Introduction à Docker et Docker Compose

Docker est une plateforme open-source pour développer, livrer et exécuter des applications dans des **conteneurs**, des environnements légers et portables contenant le code, les dépendances et les configurations pour fonctionner de manière cohérente sur tout système. **Docker Compose** est un outil pour définir et gérer des applications multi-conteneurs via un fichier YAML, simplifiant l'orchestration locale.

Docker

- **Rôle** : Créer, exécuter et gérer des conteneurs isolés, souvent pour des applications comme des bases de données (MySQL) ou des serveurs web (Apache).
- **Fonctionnalités clés** :
    - **Conteneurs** : Exécuter des applications isolées avec leur propre système de fichiers.
    - **Images** : Modèles en lecture seule pour créer des conteneurs (ex. : mysql:latest).
    - **Portabilité** : Fonctionne partout (local, cloud, production).
    - **Gestion** : Contrôle des conteneurs, réseaux, volumes et images via une interface en ligne de commande.
    - **Écosystème** : Intégration avec Kubernetes, CI/CD et registres (Docker Hub).

Docker Compose

- **Rôle** : Orchestrer plusieurs conteneurs localement en définissant services, réseaux et volumes dans un fichier docker-compose.yml.
- **Fonctionnalités clés** :
    - **Multi-conteneurs** : Gère des applications complexes (ex. : frontend React + backend Node.js + MongoDB).
    - **Configuration simple** : Un fichier YAML définit tous les services.
    - **Environnement local** : Idéal pour le développement et les tests.
    - **Automatisation** : Lance tous les conteneurs en une seule commande.

Commandes Docker

Gestion des conteneurs

- docker run [image] : Crée et démarre un conteneur.
    - Ex. : docker run -d -p 8080:80 nginx
    - Options : -d (détaché), -p (ports), --name (nom), -v (volumes), -e (variables d’env.).
- docker ps : Liste les conteneurs en cours.
    - docker ps -a : Liste tous les conteneurs (y compris arrêtés).
- docker stop [container] : Arrête un conteneur.
- docker start [container] : Redémarre un conteneur arrêté.

- docker rm [container] : Supprime un conteneur.
    - docker rm -f : Force la suppression (même en cours).
- docker exec -it [container] [command] : Exécute une commande dans un conteneur.
    - Ex. : docker exec -it my-container bash
- docker logs [container] : Affiche les logs d'un conteneur.
- docker inspect [container] : Montre les détails (config, réseau, etc.).
- docker diff [container] : Liste les fichiers modifiés dans un conteneur.

Gestion des images

- **Créer une image** :
    - **Avec un Dockerfile** :
        - Rédige un Dockerfile définissant l'image (ex. : image de base, dépendances, commandes).

Ex. : Dockerfile

```
FROM nginx:latest
COPY ./index.html /usr/share/nginx/html/
```

        - Construit : docker build -t my-nginx:latest .
        - Crée une image my-nginx:latest à partir du Dockerfile.
    - **Avec commit** :
        - Modifie un conteneur en cours (ex. : installe des paquets ou copie des fichiers).
        - Ex. : Lance docker run -it ubuntu bash, installe apache2, puis valide :

```
docker commit my-container my-ubuntu-apache:latest
```

        - Crée une image my-ubuntu-apache:latest à partir de l'état du conteneur.
- docker images : Liste les images locales.
- docker pull [image] : Télécharge une image depuis un registre.
    - Ex. : docker pull mysql:latest
- docker push [image] : Publie une image sur un registre.
- docker rmi [image] : Supprime une image.
    - docker rmi -f : Force la suppression.

Gestion des volumes

- docker volume create [name] : Crée un volume persistant.
- docker volume ls : Liste les volumes.
- docker volume rm [name] : Supprime un volume.
- docker volume inspect [name] : Affiche les détails d'un volume.

Gestion des réseaux

- docker network create [name] : Crée un réseau.
    - Ex. : docker network create my-network

- docker network ls : Liste les réseaux.
- docker network connect [network] [container] : Connecte un conteneur à un réseau.
- docker network rm [network] : Supprime un réseau.

Autres

- docker info : Affiche les infos système.
- docker version : Montre la version de Docker.
- docker system prune : Nettoie les conteneurs, images et réseaux inutilisés.
    - docker system prune -a : Supprime tout ce qui n'est pas utilisé.

Commandes Docker Compose

- **Fichier** : docker-compose.yml définit les services, réseaux et volumes.
- **Commandes modernes** : Utilise docker compose (intégré à Docker, remplace docker-compose).
- **Commandes principales** :
    - docker compose up : Démarre tous les services dans le fichier YAML.
        - -d : Mode détaché.
        - Ex. : docker compose up -d
    - docker compose down : Arrête et supprime les conteneurs et réseaux (volumes persistants par défaut).
        - --volumes : Supprime aussi les volumes.
    - docker compose ps : Liste les conteneurs du projet.
    - docker compose logs : Affiche les logs des services.
    - docker compose exec [service] [command] : Exécute une commande dans un service.
        - Ex. : docker compose exec web bash
    - docker compose build : Construit les images définies dans le YAML.
    - docker compose pull : Télécharge les images spécifiées.
    - docker compose stop : Arrête les services sans les supprimer.
    - docker compose start : Redémarre les services arrêtés.
    - docker compose rm : Supprime les conteneurs arrêtés.
    - docker compose config : Valide et affiche la configuration YAML.

Exemple : Application MERN avec Docker Compose

Voici un fichier **docker-compose.yml** pour une application **MERN** (MongoDB, Express.js, React, Node.js) :

```yaml
services:
  frontend:
    image: mern-frontend:latest
    build: ./frontend
```

```
    ports:
      - "3000:80"
    depends_on:
      - backend
  backend:
    image: mern-backend:latest
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - MONGO_URI=mongodb://mongo:27017/mern
    depends_on:
      - mongo
  mongo:
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - mongo-data:/data/db
volumes:
  mongo-data:
```

- **Explication** :
  - **Services** : frontend (React), backend (Express.js), mongo (MongoDB).
  - **Ports** : Expose 3000 (frontend), 5000 (backend), 27017 (MongoDB).
  - **Volumes** : Persiste les données MongoDB dans mongo-data.
  - **Dépendances** : Le frontend attend le backend, qui attend MongoDB.
- **Commandes** :
  - Lancer : docker compose up -d
  - Arrêter : docker compose down
  - Logs : docker compose logs backend

Relation avec DevOps

Docker et Docker Compose sont des outils clés dans les pipelines **DevOps**, soutenant l'automatisation, la collaboration et la livraison continue (CI/CD) :

- **Automatisation** : Docker standardise les environnements (build/test dans CI) via des conteneurs, et Compose orchestre des environnements locaux complexes.
- **CI/CD** : Intégration avec Jenkins/GitLab CI pour construire (docker build), tester et déployer des images (docker push) automatiquement.
- **Infrastructure as Code (IaC)** : Les Dockerfiles et fichiers docker-compose.yml sont versionnés dans Git, alignés avec les pratiques IaC.
- **Collaboration** : Les équipes Dev et Ops utilisent des images Docker pour garantir la cohérence entre développement et production.

- **Monitoring** : Les logs (docker logs, docker compose logs) et outils comme cAdvisor soutiennent l'observabilité DevOps.
- **Exemple MERN** : Un pipeline DevOps construit des images MERN avec Docker, les teste avec Compose, et les déploie sur Kubernetes, avec des logs centralisés pour le suivi.

Résumé

Docker gère des conteneurs (création via Dockerfile ou docker commit, exécution avec docker run, gestion avec docker ps, docker rm) pour des applications portables. **Docker Compose** orchestre des multi-conteneurs localement via docker-compose.yml avec docker compose up, down, logs. Exemple : Une app MERN utilise Compose pour lancer React, Express.js et MongoDB ensemble. Dans **DevOps**, Docker et Compose automatisent les builds, tests et déploiements, soutenant CI/CD et IaC.