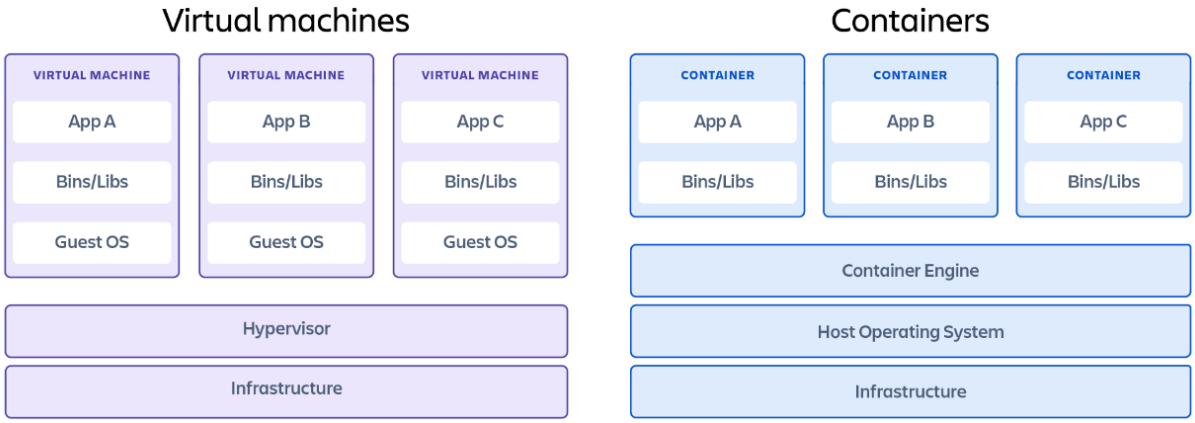
I/ Introduction

Docker est un logiciel open-source qui automatise le déploiement d'applications dans des conteneurs logiciels. Les conteneurs sont des environnements isolés qui permettent d'exécuter des applications dans un environnement prévisible, indépendamment de l'infrastructure physique ou logicielle sous-jacente. Cela signifie que les applications peuvent être déployées de manière fiable et reproductible sur différentes machines, ce qui facilite considérablement le développement, le test et la mise en production.

Docker est utile pour de nombreuses raisons, notamment :

- Il permet de regrouper les dépendances d'une application dans un seul paquet, ce qui facilite le déploiement et la gestion des applications.
- Il permet de créer des environnements de développement et de test identiques à celui de production, ce qui réduit les problèmes de compatibilité.
- Il permet de regrouper plusieurs applications sur une seule machine, ce qui permet d'optimiser l'utilisation des ressources.
- Il facilite la mise à l'échelle des applications en permettant de déployer des conteneurs sur des serveurs distants.
- Il facilite la migration d'applications vers le cloud, car les conteneurs peuvent être déployés sur différents fournisseurs de cloud.

II/ différences entre un conteneur et une machine virtuelle



Les deux sont bien des types de virtualisation cependant ils sont différents dans l'utilisation des ressources de la machine hôte.

Quand on crée une machine virtuelle, il faut allouer une certaine quantité de la mémoire de l'hôte à la machine virtuelle. L'hyperviseur (programme qui gère la virtualisation) établit donc une séparation entre les ressources des différentes VM ainsi que celles de l'hôte. Alors que la création de conteneur ne nécessite pas d'allocation de ressource physique. Ceci est possible parce que tous les conteneurs utilisent directement le Kernel de l'hôte.

III/ Avantage des conteneurs (et de Docker)

Les conteneurs permettent de packager des applications avec leurs dépendances. Ainsi l'application ne dépend plus de l'hôte sur lequel il est déployé (ou développé) mais du conteneur dans lequel il est packagé.

Par rapport à une machine virtuelle, un conteneur est beaucoup plus rapide et plus simple à utiliser. En plus, quand on utilise Docker, on a accès à des milliers d'images pour tout type d'application via Docker Hub.

Par exemple on pourrait utilisée une image MySQL qui n'est rien d'autre qu'un modèle de conteneur dans lequel MySQL est installé et prêt à être utilisé. Cela évite d'installer plein de paquets sur la machine alors que ceux-ci ne serviront plus dans quelques jours.

IV/ Les commandes de base

docker run: Cette commande permet de lancer un conteneur à partir d'une image.
Par exemple, pour lancer un conteneur à partir de l'image "nginx", vous pouvez utiliser la commande suivante : *docker run -d -p 80:80 nginx*
L'option -it permet d'interagir avec le conteneur en cours d'exécution, l'option -d pour le lancer en arrière-plan ou -p pour lier des ports entre la machine hôte et le conteneur.

docker pull: Cette commande permet de télécharger une image depuis un dépôt en ligne (comme Docker Hub).
Par exemple, pour télécharger l'image "nginx", vous pouvez utiliser la commande suivante : docker pull nginx

docker images: Cette commande permet de lister les images disponibles sur la machine locale.
Par exemple, pour lister toutes les images disponibles, vous pouvez utiliser la commande suivante : docker images

docker ps: Cette commande permet de lister les conteneurs en cours d'exécution sur la machine locale.
Par exemple, pour lister tous les conteneurs en cours d'exécution, vous pouvez utiliser la commande suivante : docker ps

docker start: Cette commande permet de démarrer un conteneur qui a été arrêté.
Par exemple, pour démarrer un conteneur avec l'ID "abcd1234", vous pouvez utiliser la commande suivante : docker start abcd1234

docker stop: Cette commande permet d'arrêter un conteneur en cours d'exécution.
Par exemple, pour arrêter un conteneur avec l'ID "abcd1234", vous pouvez utiliser la commande suivante : docker stop abcd1234

docker exec : Cette commande permet de se connecter à un conteneur en cours d'exécution.
Par exemple, pour se connecter à un conteneur avec l'ID "abcd1234" et exécuter la commande "ls", vous pouvez utiliser la commande suivante : docker exec -it abcd1234 ls

docker logs : Cette commande permet d'afficher les logs d'un conteneur.
Par exemple, pour afficher les logs d'un conteneur avec l'ID "abcd1234", vous pouvez utiliser la commande suivante : docker logs abcd1234

docker rm: Cette commande permet de supprimer un conteneur.
Par exemple, pour supprimer un conteneur avec l'ID "abcd1234", vous pouvez utiliser la commande suivante : docker rm abcd1234

docker rmi : Cette commande permet de supprimer une image.
Par exemple, pour supprimer une image avec l'ID "abcd1234", vous pouvez utiliser la commande suivante : docker rmi abcd1234

V/ Les images Docker

Pour créer une image Docker, il faut utiliser la commande "docker build". Cette commande prend en entrée un chemin vers un fichier Dockerfile, qui contient les instructions pour construire l'image.

Voici les étapes pour créer une image à partir d'un fichier Dockerfile:

- Créer un fichier Dockerfile dans le répertoire du projet avec les instructions pour construire l'image.
- Ouvrir une invite de commande ou un terminal et aller dans le répertoire du projet.
- Exécuter la commande "docker build" en spécifiant le chemin vers le fichier Dockerfile.
  Par exemple : "docker build -t myimage:latest ."
- La commande "docker build" va lire les instructions dans le fichier Dockerfile, télécharger les dépendances nécessaires et créer l'image.

Pour gérer les images Docker, vous pouvez utiliser les commandes suivantes :
- "docker images" pour lister toutes les images sur votre système
- "docker rmi" pour supprimer une image spécifique
- "docker push" pour envoyer une image sur un dépôt distant (comme Docker Hub)
- "docker pull" pour télécharger une image à partir d'un dépôt distant

Lorsqu'on utilise une commande "docker build" avec un fichier Dockerfile, cela va créer une image à partir de ces instructions. Cependant, cela ne l'enregistrera pas sur le système de fichiers local. Il faut donc utiliser la commande "docker push" pour l'enregistrer sur le système de fichiers local ou sur un registry distant.

VI/ Les réseaux Docker

Les réseaux Docker permettent de connecter des conteneurs entre eux et à l'extérieur de la machine hôte. Il existe différents types de réseaux dans Docker :
- Le réseau "bridge" (par défaut) : Ce type de réseau est utilisé par défaut lorsqu'on lance un conteneur sans spécifier de réseau. Il crée un pont virtuel entre le conteneur et l'interface de la machine hôte, permettant ainsi au conteneur de communiquer avec d'autres conteneurs et avec l'extérieur.
- Le réseau "host" : Ce type de réseau utilise les interfaces réseau de la machine hôte pour connecter les conteneurs entre eux et à l'extérieur. Les conteneurs utilisant ce type de réseau n'ont pas d'isolation réseau entre eux, ils partagent donc les mêmes ports et IP que la machine hôte.
- Le réseau "none" : Ce type de réseau désactive toute connexion réseau pour le conteneur.

Il est également possible de créer des réseaux personnalisés en utilisant la commande docker network create. Pour connecter un conteneur à un réseau existant, vous pouvez utiliser l'option --network lors de la création d'un conteneur avec la commande docker run ou utiliser la commande docker network connect pour connecter un conteneur existant à un réseau.

1) Les réseaux IPVLAN

Les réseaux IPVlan utilisent les adresses IP de la machine hôte, permettant ainsi aux conteneurs de communiquer directement avec les autres conteneurs et les hôtes du réseau physique.
Pour créer un réseau IPVlan, il faut utiliser l'option --driver pour spécifier "ipvlan" et utiliser l'option --ip-range pour spécifier la plage d'adresses IP à utiliser. Par exemple, pour créer un réseau IPVlan nommé "mynetwork" avec une plage d'adresses IP de 192.168.0.0/24, vous pouvez utiliser la commande suivante :

Il est important de noter que pour utiliser les réseaux IPVlan, il faut disposer d'un noyau Linux qui supporte les modes IPVlan et également des privilèges nécessaires pour configurer les interfaces réseau de votre machine hôte.

2) Les réseaux MACVLAN

Les réseaux macvlan permettent aux conteneurs d'avoir des adresses MAC uniques sur le réseau physique, ce qui leur permet de communiquer directement avec les autres hôtes du réseau physique.
Pour créer un réseau macvlan, il faut utiliser l'option --driver pour spécifier "macvlan" et utiliser l'option --subnet pour spécifier la plage d'adresses IP à utiliser. Par exemple, pour créer un réseau macvlan nommé "mynetwork" avec une plage d'adresses IP de 192.168.0.0/24, vous pouvez utiliser la commande suivante :

```
docker network create --driver=macvlan --subnet=192.168.0.0/24 --gateway=192.168.0.1 -o parent=eth0 mynetwork
```