

Introduction aux Conteneurs

Concepts et Historique de la Conteneurisation

Emmanuel BRUNO

Université de Toulon, LIS UMR CNRS 7020

emmanuel.bruno@univ-tln.fr

2026-10-05

Table des matières

0.1 Objectifs du cours	1
0.2 Rappel minimal : Ordinateur et Système d'exploitation	2
0.2.a Architecture simplifiée	2
0.2.b Rôle du noyau (kernel)	2
0.3 Objectifs de la Virtualisation	2
0.3.a Historique de la Virtualisation .	2
0.4 Architecture de la Virtualisation . . .	2
0.4.a Hyperviseurs et Technologies .	3
0.5 Des Machines Virtuelles	3
0.5.a Bilan sur les Machines Virtuelles	3
0.6 L'Émergence des Conteneurs (2010+)	3
0.6.a Fonctionnalités clés du noyau Linux	3
0.6.b Chronologie	4
0.7 Machines Virtuelles vs Conteneurs .	4
0.8 Philosophie et grands principes des conteneurs	4
0.8.a Principes clés	4
0.8.b Message central	4
0.9 Docker et Conteneurs	4
0.10 Docker et standards OCI	5
0.11 Concepts Fondamentaux	5
0.12 Utilisation d'une image	5
0.12.a Ce que fait réellement <code>docker</code> <code>run</code>	5

0.13 Création d'une image	5
0.14 Gestion de base	5
0.15 Shell et inspection	6
0.16 Nettoyage	6
0.17 Composants standards	6
0.18 Orchestration (aperçu)	6
0.19 Conclusion	6

0.1 Objectifs du cours

À l'issue de ce cours, vous devez être capables de :

- Comprendre l'évolution **Physique** → **VM** → **Conteneurs**
- Expliquer la différence entre **machine virtuelle** et **conteneur**
- Utiliser une **image** pour créer un conteneur
- Créer une **image simple** avec un Dockerfile

Ce cours est une **introduction conceptuelle**. Les pratiques et usages avancés seront abordés dans les TP et cours suivants.

0.2 Rappel minimal : Ordinateur et Système d'exploitation

Objectif Comprendre ce qui est virtualisé, partagé ou isolé.

0.2.a Architecture simplifiée

Applications

Système d'exploitation (Kernel)

Matériel (CPU, RAM, Disque, Réseau)

0.2.b Rôle du noyau (kernel)

- Gestion des processus
- Gestion de la mémoire
- Accès au matériel
- Sécurité et isolation

💡 Astuce

Toute application passe par le **noyau** pour accéder aux ressources.

0.3 Objectifs de la Virtualisation

La virtualisation répond à deux problèmes simples :

- **comment mieux exploiter le matériel disponible ?**
- **comment isoler les applications ?**

0.3.a Historique de la Virtualisation

Ère Physique (1990–2000)

• Infrastructure

- Serveurs dédiés mono-usage
- Faible utilisation CPU
- Scalabilité verticale uniquement

• Contraintes

- Coûts élevés
- Déploiements lents
- Maintenance complexe
- Conflits de dépendances

Ère Virtuelle (2000–2010)

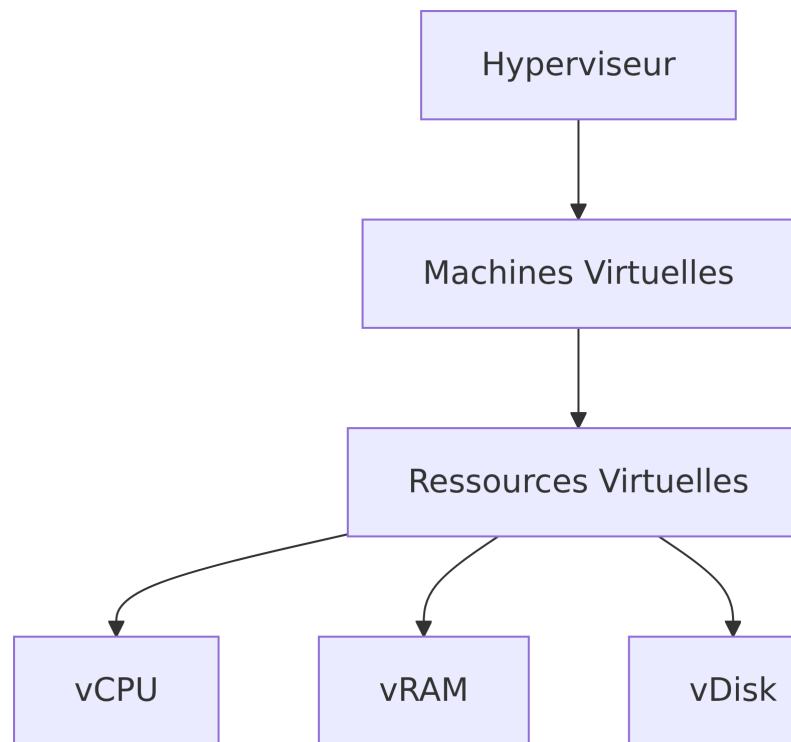
• Technologies

- Hyperviseurs : VMware ESX, Hyper-V
- Open Source : KVM, Xen

• Bénéfices

- Consolidation des serveurs
- Isolation forte
- Portabilité des VMs
- Haute disponibilité

0.4 Architecture de la Virtualisation



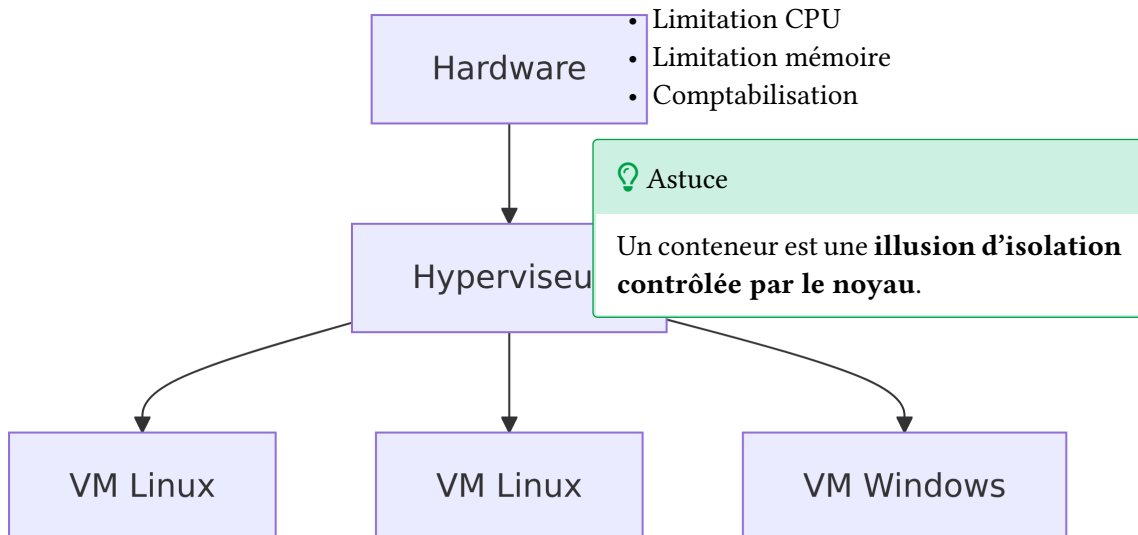
💡 Astuce

Une machine virtuelle **embarque un OS complet.**

0.4.a Hyperviseurs et Technologies

- **CPU** : Intel VT-x / AMD-V
- **Mémoire** : EPT / NPT
- **I/O** : SR-IOV, NVMe-oF
- **Stockage** : VMDK, QCOW2, SAN
- **Type 1 (Bare Metal)** : ex. VMware ESX
 - Direct sur le matériel
 - Performances élevées
- **Type 2 (Hosted)** : ex. VirtualBox, VMware Workstation
 - Sur OS existant
 - Simplicité d'usage

0.5 Des Machines Virtuelles



- Chaque VM a son propre OS
- Isolation forte via l'hyperviseur
- Ressources allouées statiquement
- Démarrage lent et consommation mémoire élevée

0.5.a Bilan sur les Machines Virtuelles

Avantages

- Isolation forte
- Snapshots
- Migration

Inconvénients

- OS complet par VM
- Démarrage lent
- Consommation mémoire

0.6 L'Émergence des Conteneurs (2010+)

Les conteneurs ne virtualisent pas une machine, mais **isolent des processus.**

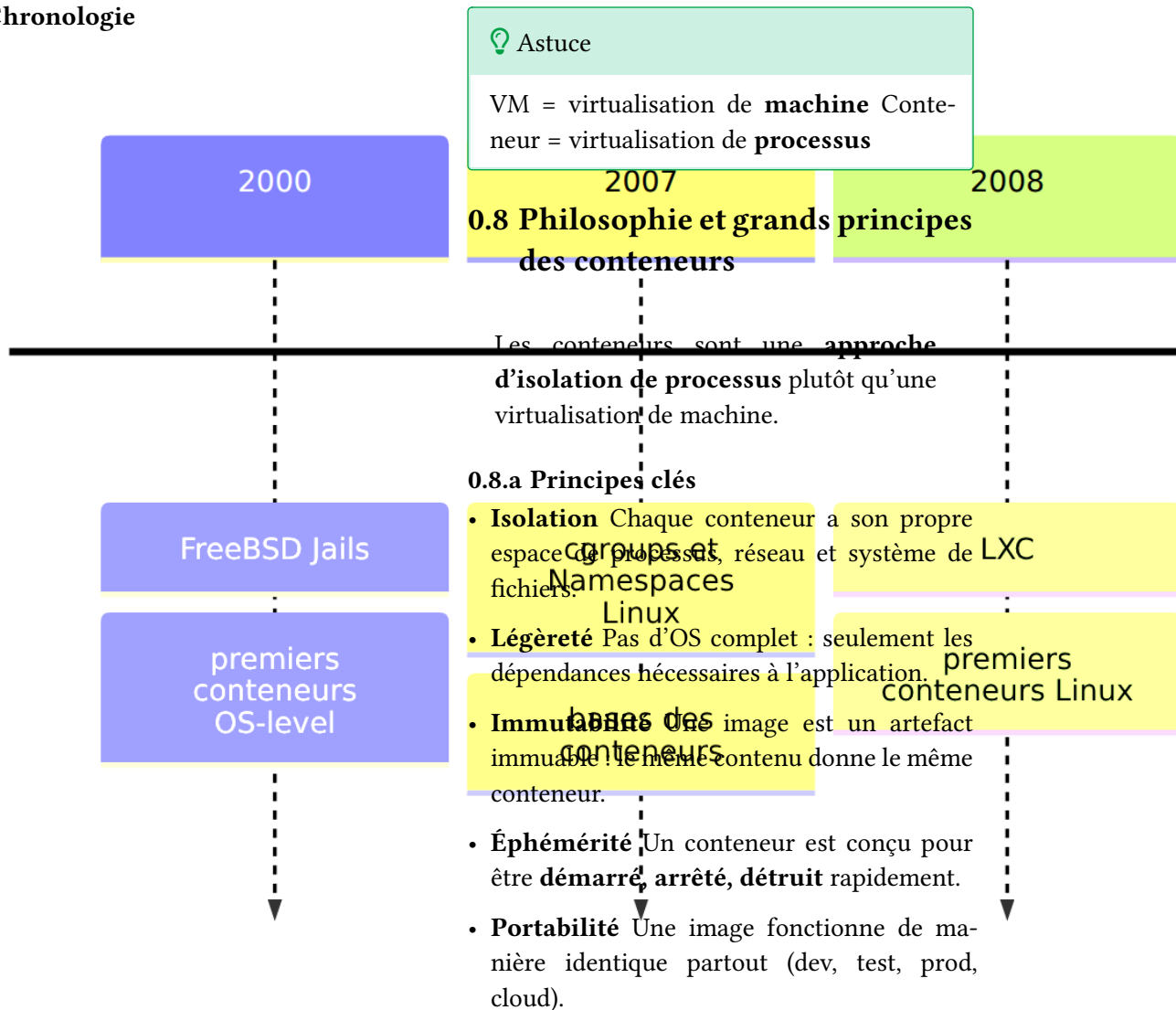
0.6.a Fonctionnalités clés du noyau Linux

1. Namespaces

- PID, Network, Mount, UTS

2. cgroups

0.6.b Chronologie



0.7 Machines Virtuelles vs Conteneurs

Critère	VM	Conteneur
Isolation	« Matérielle »	Noyau
OS	Complet	Partagé
Démarrage	Minutes	Secondes
Taille	Go	Mo
Usage	Legacy	Cloud-native

0.8.b Message central

Un conteneur est une **unité de déploiement** : un processus principal isolé, reproductible, portable et facilement orchestrable.

0.9 Docker et Conteneurs

- Docker Inc. fondé en 2010
- Docker est une plateforme complète qui popularise l'usage des conteneurs
- Il fournit :

- Un format d'image
- Un système de build
- Un registre
- Une CLI simple

⚠ Avertissement

Docker n'est pas le conteneur Docker est un ensemble d'**outils** (maintenant basé sur des standards) .

0.10 Docker et standards OCI

- Standardisation en 2015 (OCI)
 - Image Specification
 - Runtime Specification
 - Distribution Specification
- Alternatives à Docker
 - Bas niveau
 - **runc** : runtime de référence
 - **containerd** : runtime de conteneurs
 - Haut niveau
 - **CRI-O** : runtime pour Kubernetes
 - **Podman** : gestion

0.11 Concepts Fondamentaux

Image

- Template immuable
- Lecture seule
- Distribuible
- Couches de système de fichiers empilées

Conteneur

- Processus isolé
- Ressources limitées
- État éphémère

0.12 Utilisation d'une image

```
docker rm -f my_web_server >/dev/null
2>&1 || true
docker run --label ebpro-render=true
```

```
--detach \
  --name my_web_server \
  --publish :80 \
  --volume /var/www:/usr/share/nginx/
html \
  nginx
```

0.12.a Ce que fait réellement **docker run**

1. Télécharge l'image depuis un registre
2. Crée un système de fichiers isolé
3. Configure réseau et volumes (stockage de données externes)
4. Lance le processus principal

0.13 Création d'une image

- Définie via un fichier **Dockerfile**
- Instructions pour construire l'image
- Exemple simple pour une application Python

```
FROM python:3.9-slim

WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .
CMD ["python", "app.py"]
```

- Construire l'image avec la commande :

```
docker build -t myapp:0.1.0 .
```

💡 Astuce

Une image n'est pas une VM Elle contient uniquement ce qui est nécessaire à l'application.

0.14 Gestion de base

```
docker image pull nginx
docker image ls
```

```
docker container ls -a
docker container rm web
```

0.15 Shell et inspection

```
docker run --label ebpro-render=true
ubuntu </dev/null
docker exec -it web bash < /dev/null
docker logs web
docker stats
```

0.16 Nettoyage

```
docker system prune
docker system prune -a
```

0.17 Composants standards

```
Docker CLI
  ↓
Docker Engine
  ↓
containerd
  ↓
runc
  ↓
Linux Kernel
```

0.18 Orchestration (aperçu)

• Docker Compose

- Développement
- Applications simples

• Kubernetes

- Production
- Scalabilité
- Résilience

Docker Compose ≠ Kubernetes « en petit »

0.19 Conclusion

- Les conteneurs complètent la virtualisation
- Ils apportent rapidité, portabilité et standardisation
- Docker est un point d'entrée, pas une finalité
- Kubernetes s'impose comme standard d'orchestration

💡 Astuce

Un conteneur n'est pas un mini-serveur
C'est un **processus isolé et maîtrisé**

```
#| echo: false
#| output: false

# stop any running containers
docker stop $(docker ps -q) 2>/dev/null || true
docker rm $(docker ps -aq) 2>/dev/null || true

# cleanup volumes/networks (optionnel)
docker volume prune -f 2>/dev/null || true
docker network prune -f 2>/dev/null || true
```