Rapport de TP – Introduction à Docker

Samy Mehamli

1 Introduction : virtualisation lourde vs conteneurisation

La virtualisation classique repose sur l'utilisation de machines virtuelles. Chaque machine virtuelle embarque un système d'exploitation complet, ce qui entraîne une consommation importante de ressources ainsi que des temps de démarrage élevés.

Docker propose une alternative appelée **conteneurisation**. Les conteneurs partagent le noyau du système hôte et embarquent uniquement l'application et ses dépendances. Cette approche permet un démarrage rapide, une meilleure portabilité et une utilisation plus efficace des ressources.

2 Avantages de Docker

Les principaux avantages de Docker sont :
— **Portabilité** : une image fonctionne de manière identique sur toutes les machines.
— **Déploiement simplifié** : les dépendances sont intégrées dans l'image.
— **Isolation** : chaque conteneur est isolé des autres.
— **Performance** : plus léger qu'une machine virtuelle.
— **Scalabilité** : duplication facile des services.

3 Commandes Docker de base

3.1 Lister les conteneurs

— `docker ps` : affiche les conteneurs en cours d'exécution.
— `docker ps -a` : affiche tous les conteneurs.

3.2 Lancer un conteneur

```
docker run -it alpine:latest /bin/sh
```

Cette commande télécharge l'image Alpine si nécessaire et ouvre un shell interactif dans le conteneur.

3.3 Accéder à un conteneur

```
docker exec -it <conteneur> sh
```

Permet d'exécuter une commande dans un conteneur déjà lancé.

3.4 Nettoyage

```
docker system prune
```

Supprime les conteneurs arrêtés, images inutilisées et réseaux non utilisés.

4 Export et import d'un conteneur

Docker permet d'exporter le système de fichiers d'un conteneur pour créer une nouvelle image.

```
docker export <id_conteneur> -o my_container.tar
docker import my_container.tar my_new_image:v1
```

Cette méthode produit généralement une image plus lourde car l'historique des couches est perdu.

5 Gestion du stockage Docker

Par défaut, les données d'un conteneur sont supprimées lors de sa suppression. Docker propose plusieurs solutions pour gérer la persistance des données.

5.1 Bind mount

Lien direct entre un dossier de l'hôte et un dossier du conteneur.

```
docker run -v ./local:/app image
```

Avantages : pratique en développement. **Inconvénients :** dépendance forte à l'hôte.

5.2 Volumes Docker

Volumes gérés par Docker, indépendants des conteneurs.

```
docker volume create monvolume
docker run -v monvolume:/data image
```

Avantages : persistance, partage, bonnes performances. **Inconvénients :** moins visibles sur l'hôte.

5.3 tmpfs

Stockage temporaire en mémoire vive.

```
docker run --tmpfs /data image
```

Avantages : rapide et sécurisé. **Inconvénients :** données perdues à l'arrêt.

5.4 Cas pratique : MySQL

Un volume Docker a été utilisé pour stocker les données MySQL dans `/var/lib/mysql`. Après suppression et recréation du conteneur, les données restent accessibles, ce qui démontre l'intérêt des volumes pour les bases de données.

6 Gestion des réseaux Docker

Docker fournit un système réseau permettant la communication entre conteneurs et avec l'extérieur.

6.1 Types de réseaux

Bridge (par défaut)

- réseau privé entre conteneurs ;
- communication via IP ou nom de conteneur ;
- accès externe par redirection de ports.

Avantages : simplicité et isolation. **Inconvénients :** performances légèrement réduites.

Host Le conteneur partage directement le réseau de l'hôte. **Avantages :** meilleures performances. **Inconvénients :** aucune isolation, risques de conflits.

None Aucune connectivité réseau. **Avantages :** isolation maximale. **Inconvénients :** inutilisable pour une application réseau.

Réseaux personnalisés

```
docker network create monreseau
docker run -d --name site1 --network monreseau nginx
docker run -it --name client --network monreseau alpine sh
```

Les réseaux personnalisés offrent un DNS interne automatique entre conteneurs, très utilisé avec Docker Compose.

7 Exposition de ports

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

Le service devient accessible via `http://localhost:8080`.

8 Dockerfile

Un Dockerfile décrit la construction d'une image Docker.

8.1 Exemple

```
FROM openjdk:17-jdk-slim
WORKDIR /app
COPY target/app.jar app.jar
EXPOSE 8080
CMD ["java", "-jar", "app.jar"]
```

```
docker build -t mon_app_java .
docker run -p 8080:8080 mon_app_java
```

9 Applications conteneurisées

9.1 Application Flask

Dans le cadre du TP, une application Flask simple a été conteneurisée afin d'illustrer le déploiement d'une API REST à l'aide de Docker. Flask a été choisi pour sa légèreté et sa simplicité, ce qui permet de se concentrer sur les mécanismes de conteneurisation plutôt que sur la complexité applicative.

L'application expose une API HTTP sur le port 5000 et stocke les données en mémoire, sans base de données persistante. Elle permet de manipuler une collection d'étudiants à travers des requêtes REST.

Les fonctionnalités principales sont :

— la récupération de la liste des étudiants via une requête `GET` ;

— l'ajout de nouveaux étudiants via `POST` ;

— la modification et la suppression d'étudiants via `PUT` et `DELETE`.

Cette application permet de comprendre comment :

— exposer un service réseau depuis un conteneur ;

— accéder à une application conteneurisée depuis l'extérieur ;
— préparer une application à être intégrée dans une architecture multi-conteneurs.

10 Docker Compose

Docker Compose est un outil permettant de définir et de déployer une application composée de plusieurs conteneurs à l'aide d'un fichier de configuration au format YAML. Il permet de décrire l'architecture complète d'une application multi-services de manière déclarative.

Dans le TP, Docker Compose est utilisé pour mettre en place une architecture composée de deux services distincts :

— un service Flask fournissant une API REST de produits ;
— un service Apache/PHP chargé d'afficher ces produits à partir de l'API.

Chaque service est défini comme un conteneur indépendant, avec son propre environnement, ce qui respecte le principe de séparation des responsabilités.

10.1 Gestion du réseau

Docker Compose crée automatiquement un réseau virtuel dédié à l'application. Les conteneurs appartenant au même fichier `docker-compose.yml` peuvent communiquer entre eux via leurs noms de services, grâce à un DNS interne.

Ainsi, le serveur Apache/PHP peut accéder à l'API Flask sans utiliser d'adresse IP explicite, ce qui rend la configuration plus robuste et plus portable.

10.2 Avantages de Docker Compose

L'utilisation de Docker Compose apporte plusieurs avantages :

— démarrage et arrêt de l'ensemble des services avec une seule commande ;
— configuration centralisée de l'architecture applicative ;
— gestion automatique du réseau entre conteneurs ;
— simplification des tests et du déploiement en environnement local.

Docker Compose permet ainsi de se rapprocher d'une architecture réelle de production, tout en conservant une mise en œuvre simple et reproductible.

11 Conclusion

Ce TP a permis d'acquérir une compréhension globale de Docker : conteneurisation, images, Dockerfiles, réseaux, volumes et Docker Compose. Docker constitue une brique essentielle du déploiement moderne et prépare à l'utilisation d'outils d'orchestration comme Kubernetes.