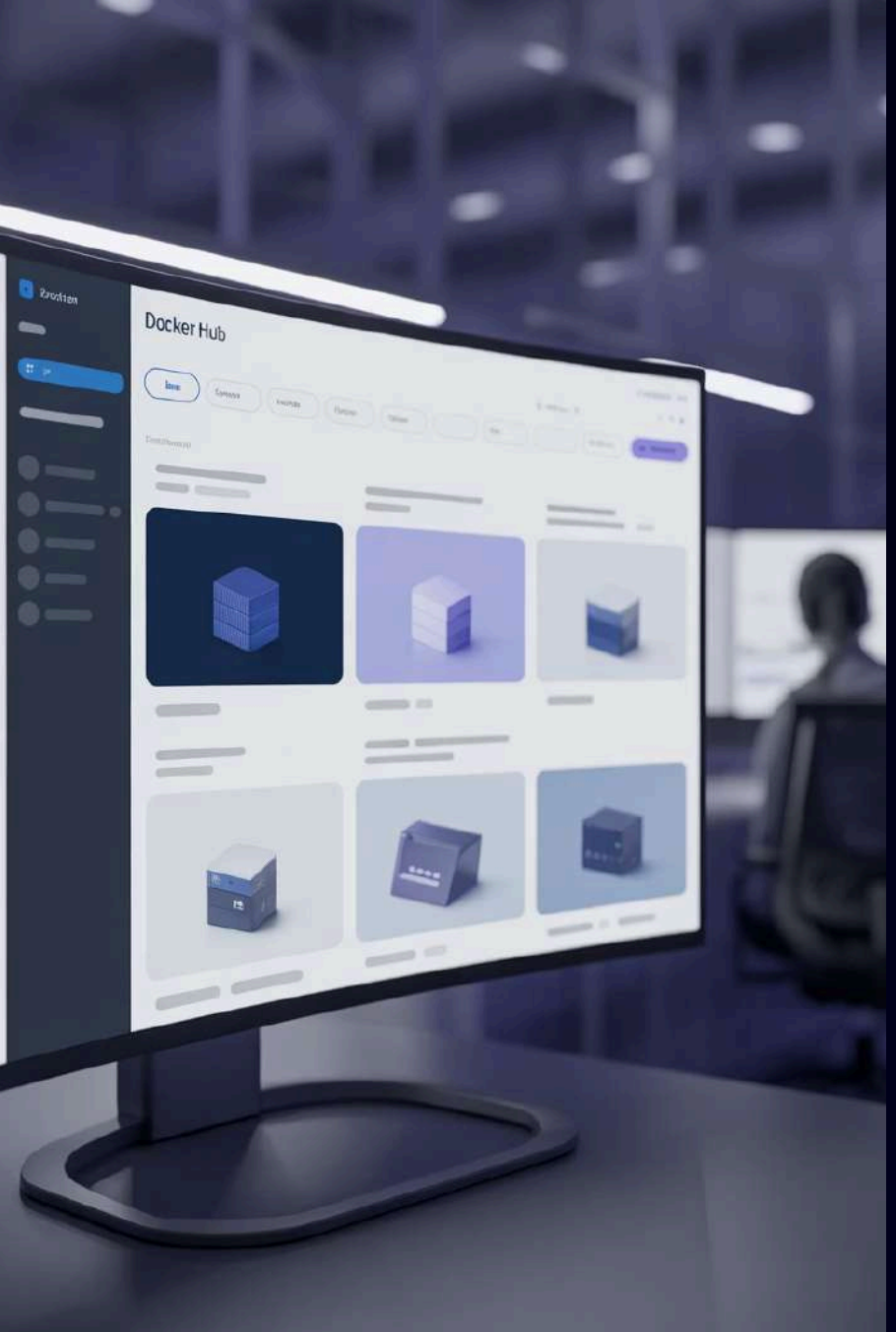


Jour 1 : Introduction à Docker et aux Conteneurs

Bienvenue dans votre formation Docker ! Aujourd'hui, nous allons découvrir l'univers fascinant de la conteneurisation, une technologie révolutionnaire qui transforme la manière dont nous développons, déployons et gérons nos applications.

Cette journée d'introduction vous donnera les fondements essentiels pour comprendre Docker et maîtriser ses concepts clés.

Conteneurs vs Machines Virtuelles

Conteneur Docker

Un conteneur est un **processus isolé** qui s'exécute sur l'hôte en partageant le noyau de l'OS. Il embarque uniquement l'application et ses dépendances, sans système d'exploitation complet.

- Partage le kernel de l'hôte
- Contient seulement l'app et ses dépendances
- Démarrage en quelques secondes

Machine Virtuelle (VM)

Une VM émule un **matériel complet** et inclut son propre système d'exploitation complet. Cela la rend beaucoup plus lourde en ressources que les conteneurs.

- OS complet avec son propre kernel
- Émulation matérielle complète
- Démarrage en plusieurs minutes

L'isolation diffère également : une VM offre une isolation forte (totalement séparée de l'hôte), tandis qu'un conteneur isole les processus de manière plus légère grâce aux espaces de noms et cgroups.

Pourquoi Docker ? Les Avantages Révolutionnaires

Portabilité des Applications

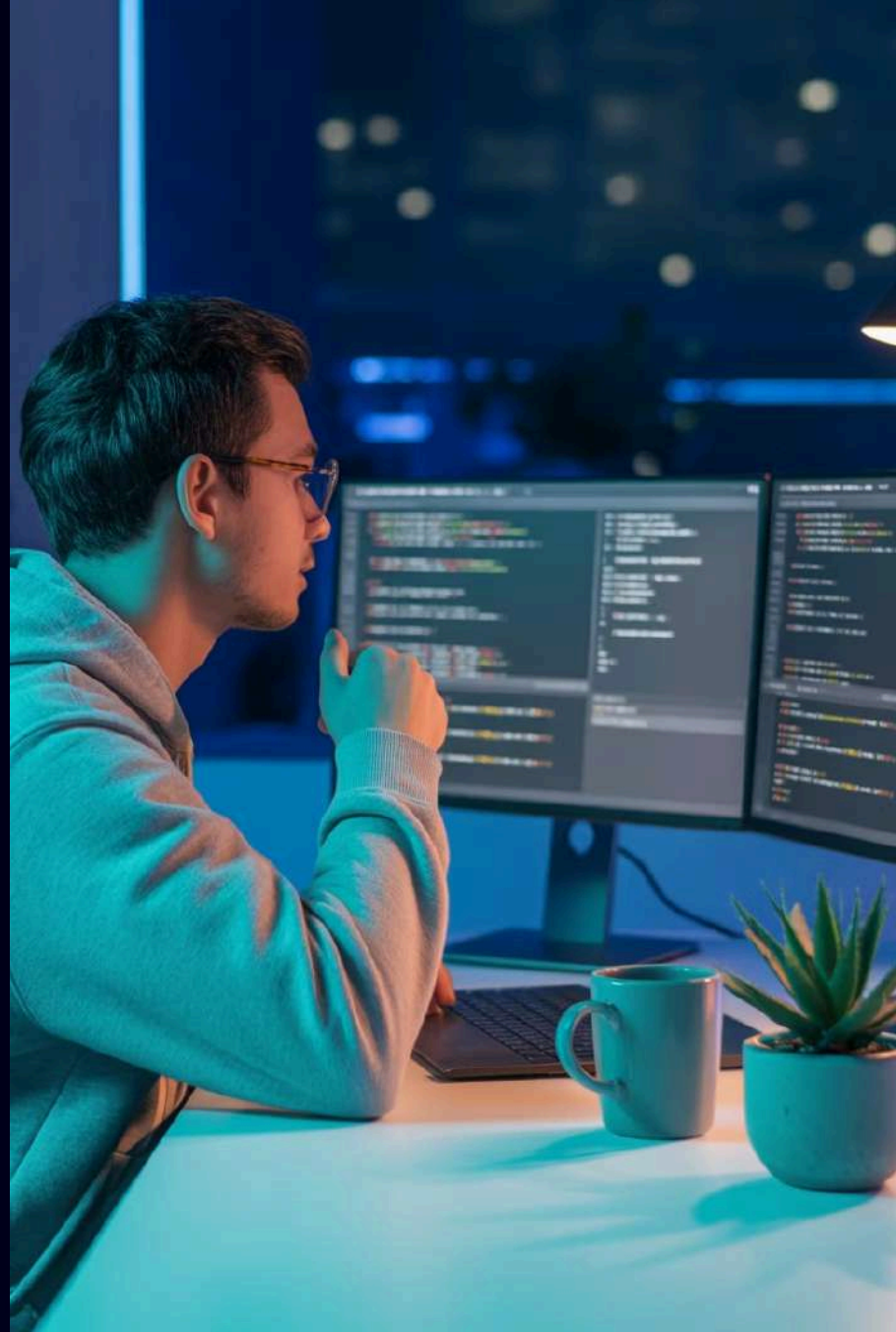
Un conteneur embarque l'application avec **toutes ses dépendances**. Cette approche garantit que l'application fonctionnera identiquement sur la machine du développeur, en staging ou en production.

Fini le fameux *"ça marche sur ma machine"* ! La cohérence est assurée partout.

Cohérence des Environnements

Grâce aux conteneurs, nous évitons les dérives de configuration. L'environnement d'exécution est isolé et identique partout.

Cette uniformité réduit drastiquement les bugs liés aux différences de configuration entre serveurs.



Plus d'Avantages Docker

🚀 Déploiements Rapides

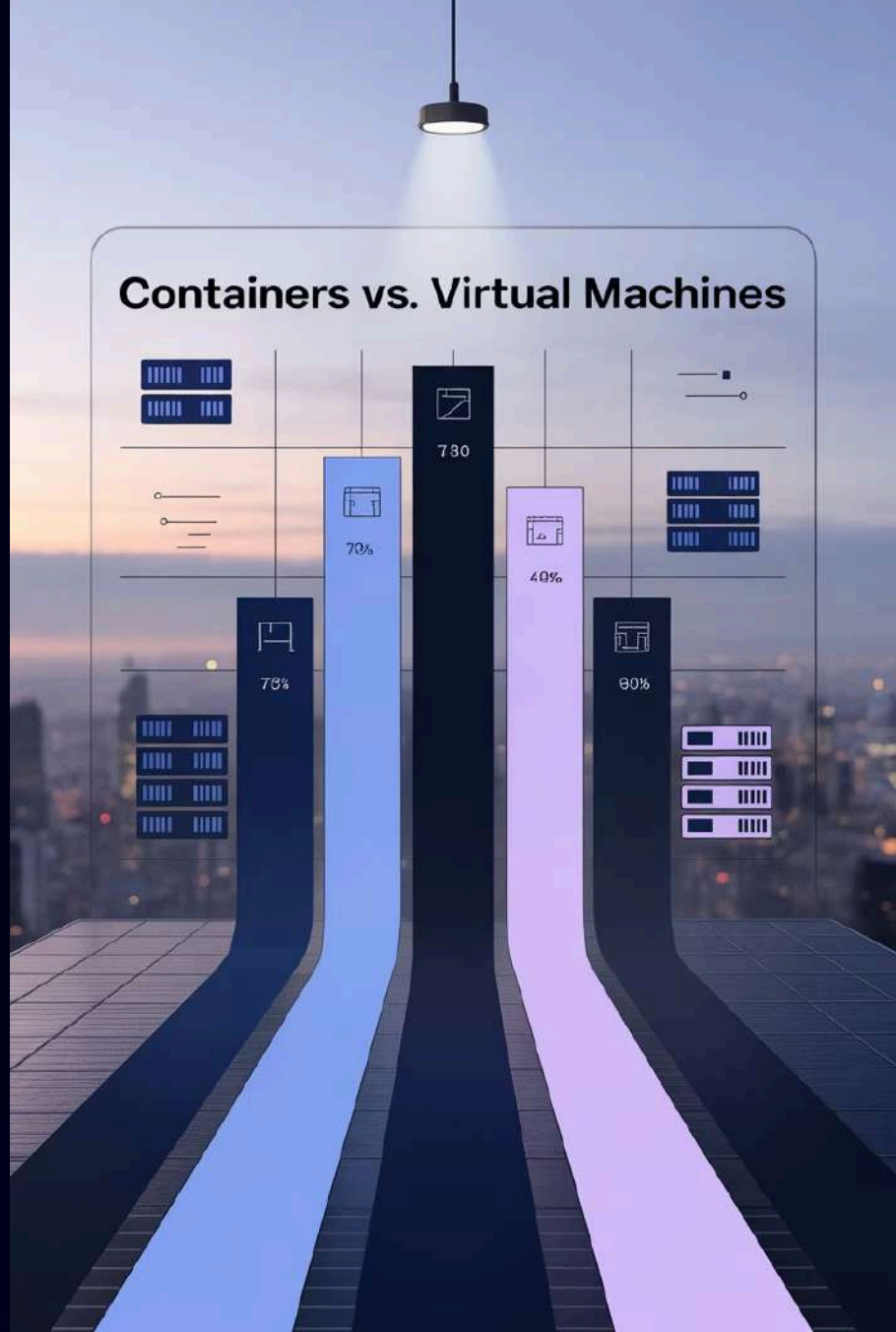
Lancer ou redémarrer un conteneur prend très peu de temps (quelques secondes) comparé à déployer une application sur une VM ou un serveur classique.

Cette rapidité permet de scaler ou mettre à jour les applications quasi-instantanément.

⚡ Optimisation des Ressources

Sur une même machine hôte, vous pouvez faire tourner de nombreux conteneurs Linux, là où vous ne pourriez faire tourner que quelques VM.

Les conteneurs **mutualisent le kernel**, réduisant considérablement l'empreinte mémoire et CPU par application.



Architecture Docker : Moteur et Client

1

Docker Engine (Daemon)

C'est le **service de fond** qui s'exécute sur la machine hôte. Il gère toutes les actions Docker :

- Création de conteneurs
- Gestion des images
- Administration des réseaux et volumes

Il s'exécute avec les droits administrateur sur l'hôte.

2

Client Docker (CLI)

L'outil en ligne de commande `docker` est le client qui communique avec le daemon via une API REST.

Chaque commande tapée (ex: `docker run`) envoie une instruction au Docker daemon qui la réalise.

Le client et le daemon peuvent être sur la même machine (Docker Desktop inclut un daemon local) ou sur des machines différentes pour contrôler un daemon distant.

Images Docker vs Conteneurs

01

Image Docker

Une image est un **gabarit en lecture-seule** qui contient tout le nécessaire pour faire tourner une application : code, dépendances, OS minimal.

Pensez à une image comme une "photo" figée d'un système de fichiers à un instant T.

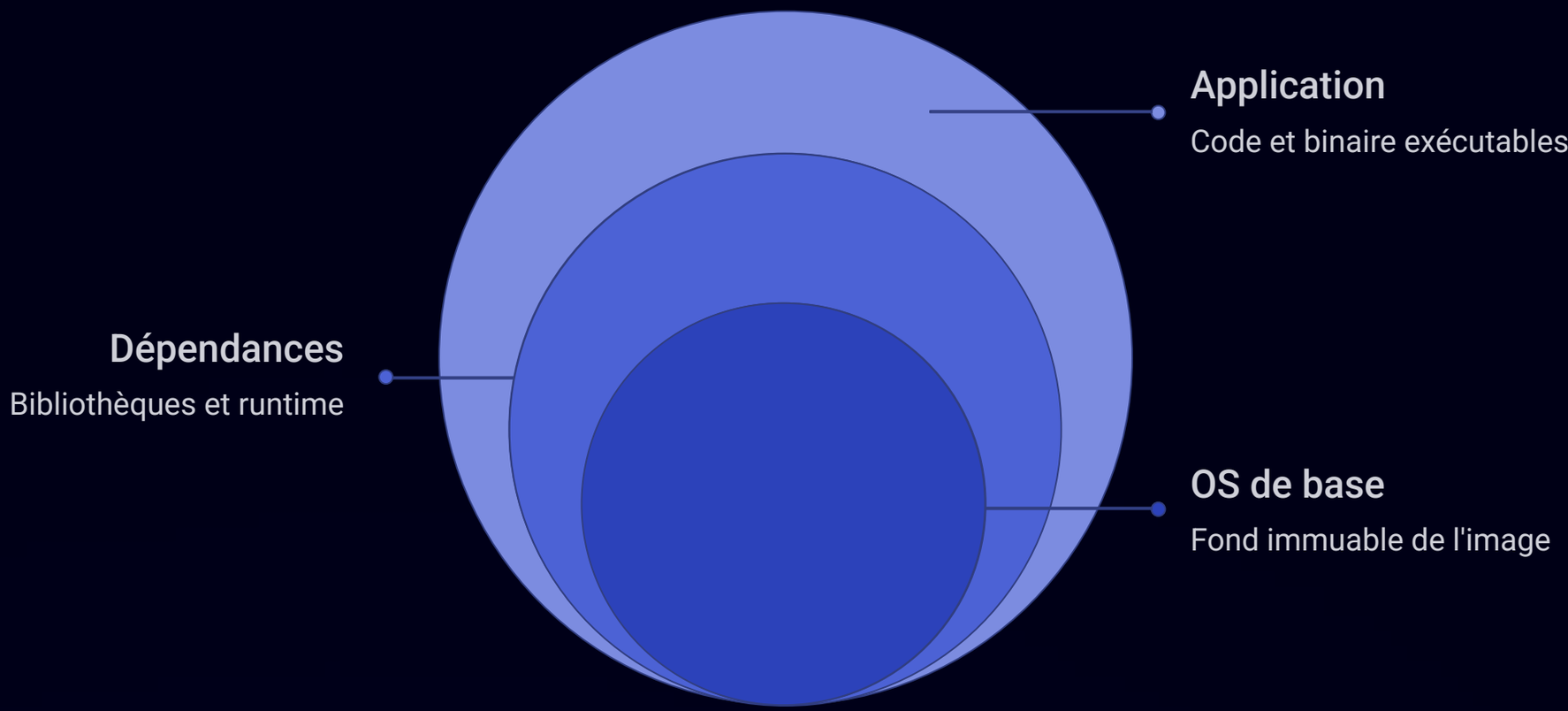
02

Conteneur

Un conteneur est une instance en cours d'exécution d'une image.

Docker ajoute une couche en lecture/écriture au-dessus de l'image - les changements durant l'exécution s'enregistrent dans cette couche temporaire.

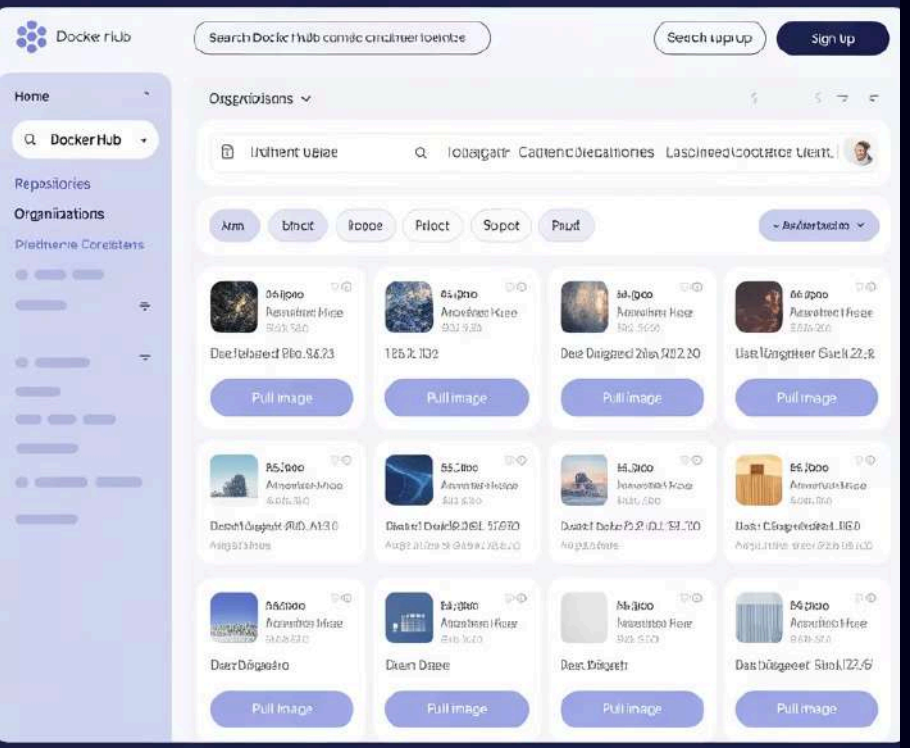
Système de Couches et Copy-on-Write



Les images sont composées de **plusieurs couches empilées** : une couche pour l'OS de base, une couche pour l'application, etc. Docker utilise un mécanisme de copy-on-write.

Principe clé : Les couches d'image ne bougent jamais. Toute modification dans un conteneur crée une nouvelle version du fichier modifié dans la couche écriture, sans altérer l'image de base.

Si plusieurs conteneurs utilisent la même image, ils partagent les couches en lecture-seule, économisant espace disque et mémoire. Chaque conteneur a sa couche propre pour ses modifications.



Registre d'Images Docker (Docker Hub)

Distribution des Images

Les images Docker sont stockées dans des **registres** accessibles en ligne. Le principal registre public est Docker Hub (hub.docker.com).

On y trouve la plupart des images officielles et communautaires.

Notation d'une Image

Format : `nom_organisation/nom_image:tag`

Exemple : `library/ubuntu:22.04` ou simplement `ubuntu:22.04`

Le **tag** identifie la version (par défaut :latest).

Docker Hub contient des images *officielles* (maintenues par Docker ou les éditeurs : nginx, node, mysql...) et des images publiées par la communauté.

Registres Privés et Alternatives

Registres d'Entreprise

Les entreprises peuvent héberger leurs propres registres pour stocker des images internes :

- JFrog Artifactory
- GitHub Container Registry
- AWS ECR
- Azure Container Registry

Solutions Open Source

Alternatives pour déployer son registre :

- Docker Registry (auto-hébergé)
- Harbor (interface web avancée)
- Portus (gestion RBAC)

Commandes Docker Essentielles (1/2)

1

docker pull <image>

Télécharger une image depuis un registre vers le stockage local.

Exemple : `docker pull ubuntu:latest`

Note : Cette commande est souvent implicite car docker run la fait automatiquement si l'image n'est pas locale.

2

docker run <image>

Créer et lancer un conteneur à partir d'une image.

Si l'image n'existe pas localement, Docker la télécharge d'abord.

Options configurables : ports, nom, mode interactif...

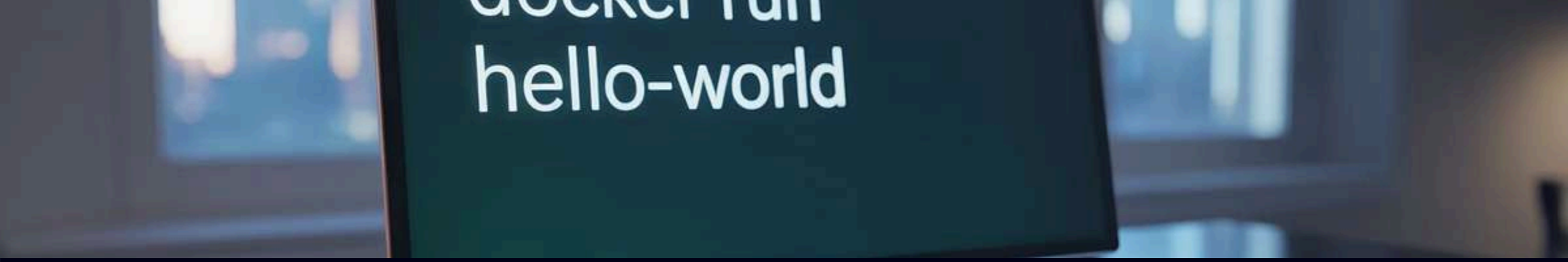
3

docker ps

Lister les conteneurs en cours d'exécution.

Option `-a` : voir tous les conteneurs (actifs + arrêtés)

Utile pour retrouver un conteneur stoppé ou terminé.



Commandes Docker Essentielles (2/2)

docker stop <id|nom>

Arrêter un conteneur de manière propre. Docker envoie un signal SIGTERM au processus principal pour lui laisser le temps de se terminer.

Après 10 secondes (délai par défaut), Docker envoie un SIGKILL pour forcer l'arrêt.

docker kill <id|nom>

Arrêt **brutal** du conteneur via SIGKILL immédiat.

À utiliser prudemment, uniquement si stop normal échoue ou en urgence.

Gestion et Nettoyage des Conteneurs

🗑️ docker rm <id|nom>

Supprimer un conteneur (nécessite qu'il soit arrêté au préalable).

Cette commande efface toutes les ressources du conteneur : système de fichiers isolé, métadonnées.

Attention : Cela n'efface pas l'image d'origine !

📋 docker images

Lister les images Docker présentes localement.

Affiche : nom, tags, identifiants, taille.

Une image reste locale tant qu'on ne la supprime pas avec `docker rmi`.

Éphéméralité des Conteneurs

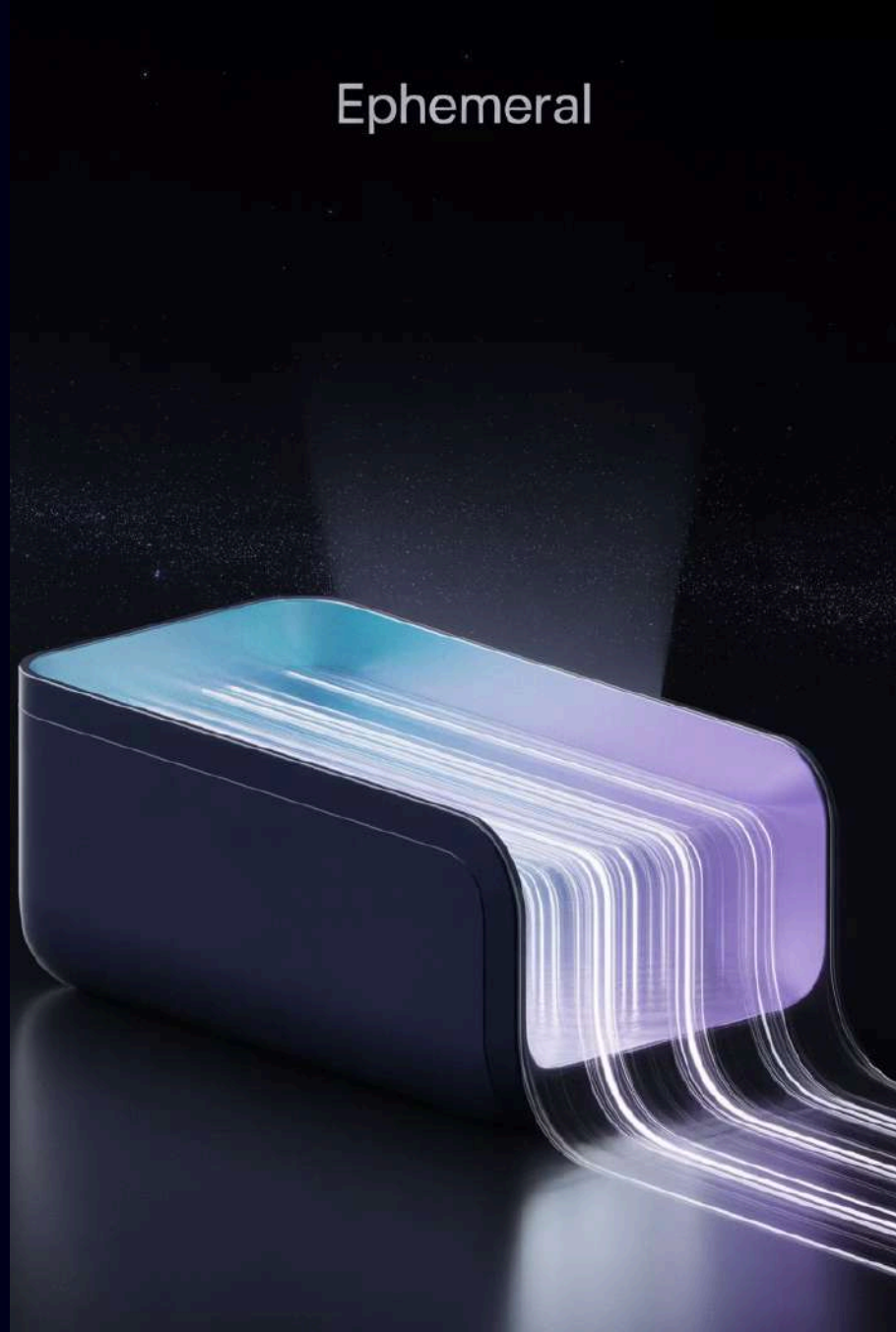
Conteneur Éphémère

Par défaut, un conteneur Docker est **éphémère**. Si vous supprimez un conteneur, tout ce qui s'est passé à l'intérieur (fichiers créés, modifications) est définitivement perdu.

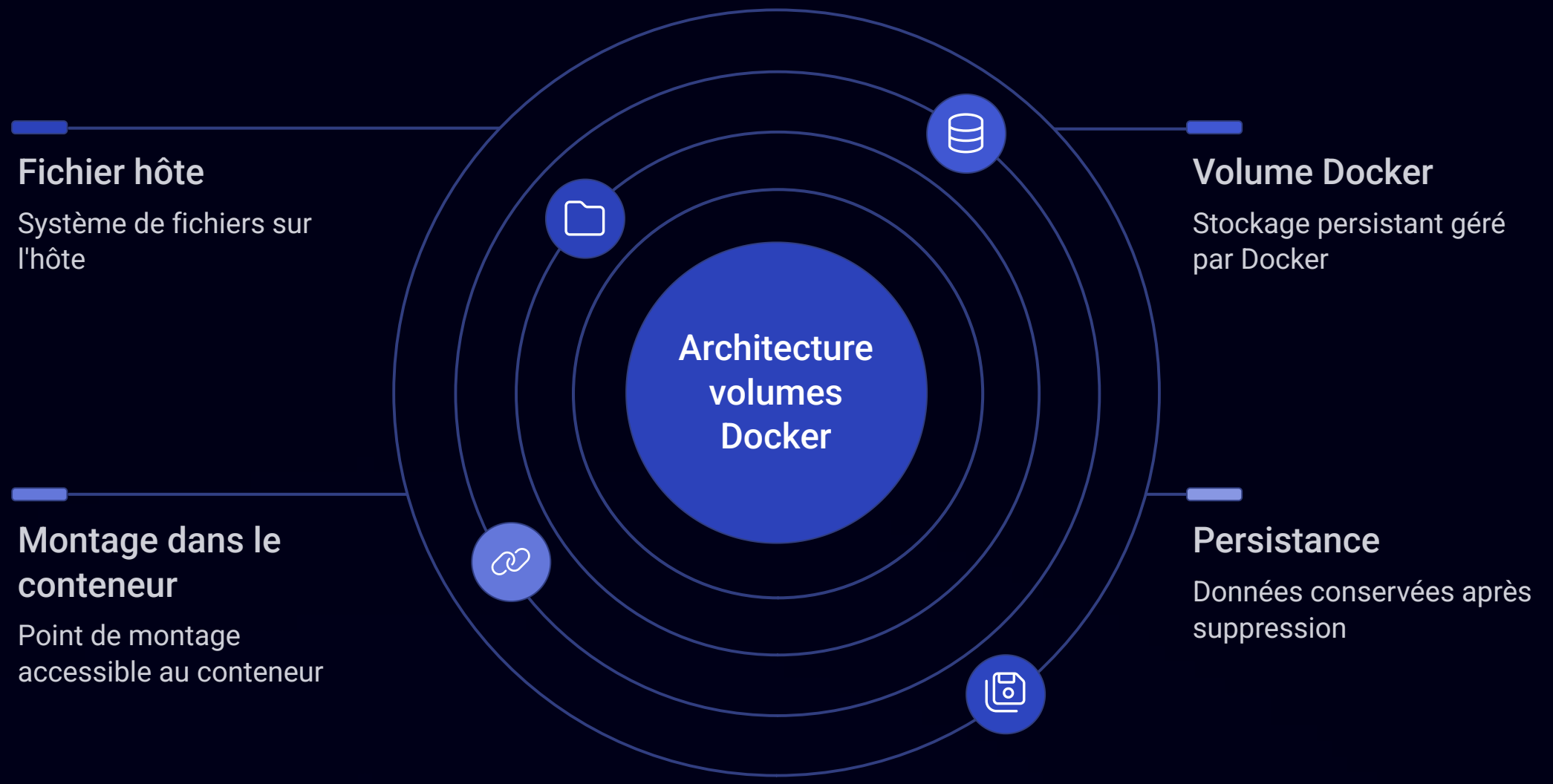
Conséquences Pratiques

Le système de fichiers interne disparaît avec le conteneur. Un nouveau conteneur depuis la même image repart toujours de l'état initial de l'image.

> **Exemple concret :** Si un conteneur exécute une base de données et que les données sont stockées à l'intérieur, supprimer ce conteneur signifie perdre toutes ces données !



Volumes Docker : La Solution de Persistance



Pour conserver des données au-delà du cycle de vie d'un conteneur, Docker propose les volumes.

Qu'est-ce qu'un Volume ?

Un volume est un **stockage persistant** (sur l'hôte ou distant) que l'on peut monter dans un conteneur.

Les données écrites sur un volume survivent à la suppression du conteneur.

Pourquoi c'est Crucial ?

Les volumes permettent de rendre un conteneur *stateful/persistant.*

Essentiel pour les bases de données, fichiers de configuration, logs...

⚠ Les volumes seront détaillés dans les prochains modules, mais il est fondamental de comprendre leur nécessité dès maintenant.

Prise en Main : Hello World Docker

01

Vérification Installation

Assurez-vous que Docker est bien installé et que le daemon fonctionne.

Commande : `docker version` pour vérifier la communication client-serveur.

02

Hello World

Exécutez : `docker run hello-world`

Docker télécharge l'image, crée un conteneur qui affiche un message de bienvenue.

03

Observer le Résultat

Message "Hello from Docker!" suivi d'une explication des étapes effectuées en interne.

04

Comprendre le Processus

Le message décrit le workflow : client → daemon → téléchargement → création conteneur → exécution → retour terminal.

Exploration d'un Conteneur Linux (Ubuntu)

Lancement Shell Interactif

Commande : `docker run -it ubuntu:latest bash`

Options : `-it` attache un terminal interactif

Dans le Conteneur

Vous êtes **root dans un système Ubuntu minimal**

Invite différente : `root@<hash>:/#`

Commandes de Vérification

`ls /` - lister la racine

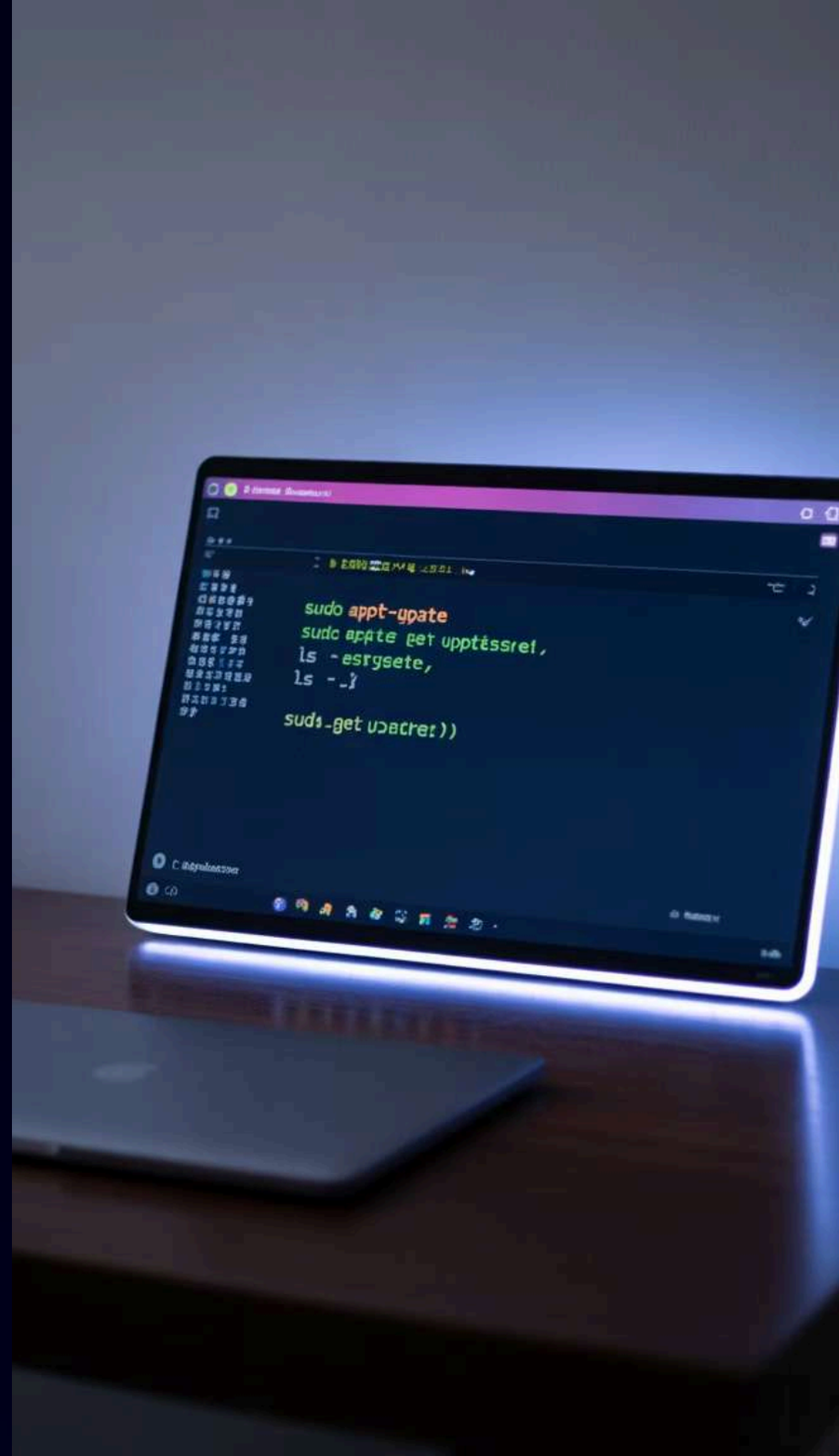
`cat /etc/os-release` - version Ubuntu

`echo "hello" > /tmp/fichier.txt` - créer fichier test

Sortie du Conteneur

`exit` ou Ctrl+D pour quitter

Le conteneur s'arrête automatiquement



Ubuntu : Isolation et Non-Persistance

1. **Relancer Ubuntu**

   Commande : `docker run -it ubuntu:latest bash`

   Nouveau conteneur créé (pas redémarrage du précédent)

2. **Constater l'Isolation**

   Chercher le fichier créé précédemment : `cat /tmp/fichier.txt`

   Surprise : Le fichier n'existe plus !

3. **Explication**

   Conteneur différent = système de fichiers différent

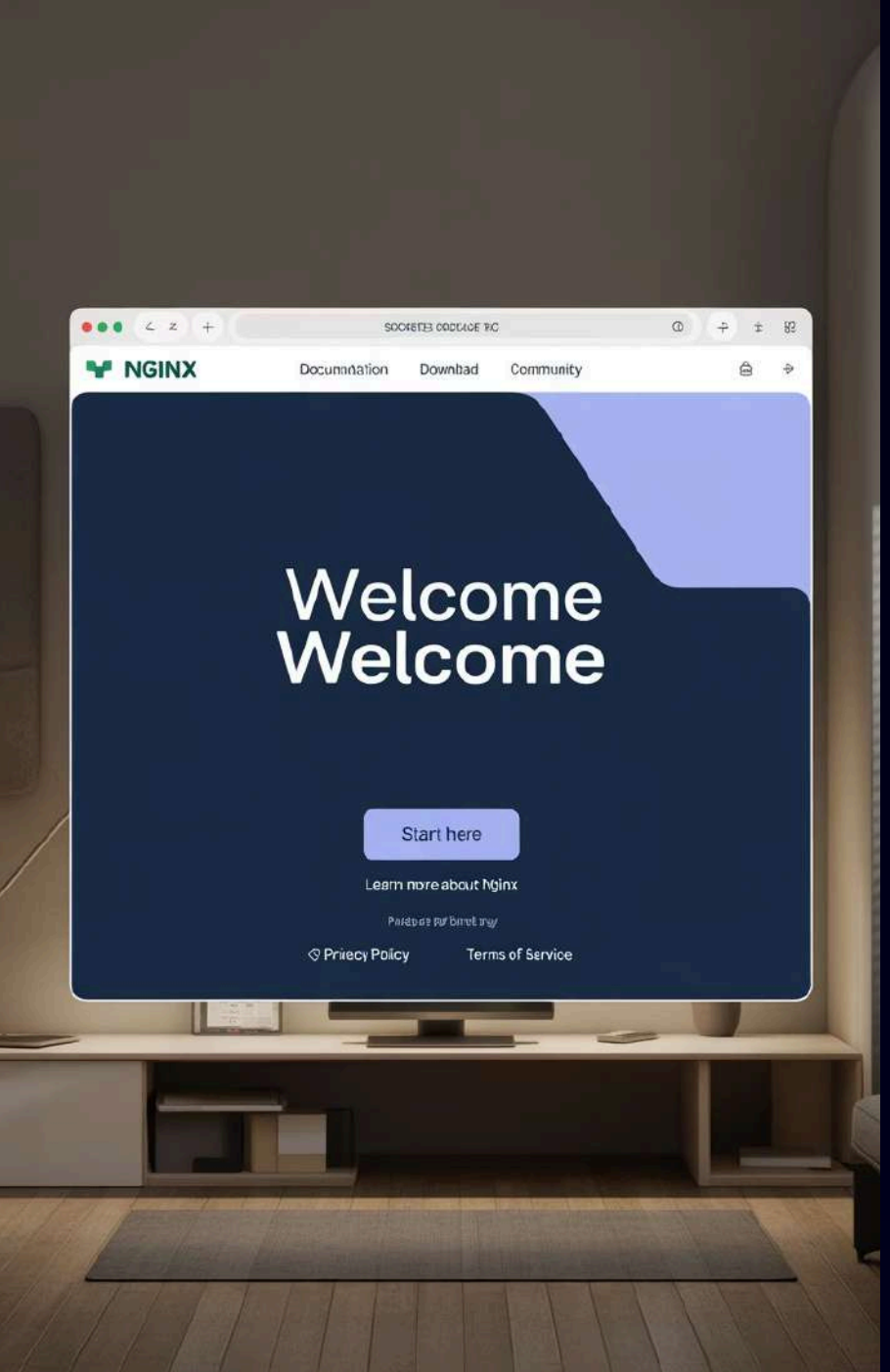
   Le fichier était dans l'ancien conteneur (maintenant supprimé)

4. **Conclusion**

   Chaque conteneur a son système de fichiers isolé

   Modifications internes = éphémères

> Cette manipulation illustre parfaitement l'isolation et l'éphéméralité : rien de ce qui se passe dans un conteneur n'affecte l'hôte ou un autre conteneur.



Service Web avec Nginx

1 Lancer Nginx

Commande : `docker run -d -p 8080:80 nginx`

- `-d` : mode détaché (arrière-plan)
- `-p 8080:80` : port 80 conteneur → port 8080 hôte

2 Vérification

`docker ps` pour voir le conteneur actif

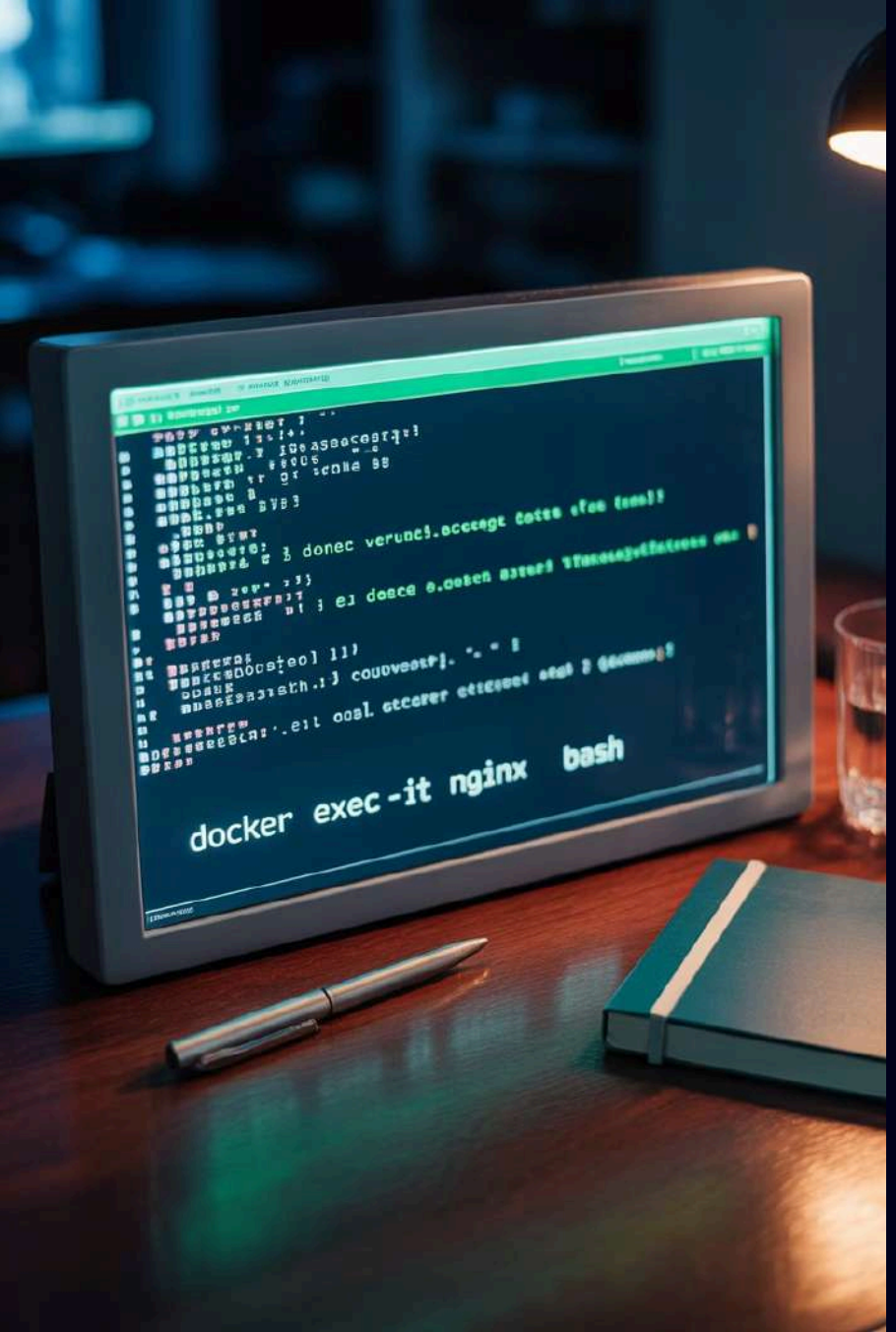
Mapping affiché : `0.0.0.0:8080->80/tcp`

3 Test du Serveur

Navigateur : `http://localhost:8080`

Page "Welcome to nginx!" s'affiche

Résultat : Déploiement d'un serveur web fonctionnel en une seule commande, sans rien installer sur l'hôte ! Le port 8080 de votre ordinateur redirige vers le port 80 du conteneur.



Gestion des Conteneurs : Logs et Exec

📋 Consultation des Logs

Commande : `docker logs <id|nom>`

Exemple : `docker logs nginx`

Affiche les logs de démarrage de Nginx et éventuellement les logs d'accès suite à votre visite de la page.

🔧 Exécuter des Commandes

Commande : `docker exec -it <id|nom> sh`

Lance un shell *dans le conteneur actif*

Note : On utilise sh car Nginx utilise souvent Alpine Linux

Inspection du Conteneur Nginx

Explorer le Système de Fichiers

Une fois dans le conteneur avec `docker exec`, parcourez l'environnement isolé.

Commande : `cat /usr/share/nginx/html/index.html`

Affiche le contenu HTML de la page "Welcome to nginx".

Lister les Processus

Commande : `ps aux`

Visualisez Nginx tournant comme processus principal dans le conteneur.

Observez l'isolation : seuls les processus du conteneur sont visibles.

Sortie Sans Arrêt

Tapez `exit` pour quitter le shell.

Important : Le conteneur Nginx continue de tourner ! Vous n'avez fait qu'entrer/sortir, pas l'arrêter.

Arrêt et Suppression d'un Conteneur

01

Arrêter le Conteneur

Commande : `docker stop <id|nom>`

Docker envoie SIGTERM puis SIGKILL après timeout.

Nginx s'arrête quasi-immédiatement. Vérifiez avec `docker ps`.

02

Conteneur Arrêté mais Existant

Le conteneur existe toujours (statut "Exited").

`docker ps -a` affiche les conteneurs inactifs.

03

Suppression Définitive

Commande : `docker rm <id|nom>`

Libère le nom et l'espace du conteneur.

`docker ps -a` : le conteneur a disparu.

04

Image Conservée

`docker images` : l'image nginx reste en cache local.

Prochain `docker run nginx` : pas de re-téléchargement !

Bilan de la Journée

Concepts Clés Maîtrisés

- **Conteneur vs VM** - différences et avantages
- **Architecture Docker** - Engine et client
- **Images et conteneurs** - isolation, couches
- **Éphéméralité** - nature temporaire des données

Pratique Réalisée

- Commandes essentielles (pull, run, ps, stop, rm)
- Exploration conteneur Linux Ubuntu
- Déploiement service Nginx
- Gestion logs et exécution de commandes

Vous avez découvert la rapidité et l'isolation qu'offre Docker, bases essentielles pour devenir expert en conteneurisation.

Perspectives et Ouverture

1

Questions & Réponses

Discussion ouverte pour clarifier les points encore flous.

"Que se passe-t-il quand on supprime un conteneur ? Pourquoi les changements sont perdus ?"

2

Prochaines Étapes

Demain : construction de **vos propres images** avec Dockerfile.

Cette semaine : gestion de la persistance et communication entre conteneurs (Docker Compose).

3

Travail Autonome

Explorez Docker Hub pour une technologie qui vous intéresse.

Testez le lancement d'autres images (base de données, langages...).

Félicitations ! Vous avez franchi la première étape vers la maîtrise de Docker. Cette technologie révolutionnaire vous accompagnera tout au long de votre carrière d'expert en informatique.