Cloud Computing

Concepts de virtualisation[1]

Danilo Carastan dos Santos

danilo.carastan-dos-santos@univ-grenoble-alpes.fr

2024

[1] Adapté du support développé par Thomas ROPARS et Renaud LACHAIZE

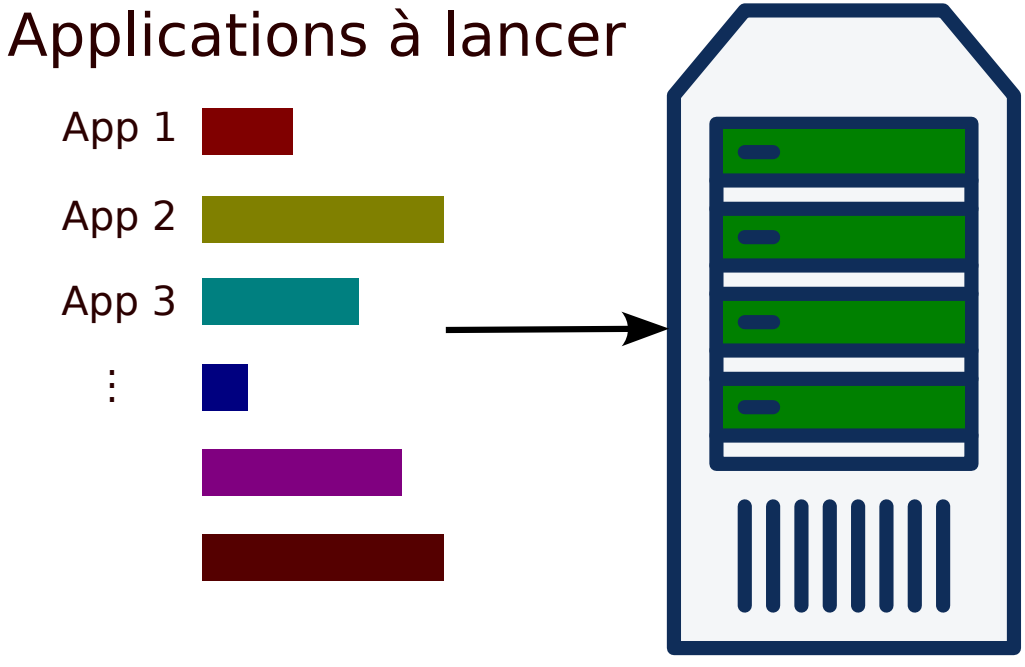
Problematique

Applications :

- Email
- Partage de données

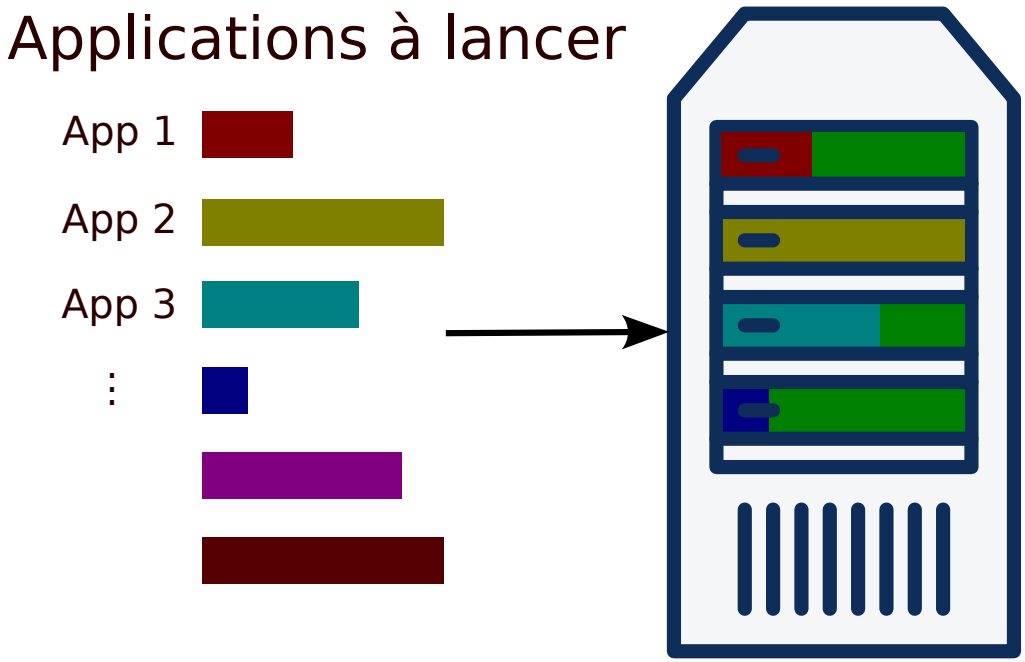
Chaque application a de prérequis spécifiques de ressources physiques/logiciels

- Utilisation de CPU
- Mémoire
- Stockage
- Système d'exploitation
- Librairies/Intergiciels spécifiques



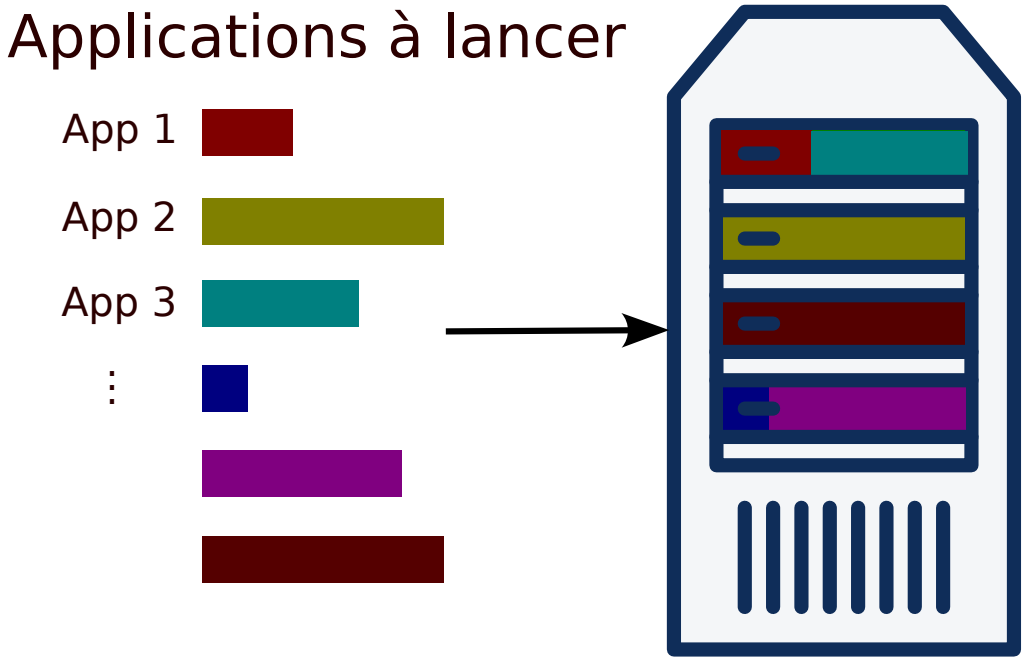
Une première solution naïve

- Une application par serveur
- Inconvénient : sous-utilisation des serveurs



Une deuxième solution

- Plusieurs applications par serveur
- Problèmes :
  - Contrôle de ressources
  - Isolation d'applications



Exemple :

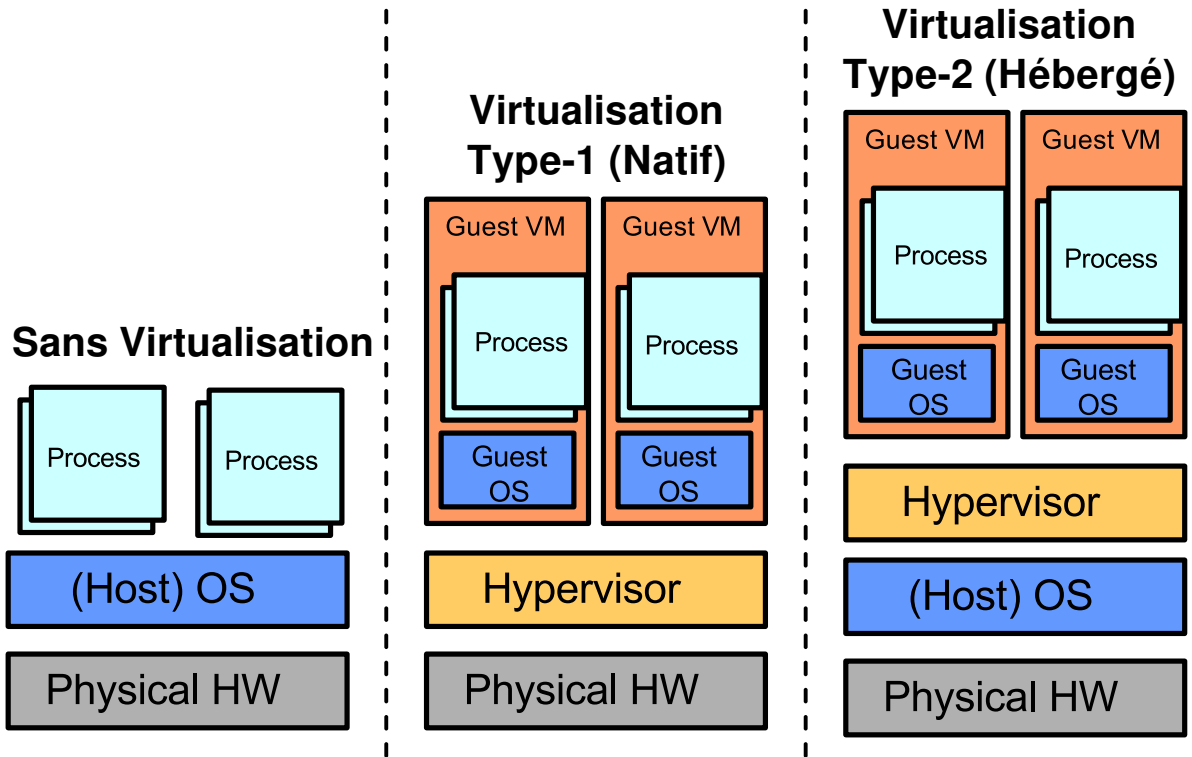
- Apps A et B tournent dans un même serveur. App A a besoin une mise à jour du système d'exploitation. Cette MàJ peut affecter B.

Une meilleure solution : virtualisation

- **Idée :** Créer plusieurs ressources virtuelles (Machine Virtuelle - VM) à partir d'une ou plusieurs ressources physiques
- Dans une VM nous ne pouvons pas distinguer entre ressources physiques ou virtuelles
- **Concept clé :** Hyperviseur
  - Est une couche logicielle
  - Fournit l'abstraction d'une ou plusieurs machines "nues" (processeurs, mémoire et périphériques) au-dessus d'une véritable machine physique
- Intérêt :
  - Faire tourner plusieurs systèmes d'exploitation simultanément sur la même plateforme matérielle
  - Sauvegarder/rembobiner l'état d'un système
  - Sécurité (renforcement de l'isolation de certaines applications)

Virtualisation

- Hyperviseur de type I (natif) : Couche logicielle de plus bas niveau. Système d'exploitation spécialisé. Plus efficace
- Exemples : VMware ESX, Xen
- Hyperviseur de type II (hébergé) : Application intermédiaire (intergiciel) qui s'appuie sur un système hôte sous-jacent
- Moins efficace (plus de couches) mais plus simple à installer/utiliser
- Exemples : VMware workstation/fusion/player, VirtualBox



Scalabilité

Scaling

- Terme générique lié au comportement d'une application si la charge de travail et/ou les ressources changent
- Deux types :
  - **Scalabilité faible (Weak scaling) :** "Puis-je faire plus de travail (traiter plus de requêtes, traiter plus de données) si j'ai plus de ressources ?"
  - **Scalabilité forte (Strong scaling) :** "Puis-je faire la même charge de travail dans moins de temps si j'ai plus de ressources ?"

Elasticité

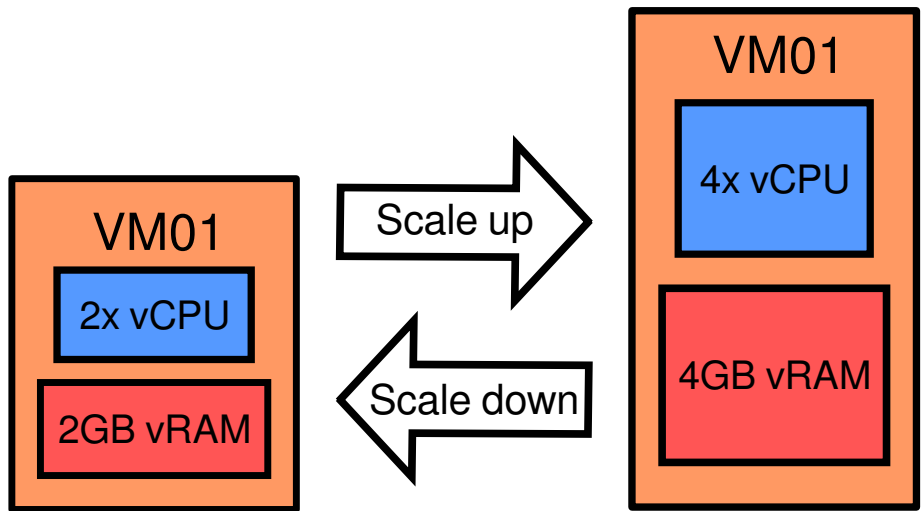
Autoscaling

- **Idée générale :** Adapter les ressources en fonction de la charge de travail courante
- Exemple : “Mon application de commerce électronique basée sur le Web devrait traiter n’importe quelle requête en moins de 500 ms, quel que soit le nombre total de requêtes qu’elle traite actuellement.”
- “Juste ce qu’il faut” :
  - Pas assez de ressources $\rightarrow$ traitement plus long (perte de qualité de service)
  - Trop de ressources $\rightarrow$ ressources non utilisées (et on les paye quand même)

Scalabilité Verticale

Vertical Scaling

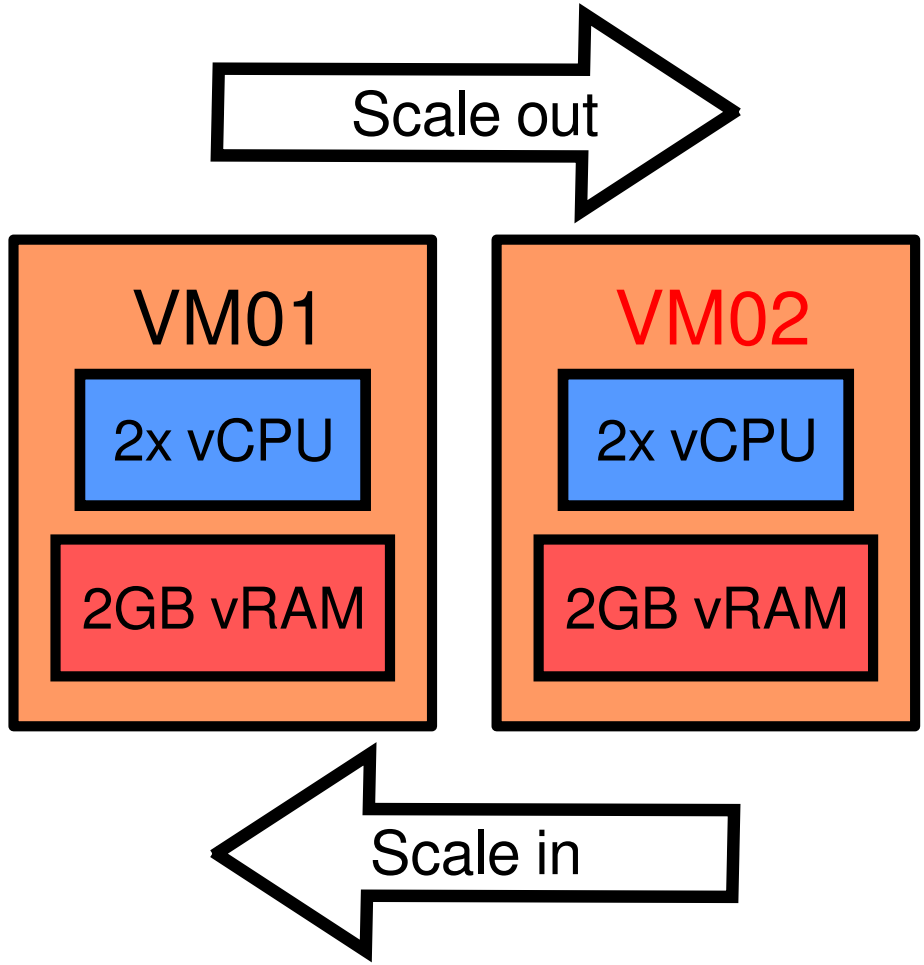
- **Idée :** remplacer les machines (virtuelles) avec des machines (virtuelles) plus puissantes
- **Avantages :**
  - Plus simple
- **Inconvénients :**
  - Scalabilité limitée
  - Coût non linéaire de ressources



Scalabilité Horizontale

Horizontal Scaling

- **Idée :** Ajouter plus d'instances de l'application. Plusieurs machines virtuelles en parallèle
- **Avantages :**
  - Système distribué
- **Inconvénients :**
  - Plus compliqué à développer : Synchronisation, débogage, etc.



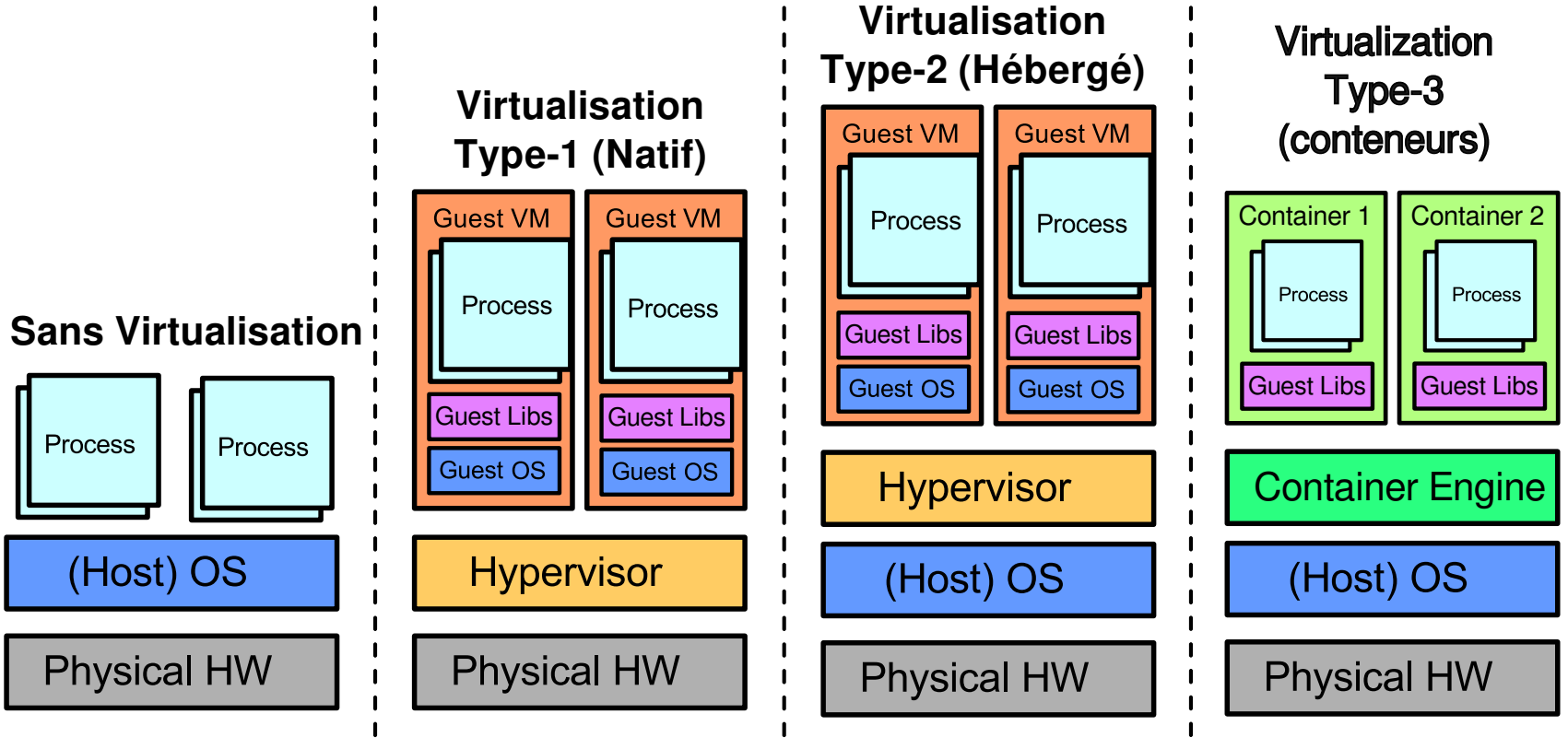
Conteneurs

Containers

- Motivation : Un microservice par VM consomme trop de ressources.
  - Par exemple : chaque VM a un OS invité (*guest OS*), ce qui est trop pour juste un microservice
- Démarrage de VM est trop long (plusieurs minutes)
  - Il faut démarrer un système complet
- Nous souhaitons partager l'OS hôte (*host OS*) entre les applications, tout en étant le plus isolé possible.

Conteneurs

OS-level virtualization : comparaison avec Machines Virtuelles



Conteneurs

Container Engine et *Container Runtime*

- Un environnement d'exécution et un ensemble de services pour manipuler des conteneurs docker sur une machine
- Exemple : Docker engine
- Logiciel client-serveur
  - Le serveur – Un daemon (processus persistant) qui gère les conteneurs sur une machine
  - Le client – Une interface en ligne de commande

Conteneurs

Support du système d'exploitation. Focus sur Linux

Plusieurs fonctionnalités fournies par des sous-systèmes et outils du noyau Linux pour gérer les conteneurs. Par exemple

- Namespaces[1] : pour configurer les ressources système visibles par un processus donné (interfaces/ports réseau, utilisateurs, PID, etc)
- Groupes de contrôle (cgroups[2]) : pour appliquer les limites d'allocation des ressources
- Capacités : pour contrôler les opérations qu'un processus/utilisateur donné est autorisé à effectuer sur différents types de ressources
- Seccomp[3] : pour filtrer les appels système légitimes qu'un processus peut effectuer

[1] `https://www.man7.org/linux/man-pages/man7/namespaces.7.html`
[2] `https://www.man7.org/linux/man-pages/man7/cgroups.7.html`
[3] `https://www.man7.org/linux/man-pages/man2/seccomp.2.html`

Conteneurs

Concept d'Image

Les technologies de conteneurs offrent une solution de *packaging* pour une application et ses dépendances.

Les images de conteneurs : Package de l'application et de ces dépendances

- Peut être exécutée sur différents environnements
- Décrites par un fichier texte (exemple : `Dockerfile`)

Un Conteneur : Une instance d'une image de conteneur

- S'exécute dans un environnement isolé

Analogie POO :

- Une image = une classe
- Un conteneur = une instance

Images de Conteneur vs Images de VM

Les images de VM :

- Sauvegarde de l'état de la VM (Mémoire, disques virtuels, etc) à un moment donné (*Snapshot*).
- La VM redémarre dans l'état qui a été sauvegardé

Les images de Conteneur :

- Une copie d'une partie d'un système de fichier
- Pas de notion d'état

Conteneurs

Applications : Intégration Continue (CI)

- **Portabilité de tests :** Environnement de test créé à partir d'une image Docker
  - Les tests peuvent être exécutés sur n'importe quelle plateforme (plateforme de CI)
- **Isolation de tests :** Un nouveau conteneur créé pour chaque étape de tests
  - Pas de pollution entre les étapes d'un test
  - Pas de pollution entre plusieurs exécutions des tests
- **Démarrage rapide :** Les tests peuvent être exécutés très souvent

Conteneurs

Applications : Déploiement Continu (CD) et *Infrastructure as Code*

- Construire notre application à partir d'images (décrites à partir de `Dockerfiles`)
- Stocker les images dans un registre
  - Stockage pérenne
  - Accessible pour tout le monde
- Exécuter en production
  - Les images contiennent toutes les dépendances pour vos applications
- Installation d'environnement automatisé
  - Fin du "pourtant ça marche sur ma machine"
  - Écrire les instructions d'installation dans un fichier `INSTALL.txt`
  - Créer un script `install.sh` qui fonctionne
  - Le transformer en un `Dockerfile`
  - Créer une image Docker à partir de ce `Dockerfile`

Machines Virtuelles et Conteneurs

Points en commun

- **Faciliter le déploiement :** Encapsuler le code (applications, librairies) et les configurations pour augmenter la portabilité entre plusieurs machines physiques et environnement de déploiement
  - Exemples : CI/CD, *infrastructure as code*
- **Partager des ressources en sécurité :** garantir l'isolation du code invité (*guest code*) entre plusieurs invités. Éviter des interactions indésirables entre :
  - L'hôte et les autres invités (code et données)
  - Les ressources physiques
- **Isoler la performance :** contrôler précisément combien de ressources chaque invité peut utiliser
  - Éviter/Réduire les interférences entre les invités
  - Différencier la qualité de service (*Quality of service, QoS*) entre les invités

Machines Virtuelles et Conteneurs

Différences

- **Empreinte mémoire/stockage :** Conteneurs sont plus légers
- **Temps de démarrage et d'arrêt :** Conteneurs sont plus rapides
- **Sécurité :** Machines virtuelles sont potentiellement plus sécurisées, mais les deux (VMs et conteneurs) ont également une grande surface d'attaque
- **Migration entre machines hôte :** VMs sont plus robustes
- **Support stateful et stateless :** VMs sont plus robustes

Machines Virtuelles et Conteneurs

En pratique

L'usage de ces deux technologies n'est pas forcément antagoniste et mutuellement exclusive.

- Fournisseurs de Cloud public typiquement déploient de Conteneurs dans des machines virtuelles
- Les systèmes d'orchestration de conteneurs peuvent remplacer de conteneurs par des machines virtuelles de façon transparente
  - Par exemple, l'interface CRI (*container runtime interface*) de Kubernetes est compatible avec machines virtuelles

Support supplémentaire

- Cours de Thomas ROPARS et Renaud LACHAIZE
  - `https://tropars.github.io/downloads/lectures/Docker/formation_docker.pdf`
  - `https://roparst.gricad-pages.univ-grenoble-alpes.fr/cloud-tutorials/lectures/Cloud--Building_blocks.pdf`
- *Cloud Fundamentals by IBM* (en Anglais)
  - `https://www.youtube.com/playlist?list=PLOspHqNVtKAC-_ZAGresP-i0okHe5FjcJ`