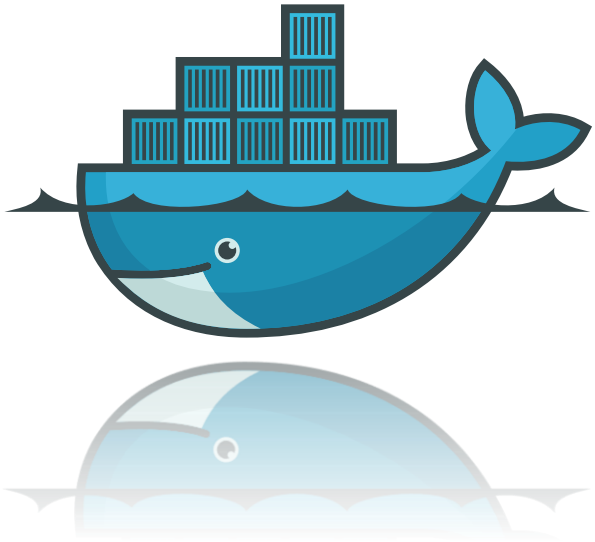


Docker

INITIATION A DOCKER DESKTOP

Léo LAROU-CHALOT
SAFEBEAR | PARIS

Vous avez dis « conteneur » ?

A la différence d'une **machine virtuelle** (VM), qui fait ce que l'on appelle de la virtualisation lourde lorsque l'on recréer un système complet dans le système hôte, un **conteneur Linux** est un **processus** ou un ensemble de processus isolés du reste du système, tout en étant **légers**.

Un 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.

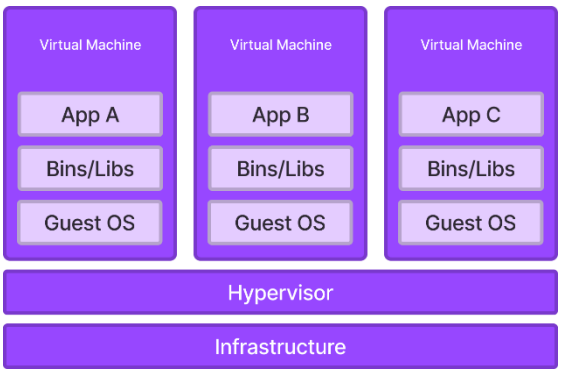
Cette documentation abordera les sujets et problématiques rencontrés en entreprises.

1 MACHINE VIRTUELLE VS CONTENEUR

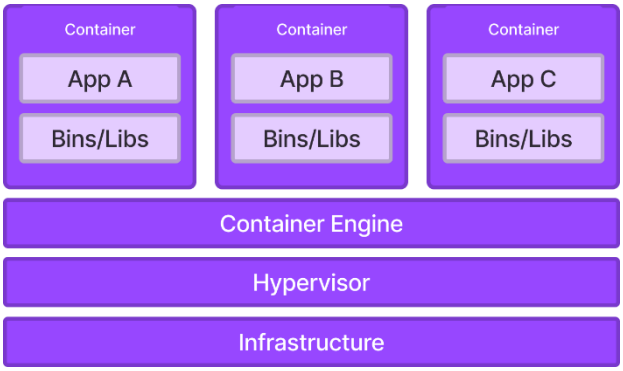
1.1 LA DECOUVERTE

Les conteneurs et les machines virtuelles sont des technologies de virtualisation des ressources très similaires. La virtualisation[1] est le processus par lequel une ressource est singulière au système (RAM, CPU, disque ou réseau) peut être « virtualisé » et représenté sous forme de ressources multiples.

La principale différence entre les conteneurs et les machines virtuelles vient du fait que la machine virtuelle virtualisent toutes une machine, jusqu'aux couches matérielles, tandis que le conteneur ne virtualise que les couches logicielles au-dessus du système d'exploitation.



1-2Machines Virtuelles



1-1Conteneurs

1.2 POURQUOI UILISER DES CONTENEURS ?

Un conteneur ne réserve pas la quantité de CPU, RAM et disque attribué auprès du système hôte. Ainsi, nous pouvons attribuer autant d'espace que nous le souhaitons, mais si celui-ci n'utilise que 2Go, le reste ne sera pas bloqué.

Le conteneurs n'ayant pas besoin d'une virtualisation des ressources mais seulement d'une isolation de celles-ci, ils peuvent démarrer beaucoup plus rapidement et plus fréquemment qu'une VM sur les serveurs hôtes, et ainsi réduire un peu plus les frais de l'infrastructure.

Les conteneurs permettent de réduire les coût, d'augmenter densité de l'infrastructure, tout en améliorant le cycle de céploiement.

2 POURQUOI UTILISER DOCKER ?

2.1 ISOLATION DES ENVIRONNEMENTS

Les conteneurs, comme ceux fournis par Docker, encapsulent une application et toutes ses dépendances, garantissant ainsi une isolation totale par rapport au reste du système. Cela résout souvent des problèmes liés à la compatibilité entre différentes versions de bibliothèques ou de dépendances.

2.2 PORTABILITE

Un conteneur Docker est léger et peut être exécuté de manière cohérente sur n'importe quel système prenant en charge Docker. Cela simplifie le processus de déploiement en garantissant que l'application fonctionne de la même manière sur les environnements de développement, de test et de production.

2.3 RAPIDITE ET EFFICACITE

Les conteneurs partagent le même noyau du système d'exploitation hôte, ce qui les rend plus légers par rapport aux machines virtuelles. Ils peuvent démarrer rapidement et consommer moins de ressources système.

2.4 FACILITE DE GESTION

Docker fournit des outils pour créer, distribuer et gérer des conteneurs. L'automatisation intégrée facilite le déploiement d'applications complexes et la gestion des mises à jour.

2.5 SCALABILITE[2]

La conteneurisation facilite le déploiement et la mise à l'échelle d'applications, que ce soit horizontalement en ajoutant des instances ou verticalement en augmentant les ressources d'une instance.

2.6 RESUME

En résumé, l'utilisation de Docker et de la conteneurisation simplifie le cycle de vie du développement logiciel, améliore la portabilité des applications et offre une gestion plus efficace des ressources. Cela répond aux besoins de l'approche DevOps en favorisant la collaboration entre les équipes de développement et d'exploitation, tout en automatisant les processus pour des déploiements plus rapides et fiables.

3 DEBUTER AVEC DOCKER DESKTOP

3.1 INSTALLATION ET REFERENCES

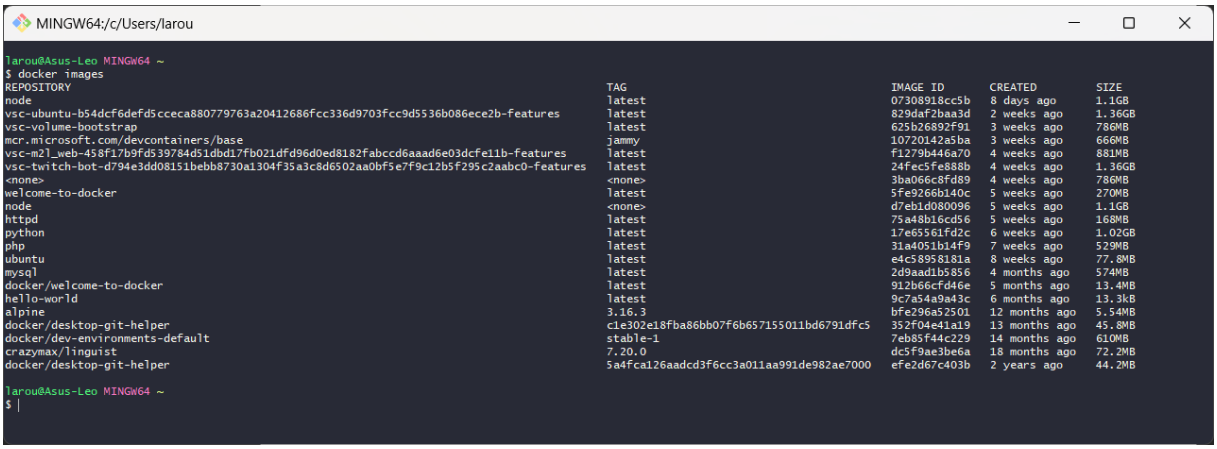
Lien de téléchargement et registry[3] officielle Docker : https://hub.docker.com

3.2 L'INTERFACE EN LIGNES DE COMMANDES

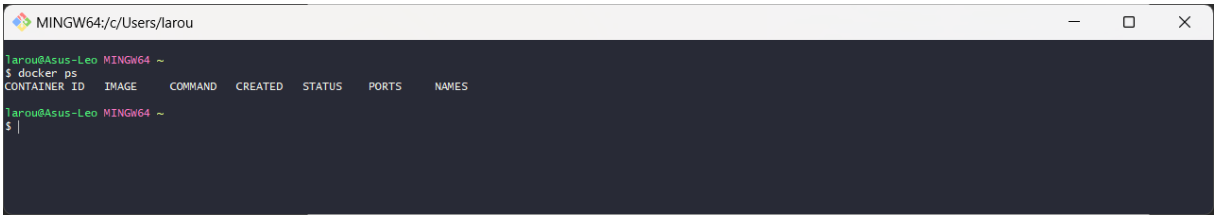
L'interface en ligne de commande est l'outils qui va nous permettre de discuter avec le daemon[4] Docker installé sur notre ordinateur.

3.2.1 Quelques commandes

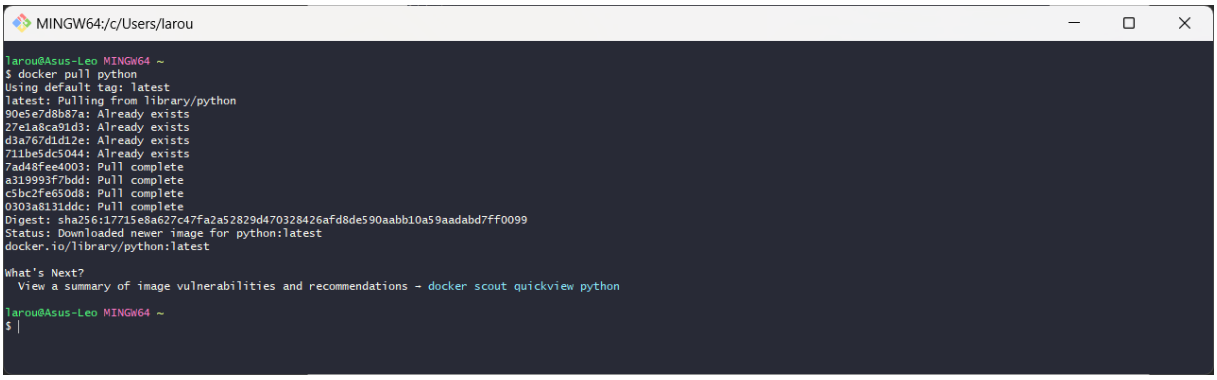
docker images : Permet de lister les images[5] disponible localement ;



docker ps : Permet de lister les conteneurs[6] disponibles localement ;

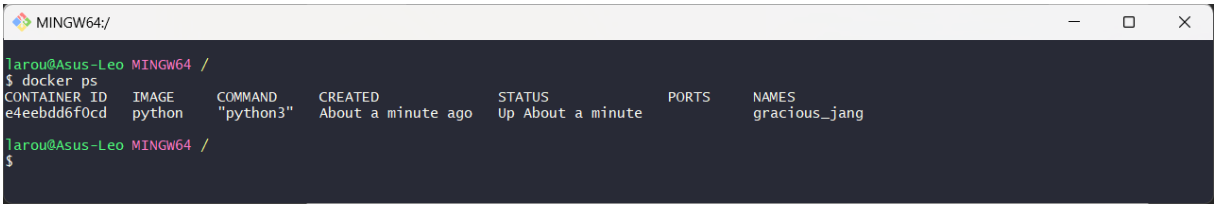


docker pull : Télécharge depuis un dossier distant ou depuis le Docker Hub une image existante ;

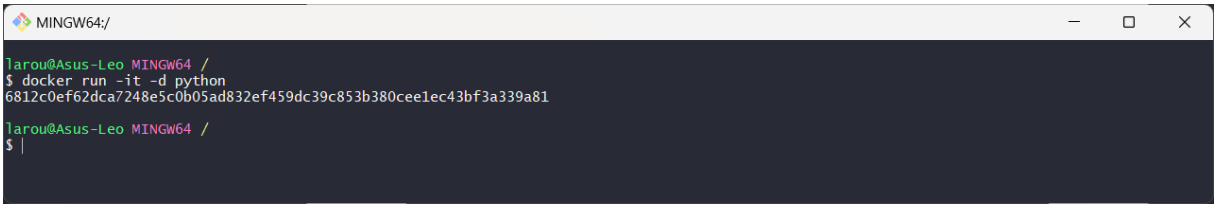


docker run -it [nom de l'image] : Permet de faire tourner le conteneur de façon interractive ;
Le drapeau -it dans le contexte de la commande docker run combine deux indicateurs : -i (interactive) et -t (tty). Ensemble, ils permettent d'ouvrir une session interactive dans le conteneur Docker, ce qui est utile pour les applications en mode terminal.

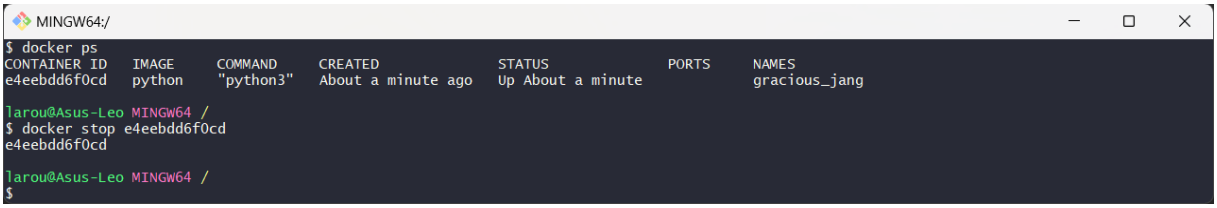
Je peux le vérifier avec **docker ps**



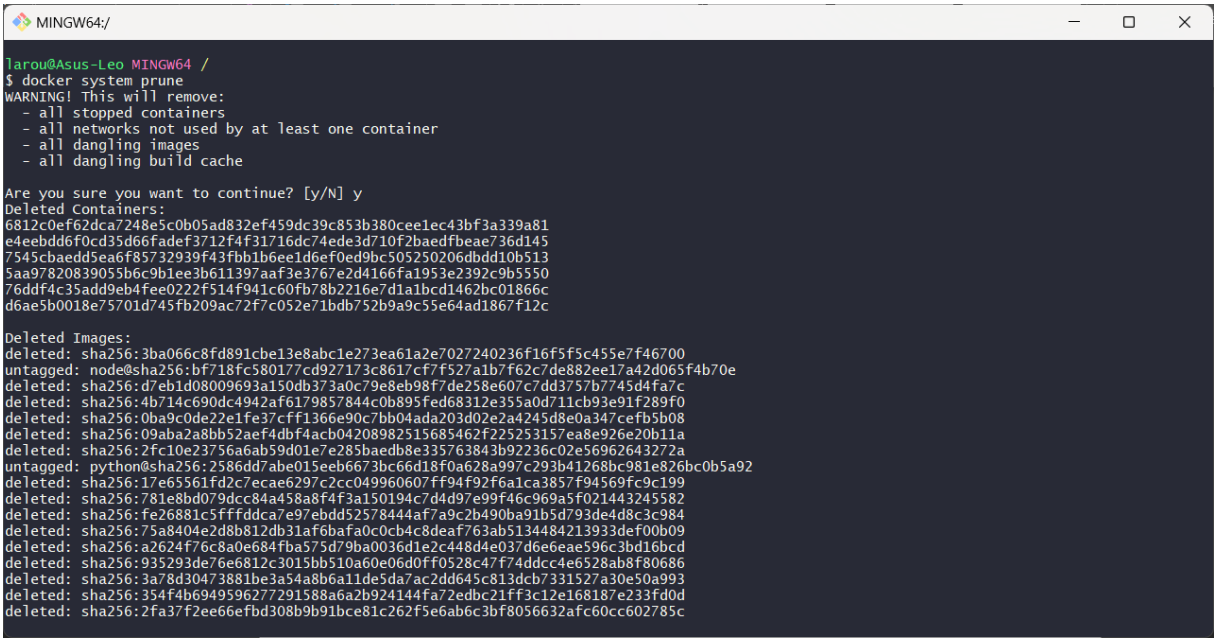
docker run -it -d [nom de l'image] : Permet de faire tourner le conteneur en détaché (arrière plan)



docker stop [id de l'image] : Permet d'arrêter l'execution du conteneur



docker system prune : Permet de nettoyer l'ensemble des conteneurs, réseaux, images et caches



3.3 CREATION D'UN DOCKERFILE

Pour pouvoir créer une image, nous aurons besoin de créer un **dockerfile.**

Un dockerfile est au conteneur ce que package.json est à Node ou ce que composer.json est à PHP.

Il faut voir cela comme un fichier de configuration.

4 ANNEXES

4.1 DEFINITIONS

4.1.1 Virtualisation

Processus par lequel une ressource est singulière au système (RAM, CPU, disque ou réseau) peut être « virtualisé » et représenté sous forme de ressources multiples.

4.1.2 Scalabilité

"Scalable" (ou "scalabilité" en français) se réfère à la capacité d'un système, d'une application ou d'une architecture à évoluer et à gérer une augmentation de la charge de travail de manière efficace, sans compromettre ses performances. En d'autres termes, un système scalable peut s'adapter à une croissance ou à une diminution de la demande sans sacrifier ses fonctionnalités ou sa qualité de service.

4.1.3 Registry

Une registry est un logiciel qui permet de partager des images à d'autres personnes. C'est un composant majeur dans l'écosystème Docker, car il permet à :

- des développeurs de distribuer des images prêtes à l'emploi et de les versionner avec un système de tags ;
- des outils d'intégration en continu de jouer une suite de tests, sans avoir besoin d'autre chose que de Docker ;
- des systèmes automatisés de déployer ces applications sur vos environnements de développement et de production.

4.1.4 Daemon

un "daemon" (ou démon en français) est un programme informatique qui s'exécute en arrière-plan de manière continue, sans l'interaction directe d'un utilisateur. Les démons sont souvent utilisés pour fournir des services ou des fonctionnalités système, et ils sont généralement initiés au démarrage du système d'exploitation.

4.1.5 Image

Une image Docker est un ensemble immuable de fichiers, de configurations et de dépendances nécessaires pour exécuter une application. Ces images sont créées à partir d'un ensemble de directives appelées "Dockerfile", qui spécifient comment l'environnement de l'application doit être configuré.

4.1.6 Conteneur

Un conteneur Docker est une instance en cours d'exécution d'une image. Il s'agit d'un processus léger qui encapsule l'application et ses dépendances, fonctionnant de manière isolée par rapport au système hôte et à d'autres conteneurs. Les conteneurs utilisent le noyau du système hôte et partagent certaines ressources tout en restant isolés les uns des autres.