Le Docker Multi-étapes

Le *multistage build* est une fonctionnalité de Docker qui permet d'utiliser **plusieurs instructions `FROM`** dans un seul `Dockerfile`.

Chaque instruction `FROM` commence un **nouvel étage de construction**. L'objectif est de séparer l'environnement nécessaire pour **construire l'application** (compilateurs, dépendances de développement, outils de test, etc.) de l'environnement nécessaire pour **exécuter l'application** en production.

Avant les constructions multi-étapes, si vous deviez compiler une application (par exemple, Go, Java, Node.js), vous aviez deux options :

1. **Méthode 1 : Un seul étage "gros"**
    - Vous incluez **tous** les outils de construction dans l'image finale.
    - **Résultat :** Image **très lourde** et contenant des dépendances inutiles, ce qui augmente la **surface d'attaque** (problème de sécurité).
2. **Méthode 2 : Deux Dockerfile séparés**
    - Vous construisez l'artefact dans une image temporaire, puis vous le copiez manuellement en dehors (sur l'hôte), puis vous le copiez dans une image finale plus petite.
    - **Résultat :** Processus **fastidieux** et **moins automatisé**.

Comment cela fonctionne? (exemple simplifié)

``FROM node:18-alpine AS builder WORKDIR /app COPY package*.json ./ RUN npm ci --omit=dev COPY . . RUN npm run build```	Cette étape est là pour **faire le travail lourd**. • Elle commence par une image de base complète (ex: node:18-alpine). • Elle installe toutes les dépendances de **développement**. • Elle **compile** votre code ou télécharge les dépendances. • Vous pouvez lui donner un nom avec AS <nom> (ex: AS builder).
``FROM nginx:alpine COPY --from=builder /app/build /usr/share/nginx/html```	L'Étape Finale (L'Image de Production): • Elle commence par une image de base légère (ex: `nginx:alpine` pour un front-end • Elle utilise l'instruction **`COPY --from=<nom_étape>`** pour ne transférer que l'artefact (le résultat compilé) de l'étape précédente.

Les Avantages Clés:

Avantage	Description
Taille Réduite	**L'image finale ne contient pas les outils de construction (ex: compilateurs, packages de développement, tests). Elle est souvent 10x à 100x plus petite.**
Sécurité Améliorée	**Moins de paquets inutiles = moins de vulnérabilités potentielles. L'image de production est propre.**
Clarté du Dockerfile	**Le processus de construction complet est contenu dans un seul fichier, ce qui simplifie le *workflow* et la maintenance.**