TP Docker

> ***Contexte :*** *Ce TP reprend l'approche du workshop officiel Docker "Getting Started".*
> ***But :*** *apprendre Docker pas à pas en containerisant une app, en la publiant, en persistant ses données, puis en passant à une architecture multi-conteneurs et à Docker Compose.*

Prérequis

- Docker Desktop installé et démarré
- Git installé
- Un terminal (PowerShell/CMD/Git Bash sur Windows, Terminal sur Mac/Linux)
- Un éditeur (VS Code recommandé)

Vérification rapide

```
docker version
docker compose version
docker run hello-world
```

Résultat attendu : affichage "Hello from Docker!".

Part 1 – Containerize an application

> *Dans cette partie, on récupère une application simple (Todo), on écrit un*

> *Dockerfile, puis on construit et lance notre première image Docker.*

1. Get the app

1. Clone le projet :

```
git clone https://github.com/docker/getting-started-app.git
cd getting-started-app
```

2. Vérifie le contenu du dossier : `package.json` , `src/` , `yarn.lock` , `.dockerignore` , etc.

2. Build the app's image

2.1 Créer le Dockerfile

Dans la racine du projet, crée un fichier **Dockerfile** (sans extension) :

```
# syntax=docker/dockerfile:1

FROM node:lts-alpine
WORKDIR /app
COPY . .
RUN yarn install --production
CMD ["node", "src/index.js"]
EXPOSE 3000
```

> **Idée :** on part d'une image Node légère, on copie le code, on installe les dépendances, puis on démarre l'app sur le port 3000.
>
> - `FROM node:lts-alpine` : image de base légère avec Node + Yarn.
> - `WORKDIR /app` : dossier de travail interne.
> - `COPY . .` : copie du code.

- `RUN yarn install --production` : installe les dépendances.
- `CMD ...` : commande de démarrage.
- `EXPOSE 3000` : l'app écoute sur 3000.

2.2 Construire l'image

```
docker build -t getting-started .
```

Vérification

```
docker image ls
```

Tu dois voir l'image `getting-started`.

3. Start an app container

1. Lance le conteneur :

```
docker run -d -p 127.0.0.1:3000:3000 getting-started
```

2. Ouvre `http://localhost:3000` dans ton navigateur.
3. Ajoute 1–2 items Todo pour valider le fonctionnement.
4. Liste les conteneurs :

```
docker ps
```

Tu dois voir le conteneur publié sur `127.0.0.1:3000`.

Q1. Quelle est la différence entre une **image Docker** et un **conteneur Docker** ?

Part 2 – Update the application

> *Ici, on modifie le code, on reconstruit l'image, puis on remplace l'ancien conteneur par le nouveau.*

1. Update the source code

1. Ouvre `src/static/js/app.js` .
2. Modifie le texte affiché quand la liste est vide :

```
- No items yet! Add one above!
+ You have no todo items yet! Add one above!
```

2. Rebuild the image

```
docker build -t getting-started .
```

3. Start a new container (observe l'erreur)

```
docker run -dp 127.0.0.1:3000:3000 getting-started
```

> **Erreur attendue :** port déjà utilisé.

4. Remove the old container

1. Trouve l'ancien conteneur :

```
docker ps
```

2. Supprime-le :

```
docker rm -f <container-id>
```

5. Start the updated container

```
docker run -dp 127.0.0.1:3000:3000 getting-started
```

Recharge `http://localhost:3000` → le nouveau texte doit s'afficher.

> **Q2.** Pourquoi doit-on **supprimer l'ancien conteneur** avant de lancer le nouveau après un rebuild ?

Part 3 – Share the application

> *Cette partie montre comment publier une image vers Docker Hub pour pouvoir la partager et la déployer ailleurs.*

1. Create a repository

Sur Docker Hub :

1. Se connecter / créer un compte
2. **Create Repository**

3. Nom : `getting-started`
4. Visibilité : **Public**
5. Create

2. Push the image

1. Tentative de push "exemple" (pour voir l'erreur) :

```
docker push docker/getting-started
```

2. Login :

```
docker login
```

3. Tague l'image locale :

```
docker tag getting-started YOUR-USER-NAME/getting-started
```

4. Push :

```
docker push YOUR-USER-NAME/getting-started
```

Résultat attendu : l'image apparaît dans ton repo Docker Hub.

> **Q3.** À quoi sert la commande `docker tag` avant un `docker push` ?

Part 4 – Persist the DB (Volumes)

> *On apprend à ne pas perdre les données quand on supprime un conteneur, grâce aux volumes Docker.*

1. Create a volume

```
docker volume create todo-db
```

2. Remove the current container

```
docker ps
docker rm -f <container-id>
```

3. Start a new container with the volume mounted

```
docker run -dp 127.0.0.1:3000:3000 \
  --mount type=volume,src=todo-db,target=/etc/todos \
  getting-started
```

> **Windows Git Bash (si besoin) :**
>
> ```
> docker run -dp 127.0.0.1:3000:3000 \
>   --mount type=volume,src=todo-db,target=//etc/todos \
>   getting-started
> ```

4. Verify that the data persists

1. Ajoute quelques todos.

2. Supprime le conteneur :

```
docker rm -f <container-id>
```

3. Relance la commande `docker run ...` avec volume.
4. Les todos doivent rester visibles.

Q4. Quel problème résout un **volume Docker** dans cette application Todo ?

Part 6 – Multi-container apps (MySQL + App)

> *On sépare l'application et la base de données dans deux conteneurs, reliés par un réseau Docker.*

1. Create a network

```
docker network create todo-app
```

2. Start MySQL

Start a MySQL container and attach it to the network. You're also going to define a few environment variables that the database will use to initialize the database.

Mac / Linux / Git Bash

```
docker run -d \
  --network todo-app --network-alias mysql \
  -v todo-mysql-data:/var/lib/mysql \
  -e MYSQL_ROOT_PASSWORD=secret \
  -e MYSQL_DATABASE=todos \
  mysql:8.0
```

PowerShell

```
docker run -d `
  --network todo-app --network-alias mysql `
  -v todo-mysql-data:/var/lib/mysql `
  -e MYSQL_ROOT_PASSWORD=secret `
  -e MYSQL_DATABASE=todos `
  mysql:8.0
```

CMD

```
docker run -d ^
  --network todo-app --network-alias mysql ^
  -v todo-mysql-data:/var/lib/mysql ^
  -e MYSQL_ROOT_PASSWORD=secret ^
  -e MYSQL_DATABASE=todos ^
  mysql:8.0
```

3. Quick DB check

```
docker ps
docker exec -it <mysql-id> mysql -u root -p
```

Mot de passe : `secret`

```
SHOW DATABASES;
exit;
```

Tu dois voir la base `todos` .

4. Run the app connected to MySQL

The todo app supports the setting of a few environment variables to specify MySQL connection settings. They are:

- `MYSQL_HOST` - the hostname for the running MySQL server
- `MYSQL_USER` - the username to use for the connection
- `MYSQL_PASSWORD` - the password to use for the connection
- `MYSQL_DB` - the database to use once connected

Mac / Linux / Git Bash

```
docker run -dp 127.0.0.1:3000:3000 \
  -w /app -v "$(pwd):/app" \
  --network todo-app \
  -e MYSQL_HOST=mysql \
  -e MYSQL_USER=root \
  -e MYSQL_PASSWORD=secret \
  -e MYSQL_DB=todos \
  node:lts-alpine \
  sh -c "yarn install && yarn run dev"
```

PowerShell

```
docker run -dp 127.0.0.1:3000:3000 `
  -w /app -v "$(pwd):/app" `
  --network todo-app `
  -e MYSQL_HOST=mysql `
  -e MYSQL_USER=root `
  -e MYSQL_PASSWORD=secret `
  -e MYSQL_DB=todos `
  node:lts-alpine `
  sh -c "yarn install && yarn run dev"
```

CMD

```
docker run -dp 127.0.0.1:3000:3000 ^
  -w /app -v "%cd%:/app" ^
  --network todo-app ^
  -e MYSQL_HOST=mysql ^
  -e MYSQL_USER=root ^
  -e MYSQL_PASSWORD=secret ^
  -e MYSQL_DB=todos ^
  node:lts-alpine ^
  sh -c "yarn install && yarn run dev"
```

Résultat attendu : l'app tourne sur `localhost:3000` et utilise MySQL.

> **Q5.** Pourquoi utilise-t-on un réseau Docker et un alias `mysql` dans cette partie ?

Part 7 – Use Docker Compose

> *On remplace les longues commandes `docker run` par un fichier*

> *`compose.yaml` pour lancer toute la stack en une seule commande.*

In part 6, you used the following command to start the application service.

```
docker run -dp 127.0.0.1:3000:3000 \
  -w /app -v "$(pwd):/app" \
  --network todo-app \
  -e MYSQL_HOST=mysql \
  -e MYSQL_USER=root \
  -e MYSQL_PASSWORD=secret \
  -e MYSQL_DB=todos \
  node:lts-alpine \
  sh -c "yarn install && yarn run dev"
```

You'll now define this service in the compose.yaml file.

1. Create a compose file

Dans la racine, crée `compose.yaml` .

2. Define the services

```
services:
  app:
    image: node:lts-alpine
    command: sh -c "yarn install && yarn run dev"
    ports:
      - 127.0.0.1:3000:3000
    working_dir: /app
    volumes:
      - ./:/app
    environment:
      MYSQL_HOST: mysql
      MYSQL_USER: root
      MYSQL_PASSWORD: secret
      MYSQL_DB: todos

  mysql:
    image: mysql:8.0
    volumes:
      - todo-mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: secret
      MYSQL_DATABASE: todos

volumes:
  todo-mysql-data:
```

Explication :

- `services` : définit les conteneurs.
- `app` dépend de `mysql` via les variables.
- `ports` publie l'app.
- `volumes` assure la persistance MySQL.

3. Run the app stack

1. Supprime les anciens conteneurs si besoin :

```
docker ps
docker rm -f <ids>
```

2. Lance :

```
docker compose up -d
```

3. Vérifie :

```
docker compose ps
```

4. View logs

```
docker compose logs -f
```

5. Tear down

```
docker compose down
```

Pour supprimer aussi le volume :

```
docker compose down --volumes
```

Q6. Quel est l'avantage principal de Docker Compose par rapport à plusieurs commandes `docker run` ?