Conteneurisation et Orchestration

Partie 3 : Docker

Plongeons dans l'écosystème Docker

Instances:

- immutables
- éphémères (non persistentes)
- Portables

Le Problème Classique...

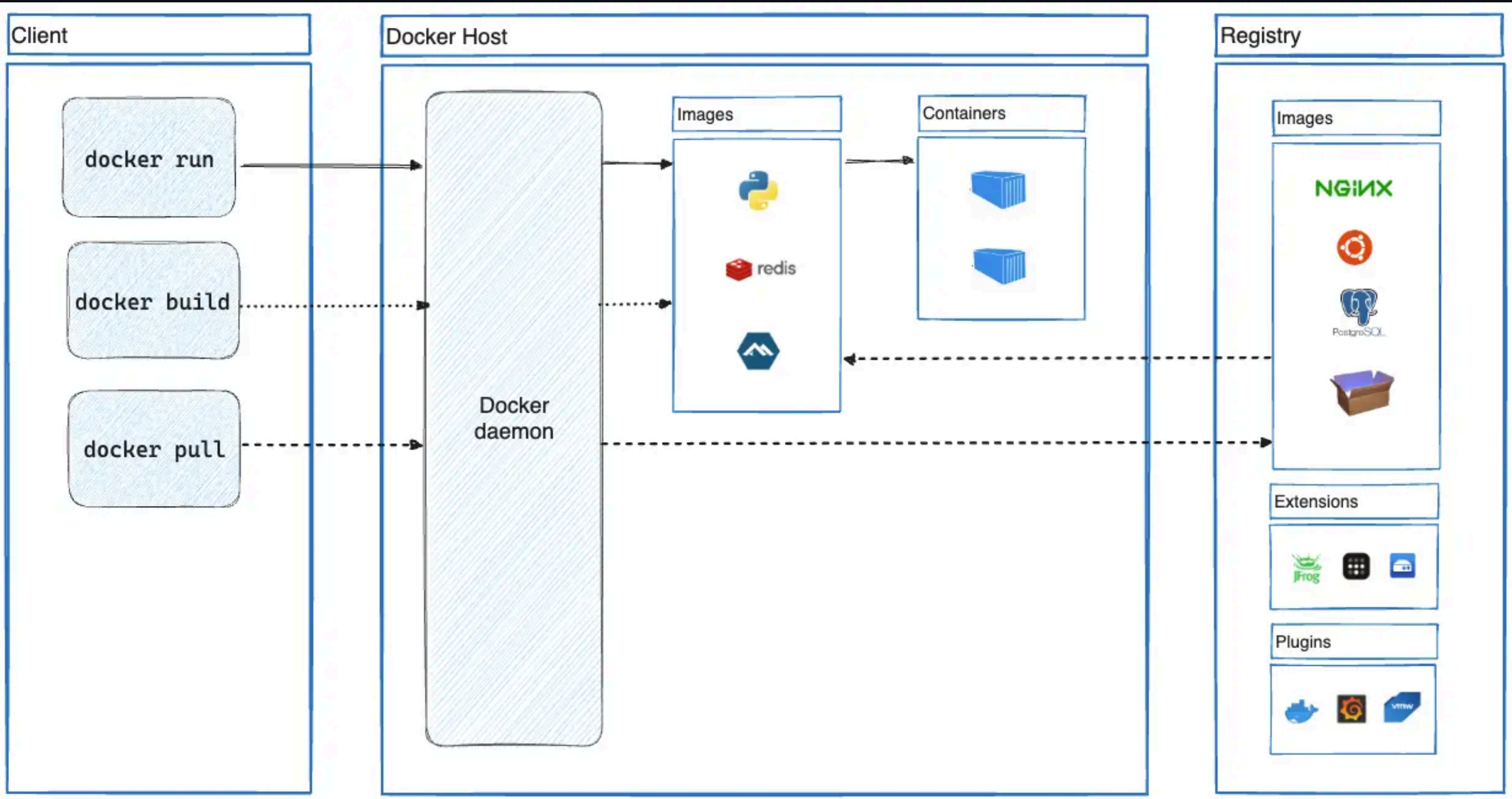
"Ça marche sur ma machine !"

- Différences d'OS
- Conflits de versions
- Variables d'environnement
- Configuration manquante

C'est quoi Docker ? (Définition Technique)

Docker est une plateforme de conteneurisation qui utilise une architecture client-serveur.

- **Le Démon Docker (`dockerd`)** : C'est le service de fond (le serveur) qui écoute les requêtes de l'API Docker et gère les objets : images, conteneurs, réseaux et volumes.
- **Le Client Docker (CLI `docker`)** : C'est l'outil en ligne de commande (le client) que nous utilisons pour interagir avec le démon.
- **L'API REST** : Le client communique avec le démon via l'API.


Client
Docker Host
Registry
docker run
docker build
docker pull
Docker daemon
Images
Containers
redis
Images
NGINX
PostgreSQL
Extensions
JFrog
Plugins

L'Histoire de Docker

Docker a démocratisé la conteneurisation et a transformé le cycle de vie du développement logiciel.

- **Créateur** : **Solomon Hykes**, un ingénieur franco-américain.
- **Origine (2010)** : Le projet a démarré comme une initiative personnelle au sein de la société **dotCloud**, une plateforme PaaS. L'objectif, simplifier le déploiement d'applications.
- **Lancement (2013)** : Docker est présenté au public et publié en open-source. Son adoption est fulgurante.
- **Docker 1.0 (2014)** : La première version stable est lancée, marquant le début de son utilisation en production.

Concepts Fondamentaux

- **Image Docker** : Un template **immuable (en lecture seule)** qui contient le code de l'application, les bibliothèques, les dépendances et les outils nécessaires à son exécution.
- **Conteneur Docker** : Une **instance exécutable** d'une image. C'est un processus isolé qui possède son propre système de fichiers, son propre réseau et son propre espace de processus, mais qui partage le noyau du système d'exploitation hôte.

Docker Hub : Le Registre d'Images

Docker Hub est un service de **registre** hébergé dans le cloud pour les images Docker.

- **Référentiel Central** : Permet de stocker, partager et gérer des images.
- **Images Officielles** : Contient des images pré-construites et sécurisées pour des logiciels populaires (maintenues par les éditeurs).
- **Référentiels Publics et Privés** : Permet de

Les Volumes : Persistance des Données

Un **volume** est le mécanisme privilégié pour rendre les données des conteneurs persistantes.

- **Gestion par Docker** : Les volumes sont créés et gérés par Docker. Ils sont stockés dans une partie du système de fichiers de l'hôte qui est gérée par Docker (`/var/lib/docker/volumes/` on Linux).
- **Découplage** : Le cycle de vie d'un volume est indépendant du cycle de vie d'un conteneur.

Les Réseaux : Communication des Conteneurs

Le système de réseau de Docker permet aux conteneurs de communiquer entre eux et avec des systèmes externes.

- **Réseau `bridge` (par défaut)** : Crée un réseau privé interne à l'hôte. Les conteneurs sur le même réseau bridge peuvent communiquer entre eux en utilisant leur nom comme nom DNS.
- **Exposition de ports (`-p HOTE:CONTENEUR`)** : Mappe un port du conteneur à un port de la machine hôte, rendant le service accessible de l'extérieur.

Le Risque de Sécurité : Le Daemon Docker

Le modèle traditionnel de Docker présente un risque de sécurité majeur :

- **Le Daemon tourne en `root`** : Le processus central de Docker (le daemon) s'exécute avec les privilèges les plus élevés. Si un attaquant compromet ce daemon, il obtient un accès complet au système hôte.
- **Point de Défaillance Unique** : Tout le cycle de vie des conteneurs est géré par ce seul processus centralisé.

Une vulnérabilité dans un conteneur pourrait permettre à un attaquant de "s'échapper" et de prendre le contrôle de l'hôte.

L'Alternative : Podman (Rootless & Daemonless)

Podman est une alternative à Docker qui a été conçue avec la sécurité comme priorité, en reprenant la philosophie de `rkt`.

- **Rootless (Sans `root`)** : Les conteneurs peuvent être gérés par des utilisateurs non-privilégiés. Même si un conteneur est compromis, l'attaquant n'obtient que les droits limités de cet utilisateur, pas ceux de l'hôte.
- **Daemonless (Sans daemon)** : Il n'y a pas de processus central qui tourne en tâche de fond. Chaque conteneur est un processus enfant direct de l'utilisateur, ce qui réduit la surface d'attaque.

Podman offre une meilleure isolation et une sécurité renforcée, ce qui en fait un choix de plus en plus populaire dans les environnements de production.

Installation et Prise en Main

Docker est disponible sur les principaux systèmes d'exploitation.

- **Linux** : Installation native. C'est l'environnement de prédilection de Docker.
  - `sudo apt-get install docker-ce` (sur Debian/Ubuntu)
- **Windows** : Via **Docker Desktop**, qui s'appuie sur **WSL 2**.
- **macOS** : Via **Docker Desktop**, qui utilise une machine virtuelle légère.
  - **Alternative Open Source** : **Colima**, qui est plus léger.

Vérifier l'Installation

La première commande à exécuter après l'installation est `hello-world`.

```
docker run hello-world
```

Cette simple commande effectue plusieurs actions :

1. Le client Docker contacte le démon Docker.
2. Le démon cherche l'image `hello-world` localement.
3. Ne la trouvant pas, il la télécharge (`pull`) depuis Docker Hub.
4. Il crée un nouveau conteneur à partir de cette image.
5. Il exécute le programme dans le conteneur, qui affiche un message de bienvenue.
6. Le conteneur se termine et est supprimé.

La CLI Docker : Commandes Essentielles

Gérer les Images (1/2)

- Chercher une image sur Docker Hub

  ```
  docker search nginx
  ```

- Télécharger une image

  ```
  docker pull nginx:latest
  ```

- Lister les images locales

  ```
  docker images
  ```

- Construire une image depuis un `Dockerfile`

  ```
  docker build -t mon-app:1.0 .
  ```

La CLI Docker : Commandes Essentielles

Gérer les Images (2/2)

- Gérer les versions (tags)

  ```
  docker tag mon-app:1.0 mon-app:latest
  ```

- Supprimer une image

  ```
  docker rmi <ID_IMAGE>
  ```

La CLI Docker : Commandes Essentielles

Gérer les Conteneurs

- Démarrer un conteneur
  `docker run -d -p 8080:80 --name web nginx`
- Lister les conteneurs actifs / tous les conteneurs
  `docker ps` / `docker ps -a`
- Arrêter / Démarrer / Redémarrer un conteneur
  `docker stop web` / `docker start web` / `docker restart web`
- Voir les logs d'un conteneur
  `docker logs web`
- Supprimer un conteneur (doit être arrêté)
  `docker rm web`

La CLI Docker : Commandes Essentielles

Nettoyer les Ressources

Docker peut rapidement consommer de l'espace disque avec des objets inutilisés.

`docker system prune` est la commande de nettoyage.

- **Supprime les conteneurs arrêtés.**
- **Supprime les réseaux non utilisés.**
- **Supprime les images "dangling"** (images sans tag, souvent des couches intermédiaires de build).

Pour un nettoyage agressif (supprime aussi les images `unused` et les volumes) :

```
# Attention, cette commande est destructive !
docker system prune -a -f --volumes
```

Conclusion sur Docker

- **Docker** a standardisé la conteneurisation, la rendant accessible à tous.
- Les **images**, **conteneurs**, **volumes** et **réseaux** sont les briques de base de tout projet conteneurisé.
- La **CLI Docker** est un outil puissant pour gérer l'ensemble du cycle de vie de vos applications.