

Introduction à Docker pour DevOps

Conteneurisation et intégration continue dans le cycle DevOps



Développement



Intégration



Déploiement



Monitoring

Qu'est-ce que Docker?

Docker est une plateforme de **conteneurisation** qui permet d'empaqueter une application et ses dépendances dans un conteneur isolé, garantissant son fonctionnement uniforme dans différents environnements.



Code

Application



Dépendances

Bibliothèques, configurations



Isolation

Environnement isolé



Portabilité

Fonctionnement uniforme

De "ça marche sur ma machine" à "ça marche partout"

Le problème de la portabilité

"ça marche sur ma machine" - le plus grand défi du développement logiciel

Le défi des environnements divergents

Lorsqu'une application fonctionne parfaitement dans un environnement, mais échoue dans un autre, cela entraîne des retards, des erreurs et des coûts supplémentaires.



Développement

Environnement local du développeur

✅ Fonctionne



Test

Environnement contrôlé

⚠️ Peut échouer



Production

Environnement réel

❌ Souvent échoue

La solution avec Docker



Docker encapsule l'application et ses dépendances dans un conteneur isolé, garantissant que l'environnement d'exécution est identique quelle que soit la machine hôte.

Conteneurisation vs Virtualisation

Caractéristique	Conteneur Docker	Machine Virtuelle (VM)
 Isolation	 Partage le noyau du système d'exploitation hôte	 Complète (OS invité)
 Taille	 Mégaoctets	 Gigaoctets
 Démarrage	 Secondes	 Minutes
 Ressources	 Plus légères	 Plus gourmandes

Les conteneurs offrent une alternative plus légère et plus efficace aux VM pour la plupart des cas d'usage

Images Docker vs Conteneurs Docker

Images Docker



Modèle statique et immuable

En lecture seule, ne peut pas être modifiée



Composée de couches

Chaque instruction du Dockerfile crée une couche



Partageable et distribuable

Peut être partagée via des registres Docker



Définit le "quoi"

Détermine ce qui est inclus dans le conteneur



Conteneurs Docker



Instance exécutable

Version "vivante" d'une image en cours d'exécution



Modifiable temporairement

Peut être configurée lors de l'exécution



Isolation

Environnement isolé partageant le noyau hôte



Définit le "comment"

Détermine comment l'application est exécutée

Une image Docker devient un conteneur Docker lorsqu'elle est exécutée

Avantages de Docker



Portabilité

Applications fonctionnent identiquement dans n'importe quel environnement, quel que soit le système hôte.



Isolation

Environnements indépendants pour chaque application, évitant les conflits de dépendances entre systèmes.



Légèreté

Plus petits et plus rapides à démarrer que les machines virtuelles, avec une consommation de ressources moindre.



Reproductibilité

Même environnement peut être recréé à l'identique à tout moment, garantissant la cohérence des résultats.



Rapidité

Démarrage quasi instantané des conteneurs et rapidité de construction des images accélèrent les cycles de développement.

Ces avantages font de Docker l'outil idéal pour le développement et le déploiement d'applications dans un contexte DevOps

Docker dans l'écosystème DevOps



Tests

Automatisation des tests
Intégration continue



Docker

Plateforme de
conteneurisation



Monitoring

Observabilité
Déploiement continu



Collaboration

Environnements identiques pour tous les membres de l'équipe, éliminant les différences d'environnement.



Automatisation

Intégration parfaite dans les pipelines CI/CD, automatisant la construction, les tests et le déploiement.



Cohérence

Environnements identiques de développement à la production, résolvant le problème "ça marche sur ma machine".

Intégration et Déploiement Continu (CI/CD)

L'utilisation des images Docker comme artefacts idéaux dans les pipelines CI/CD, garantissant que l'application testée est identique à celle déployée.



Code

Dépôt Git



Build

Construction Docker



Test

Tests automatisés



Push

Enregistre Docker



Deploy

Déploiement

Avantages de Docker dans CI/CD



Identité garantie entre les environnements de test et de production



Temps de déploiement réduit par démarrage instantané des conteneurs



Isolation des applications pour une meilleure stabilité



Facilité des rollbacks avec des images immutables

Architecture Microservices

Docker facilite le développement et le déploiement d'architectures microservices en permettant la conteneurisation indépendante de chaque service.



Chaque service peut être développé, testé et déployé indépendamment

Structure d'un Dockerfile

Instructions clés



FROM

Définit l'image de base



WORKDIR

Définit le répertoire de travail



COPY

Copie des fichiers



RUN

Exécute une commande



EXPOSE

Déclare le port à écouter



CMD

Définit la commande par défaut

Exemple: Dockerfile Node.js

```
# Utilise une image de base officielle Node.js version 18
FROM node:18-alpine

# Définit le répertoire de travail
WORKDIR /app

# Copie les fichiers package.json et package-lock.json
COPY package*.json ./

# Installe les dépendances
RUN npm install

# Copie le reste du code de l'application
COPY . .

# Expose le port 3000
EXPOSE 3000

# Commande à exécuter lorsque le conteneur démarre
CMD ["node", "server.js"]
```

Fonctionnement du Dockerfile

Chaque instruction crée une nouvelle couche dans l'image Docker. Les couches sont immuables et partagées, ce qui rend les images légères et efficaces.

Cycle de vie du Dockerfile



1. Écriture du Dockerfile

Définition des instructions pour créer l'image

```
FROM node:18-alpine
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
EXPOSE 3000
CMD ["node", "server.js"]
```



2. Construction de l'image

Transformation du Dockerfile en image Docker

```
docker build -t mon-app-nodejs:1.0 .
```

- Crée une nouvelle image
- -t : tag de l'image
- . : contexte de construction



3. Exécution du conteneur

Démarrage de l'application dans un conteneur

```
docker run -p 80:3000 mon-app-nodejs:1.0
```

- -p : mappage de ports
- 80:3000 → hôte:conteneur
- Image à utiliser

Chaque phase dépend de la précédente pour fonctionner correctement

Construction d'une image Docker

```
docker build -t mon-application:1.0 .
```



Contexte de construction

Le point final `.` spécifie le contexte de construction, c'est-à-dire le chemin où se trouve le Dockerfile et les fichiers nécessaires à la construction.



Système de cache

Docker utilise un mécanisme de cache intelligent : si une instruction et son contexte n'ont pas changé depuis la dernière construction, Docker réutilise la couche mise en cache.



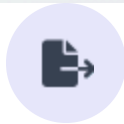
FROM

Définit l'image de base



WORKDIR

Définit le répertoire de travail



COPY

Copie les fichiers



RUN

Exécute une commande



EXPOSE

Expose un port



CMD

Définit la commande par défaut

Exécution d'un conteneur

Commande

```
docker run[OPTIONS]IMAGE[:TAG]  
// Lance un nouveau conteneur à partir d'une image
```

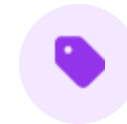


Mappage de ports

L'option `-p` permet de mapper un port de l'hôte à un port du conteneur.

```
docker run -p 80:3000 mon-app:1.0
```

Exemple: Le port 3000 du conteneur est accessible via le port 80 de l'hôte



Nommage des conteneurs

L'option `--name` permet de donner un nom à votre conteneur.

```
docker run --name mon-conteneur mon-app:1.0
```

Exemple: Le conteneur sera accessible par le nom "mon-conteneur" au lieu d'un ID généré



Plus d'options disponibles: `-d` (détaché), `-e` (variables d'environnement), `-v` (volumes)

Commandes de gestion des images

docker build

docker build

Construit une image Docker à partir d'un Dockerfile et d'un contexte.

```
docker build -t mon-app:1.0 .
```

docker images

docker images

Liste toutes les images Docker disponibles localement.

```
docker images
```

docker pull

docker pull

Télécharge une image ou un dépôt depuis un registre Docker.

```
docker pull ubuntu:latest
```

docker push

docker push

Télécharge une image ou un dépôt vers un registre Docker.

```
docker push mon-registre/mon-app:1.0
```

docker rmi

docker rmi

Supprime une ou plusieurs images Docker.

```
docker rmi mon-app:1.0
```

 **Conseil:** Utilisez des tags pour identifier et gérer vos images. Par convention, les tags incluent le nom du projet et le numéro de version.

Commandes de gestion des conteneurs

Les commandes fondamentales pour manipuler les conteneurs Docker



docker run

Crée et démarre un nouveau conteneur

```
docker run -p 80:80 mon-app:1.0
```



docker ps

Liste les conteneurs en cours d'exécution

```
docker ps -a # Inclut les conteneurs arrêtés
```



docker stop

Arrête un conteneur en cours d'exécution

```
docker stop mon-conteneur
```



docker start

Démarre un conteneur arrêté

```
docker start mon-conteneur
```



docker rm

Supprime un conteneur

```
docker rm mon-conteneur
```



docker logs

Affiche les logs d'un conteneur

```
docker logs mon-conteneur
```

Ces commandes sont essentielles pour gérer le cycle de vie de vos conteneurs Docker

Interaction avec les conteneurs

Commande `docker exec`

Volumes Docker



Définition

Les volumes Docker sont des mécanismes de stockage de données qui permettent de **stocker et partager des données** entre conteneurs et avec l'hôte, garantissant la persistance des données même lorsque les conteneurs sont supprimés.

Cas d'utilisation



Stockage de données persistantes (bases de données, fichiers de configuration)



Partage de données entre plusieurs conteneurs en cours d'exécution



Mise à jour d'applications sans perte de données

Types de volumes

Volumes nommés

Identifiés par un nom et gérés par Docker

Volumes anonymes

Créés automatiquement par Docker sans nom spécifique

Mounts de bind

Liens entre des chemins sur le système hôte et des chemins dans le conteneur

Bonnes pratiques

- Créez des volumes nommés pour une meilleure gestion des données
- Séparez les données d'état des données de configuration
- Utilisez des volumes read-only pour les données qui ne doivent pas être modifiées

Docker Compose

Introduction

Docker Compose est un outil qui permet de définir et d'exécuter des applications multi-conteneurs avec un simple fichier YAML.

Fonctionnalités clés



Définition simple

Définissez vos services avec un fichier YAML



Applications multi-conteneurs

Composez des applications complexes avec plusieurs services



Réseaux et volumes

Gestion simplifiée des réseaux et volumes



Gestion du cycle de vie

Démarrage, arrêt et déploiement simplifiés

Exemple de fichier Compose

`docker-compose.yml`

```
version: '3'
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
    volumes:
      - ./html:/usr/share/nginx/html
  db:
    image: mysql:5.7
    environment:
      MYSQL_ROOT_PASSWORD: example
    volumes:
      - db_data:/var/lib/mysql
volumes:
  db_data:
```

Commandes utiles

```
docker-compose up
docker-compose down
docker-compose restart
docker-compose logs
```

"Docker Compose est l'outil idéal pour gérer des applications multi-conteneurs dans un environnement DevOps"

Bonnes pratiques Docker

Conseils pour optimiser vos images Docker



Utiliser des images de base légères

Préférez des images de base minimalistes comme Alpine Linux pour réduire la taille de votre image finale. Cela réduit l'espace disque utilisé et accélère le démarrage de vos conteneurs.



Minimiser les couches

Combinez autant d'instructions que possible dans une seule commande RUN pour réduire le nombre de couches dans votre image. Moins de couches signifie moins de métadonnées et une image plus compacte.



Nettoyer les caches

Supprimez les fichiers temporaires, les caches et les dépendances non nécessaires avant de construire votre image. Utilisez des commandes multi-lignes avec des `&&` pour nettoyer automatiquement après chaque installation.



Sécuriser vos conteneurs

Limitez les privilèges des conteneurs, exécutez-les avec l'utilisateur non privilégié le plus approprié, et réduisez l'empreinte de surface d'attaque en exposant uniquement les ports nécessaires.



Conseil supplémentaire: Utilisez des outils comme `docker-slim`

Conclusion et ressources

Avantages clés de Docker dans DevOps

-  **Portabilité** : Applications fonctionnent identiquement dans n'importe quel environnement
-  **Isolation** : Environnements isolés pour chaque application, évitant les conflits
-  **Performance** : Légèreté des conteneurs par rapport aux VM, avec démarrage quasi instantané
-  **CI/CD** : Intégration parfaite dans les pipelines d'intégration et déploiement continus

Ressources pour approfondir

-  **Documentation officielle** : docker.com/docs
-  **Tutoriels** : Docker Hub, freeCodeCamp, W3Schools
-  **Communautés** : Docker Forums, Stack Overflow, GitHub
-  **Vidéos** : YouTube, DockerCon, TechTalks

Maintenez-vous à jour avec les dernières versions de Docker pour exploiter pleinement ses fonctionnalités DevOps