DOCKER

Introduction • Conteneurisation • VM vs conteneur

Objectif — Comprendre ce qu'est Docker, comment fonctionne un conteneur, le rôle du noyau Linux, les différences avec une machine virtuelle et les enjeux de portabilité, performance et sécurité.

Plan du cours

- Démarrer un conteneur
  - Comment démarrer un conteneur depuis une image Docker ?
  - Commandes utiles à la gestion des conteneurs
- Orchestrer les conteneurs avec Docker Compose
  - But de Docker Compose
  - Installation / disponibilité de Docker Compose
  - Contenu d'un fichier Compose (compose.yaml / docker-compose.yml)
  - Commandes utiles à la gestion des conteneurs
  - Création d'un fichier Compose
- Portainer : interface Web tierce permettant d'administrer Docker

Note actuelle — Docker Compose v2 s'utilise avec la commande « docker compose ». Le nom compose.yaml est recommandé, même si docker-compose.yml reste reconnu.

1. Docker : définition et principe

- Docker permet d'empaqueter une application avec les fichiers et dépendances nécessaires à son exécution dans un conteneur isolé.
- L'objectif est d'obtenir une exécution plus fiable, reproductible et portable entre environnements compatibles.
- Docker relève de la conteneurisation : on parle souvent de « virtualisation légère », mais un conteneur n'est pas une machine virtuelle complète.
- Un conteneur est avant tout un processus isolé avec son propre système de fichiers, son réseau et son arborescence de processus isolée.

À retenir — Plusieurs conteneurs Linux exécutés sur un même hôte partagent le même noyau Linux. Ils n'embarquent pas chacun leur propre noyau comme le ferait une VM.

Flexibilité, portabilité et environnements distribués

- Docker augmente la flexibilité et la portabilité d'une application en standardisant son environnement d'exécution.
- Une même image peut être utilisée sur une grande variété d'hôtes compatibles : machine locale, serveur dédié, cloud privé/public ou machine virtuelle.
- Les conteneurs sont adaptés aux environnements distribués : plusieurs services peuvent fonctionner de manière autonome sur un même hôte ou être répartis sur plusieurs nœuds d'un cluster.

- La conteneurisation simplifie le déploiement d'applications, tâches de fond et autres processus sur un ou plusieurs hôtes.

2. Comment fonctionne Docker ?

- Un conteneur n'embarque pas un système d'exploitation invité complet avec son propre noyau.
- Il contient toutefois un espace utilisateur : fichiers, bibliothèques, binaires et dépendances nécessaires à l'application.
- L'exécution repose sur les fonctionnalités du noyau de l'hôte : isolation, réseau, montages, contrôle des ressources, utilisateurs, etc.
- Docker Engine s'appuie aujourd'hui sur containerd et un runtime conforme OCI tel que runc pour interagir avec les mécanismes du noyau.

Correction historique — Les premières versions de Docker utilisaient LXC. Ce n'est plus l'architecture actuelle de Docker : le runtime moderne repose sur l'écosystème OCI (containerd/runc).

Fonctions du noyau : cgroups et namespaces

- cgroups (control groups) : permettent de mesurer et limiter l'utilisation des ressources d'un processus ou d'un groupe de processus : CPU, mémoire, etc.
- namespaces : fournissent l'isolation en donnant au processus du conteneur sa propre vue de certaines ressources système.
  - Processus / PID et IPC (communication inter-processus)
  - Réseau
  - Points de montage et système de fichiers
  - Utilisateurs et identifiants
  - Autres espaces de noms selon le noyau et la configuration

Important — Docker utilise bien les namespaces ET les cgroups. Les namespaces isolent la vue des ressources ; les cgroups contrôlent et limitent leur consommation.

3. Intérêts de Docker

- Déploiement basé sur une image : on part d'une image contenant l'environnement de l'application, puis on la versionne et on la distribue.
- Partage de l'image et de sa définition : l'application et ses dépendances peuvent être reproduites dans différents environnements compatibles.
- Automatisation du déploiement : création, lancement et remplacement rapides des conteneurs.
- Contrôle des versions des images et intégration aux chaînes CI/CD (par exemple GitLab CI/CD).
- Portabilité accrue par rapport à une installation manuelle directement sur un hôte.

Docker et LXC

LXC / conteneur système	Docker / conteneur applicatif
Approche proche d'un environnement système complet isolé.	Approche centrée sur l'application et ses processus.
Partage le noyau de l'hôte.	Partage également le noyau de l'hôte ; Docker ajoute une chaîne d'outils pour images, distribution, réseau, volumes et cycle de vie.

4. Application installée directement sur l'hôte

- Les applications installées directement sur l'hôte dépendent des bibliothèques, paquets et versions présents sur le système.
- Les conflits de dépendances et de versions peuvent compliquer la maintenance et rendre le déploiement moins reproductible.
- Une application n'est pas automatiquement portable : il faut reproduire son environnement sur la nouvelle machine.
- Un incident ou une mauvaise configuration d'une application peut affecter d'autres services si elles partagent étroitement le même environnement système.
- La mise à l'échelle et la migration restent possibles, mais sont généralement moins standardisées qu'avec des conteneurs ou des VM.

5. Machine virtuelle

- Une machine virtuelle émule ou virtualise tout ou partie du matériel d'un ordinateur grâce à un hyperviseur.
- L'hyperviseur permet de faire fonctionner plusieurs VM sur une machine hôte et d'allouer/partager les ressources entre elles.
- Chaque VM contient son propre système d'exploitation invité complet, avec son propre noyau, ses bibliothèques et ses applications.
- Les images de VM sont généralement volumineuses et leur démarrage est plus lourd que celui d'un conteneur.
- Le coût en ressources est supérieur, mais l'isolation est généralement plus forte car chaque VM possède son propre noyau invité.

6. Conteneur

- Un conteneur permet d'exécuter une application et les bibliothèques requises de manière isolée.
- Il utilise la virtualisation au niveau du système d'exploitation : les mécanismes du noyau isolent les processus au lieu d'émuler une machine complète.
- Il est généralement plus petit, démarre rapidement et consomme moins de ressources qu'une VM équivalente.
- Il est facilement reproductible à partir d'une image, sous réserve de compatibilité de plateforme (OS / architecture / runtime).

Architecture : VM vs conteneur

Machine virtuelle	Conteneur
Application + bibliothèques	Application + bibliothèques
OS invité complet	Pas d'OS invité complet
Hyperviseur	Docker Engine / runtime
Noyau de l'hôte	**Noyau de l'hôte partagé**

7. Comparaison VM vs conteneur : sécurité

Machine virtuelle

- Chaque VM dispose de son propre système d'exploitation et de son propre noyau invité.
- L'isolation est généralement plus forte : l'hyperviseur constitue une frontière supplémentaire entre les VM et l'hôte.

- L'hyperviseur contrôle l'accès aux ressources virtualisées et limite les interactions directes avec l'hôte.

Conteneur / Docker

- Les conteneurs d'un même hôte partagent le noyau : une vulnérabilité du noyau ou une mauvaise configuration peut donc avoir un impact plus large.
- Les risques incluent notamment l'évasion de conteneur (container breakout), l'élévation de privilèges, les capacités Linux excessives ou l'exposition du socket Docker.
- Docker utilise des namespaces, cgroups, capacités Linux et autres mécanismes de sécurité : il est donc faux de dire que Docker « partage les ressources sans namespaces ».
- Le niveau réel de sécurité dépend fortement de la configuration : utilisateur non-root, limitations de capacités, profils seccomp/AppArmor/SELinux, images de confiance, mises à jour, etc.

Synthèse sécurité — VM = séparation plus forte par un noyau invité distinct. Conteneur = isolation efficace mais noyau partagé : il faut durcir la configuration.

8. Comparaison VM vs conteneur : portabilité

Machines virtuelles

- Une VM embarque un système d'exploitation complet : son image est donc plus volumineuse.
- La migration est possible, mais dépend du format de VM, de l'hyperviseur, de la configuration matérielle virtuelle et des outils utilisés.

Conteneurs Docker

- Une image de conteneur n'embarque pas un noyau invité complet : elle est généralement plus légère qu'une image de VM.
- Les images sont faciles à distribuer et à recréer sur un autre hôte compatible.
- La portabilité n'est pas absolue : une image cible une plateforme (par exemple linux/amd64, linux/arm64 ou windows/amd64). Les images multi-plateformes permettent de couvrir plusieurs couples OS/architecture.
- Sur macOS et Windows, Docker Desktop peut faire fonctionner des conteneurs Linux grâce à un environnement Linux virtualisé/WSL selon la plateforme et la configuration.

9. Comparaison VM vs conteneur : performance

- VM et conteneurs répondent à des besoins différents : il n'est pas pertinent de résumer le choix à un simple « plus rapide / plus lent ».
- L'architecture des conteneurs est plus légère car elle évite un noyau invité complet par instance.
- Les conteneurs démarrent généralement très rapidement.
- La consommation de ressources peut être limitée et adaptée à la charge via les mécanismes du noyau et les options Docker.
- La réplication et la mise à l'échelle sont généralement simples : on recrée de nouvelles instances à partir de la même image.

10. Tableau récapitulatif

Critère	Machine virtuelle	Conteneur
Isolation	Hyperviseur + OS invité + noyau invité distinct	Isolation au niveau OS ; noyau de l'hôte partagé

Système d'exploitation	OS invité complet	Pas d'OS invité complet ; espace utilisateur + dépendances
Démarrage	Secondes à minutes selon la VM	Souvent très rapide (secondes ou moins)
Taille	Souvent plusieurs Go	Souvent de quelques dizaines à quelques centaines de Mo ; peut varier fortement
Distribution	Images de VM, formats dépendant des outils	Images de conteneur et registres ; distribution standardisée
Migration / remplacement	Migration de la VM ou restauration ailleurs	Souvent détruit puis recréé depuis l'image ; données persistantes séparées
Création	Installation/configuration d'un OS complet	Création rapide depuis une image
Ressources	Surcoût d'un OS invité par VM	Surcoût plus faible ; ressources contrôlables par cgroups

11. Notes de cours corrigées et complétées

Linux, Windows et noyau partagé

- Dans le contexte de votre TP sur Debian, Docker exécute des conteneurs Linux qui utilisent le noyau Linux de l'environnement Docker.
- Il ne faut toutefois pas retenir « Docker = uniquement Linux » : Docker prend aussi en charge les conteneurs Windows sur des hôtes Windows compatibles.
- Une VM peut, elle, exécuter un système invité différent de celui de l'hôte si l'hyperviseur et l'architecture le permettent, car elle dispose de son propre noyau invité.

Pourquoi dit-on qu'un conteneur est « sans OS » ?

- La formulation correcte est : le conteneur n'embarque pas un OS invité complet avec son propre noyau.
- Il embarque tout de même les fichiers et bibliothèques nécessaires à l'application : on parle d'espace utilisateur (user space).
- Il n'y a donc pas d'émulation d'une machine complète : les processus s'appuient directement sur les fonctions du noyau fourni par l'environnement Docker.

Docker dans Docker (DinD)

- Oui, on peut lancer un démon Docker à l'intérieur d'un conteneur : c'est le principe de Docker-in-Docker (DinD).
- L'image officielle docker:<version>-dind est conçue pour ce cas, souvent utilisé en CI/CD ou en environnement de test.
- Ce n'est pas un « nouveau noyau Docker dans Docker » : le démon interne finit toujours par utiliser les capacités du noyau de l'environnement hôte.
- DinD nécessite généralement des privilèges élevés (par exemple --privileged), ce qui augmente les risques de sécurité. Il faut donc l'utiliser seulement lorsque le besoin le justifie.

À distinguer — Monter /var/run/docker.sock dans un conteneur permet à ce conteneur de piloter le démon Docker de l'hôte. Ce n'est pas du vrai DinD et cela donne un accès très puissant au Docker de l'hôte.

12. Mémo express

Conteneur — Processus isolé + fichiers nécessaires + noyau partagé.

Machine virtuelle — Machine virtualisée + OS invité complet + noyau invité distinct.

cgroups — Mesurent et limitent les ressources.

namespaces — Isolent la vue des processus et des ressources.

Image Docker — Modèle immuable servant à créer des conteneurs reproductibles.

Docker Engine — Moteur qui gère images, conteneurs, réseaux et volumes.

Sources de vérification

Références officielles Docker utilisées pour corriger les formulations ambiguës ou devenues historiques :

- **Docker Docs — What is a container? :** https://docs.docker.com/get-started/docker-concepts/the-basics/what-is-a-container/
- **Docker Docs — Running containers :** https://docs.docker.com/engine/containers/run/
- **Docker Docs — Docker Engine security :** https://docs.docker.com/engine/security/
- **Docker Docs — dockerd / runtime OCI :** https://docs.docker.com/reference/cli/dockerd/
- **Docker Docs — Resource constraints :** https://docs.docker.com/engine/containers/resource_constraints/
- **Docker Docs — Rootless Docker-in-Docker :** https://docs.docker.com/engine/security/rootless/tips/
- **Docker Docs — Multi-platform builds :** https://docs.docker.com/build/building/multi-platform/
- **Docker Docs — Docker Desktop / Windows containers :** https://docs.docker.com/desktop/setup/install/windows-install/
- **Docker Docs — Docker Desktop on Linux uses a VM :** https://docs.docker.com/desktop/setup/install/linux/