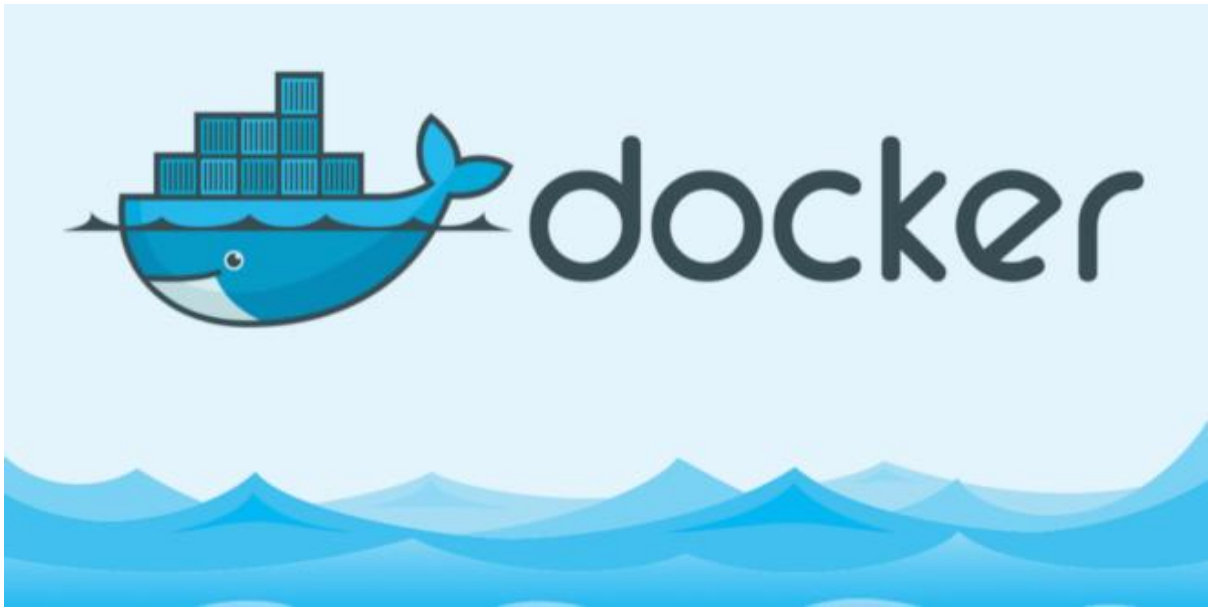
E6 - Compétence

Docker

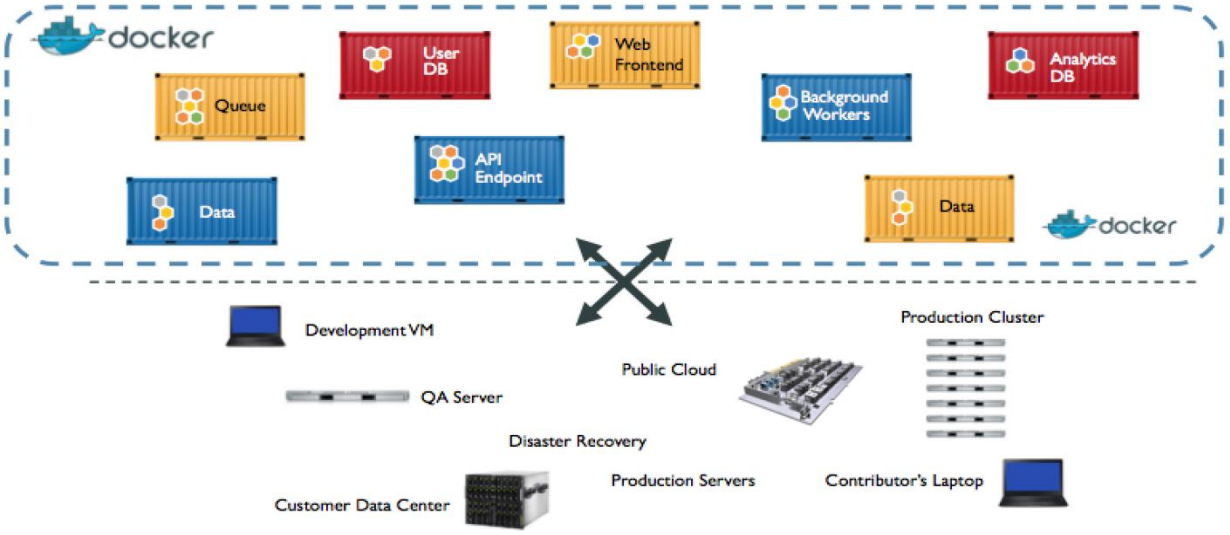


Introduction

Docker est un logiciel libre permettant facilement de lancer des applications dans des conteneurs logiciels. Il permet d'empaqueter (faire un paquet), Docker est un outil qui peut empaqueter une application et ses dépendances dans un conteneur isolé, qui pourra être exécuté sur n'importe quel serveur.

Il ne s'agit pas de virtualisation, mais de conteneurisation, une forme plus légère qui s'appuie sur certaines parties de la machine hôte pour son fonctionnement. Cela permet d'accroître la flexibilité et la portabilité d'exécution d'une application, laquelle va pouvoir tourner de façon fiable et prévisible sur une grande variété de machines hôtes, que ce soit sur la machine locale, un cloud privé ou public, etc.

La technologie de conteneur de Docker peut être utilisée pour étendre des systèmes distribués de façon à ce qu'ils s'exécutent de manière autonome depuis une seule machine ou une seule instance par nœud. Cela permet aux nœuds d'être déployés au fur et à mesure que les ressources sont disponibles, offrant un déploiement transparent et similaire aux PaaS (plate-forme en tant que service).



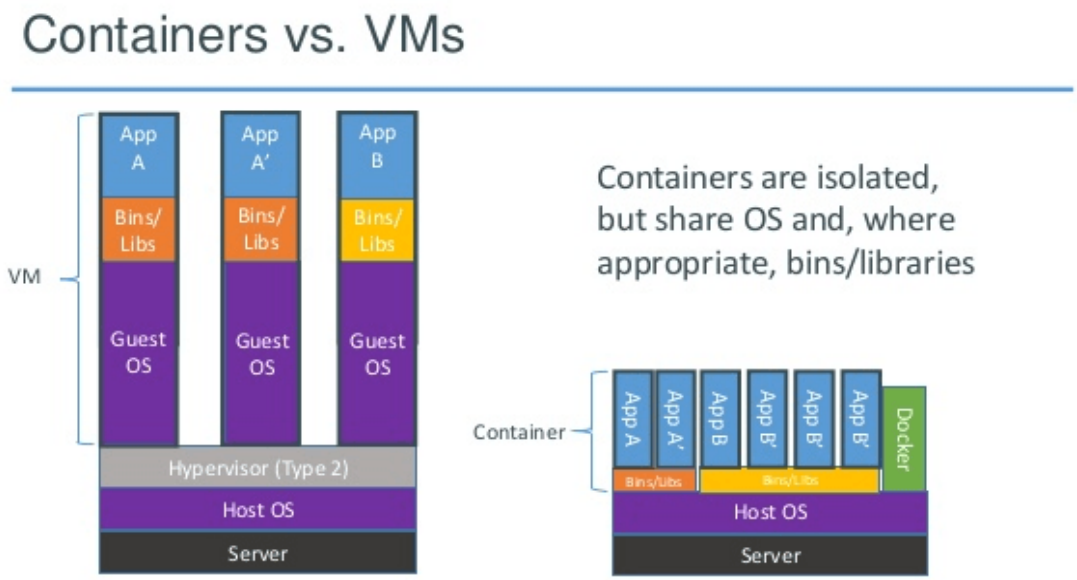
Présentation du concept

Docker permet la mise en œuvre de conteneurs s'exécutant en isolation, via une API. Construit sur des capacités du noyau Linux, un conteneur Docker, à l'opposé de machines virtuelles traditionnelles, il ne requiert aucun système d'exploitation séparé et n'en fournit aucun.

Il s'appuie plutôt sur les fonctionnalités du noyau et utilise l'isolation de ressources (le processeur, la mémoire, les entrées et sorties et les connexions réseau) ainsi que des espaces de noms séparés pour isoler le système d'exploitation. Docker accède aux capacités de virtualisation du noyau Linux, soit directement avec la bibliothèque runc, ou indirectement via libcrt LXC.

Utiliser Docker pour créer et gérer des conteneurs peut simplifier la mise en œuvre de systèmes distribués en permettant à de multiples applications, tâches de fond et autres processus de s'exécuter de façon autonome sur une seule machine physique ou à travers diverses machines isolées.

Ceci permet de déployer des nœuds en tant que ressources sur les besoins, fournissant ainsi une plateforme de déploiement de style PAAS, docker apporte une simplification lors de la création et maintenance de queues de tâches ou autres systèmes distribués.



La notion de conteneur

L'objectif d'un conteneur est le même que pour un serveur dédié virtuel, héberger des services sur un même serveur physique tout en les 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.

Le conteneur permet d'isoler chaque service, il reste très proche des machines virtuelles. Contrairement à la virtualisation, 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

Les images de Docker contiennent plusieurs couches d'application entassée les unes au-dessus des autres, elles dépendent toujours de la couche mère qui est leurs références. Par exemple pour créer un serveur Web avec une base phpMyAdmin sous MySQL, il faut connaître l'OS (à définir l'OS). Docker utilise l'OS de l'hôte, puis il faut que l'on utilise un serveur Web apache et enfin le PHP pour installer phpMyAdmin et MySQL server. L'image crée dispose bien de différentes couches. Elle reste toutes indépendante et modifiable a tous moment (pour changer de version, c'est très pratique ainsi que très rapide.).

- L'image est un fichier compilé (gardé en attendant d'être utilisé)
- Le Dockerfile est un fichier source (on écrit ce que l'on veut)
- Le container est une instance de notre classe (on peut changer ses propriétées, et appeler des méthodes)

Dockerfiles

Un Dockerfile peut être inclus dans d'autres Dockerfile, et être à la base de plusieurs images différentes. Par exemple, si tous les projets utilisent MySQL comme SGDB, vous pouvez créer un Dockerfile qui installe MySQL, et ensuite créer un autre Dockerfile pour chacun des projets. Si on met à jour Dockerfile MySQL, les images, des projets pourront être mis à jour en même temps. On peut donc utiliser une même image pour créer plusieurs conteneurs différents, mais avec les mêmes propriétés.

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 :

Containeur

Le conteneur de docker est un espace, crée sur-mesure avec une taille définit et modifiable à tout instant. Il contient une application ainsi que toutes ses dépendances (bibliothèque et librairie). Plusieurs conteneurs peuvent être créés et être exécuté sur la même machine, elle se partage les ressources du système sur lequel le docker est installé.

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

En conclusion, nous permettons de multiplier les environnements sur notre machine sans limite de performances. Les ressources sont partagées avec la machine hôte donc chaque environnement peut-être configuré avec le Dockerfile, présent à la racine.

Ainsi Docker permet de répondre à de nombreux problèmes lorsque l'on travaille sur des environnements de développement qui ne sont pas les mêmes. Il apparaît comme un outil très intéressant pour le partage d'image entre développeurs, la maintenance de ces images, et avoir des environnements de dev/test indépendant de l'OS.

Bibliographie :

https://fr.wikipedia.org/wiki/Docker_(logiciel)
https://www.lebigdata.fr/docker-definition
https://www.grafikart.fr/tutoriels/docker-intro-634
https://blog.ippon.fr/2014/10/20/docker-pour-les-nu-pour-les-debutants/
https://docs.docker.com/compose/overview/
https://docs.docker.com/compose/
https://actu.alfa-safety.fr/devops/conteneur-docker-fonctionnement-et-avantages-pour-heberger-ses-applications/
https://blog.octo.com/pourquoi-utiliser-docker-en-tant-que-dev/
https://www.lemagit.fr/conseil/Cinq-inconvenients-des-conteneurs-et-comment-y-remedier