

TP2 Docker - MAZOYER Alexandre

Fichier models/inex.js adapté :

```
Docker > src > models > JS index.js > ...
1  const { Sequelize } = require('sequelize');
2  const dbConfig = require('../db.config');
3
4  // Comment this block to disable Sqlite
5  /*const instance = new Sequelize({
6    dialect: dbConfig.dialect,
7    storage: dbConfig.storage
8  });*/
9
10 // Uncomment this block to use Mysql, don't forget to adapt db.config.js
11 const instance = new Sequelize(dbConfig.database, dbConfig.username, dbConfig.password, {
12   host: dbConfig.hostname,
13   port: dbConfig.port,
14   dialect: "mysql"
15 });
16
17 module.exports = {
18   instance,
19   books: require('./books')(instance)
20 };
```

Fichier db.config.js adapté :

```
Docker > src > JS db.config.js > ...
1  // Comment this block to disable sqlite
2  /*module.exports = {
3    dialect: "sqlite",
4    storage: "./my-db.sqlite",
5  };*/
6
7  // Uncomment this block to use mysql
8  module.exports = {
9    dialect: "mysql",
10    hostname: process.env.DB_HOST || "localhost",
11    username: process.env.DB_USER || "root",
12    password: process.env.DB_PASSWORD || "",
13    database: process.env.DB_NAME || "mydatabase",
14    port: process.env.DB_PORT || 3306
15  };
16
17  // TODO : adapt this file to load parameters from environment variables (process.env.VARIABLE_NAME)
```

Reconstruction de l'image Docker :

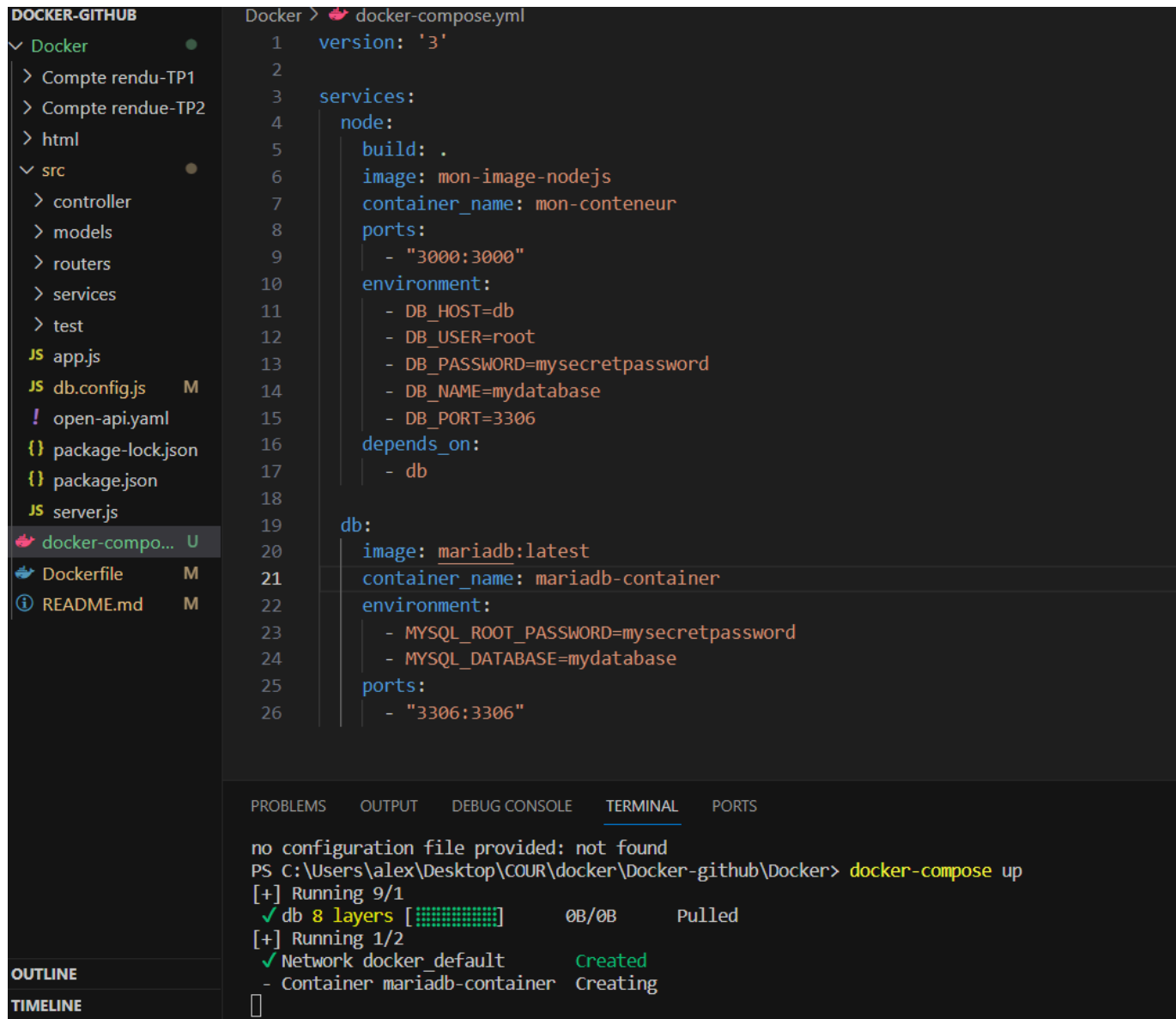
```
PS C:\Users\alex\Desktop\COUR\docker\Docker-github\Docker> docker build -t mon-image-nodejs .
[+] Building 10.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 558B
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load metadata for docker.io/library/node:18-alpine
=> [1/5] FROM docker.io/library/node:18-alpine
=> [internal] load build context
=> => transferring context: 442.61kB
=> [2/5] WORKDIR /app
=> [3/5] COPY src/package*.json ./
=> [4/5] RUN npm install
=> [5/5] COPY src/ .
=> exporting to image
=> => exporting layers
=> => writing image sha256:b07d12374d7eeddade84e4e1c0715f603c8da080cc9a4f6e234a7a28fa9c5649
=> => naming to docker.io/library/mon-image-nodejs

What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview
PS C:\Users\alex\Desktop\COUR\docker\Docker-github\Docker> 
```

Lancement du conteneur avec la nouvelle image :

```
mon-conteneur
PS C:\Users\alex\Desktop\COUR\docker\Docker-github\Docker\src> docker run -d --name mon-conteneur -p 3000:3000 --network my_network -e DB_HOST=mysql-container -e DB_USER=root -e DB_PASSWORD=mysecretpassword -e DB_NAME=mydatabase -e DB_PORT=3306 node:18-alpine
66641d53d3e87e2ba094a698dbdc3a56ed160cfc40022863c67aaef841ffca1
PS C:\Users\alex\Desktop\COUR\docker\Docker-github\Docker\src> 
```

la commande ***docker-compose up*** construira les images, démarrera les deux services et configurera les conteneurs selon les spécifications du fichier ***docker-compose.yml***.



The screenshot shows a Visual Studio Code editor with a project named 'DOCKER-GITHUB'. The left sidebar displays a file explorer with a tree structure: 'Docker' (expanded), 'Compte rendu-TP1', 'Compte rendue-TP2', 'html', 'src' (expanded), 'controller', 'models', 'routers', 'services', 'test', 'app.js', 'db.config.js', 'open-api.yaml', 'package-lock.json', 'package.json', 'server.js', 'docker-compo...', 'Dockerfile', and 'README.md'. The main editor area shows the 'docker-compose.yml' file with the following content:

```
1 version: '3'
2
3 services:
4   node:
5     build: .
6     image: mon-image-nodejs
7     container_name: mon-conteneur
8     ports:
9       - "3000:3000"
10    environment:
11      - DB_HOST=db
12      - DB_USER=root
13      - DB_PASSWORD=mysecretpassword
14      - DB_NAME=mydatabase
15      - DB_PORT=3306
16    depends_on:
17      - db
18
19 db:
20   image: mariadb:latest
21   container_name: mariadb-container
22   environment:
23     - MYSQL_ROOT_PASSWORD=mysecretpassword
24     - MYSQL_DATABASE=mydatabase
25   ports:
26     - "3306:3306"
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
no configuration file provided: not found
PS C:\Users\alex\Desktop\COUR\docker\Docker-github\Docker> docker-compose up
[+] Running 9/1
  ✓ db 8 layers [██████████] 0B/0B Pulled
[+] Running 1/2
  ✓ Network docker_default Created
  - Container mariadb-container Creating
```

Pour que l'application Node.js puisse utiliser la base de données conteneurisée dans Docker Compose, nous devons nous assurer que les services peuvent se connecter les uns aux autres.

```
Docker > docker-compose.yml
1  version: '3'
2
3  services:
4    node:
5      build: .
6      image: mon-image-nodejs
7      container_name: mon-conteneur
8      ports:
9        - "3000:3000"
10     environment:
11       - DB_HOST=db
12       - DB_USER=root
13       - DB_PASSWORD=mysecretpassword
14       - DB_NAME=mydatabase
15       - DB_PORT=3306
16     depends_on:
17       - db
18     networks:
19       - my_network
20
21   db:
22     image: mariadb:latest
23     container_name: mariadb-container
24     environment:
25       - MYSQL_ROOT_PASSWORD=mysecretpassword
26       - MYSQL_DATABASE=mydatabase
27     ports:
28       - "3306:3306"
29     networks:
30       - my_network
31
32   networks:
33     my_network:
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS docker-com

```
mariadb-container | Version: '11.3.2-MariaDB-1:11.3.2+maria-ubu2204' socket: '/run/mysqld/mysqld.sock' port: 3306 mariadb.org binary distribution
```

Voici le fichier **docker-compose.yml** configurer pour utiliser des variables d'environnement :

```
Docker > 🐳 docker-compose.yml
1  version: '3'
2  services:
3    node:
4      build: .
5      image: mon-image-nodejs
6      container_name: mon-conteneur
7      ports:
8        - "3000:3000"
9      environment:
10       - DB_HOST=${DB_HOST:-db}
11       - DB_USER=${DB_USER:-root}
12       - DB_PASSWORD=${DB_PASSWORD:-mysecretpassword}
13       - DB_NAME=${DB_NAME:-mydatabase}
14       - DB_PORT=${DB_PORT:-3306}
15      depends_on:
16        - db
17      networks:
18        - my_network
19
20    db:
21      image: mariadb:latest
22      container_name: mariadb-container
23      environment:
24        - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD:-mysecretpassword}
25        - MYSQL_DATABASE=${MYSQL_DATABASE:-mydatabase}
26      ports:
27        - "3306:3306"
28      networks:
29        - my_network
30  networks:
31    my_network:
```

Publié le port 3000 et exposer le port 3006 sans le publier :

```
Docker > docker-compose.yml
1  version: '3'
2  services:
3    node:
4      build: .
5      image: mon-image-nodejs
6      container_name: mon-conteneur
7      ports:
8        - "3000:3000" # Publie le port 3000 de l'application Node.js
9      environment:
10       - DB_HOST=db
11       - DB_USER=root
12       - DB_PASSWORD=mysecretpassword
13       - DB_NAME=mydatabase
14       - DB_PORT=3306
15      depends_on:
16        - db
17      networks:
18        - my_network
19
20    db:
21      image: mariadb:latest
22      container_name: mariadb-container
23      environment:
24        - MYSQL_ROOT_PASSWORD=mysecretpassword
25        - MYSQL_DATABASE=mydatabase
26      expose:
27        - "3306" # Expose le port 3306 de l'application DB sans le publier
28      networks:
29        - my_network
30  networks:
31    my_network:
```

Q1 : Que se passe t'il si un de mes ports publiés est déjà utilisé ?

Docker affichera un message d'erreur et ne pourra pas démarrer le conteneur.

```
Error response from daemon: driver failed programming external connectivity
d for 0.0.0.0:3307 failed: port is already allocated
```

Dans ce cas, on doit choisir un autre port disponible ou libérer le port déjà utilisé sur votre machine avant de relancer Docker Compose.

Q2 : quelle option de la commande npm install permet de n'installer que les dépendances de production ?

npm install --production

Cela installera uniquement les dépendances listées dans la section "dependencies" de votre fichier package.json.

Q2bis : pourquoi faire cela ?

L'utilisation de l'option **--production** est une bonne pratique pour garantir que votre environnement de production n'inclut que ce qui est strictement nécessaire pour exécuter votre application, tout en évitant les éléments superflus liés au développement.

Q3 : Comment peut-on analyser la sécurité d'une application comme celle-ci (dépendances & image docker)

Il y a des Outils de sécurité des dépendances : **npm audit**

De plus, il existe des outils de scanning d'images Docker tels que **Clair** ou **Trivy** pour analyser les images Docker à la recherche de vulnérabilités connues.

Q4 : Pourquoi à l'étape 6 mon container node n'arrive pas à communiquer avec ma base de données si je laisse "localhost" en hostname ?

Quand vous utilisez "localhost" comme nom d'hôte dans votre application Node.js pour se connecter à la base de données, cela signifie que votre application tente de se connecter à la base de données sur la même machine (le même conteneur Docker, dans ce cas). Cependant, dans un environnement Docker, chaque conteneur a son propre espace réseau isolé.