les autres commandes (par exemple RUN, CMD, ENTRYPOINT et ADD).
- ENV permet de définir des variables d'environnements qui pourront ensuite être modifiées grâce au paramètre de la commande run --env <key>=<value>.
- VOLUMES permet de créer un point de montage qui permettra de persister les données. On pourra alors choisir de monter ce volume dans un dossier spécifique en utilisant la commande `run -v :

Dockerfile

```
FROM microsoft/dotnet:sdk AS build-env
WORKDIR /Docker

# Copy csproj and restore as distinct layers

RUN dotnet restore

# Copy everything else and build
COPY . ./
RUN dotnet publish -c Release -o out


# Build runtime image
FROM microsoft/dotnet:aspnetcore-runtime
WORKDIR /app
COPY --from=build-env /app/out .
ENTRYPOINT ["dotnet", "aspnetapp.dll"]
```

Docker Compose

Compose et un outil qui permet de définir et d'exécuter toutes les applications de Docker. Le Docker Compose nous utilisons un fichier YAML pour la configuration des services de notre application. Pour démarrer notre conteneur, il faut utiliser la commande "docker-compose up".

Compose fonctionne dans différents environnements tels que : la production, mise en scène et développement.

Compose est un processus en trois étapes :

1. Pour commencer nous devons définir l'environnement de l'application de manière à ce que Dockerfile puisse être reproduit n'importe où.
2. Ensuite, il faudra définir les services qui composent l'application docker-compose.yml afin qu'ils puissent être exécutés ensemble dans l'environnement isolé.
3. Exécuter docker-compose up et composer démarre et exécute l'intégralité de votre application.

Fichier de configuration

Le docker-compose.yml et le fichier de configuration, ce fichier contient les conteneurs. Exemple d'un docker-compose.yml :

```
version: '3'
services:
  web:
    build: .
    ports:
    - "5000:5000"
    volumes:
    - .:/code
    - logvolume01:/var/log
    links:
    - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

docker-compose up démarre les services décrits dans mon docker-compose.yml et ne me rend pas la main.

docker-compose up -d fait la même chose mais me rend la main une fois que les services sont démarrés.

docker-compose up –build reconstruit les services avant de les lancer.

docker-compose down stoppe les services.

docker-compose restart redémarre l'ensemble des services.

docker-compose restart nginx redémarre un des services (ici **nginx**).

docker-compose exec rails bash me fournit une console bash au sein du conteneur **rails**.

docker-compose exec rails bin/rails db:migrate effectue un **rails db:migrate** au sein du conteneur rails.

docker-compose logs me retourne l'ensemble des logs des services depuis le dernier démarrage et me rend la main.

docker-compose logs -f affiche les logs des services et continue à les « écouter » sans me rendre la main.

docker-compose logs -f rails fait la même chose pour le conteneur **rails** uniquement.

Avantages et Inconvénients

Avantages :

- Permet le partage du conteneur de manière très simple avec d'autres personnes.
- Légèreté des conteneurs pour fonctionner le conteneur n'as pas besoin de prendre beaucoup de MO sur notre disque.
- Gain de temps : rapidité de mise en place de VM.

Inconvénients :

- Temps d'adaptation pour connaître les commandes et prendre ses repères.
- Docker et encore en développement.
- Une communauté limitée
- Multiplication des liens si l'on sépare le PHP de MySQL et de Appache.

Conclusion

Pour conclure, Docker est un outil tres puissance qui permet de crée de multiple environnement sur une meme machine sans limite de performance et sans demander trop de ressource. Il permet donc de ressoudre de nombreux problèmes lorsque l'on travaille sur des environnements different les un des autres.

Sources:

https://fr.wikipedia.org/wiki/Docker_(logiciel)

https://www.grafikart.fr/tutoriels/docker-intro-634

https://docs.docker.com/