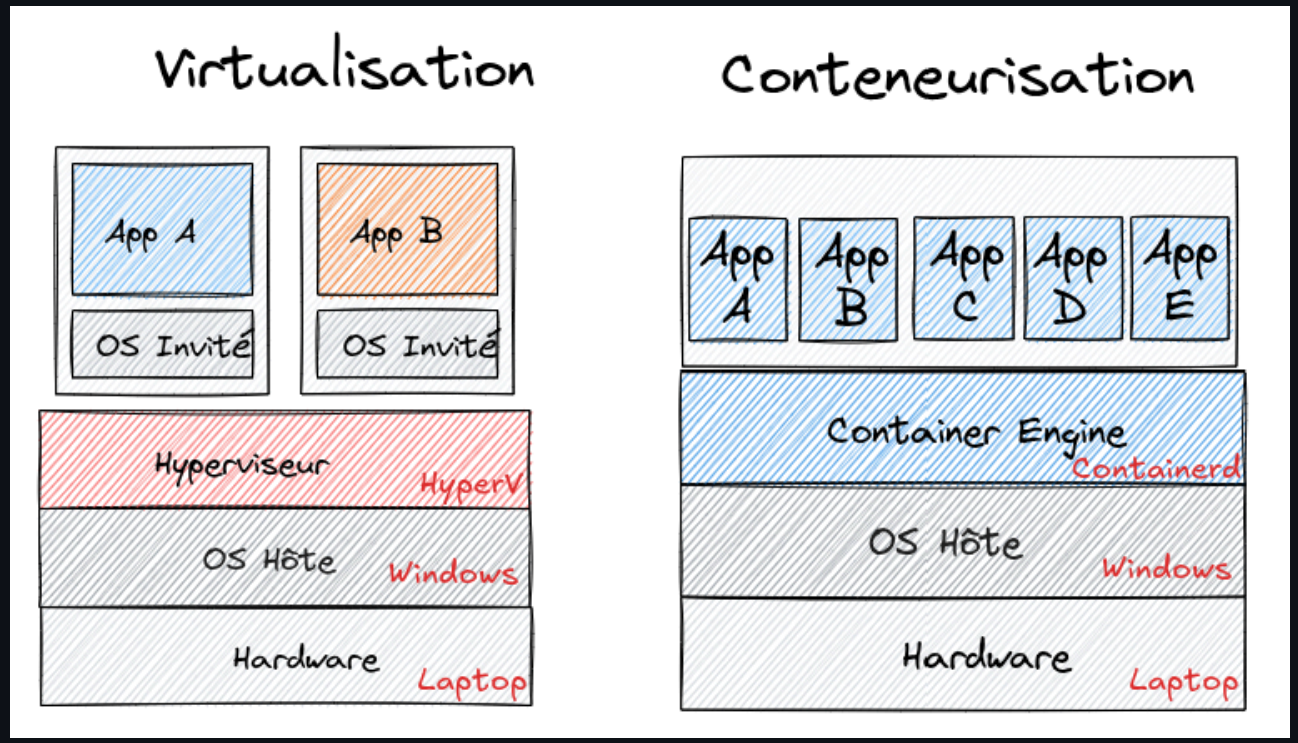
Conteneurisation et Orchestration

Partie 2 : La Conteneurisation

La conteneurisation isole une application et ses dépendances dans un **conteneur**. C'est une alternative légère à la virtualisation.



FreeBSD Jails expand on Unix chroot to isolate files

Jails

2000

VServer

2001

Linux-VServer ports kernel isolation, but requires recompilation

Solaris Zones bring the concept of snapshots

SOLARIS

Zones

2004

cgroups

Google

Google introduces Process Containers, merged as cgroups

2006

Red Hat adds user namespaces, limiting root access in containers

redhat

Namespaces

2008

LXC

IBM

IBM creates LXC, providing user tools for cgroups and namespaces

2008

Docker provides simple user tools and images. Containers go mainstream

2013

docker

Docker

Comment ça marche ? La magie du noyau Linux

Les conteneurs exploitent des fonctionnalités natives du noyau Linux pour l'isolation :

- **Namespaces** : Cloisonnent les ressources système. Chaque conteneur a sa propre vue des processus, des réseaux, des montages, etc.
- **cgroups (Control Groups)** : Limitent et contrôlent l'utilisation des ressources (CPU, RAM, I/O) pour chaque conteneur.
- **Capabilities** : Affinent les privilèges `root` pour renforcer la sécurité.

Résultat : Les conteneurs sont des processus isolés et contrôlés sur le même OS hôte.

Les Namespaces

Les **Namespaces** créent une illusion : chaque conteneur pense avoir son propre système d'exploitation.

Namespace	Rôle
PID	Isole les processus (un conteneur ne voit que ses propres processus).
NET	Isole les interfaces réseau, les ports, les tables de routage.
MNT	Isole les points de montage du système de fichiers.
UTS	Isole le nom d'hôte (hostname).
IPC	Isole les communications inter-processus.
USER	Mappe les utilisateurs (un `root` dans le conteneur n'est pas `root` sur l'hôte).

Les cgroups (Control Groups)

Les **cgroups** sont le "gouverneur" des ressources. Ils garantissent qu'un conteneur "gourmand" ne mettra pas en péril tout le système.

- **Limiter la consommation** :
  - "Ce conteneur ne peut pas utiliser plus de 2 Go de RAM."
  - "Ce conteneur est limité à 50% d'un cœur de CPU."
- **Prioriser les ressources** :
  - Donner plus de temps CPU à un conteneur critique.
- **Monitorer l'utilisation** :
  - Suivre la consommation de chaque conteneur.

L'Évolution vers cgroups v2

La version 2 des cgroups, maintenant standard, simplifie grandement la gestion des ressources.

Principale amélioration : une hiérarchie unifiée.

- **cgroups v1** : Hiérarchies multiples et complexes (une par ressource : CPU, mémoire...).
- **cgroups v2** : Une seule arborescence pour gérer toutes les ressources, ce qui rend la configuration plus simple et cohérente.

Contrôles simplifiés :

- **CPU** : `cpu.max` pour limiter le temps CPU.
- **Mémoire** : `memory.max` pour plafonner la RAM.
- **I/O** : `io.max` pour contrôler les accès disque.

Les Capabilities

Le modèle de sécurité de Linux est traditionnellement binaire : soit on est `root` (tous les droits), soit on est un utilisateur (droits limités).

Les **Capabilities** découpent les privilèges de `root` en unités plus petites et granulaires.

Exemple :

- Un conteneur a besoin d'écouter sur le port 80 (privilégié).
- Au lieu de lui donner `root`, on lui donne uniquement la capability `CAP_NET_BIND_SERVICE`.

Cela réduit considérablement la surface d'attaque en cas de compromission.

LXC vs Containerd : Les moteurs sous le capot

LXC (Linux Containers) - 2008

- **Le pionnier** : LXC est l'implémentation originale de la conteneurisation sous Linux.
- **Approche "VM légère"** : Il vise à simuler un environnement de système d'exploitation complet, avec un `init`, des services, etc.
- **Gestion d'OS** : On peut se connecter en SSH à un conteneur LXC comme on le ferait avec une VM.
- **Utilisation** : Idéal pour isoler des applications complètes ou des environnements qui nécessitent une structure d'OS traditionnelle. Proxmox l'utilise beaucoup.

Containerd - 2015

- **Le standard industriel** : Initialement un composant de Docker, `containerd` est maintenant un projet indépendant de la CNCF.
- **Focalisé sur l'exécution** : Son rôle est simple et robuste : gérer le cycle de vie des conteneurs (démarrage, arrêt, etc.). Il n'incorpore pas la gestion d'images, de volumes ou de réseau (délégué à d'autres composants).
- **Le cœur de l'écosystème** : C'est le moteur d'exécution par défaut de Docker et de Kubernetes.
- **Minimaliste et performant** : Conçu pour être intégré dans des plateformes plus larges.

LXC vs. Containerd

Caractéristique	LXC (Linux Containers)	Containerd
Philosophie	VM légère, OS complet	Exécution de conteneurs minimaliste
Cas d'usage	Isoler des systèmes complets	Exécuter des applications uniques (microservices)
Écosystème	Proxmox, environnements legacy	**Docker, Kubernetes**, Cloud Native
Gestion	Similaire à une VM	Orienté API, intégré à des orchestrateurs
Standardisation	Moins répandu aujourd'hui	**Standard de l'industrie**

En résumé : LXC est pour les "systèmes", Containerd est pour les "applications".

Avantages de la Conteneurisation

- **Portabilité** : Fonctionne sur n'importe quel système supportant les conteneurs.
- **Efficacité** : Partage du noyau hôte, donc moins de ressources consommées.
- **Déploiement Rapide** : Démarrage quasi-instantané.
- **Isolation** : Les applications ne s'interfèrent pas entre elles.
- **Scalabilité** : Facilite la mise à l'échelle horizontale des applications.
- **Développement Simplifié** : Environnements de dev, test et prod identiques.

Conteneur vs. Machine Virtuelle (VM)

Caractéristique	Machine Virtuelle (VM)	Conteneur
Isolation	**Matérielle** (Hyperviseur)	**OS / Processus** (Noyau partagé)
OS	OS invité complet par VM	Partage le noyau de l'OS hôte
Taille	Lourde (plusieurs Go)	Légère (quelques Mo)
Démarrage	Lent (minutes)	Rapide (secondes)
Performance	Moins performante (overhead)	Quasi-native
Portabilité	Limitée par l'hyperviseur	Très portable (sur tout OS)

Développer sur Windows avec Linux ?

Historiquement, utiliser des outils Linux sur Windows était un défi. Mais une solution a tout changé...

C'est quoi WSL ?

WSL (Windows Subsystem for Linux) est une fonctionnalité de Windows qui permet d'exécuter un environnement Linux complet directement sur Windows, sans avoir besoin d'une machine virtuelle traditionnelle.

- **Objectif** : Offrir aux développeurs un accès facile aux outils et utilitaires Linux (bash, grep, etc.) tout en restant sur leur OS Windows principal.
- **Intégration** : Permet une interaction fluide entre les fichiers Windows et Linux.
- **Deux versions majeures** : WSL 1 et WSL 2, chacune avec une architecture différente et des capacités distinctes.

WSL a révolutionné le développement multi-plateforme sur Windows.

Linux sur Windows : L'ère WSL (2016)

Avant WSL 2, faire tourner Linux sur Windows était complexe.

WSL 1 (Windows Subsystem for Linux) :

- **Principe** : Une couche de traduction qui convertissait les appels système Linux en appels système Windows.
- **Avantage** : Permettait d'exécuter des binaires Linux (comme `bash`, `grep`) directement sur Windows.
- **Inconvénient majeur pour Docker** : Ne possédait pas de vrai noyau Linux. Docker ne pouvait donc pas y tourner nativement. Les performances étaient souvent médiocres.

Cette limitation a conduit au développement de WSL 2.

Docker sur Windows : La Révolution WSL 2 (2019)

WSL 2 a transformé l'utilisation de Docker sur Windows.

- **Un vrai noyau Linux** : Contrairement à WSL 1, WSL 2 intègre une machine virtuelle légère avec un véritable noyau Linux.
- **Compatibilité native avec Docker** : Cela permet à Docker Desktop de fonctionner de manière native et performante, sans la surcharge d'une VM traditionnelle.
- **Performances améliorées** : Accès aux fichiers et exécution des conteneurs beaucoup plus rapides.
- **Simplification du développement** : Les développeurs Windows peuvent désormais profiter pleinement de l'écosystème Linux et Docker directement depuis leur machine.

L'Avenir de la Conteneurisation

La conteneurisation va au-delà du simple déploiement pour devenir le socle unifié de l'informatique, du cloud au poste de travail.

- **Fusion avec la Virtualisation** : Les conteneurs s'exécuteront par défaut dans des **micro-VMs** (Kata Containers, Firecracker), alliant la **sécurité de la VM** à la **légèreté du conteneur**. Ce sera le standard pour les workloads sécurisés.
- **Le Bureau "Cloud Native"** : Les OS de bureau (comme UBlue/Bluefin) seront gérés comme des images de conteneurs, garantissant des environnements de développement **reproductibles, immuables et parfaitement alignés avec la production**.
- **Standardisation IA/ML** : Le conteneur deviendra le format universel pour **packager, entraîner et déployer les modèles d'IA**, assurant la portabilité et la reproductibilité des workloads complexes.

Conclusion sur la conteneurisation

Elle a révolutionné le développement et le déploiement d'applications grâce à :

- **L'isolation et la légèreté** : Partage du noyau hôte pour une efficacité maximale.
- **La portabilité** : "Build once, run anywhere" sur tout environnement compatible.
- **La standardisation** : Un format universel pour packager et distribuer les applications.
- **L'accélération du DevOps** : Facilite l'intégration et le déploiement continus.
- **Le Cloud Native** : Pilier des architectures microservices et du cloud moderne.

La conteneurisation est désormais un élément incontournable de l'écosystème logiciel.