Données Massives

Docker : un environnement de virtualisation léger pour préparer les environnements Big Data

Chahrazed Labba
chahrazed.labba@univ-lorraine.fr

Plan

À propos de ce cours

Organisation du cours

- **Docker**
  Préparer et gérer des environnements reproductibles pour les TP
- **Introduction aux données massives**
  Définition, caractéristiques (les 3V,5V, ..), enjeux et applications
- **Mouvement NoSQL**
  Nouvelles bases de données adaptées aux données massives
- **Hadoop**
  Écosystème de stockage et de traitement distribué
- **Apache Spark**
  Traitement des données en mémoire, batch et streaming
- **Architectures Big Data**
  Data Lake, Data Warehouse, Lambda et Kappa
- **Un projet à réaliser**

Docker

Sources et Références

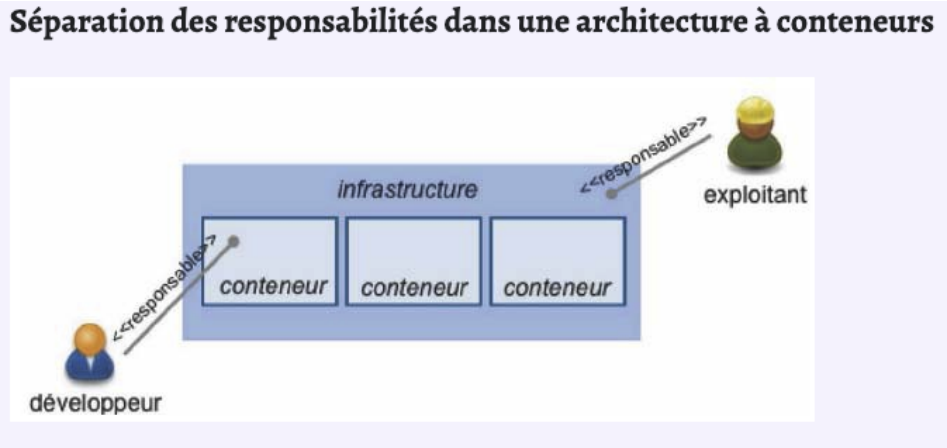
- Docker Documentation, *https://docs.docker.com/*
- Docker et conteneurs : Architectures, développement, usages et outils Cloux, Pierre-Yves; Garlot, Thomas; Kohler, Johann
- Comprendre Docker - Prise en main des conteneurs, Jean-Philippe GOUIGOUX

Un peu d'histoire : la conteneurisation

- **Conteneurs physiques (intermodaux)** : grandes boîtes métalliques standardisées pour le transport mondial
- **Avant la standardisation** :
  - Chargement manuel en petits lots (« break bulk cargo »)
  - Processus coûteux, lent et peu fiable
- **Premières tentatives** :
  - Fin XVII^{e} siècle : transport du charbon en Angleterre
  - Ligne Liverpool–Manchester : boîtes en bois standardisées
- **Standardisation moderne** :
  - Années 1930 : tests pour réduire les coûts après la crise de 1929
  - 1933 : création du Bureau International des Conteneurs (BIC)
- **Aujourd'hui** : plus de 494 millions de conteneurs en circulation

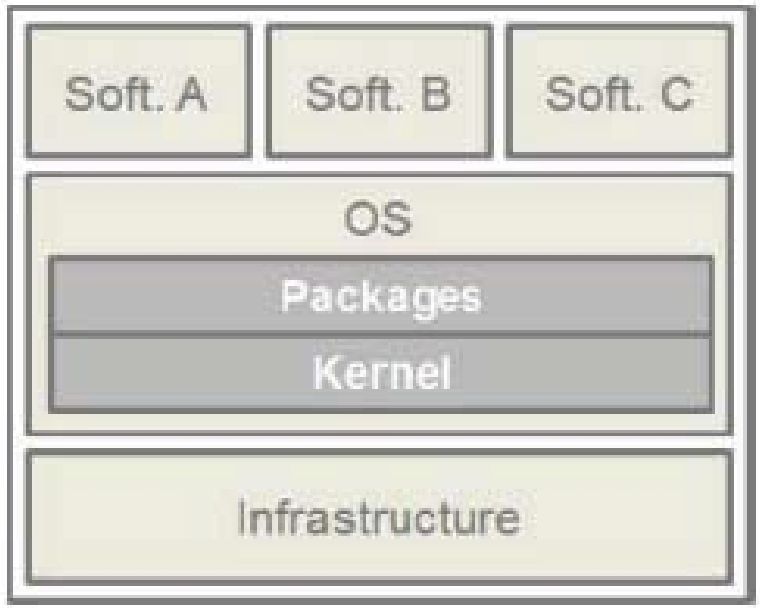
Extrapolation au monde logiciel

- Le déploiement logiciel est historiquement artisanal
  - Multiples incidents liés aux environnements
  - Difficultés lors de la livraison et de l'exploitation
- **Les conteneurs logiciels apportent une réponse** :
  - **Développeur** : responsable du contenu du conteneur (code + dépendances + OS)
  - **Exploitant** : responsable de l'infrastructure (stockage, réseau, puissance de calcul)
- Standardisation et séparation claire des rôles



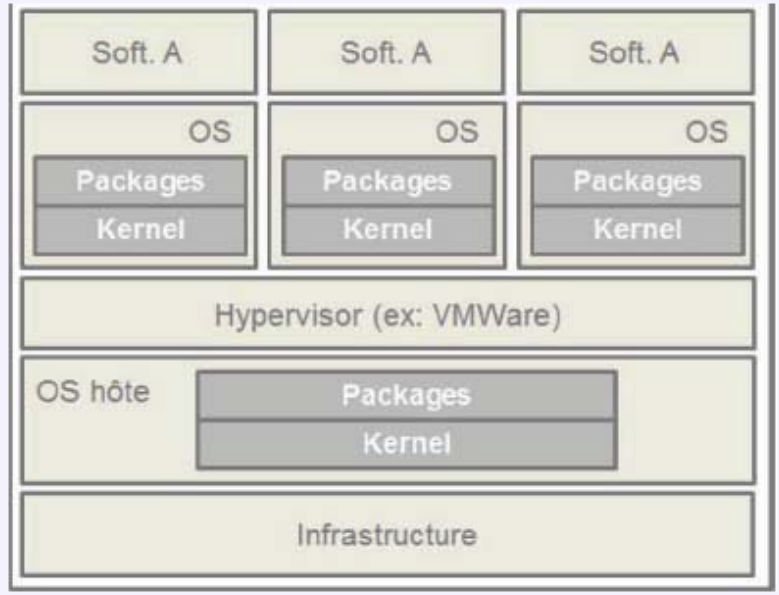
Différences avec la virtualisation matérielle

- **Installation classique (sans VM ni conteneur)** :
  - Conflits possibles entre logiciels :
    - Dépendances et versions incompatibles
    - Ports réseaux identiques
    - Accès concurrents (I/O, CPU)
  - Mise à jour de l'OS impacte tous les logiciels
  - Mise à jour d'un logiciel peut perturber les autres
- **Conséquence** : cohabitation souvent problématique
- **Question** : la virtualisation matérielle ne résout-elle pas cela ?



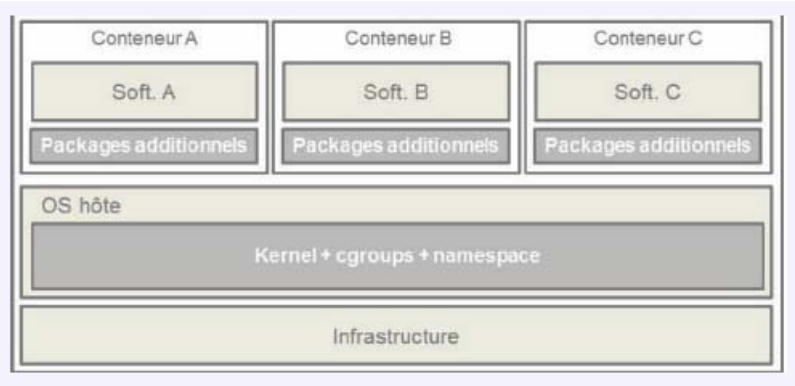
Virtualisation matérielle : « oui, mais… »

- **Isolation élevée** : chaque logiciel dans sa *sandbox*
- **Mais de nouvelles limites** :
  - **Poids des VM** : OS invité → mémoire de plusieurs Go, distribution lourde
  - **VM « trop pleine »** : embarque des choix infra (stockage, CPU, réseau) → moins de flexibilité
  - **Couplage aux outils** : gestion via vCenter, impact potentiel sur l'app
- **Conclusion** : oui pour l'isolation, mais coûts, lourdeur et rigidité



Architectures à base de conteneurs

- **Conteneurs : compromis efficace**
  - Isolation + dépendances logicielles incluses
  - Partagent le noyau de l'OS hôte → légers et rapides
  - Un hôte peut exécuter bien plus de conteneurs que de VM



Les conteneurs : avant Docker

- Les conteneurs existent depuis longtemps dans le monde Linux
- Basés sur deux mécanismes du noyau :
  - **Namespaces** : isolation des processus
  - **Control Groups (cgroups)** : limitation des ressources
- Avant Docker :
  - Outils comme **chroot**, **LXC** (Linux Containers)
  - Utilisation réservée aux experts → configuration complexe
- **Docker (2013)** :
  - N'a pas inventé les conteneurs
  - A simplifié leur usage et standardisé le format d'image

Docker : origine et vision

- **Origine** :
  - Créé par **DotCloud**, société de PaaS
  - Mars 2013 : création de Docker Inc. et mise en open source
- **Vision Docker** : un conteneur = un seul processus
- **Exemple : stack LAMP** (Linux, Apache, MySQL, PHP)
  - Apache → 1 conteneur
  - MySQL → 1 conteneur
  - PHP → 1 conteneur
- **Différence clé** :
  - **Avec LXC** : plusieurs processus dans un seul conteneur
  - **Avec Docker** : chaque service isolé dans son propre conteneur → modularité et flexibilité

Les points forts de Docker

- **Facilité de déploiement** :
  - Instanciation d'un conteneur en une seule commande
  - Automatisation via un **Dockerfile**
- **Manutention et diffusion** :
  - Partage simple d'un conteneur configuré
  - Garantit une exécution identique partout
- **Rapidité** :
  - Conteneur déjà packagé → démarrage quasi immédiat
- **Interchangeabilité** :
  - Même résultat quel que soit le système sous-jacent
- **Indépendance de l'infrastructure** :
  - Local, cluster, plateforme, cloud (CaaS)

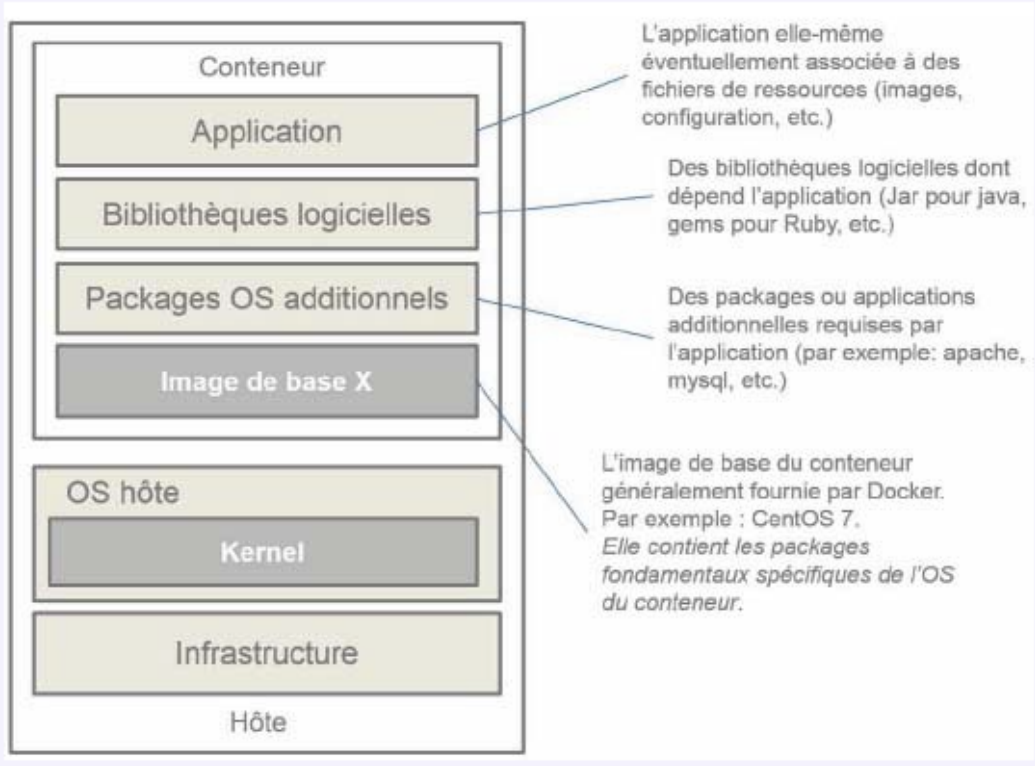
Concepts fondamentaux de Docker

- **Image Docker** :
  - Modèle en lecture seule (read-only)
  - Contient un système de base + une application
  - Sert de *template* pour créer des conteneurs
- **Docker Hub** :
  - Registre public d'images (par Docker Inc.)
  - Permet de télécharger des images officielles ou partagées
  - Adresse : `https://store.docker.com`

La notion d'image Docker

- **Question de départ** : d'où viennent les fichiers d'un conteneur ?
- **Problème initial** :
  - Construction manuelle fastidieuse (arborescence Linux)
  - Distribution et réutilisation difficiles
- **Apport de Docker** :
  - Conditionner le contenu en **images**
  - Archives réutilisables et partageables entre hôtes
  - Organisation en **couches** (layers) → mutualisation et efficacité
- **Standardisation** :
  - Depuis 2015 : **Open Container Initiative (OCI)**
  - Assure portabilité des images entre moteurs de conteneurs

Organisation en couches : Union File System

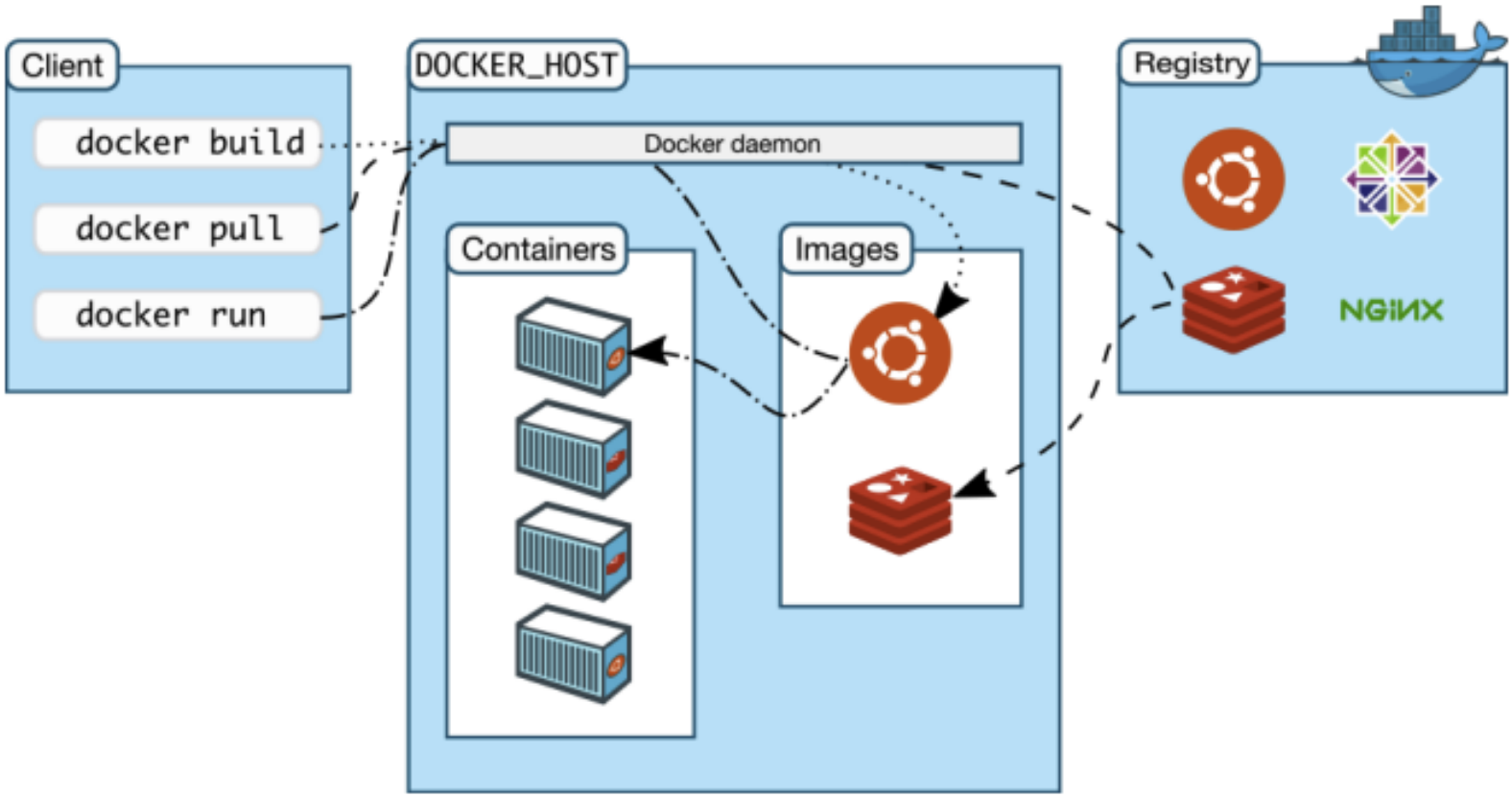


- Une image = **empilement de couches** (base → app)
- Chaque couche ajoute ou modifie la précédente
- Les couches sont partagées entre plusieurs images
- Assemblage possible grâce à l'**Union File System**
- Résultat final = un conteneur ou une nouvelle image

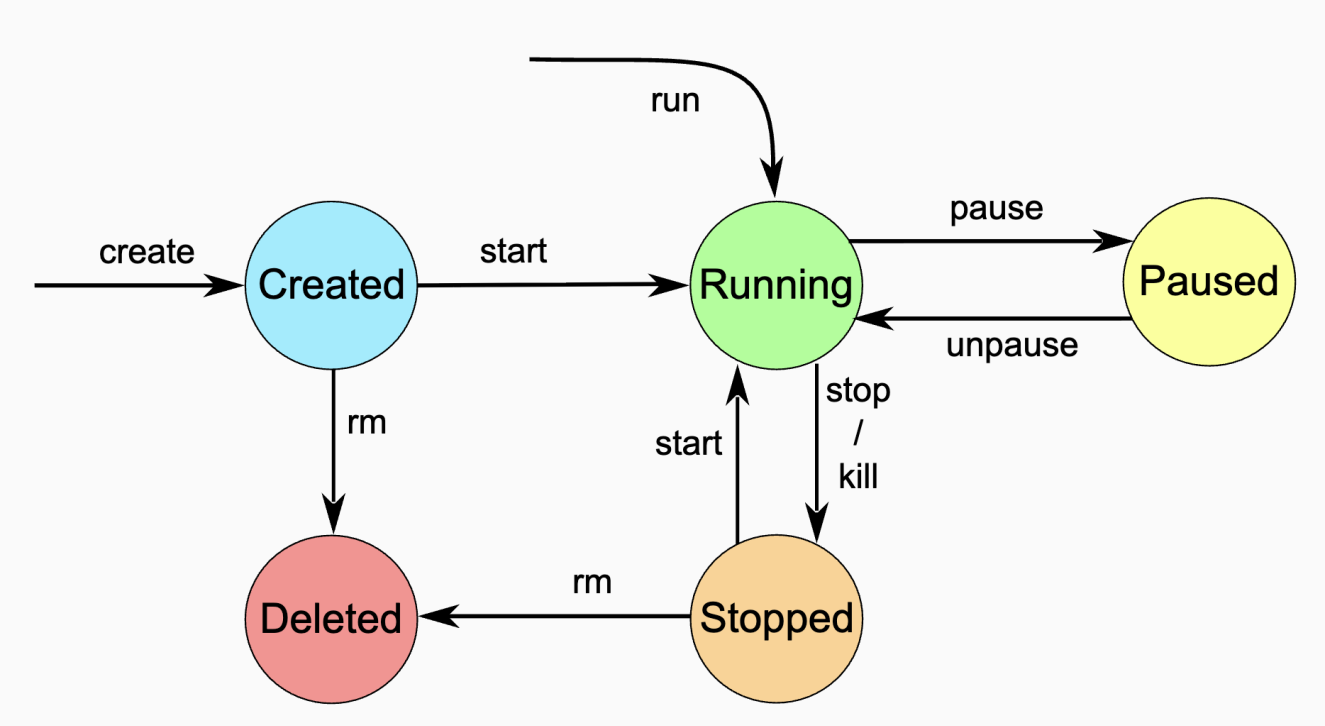
Docker Hub et Registry

- **Registry** :
  - Annuaire et centre de stockage des images
  - Point de téléchargement pour construire des conteneurs
- **Docker Hub** (https://hub.docker.com) :
  - Registry public géré par Docker Inc.
  - Publier des images publiques (gratuit) ou privées (payant)
  - Images de base officielles : Ubuntu, Debian, Rocky Linux, MySQL, Nginx, etc.
- **Registries privés** :
  - Utilisés par les entreprises pour leurs propres images
  - Fonctionnent comme Docker Hub mais en interne

Architecture Docker



Cycle de vie d'un conteneur docker



Les versions de Docker

- **Docker Community Edition (CE)** :
  - Version gratuite et open source
  - Disponible uniquement pour Linux
- **Docker Desktop** :
  - Version pour Windows et macOS
  - Inclut Docker Engine, CLI, Compose, et une interface graphique
- **Docker Enterprise** :
  - Version payante pour Linux
  - Destinée aux environnements de production et aux grandes entreprises
  - Inclut support, sécurité renforcée et outils avancés

Installer Docker sur votre machine

- **Créer un compte sur Docker Hub** :
  - `https://app.docker.com/signup`
- **Installer Docker Desktop** :
  - Windows : `https://docs.docker.com/desktop/setup/install/windows-install/`
  - Mac : `https://docs.docker.com/desktop/setup/install/mac-install/`

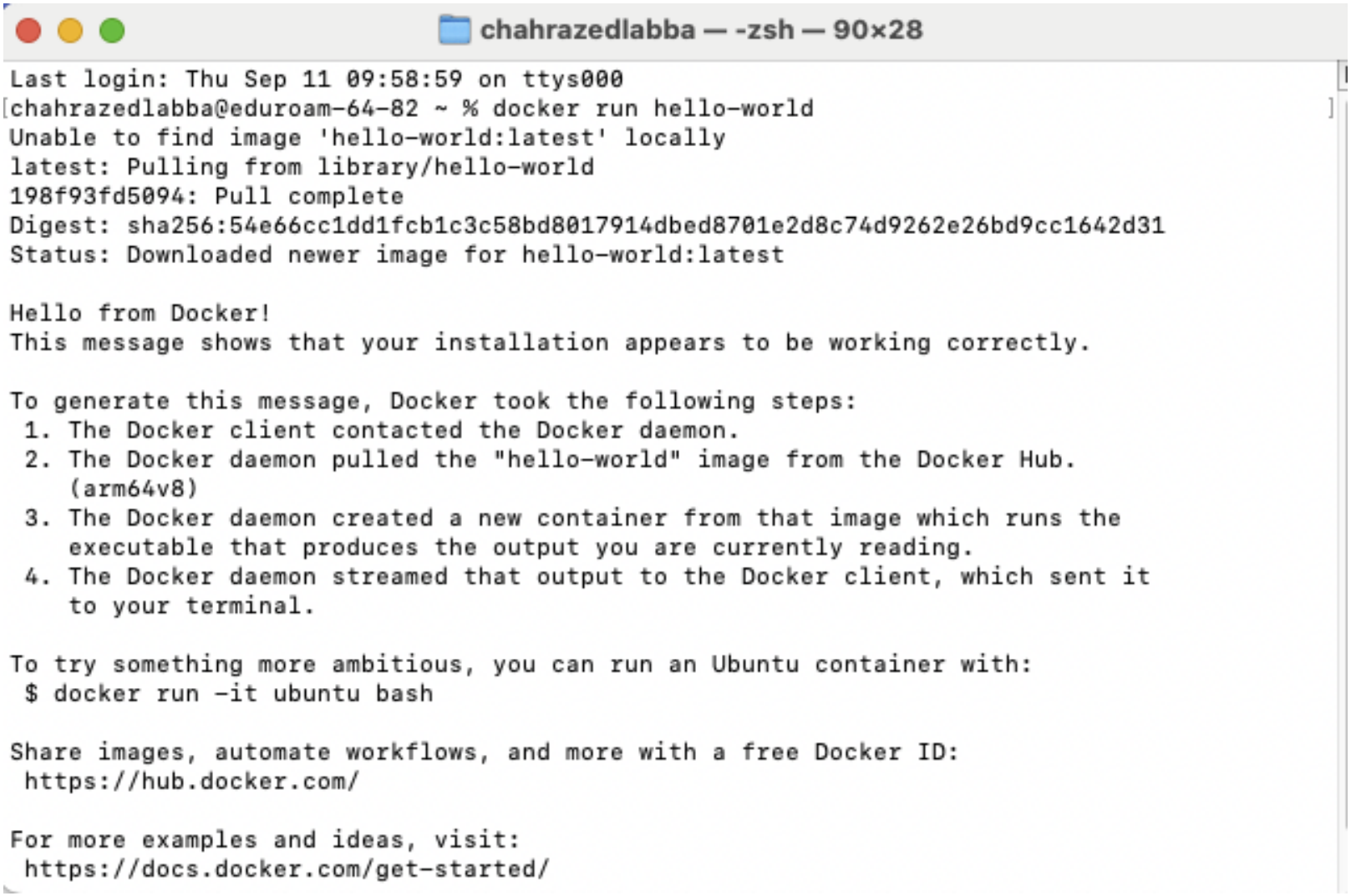
Tester l'installation : Hello World

- **But** :
  - Vérifier que Docker est bien installé et fonctionne
  - Valider toute la chaîne logicielle (client ↔ daemon ↔ registry ↔ conteneur)
- **Commande à exécuter** :

```
docker run hello-world
```

- **Résultat attendu** :
  - Téléchargement de l'image `hello-world` depuis Docker Hub
  - Création et exécution d'un conteneur minimaliste
  - Affichage d'un message de bienvenue confirmant l'installation

Test de l'installation : Hello World



Lancer un conteneur Docker (Ubuntu)(1/3)

- **Ouvrir un terminal :**
- **Télécharger une image Ubuntu :**

```
docker pull ubuntu
```

```
chahrazedlabba@eduroam-64-82 ~ % docker pull ubuntu
Using default tag: latest
latest: Pulling from library/ubuntu
cc43ec4c1381: Pull complete
Digest: sha256:9cbed754112939e914291337b5e554b07ad7c392491dba6daf25eef1332a22e8
Status: Downloaded newer image for ubuntu:latest
docker.io/library/ubuntu:latest
```

- **Lister les images disponibles :**

```
docker images
```

```
REPOSITORY    TAG       IMAGE ID       CREATED        SIZE
ubuntu        latest    9cbed7541129   3 weeks ago    139MB
hello-world   latest    54e66cc1dd1f   4 weeks ago    16.9kB
```

Lancer un conteneur Docker (Ubuntu) (2/3)

- **Lancer un conteneur interactif :**

```
docker run -it ubuntu
```

- **Exécuter des commandes Ubuntu :**
  - `ls`, `apt-get update`, `pwd`, etc.
- **Lister les conteneurs en cours d'exécution :**

```
docker ps
```

```
chahrazedlabba@eduroam-64-82 ~ % docker ps
CONTAINER ID   IMAGE     COMMAND       CREATED          STATUS          PORTS     NAMES
3bbb1ddec975   ubuntu    "/bin/bash"   11 seconds ago   Up 10 seconds             infallible_clarke
```

Lancer un conteneur Docker (Ubuntu)(3/3)

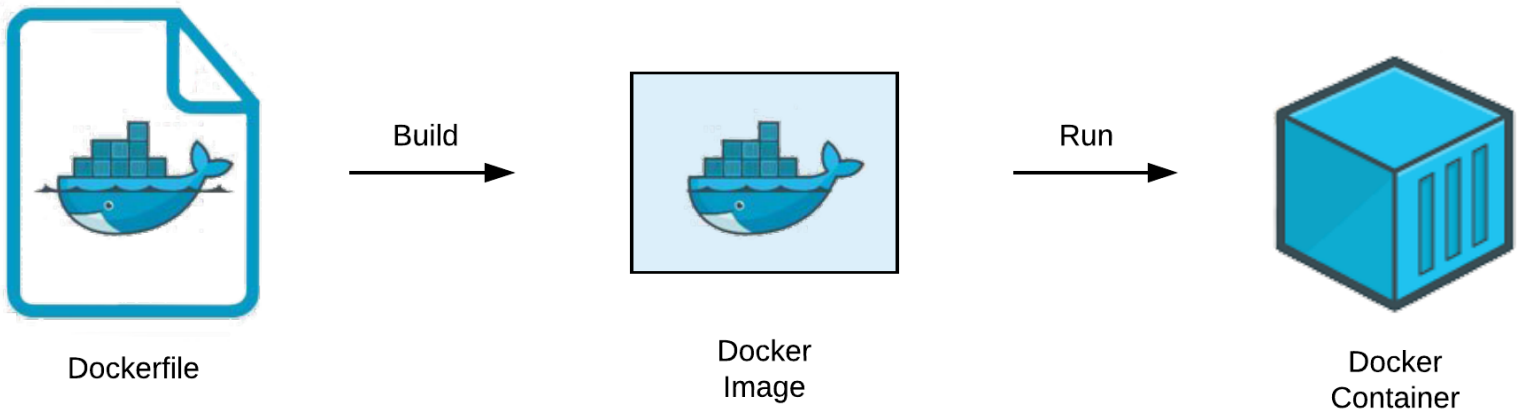
- **Exécuter un conteneur en arrière-plan (détaché) :**

```
docker run -it -d ubuntu
```

- **Explication des options :**
  - `-i` : mode interactif
  - `-t` : allocation d'un pseudo-terminal
  - `-d` : mode détaché (arrière-plan)
- **Arrêter un conteneur en cours d'exécution :**

```
docker stop <id_conteneur>
```

Construire une image Docker



Le Dockerfile : une recette de construction

- **Dockerfile** = fichier texte décrivant la construction d'une image
- Utilise une **grammaire spécifique à Docker**
- Contient toutes les **opérations nécessaires** à la préparation de l'image
- Permet de **compiler une image automatiquement** à partir d'une description textuelle

Exemple de Dockerfile

```
# Étape 1 : Image de base
FROM ubuntu:20.04
# Étape 2 : Installer Nginx
RUN apt-get update && \
    apt-get install -y nginx && \
    rm -rf /var/lib/apt/lists/*
# Étape 3 : Définir une variable d'environnement
ENV APP_HOME=/usr/share/nginx/html
# Étape 4 : Copier la page index.html dans le répertoire web
COPY index.html $APP_HOME/index.html
# Étape 5 : Définir un volume pour les fichiers web
VOLUME ["/usr/share/nginx/html"]
# Étape 6 : Exposer le port HTTP
EXPOSE 80
# Étape 7 : Définir l'ENTRYPOINT
ENTRYPOINT ["nginx"]
# Étape 8 : Définir la commande par défaut
CMD ["-g", "daemon off;"]
```

Instruction FROM

```
FROM ubuntu:20.04
```

- Toujours la première instruction d'un Dockerfile
- Définit l'**image de base**
- Sert de point de départ pour la construction
- Exemple : ici une distribution **Ubuntu 20.04**

Instruction RUN

```
RUN apt-get update && apt-get install -y nginx
```

- Exécute des commandes au moment de la **construction**
- Permet d'installer des paquets ou modifier l'image
- Chaque RUN crée une nouvelle couche
- Exemple : installation de **Nginx**

Instruction ENV

```
ENV APP_HOME=/usr/share/nginx/html
```

- Définit une **variable d'environnement** avec une valeur par défaut
- Utilisable :
  - Pendant la construction de l'image (dans d'autres instructions RUN, COPY...)
  - À l'exécution du conteneur (accessible comme une variable système)
- Ces valeurs peuvent être **surchargées au lancement**
  - Exemple : `docker run -e APP_HOME=/tmp nginx`
- Ici : `APP_HOME` pointe vers le répertoire web de Nginx

Instruction VOLUME

```
VOLUME ["/usr/share/nginx/html"]
```

- Déclare un répertoire comme **volume persistant**
- Sépare les **données** (contenu web, bases, logs) de l'application
- Permet de **partager** des données entre plusieurs conteneurs
- Les données du volume ne sont pas supprimées quand le conteneur est détruit
- Deux types d'utilisation :
  - **Volume anonyme** : géré automatiquement par Docker
  - **Volume monté** : on connecte un dossier local

Instruction COPY

```
COPY index.html $APP_HOME/index.html
```

- Copie des fichiers locaux vers l'image
- Permet d'ajouter du code, des configs, des pages web...
- Exemple : notre page `index.html`

Instruction EXPOSE

```
EXPOSE 80
```

- Indique le port utilisé par l'application à l'intérieur du conteneur
- **Ne rend pas le port accessible** depuis l'extérieur (il faut utiliser `-p`)
- Sert de **documentation** pour l'image et pour les outils (docker inspect, Compose, Kubernetes)
- Ici : port **80**, standard HTTP

Instructions ENTRYPOINT et CMD

```
ENTRYPOINT ["nginx"]
CMD ["-g", "daemon off;"]
```

- **ENTRYPOINT** : définit le programme principal
- **CMD** : définit les arguments par défaut
- Ici : exécute `nginx` en mode avant-plan

Une activité pratique

- Connectez-vous à **Arche**
- Téléchargez :
  - le fichier `Dockerfile`
  - le fichier `index.html`
- Créez un répertoire sur votre machine, par exemple `mon_site/`
- Placez `Dockerfile` et `index.html` dans ce répertoire
- Ouvrez un terminal et placez-vous dans ce dossier :
  - `cd mon_site`
- Construisez l'image Docker :
  - `docker build -t monsite .`

Lancer le conteneur et accéder au site

Commande

```
docker run -d -p 8080:80 monsite
```

- `docker run` : lance un nouveau conteneur
- `-d` : mode détaché (le conteneur tourne en arrière-plan)
- `-p 8080:80` : mappe le port 80 du conteneur sur le port 8080 de l'hôte
- `monsite` : nom de l'image à utiliser

Vérification

- Ouvrez un navigateur
- Tapez l'adresse : `http://localhost:8080`
- La page `index.html` doit s'afficher

Connexion à Docker Hub et publication d' une image docker

- Rendez-vous sur `https://hub.docker.com`.
- Créez un compte si vous n'en avez pas déjà un.
- Une fois le compte créé, connectez-vous à Docker Hub.

Connexion à Docker Hub et publication d' une image docker

- Ouvrez votre **terminal** (cmd sous Windows, terminal sous Linux/Mac)
- Tapez la commande :

```
docker login
```

- Entrez vos identifiants Docker Hub :
  - Nom d'utilisateur
  - Mot de passe
- Si la connexion réussit, vous verrez :

```
Login Succeeded
```

Préparer une image pour Docker Hub

- Pour publier une image, son nom doit inclure votre **nom d'utilisateur Docker Hub**. Cela garantit que l'image est associée à votre compte
- Identifiez votre image locale :

```
docker images
```

- Renommez (taggez) l'image avec votre identifiant :

```
docker tag nom\_image:latest \
  <votre_nom_utilisateur>/nom\_image:latest
```

- Pour notre exemple :

```
docker tag monsite:latest \
  <votre_nom_utilisateur>/monsite:latest
```

Publier une image sur Docker Hub

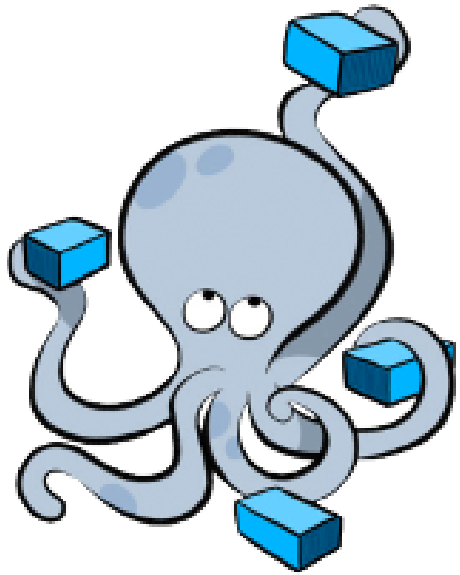
- Utilisez la commande `docker push` pour envoyer l'image :

  `docker push <votre_nom_utilisateur>/monsite:latest`
- Connectez-vous à votre compte Docker Hub via le navigateur
- Accédez à **Repositories** dans votre tableau de bord
- Vous devriez voir votre image `mon-serveur-apache` publiée avec le tag `latest`

Docker Compose : introduction

- Outil permettant de gérer des applications multi-conteneurs
- Décrit l'architecture dans un fichier YAML (`docker-compose.yml`)
- Permet de définir :
  - Les services (web, base de données, cache, etc.)
  - Les réseaux et volumes partagés
  - Les variables d'environnement
- Une seule commande pour lancer toute l'application :

```
docker compose up
```



Exemple de fichier docker-compose.yml

```yaml
version: "3.9"

services:
  web:
    image: nginx:latest
    ports:
      - "8080:80"
    volumes:
      - ./html:/usr/share/nginx/html   # bind mount pour ton code

  db:
    image: mysql:8.0
    environment:
      MYSQL_ROOT_PASSWORD: root
      MYSQL_DATABASE: demo
      MYSQL_USER: student
      MYSQL_PASSWORD: studentpass
    volumes:
      - db_data:/var/lib/mysql         # volume nommé pour persister les données

volumes:
  db_data:   # volume nommé, géré automatiquement par Docker
```

Structure d'un fichier docker-compose.yml

- **Version** : indique la version de syntaxe à utiliser
- **Services** : définit les conteneurs de l'application (depuis une image ou un Dockerfile)
- **Réseaux** : organisent la communication entre services (réseaux par défaut ou personnalisés)
- **Volumes** : permettent de stocker des données persistantes ou de partager des fichiers
- **Variables d'environnement** : configurent le comportement des services, définies dans le fichier ou via un `.env`

Commandes Docker Compose

- Démarrer les services :

  `docker compose up -d`
- Arrêter les services :

  `docker compose down`
- Voir les logs :

  `docker compose logs -f`
- Relancer avec reconstruction :

  `docker compose up -d --build`

Conclusion