

# Rapport de Mini-Projet

# Conteneurisation des Applications

---

Réalisé par :

- Elias Dounas

Année universitaire : 2024-2025



## Table des matières

<b>Introduction.....</b>	<b>2</b>
<b>Partie 0.....</b>	<b>2</b>
<b>Partie 1.....</b>	<b>2</b>
<b>Partie 2.....</b>	<b>31</b>
<b>Partie 3.....</b>	<b>46</b>

# Introduction

Ce projet vise à développer une application web 3-tiers intégrant un frontend réalisé avec React, un backend développé en Java/Spring, et une base de données relationnelle (MySQL ou PostgreSQL). L'objectif est de concevoir une solution permettant de gérer des étudiants et leurs notes via des fonctionnalités interactives telles que l'ajout d'étudiants, l'affichage des listes et des détails des notes, avec des surlignages visuels basés sur les moyennes. Le projet comprend également une conteneurisation complète à l'aide de Docker et Docker Compose pour assurer la portabilité, ainsi que des déploiements avancés avec Docker Swarm et Kubernetes pour garantir la haute disponibilité, la gestion des ressources, et la scalabilité des services.

## Partie 0

Le code source, ainsi que les fichiers des autres parties, sont disponibles dans la repo suivante :

<https://github.com/eliasDounas/mini-projet-conteneurisation>

## Partie 1

1. On crée un réseau docker appelé **mon\_reseau** avec le driver bridge. On choisit ce driver car il permet la communication mutuelle entre les conteneurs ainsi que la communication avec la machine hôte.

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker network create --driver bridge mon_reseau
● da0d81078264df4ab7f5c6f934004d5b67dea71983271f90136d832434ce4f4a
PS C:\Users\Elias\OneDrive\Bureau\tp> docker network ls
● NETWORK ID      NAME      DRIVER      SCOPE
  0c1c160f521b    appnetwork  bridge      local
  f157e1de0195    bridge     bridge      local
  bc353b1de230    host       host       local
  da0d81078264    mon_reseau  bridge      local
  19541869002b    none      null      local
○ PS C:\Users\Elias\OneDrive\Bureau\tp>
```

## 2. On pull d'abord la dernière image postgres

```
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker pull postgres:latest
latest: Pulling from library/postgres
bc0965b23a04: Pull complete
002e1a8eb6f9: Pull complete
a24f300391ed: Pull complete
627f580b7ad7: Pull complete
cfb3c2203f88: Pull complete
9e592465b243: Pull complete
8d4265d09d9c: Pull complete
e3a8293e92fd: Pull complete
2cb801c39436: Pull complete
c5fdb20d8658: Pull complete
67c5fe618f0c: Pull complete
c9cdd1fe82e4: Pull complete
8f152c4aceed: Pull complete
2cd360f3b7db: Pull complete
Digest: sha256:fe4efc6901dda0d952306fd962643d8022d7bb773ffe13fe8a21551b9276e50c
Status: Downloaded newer image for postgres:latest
docker.io/library/postgres:latest
```

On crée un volume Docker pour rendre les données persistantes, ce qui garantit que les données survivent même si le conteneur est supprimé. Puis on lance le conteneur PostgreSQL avec ce volume:

```
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker volume create db_data
db_data
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker run --name etudiantsDB --network mon_reseau -e POSTGRES_USER=postgres -e POSTGRES_PASSWORD=root -e POSTGRES_DB=etudiants -v db_data:/var/lib/postgresql/data -d postgres:latest
97fed4347795e052526009887fdcd4568c8826b23e57e605a57fb6def049c6b5
○ PS C:\Users\Elias\OneDrive\Bureau\tp> █
```

```

backend > 🐳 Dockerfile > ...
 2  FROM maven:3.9.
 3  # Set the working directory inside the container
 4  WORKDIR /app
 5
 6  # Copy only the pom.xml and settings.xml for dependency resolution
 7  COPY pom.xml .
 8  COPY src ./src
 9
10 # Install dependencies and package the application
11 RUN mvn clean package -DskipTests
12
13 # Use a lightweight JRE image for the runtime environment
14 FROM eclipse-temurin:17-jre-alpine@sha256:fcf70ae7ba37872c7d1da875593321c3e90bd9a02c6b4bfde5a1260b08b8f178
15
16 # Set the working directory inside the runtime container
17 WORKDIR /app
18
19 # Copy the packaged JAR file from the build stage
20 COPY --from=builder /app/target/*.jar app.jar
21
22 # Expose the port the application will run on
23 EXPOSE 8080
24 # Run the application
25 ENTRYPOINT ["java", "-jar", "app.jar"]

```

3.

Ln 23, Col 12 Spaces: 4 UTF-8 CRLF Dockerfile ⚙ Go Live ⚡ Prettier 🔍

#### 4. Bonnes pratiques appliquées :

**Utiliser un build multi-étages** : Réduit la taille de l'image finale en excluant les outils de build.

SHA256 pinning ensures image consistency and security

**Utiliser des images officielles et légères** : Améliore la sécurité et réduit la taille.

**Spécifier une version fixe pour les images** : Évite les incompatibilités dues aux mises à jour.

**Définir un répertoire de travail (WORKDIR)** : Maintient l'organisation et évite les problèmes de chemin relatif.

**Copier les fichiers par étapes** : Optimisez le cache Docker en copiant pom.xml avant src.

**Nettoyer les dépendances non nécessaires** : Utiliser -DskipTests pour ignorer les tests en production.

**Exclure les fichiers inutiles avec .dockerignore** : Réduit la taille du contexte de build.

**Exposer uniquement les ports nécessaires** : Améliore la sécurité et facilite la gestion réseau.

**Utiliser ENTRYPPOINT pour définir le point d'entrée** : Permet de passer des arguments à la commande principale.

**Commandes spécifiques au runtime** : Utilisation de CMD au lieu de RUN pour des actions à l'exécution.

5. A- Avant de créer l'image, on crée un fichier .Dockerignore qui permet d'identifier les fichiers à ignorer pendant la copie des fichiers dans le conteneur ce qui permet la réduction de la taille de l'image créée.

```
📦 Dockerfile      📷 .Dockerignore X
backend > 📷 .Dockerignore
1   # Ignore Maven's local repository
2   .mvn
3   target
4
5   # Ignore IntelliJ IDEA files
6   .idea
7
8   # Ignore Eclipse/VS Code files
9   .project
10  .classpath
11  .settings
12  .vscode
13
14  # Ignore logs
15  *.log
16
17  # Ignore OS-generated files
18  .DS_Store
```

On crée maintenant une image appelée backend-image à partir du Dockerfile avec la commande docker build --tag backend-image .

```

● PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker build -t backend-image:1.0 .
[+] Building 655.7s (16/16) FINISHED
  => [internal] load build definition from Dockerfile
  => => transferring dockerfile: 774B
  => [internal] load metadata for docker.io/library/maven:3.8.6-eclipse-temurin-17
  => [internal] load metadata for docker.io/library/eclipse-temurin:17-jre
  => [auth] library/eclipse-temurin:pull token for registry-1.docker.io
  => [auth] library/maven:pull token for registry-1.docker.io
  => [internal] load .dockerrcignore
  => => transferring context: 2B
  => [stage-1 1/3] FROM docker.io/library/eclipse-temurin:17-jre@sha256:260e77f3c41795c73c24d8c7e5fb3 296.0s
  => => resolve docker.io/library/eclipse-temurin:17-jre@sha256:260e77f3c41795c73c24d8c7e5fb334939e333e 0.1s
  => => sha256:d6a4ff3a51b10a282cf00ef272a3084ec2aa69a242bc9d7ccf846b49d8943000 1.94kB / 1.94kB 0.0s
  => => sha256:44151ec97d36e58cb03267dfd9aae788be0785cb5fc6ae916a72c1d64b3ca8c1 5.96kB / 5.96kB 0.0s
  => => sha256:de44b265507ae44b212defcb50694d666f136b35c1090d9709068bc861bb2d64 29.75MB / 29.75MB 202.2s

```

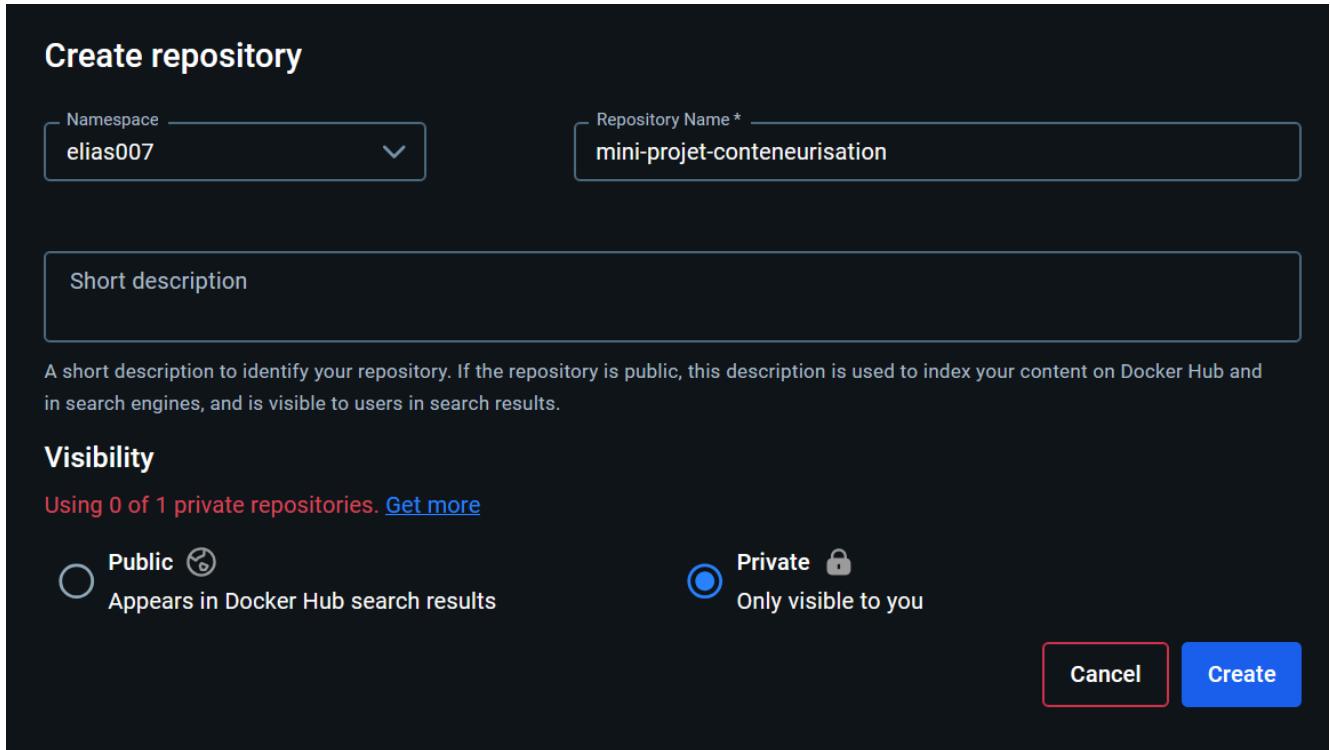
```

PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker image ls
● REPOSITORY          TAG      IMAGE ID      CREATED        SIZE
  backend-image        1.0      bff082f08ce5   3 minutes ago  314MB
  postgres             latest    810c36706d00   3 weeks ago   435MB

```

B - On peut utiliser la commande `docker scan` pour scanner les vulnérabilités. Cependant, j'ai opté pour un autre moyen qui est Docker Scout image analysis qu'on va pouvoir lancer une fois on push la repo sur Docker Hub ci-dessous.

C - Pour publier cette image dans le Docker hub, on crée d'abord un repository appelé mini-projet-conteneurisation



Par la suite on génère un Token à l'utiliser au moment de l'authentification à travers le terminal (pour

éviter d'écrire le mot de passe directement dans le terminal)

The screenshot shows the Docker web interface with the title 'docker EARLY ACCESS'. The current page is 'Personal access tokens / New access token'. The form for creating a new access token has the following fields: 'Access token description' set to 'main', 'Expiration date' set to 'None', and 'Access permissions' set to 'Read, Write, Delete'. Below the form, a note states: 'Read, Write, Delete tokens allow you to manage your repositories.' At the bottom are 'Cancel' and 'Generate' buttons.

```
PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker login -u elias007
```

● Password:

```
Login Succeeded
```

On tag maintenant notre image avec le tag username/repo:newimagename avec la commande docker tag backend-image elias007/mini-projet-conteneurisation:springboot-image avant de la publier à l'aide de la commande docker push elias007/mini-projet-conteneurisation :springboot-image

```
PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker tag backend-image:1.0 elias007/mini-projet-conteneurisation:springboot-image
```

```
PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker push elias007/mini-projet-conteneurisation:springboot-image
● ge
The push refers to repository [docker.io/elias007/mini-projet-conteneurisation]
25e50fad793a: Pushed
bd528db59361: Pushed
2aeb2db3bef1: Mounted from library/eclipse-temurin
7824d009e26c: Mounted from library/eclipse-temurin
cd729742fb88: Mounted from library/eclipse-temurin
11e7186a0470: Mounted from library/eclipse-temurin
687d50f2f6a6: Mounted from library/eclipse-temurin
springboot-image: digest: sha256:9d8ff086883ea81f686491090a7aec42c062b6cdba73c512d9ec4b3357b68a0b size: 1786
```

## Remarque:

J'ai eu des soucis avec la commande docker push: "denied: requested access to the resource is denied", après quelque recherche, j'ai opté pour une nouvelle façon de login USING WEB BASED LOGIN qui a permis de résoudre le problème

```
PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker login

● USING WEB BASED LOGIN
To sign in with credentials on the command line, use 'docker login -u <username>'

Your one-time device confirmation code is: [REDACTED]
Press ENTER to open your browser or submit your device code here: https://login.docker.com/activate

Waiting for authentication in the browser...

Login Succeeded
```

## Remarque 2:

The screenshot shows the Docker Hub interface for analyzing the image `elias007/mini-projet-conteneurisation:springboot-image`. The top navigation bar includes links for "View recommended base image fixes" and "Give feedback". The main content area displays the following information:

- Image details:** elias007/mini-projet-conteneurisation:springboot-image, MANIFEST DIGEST: sha256:9d8ff086883ea81f686491090a7aec42c062b6cd8a73c512d9ec4b3357b68a0b.
- Metrics:** OS/ARCH: linux/amd64, COMPRESSED SIZE: 133.16 MB, LAST PUSHED: an hour ago by elias007, TYPE: Image, VULNERABILITIES: 0 0 4 8 0, MANIFEST DIGEST: sha256:9d8ff0868...
- Vulnerabilities:** 12 vulnerabilities found, including 4 fixable. A search bar allows filtering by package or CVE name, with options to "Fixable" and "Show excepted".
- Layers:** 19 layers listed, including ARG RELEASE, ARG LAUNCHPAD\_BUILD\_ARCH, LABEL org.opencontainers.image.ref.name=ubuntu, and LABEL org.opencontainers.image.version=24.04.
- Layer Details:** A table on the right provides a breakdown of vulnerabilities for each layer, showing the number of vulnerabilities for each package (ubuntu/pam, ubuntu/krb5, ubuntu/waet) across different severity levels (0, 1, 2, 3).

Une fois l'image dans Docker Hub, on peut activer docker scout image analysis qui va permettre d'analyser les vulnérabilités possibles et de proposer des optimisations. Dans notre cas, il a trouvé quelques vulnérabilités de moyenne sévérité.

D - On instancie maintenant l'image en créant un conteneur (nommé backend) s'exécutant en local et partageant le même réseau avec le conteneur de la base de données postgres. On spécifie un

mapping 8080 :8080 pour ce conteneur avec l'option -p

```
PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker run --name backend-springboot --network mon_reseau -p 8080:80 80 backend2-image
```

:: Spring Boot :: (v3.4.0)

```
2024-12-13T22:18:11.778Z INFO 1 --- [Notes Etudiants] [           main] c.n.NotesEtudiantsApplication
: Starting NotesEtudiantsApplication v0.0.1-SNAPSHOT using Java 17.0.13 with PID 1 (/app/app.jar started by root i
n /app)
2024-12-13T22:18:11.792Z INFO 1 --- [Notes Etudiants] [           main] c.n.NotesEtudiantsApplication
: No active profile set, falling back to 1 default profile: "default"
2024-12-13T22:18:12.584Z INFO 1 --- [Notes Etudiants] [           main] .s.d.r.c.RepositoryConfigurationDelegate
: Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2024-12-13T22:18:12.684Z INFO 1 --- [Notes Etudiants] [           main] .s.d.r.c.RepositoryConfigurationDelegate
: Finished Spring Data repository scanning in 81 ms. Found 2 JPA repository interfaces.
2024-12-13T22:18:13.302Z INFO 1 --- [Notes Etudiants] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer
: Tomcat initialized with port 8080 (http)
2024-12-13T22:18:13.319Z INFO 1 --- [Notes Etudiants] [           main] o.apache.catalina.core.StandardService
: Starting service [Tomcat]
2024-12-13T22:18:13.320Z INFO 1 --- [Notes Etudiants] [           main] o.apache.catalina.core.StandardEngine
: Starting Servlet engine: [Apache Tomcat/10.1.33]
2024-12-13T22:18:13.358Z INFO 1 --- [Notes Etudiants] [           main] o.a.c.c.C.[Tomcat].[localhost].[/]
: Initializing Spring embedded WebApplicationContext
```

## E - On inspecte le conteneur

```
● PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker inspect backend-springboot
[
  {
    "Id": "2ec35e18091e749565ef4c1b10d87e036fe96a9575597ee5b9351ea81a64289b",
    "Created": "2024-12-13T22:18:10.10735157Z",
    "Path": "java",
    "Args": [
      "-jar",
      "app.jar"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 3171,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2024-12-13T22:20:55.254218536Z",
      "FinishedAt": "2024-12-13T22:20:49.674597085Z"
    },
    "Image": "sha256:66d2e7bf7099a9e4359936efdf472fe8b074c577df1eb8948750021eaf4c0f46",
    "ResolvConfPath": "/var/lib/docker/containers/2ec35e18091e749565ef4c1b10d87e036fe96a9575597ee5b9351ea81a64289b/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/2ec35e18091e749565ef4c1b10d87e036fe96a9575597ee5b9351ea81a64289b/hostname",
    "HostsPath": "/var/lib/docker/containers/2ec35e18091e749565ef4c1b10d87e036fe96a9575597ee5b9351ea81a64289b/
```

## F - Logs:

6. On crée d'abord notre fichier Dockerfile avec les bonnes pratiques, un fichier .Dockerignore et un dossier nginx ou dedans on ajoute notre fichier nginx.config

```
frontend > ⚡ .Dockerignore
```

1	node_modules
2	npm-debug.log
3	Dockerfile
4	.dockerignore
5	.git
6	.gitignore
7	env

```
frontend > ⚙ nginx.config
 1 server {
 2     listen 80;
 3
 4     server_name localhost;
 5
 6     root /usr/share/nginx/html;
 7     index index.html;
 8
 9     location / {
10         # Serveur les fichiers statiques
11         try_files $uri /index.html;
12     }
13
14     error_page 404 /index.html;
15 }
16
```

```
frontend > 🛠 Dockerfile > ...
 1 # Build Stage
 2 FROM node:18-alpine AS build
 3 RUN mkdir -p /usr/share/nginx/html
 4 WORKDIR /app
 5 COPY package*.json ./
 6 RUN npm install
 7 COPY ..
 8 RUN npm run build
 9
10 # Production Stage
11 FROM nginx:stable-alpine3.20-slim
12 COPY --from=build /app/dist /usr/share/nginx/html
13 EXPOSE 80
14 CMD ["nginx", "-g", "daemon off;"]
15
```

Les bonnes pratiques utilisées lors de la création de ce dockerfile:

**Utilisation des multi-stage builds** : Permet de séparer la phase de construction et de production pour réduire la taille de l'image finale.

**Utilisation d'images de base légères** : Réduit la taille des images en utilisant des variantes Alpine pour Node.js et NGINX.

**Définition du répertoire de travail avec WORKDIR** : Simplifie la gestion des chemins et améliore la lisibilité.

**Utilisation de package\*. json pour installer les dépendances** : Assure que seules les dépendances nécessaires sont installées.

**Exposition explicite du port avec EXPOSE** : Clarifie quel port est utilisé par le conteneur pour le trafic réseau.

**Utilisation de CMD pour démarrer le service** : Garantit que le conteneur reste actif en exécutant NGINX en mode non-démon.

**Suppression des fichiers inutiles dans l'image finale** : Seules les ressources essentielles sont copiées dans l'étape de production grâce à COPY --from=build.

Ensuite, on crée une image à partir de ce dockerfile

```
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend> docker build --tag frontend-image .
● [+] Building 3.7s (17/17) FINISHED                                            docker:desktop-linux
=> [internal] load build definition from Dockerfile                           0.1s
=> => transferring dockerfile: 358B                                         0.0s
=> [internal] load metadata for docker.io/library/nginx:stable-alpine3.20-slim   1.3s
=> [internal] load metadata for docker.io/library/node:18-alpine                 1.4s
=> [auth] library/node:pull token for registry-1.docker.io                      0.0s
=> [auth] library/nginx:pull token for registry-1.docker.io                      0.0s
=> [internal] load .dockerrcignore                                              0.0s
=> => transferring context: 2B                                                 0.0s
=> [stage-1 1/2] FROM docker.io/library/nginx:stable-alpine3.20-slim@sha256:c13d84b525bee78b8761 0.0s
=> [internal] load build context                                               2.0s
=> => transferring context: 714.45kB                                         1.7s
=> [build 1/7] FROM docker.io/library/node:18-alpine@sha256:6eb9c3d9bd191bd2cc6ce7ec3d5ec4c21276 0.0s
=> CACHED [build 2/7] RUN mkdir -p /usr/share/nginx/html                         0.0s
=> CACHED [build 3/7] WORKDIR /app                                              0.0s
=> CACHED [build 4/7] COPY package*.json ./                                       0.0s
=> CACHED [build 5/7] RUN npm install                                           0.0s
```

Après l'avoir scanné des potentielles vulnérabilités, on la tag puis on la publie au docker hub

```
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend> docker tag frontend-image elias007/mini-projet-conteneurisation:react-image
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend> docker push elias007/mini-projet-conteneurisation:react-image
The push refers to repository [docker.io/elias007/mini-projet-conteneurisation]
5e301605250f: Pushed
bdfaf1e3a3b6: Mounted from library/nginx
2a9ed7a63dd9: Mounted from library/nginx
5203171b6392: Mounted from library/nginx
adae2ffee4c3: Mounted from library/nginx
34dcab1046f4: Mounted from library/nginx
38efb9d2ae82: Mounted from library/nginx
75654b8eee9bd: Mounted from library/nginx
react-image: digest: sha256:c30086e39bb1d8851b0060d66dc47405cc40b022ac73a4570ac1cb0a829354c6 size: 1985
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend>
```

The screenshot shows the Docker Hub interface for the image `elias007/mini-projet-conteneurisation:react-image`. At the top, there's a manifest digest link and a 'View recommended base image fixes' button. Below that, it says 'Analyzed by'. The main card displays basic info: OS/ARCH (linux/amd64), COMPRESSED SIZE (7.36 MB), LAST PUSHED (an hour ago by `elias007`), TYPE (Image), and VULNERABILITIES (No Vulnerabilities Found). A manifest digest link is also present. The 'Vulnerabilities (0)' tab is selected. The 'Image hierarchy' section shows the image structure: FROM alpine:05a56cc5acbd9c9c5b7ba5ec88d866a0ddc76b586828f8288d29c57ccaa15a..., FROM nginx:1.26-alpine-slim, 1.26-alpine3.20-slim, 1.26.2-alpine-slim, 1.26.2-alpine3.20-slim, and ALL elias007/mini-projet-conteneurisation:react-image. The 'Layers (19)' section lists the layers with their respective file paths, sizes, and status (all green checkmarks). A note says 'No vulnerabilities were detected in this image'.

Une fois l'image dans Docker Hub, on peut activer docker scout image analysis qui va permettre d'analyser les vulnérabilités possibles et de proposer des optimisations. Dans notre cas, tout est OK!

On l'instancie par un conteneur nommé `frontend-etudiants` qui partage le même réseau avec les autres conteneurs. On fait le mapping du port 3000:80

```
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend> docker run -d --name frontend-etudiants --network mon_reseau -p 3000:80 frontend-image
7102bab8e63e361e30e2addbf13fe6229b1d299d4b0b8ca53378802ab98fe44
```

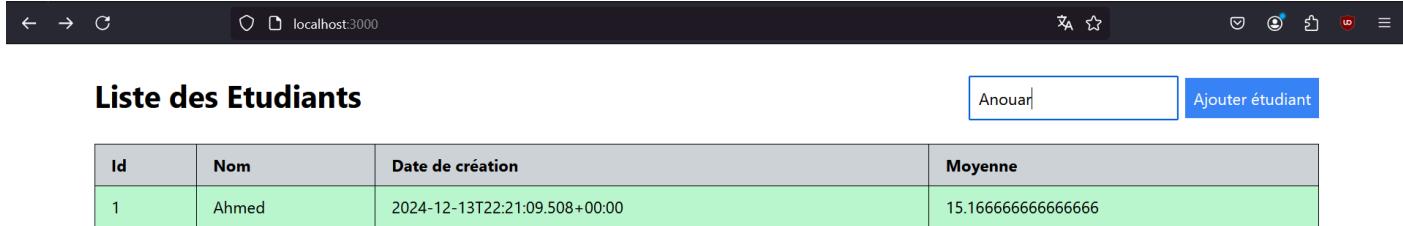
On inspecte le conteneur:

```
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend> docker inspect frontend-etudiants
[
  {
    "Id": "7102bab8e63e361e30e2addbf13fe6229b1d299d4b0b8ca53378802ab98fe44",
    "Created": "2024-12-13T21:34:58.507052379Z",
    "Path": "/docker-entrypoint.sh",
    "Args": [
      "nginx",
      "-g",
      "daemon off;"
    ],
    "State": {
      "Status": "running",
      "Running": true,
      "Paused": false,
      "Restarting": false,
      "OOMKilled": false,
      "Dead": false,
      "Pid": 605,
      "ExitCode": 0,
      "Error": "",
      "StartedAt": "2024-12-13T21:34:59.185960043Z",
      "FinishedAt": "0001-01-01T00:00:00Z"
    },
    "Image": "sha256:28573b1bde5b0cba939a656d26a9ae1439060bd4fe43cdfb5505c0d30e70c050",
    "ResolvConfPath": "/var/lib/docker/containers/7102bab8e63e361e30e2addbf13fe6229b1d299d4b0b8ca53
378802ab98fe44/resolv.conf",
    "HostnamePath": "/var/lib/docker/containers/7102bab8e63e361e30e2addbf13fe6229b1d299d4b0b8ca5337
```

On affiche les log

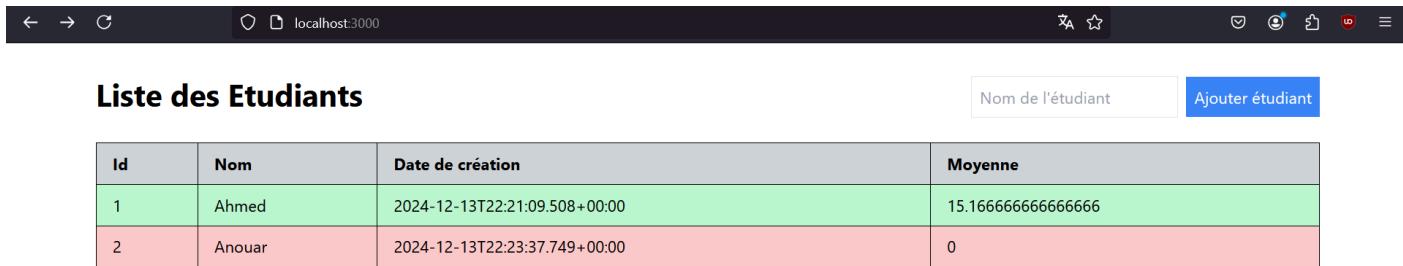
```
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend> docker logs frontend-etudiants
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2024/12/13 21:35:00 [notice] 1#1: using the "epoll" event method
2024/12/13 21:35:00 [notice] 1#1: nginx/1.26.2
2024/12/13 21:35:00 [notice] 1#1: built by gcc 13.2.1 20240309 (Alpine 13.2.1_git20240309)
2024/12/13 21:35:00 [notice] 1#1: OS: Linux 5.15.153.1-microsoft-standard-WSL2
2024/12/13 21:35:00 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
2024/12/13 21:35:00 [notice] 1#1: start worker processes
2024/12/13 21:35:00 [notice] 1#1: start worker process 31
2024/12/13 21:35:00 [notice] 1#1: start worker process 32
2024/12/13 21:35:00 [notice] 1#1: start worker process 33
2024/12/13 21:35:00 [notice] 1#1: start worker process 34
2024/12/13 21:35:00 [notice] 1#1: start worker process 35
2024/12/13 21:35:00 [notice] 1#1: start worker process 36
2024/12/13 21:35:00 [notice] 1#1: start worker process 37
2024/12/13 21:35:00 [notice] 1#1: start worker process 38
2024/12/13 21:35:00 [notice] 1#1: start worker process 39
2024/12/13 21:35:00 [notice] 1#1: start worker process 40
2024/12/13 21:35:00 [notice] 1#1: start worker process 41
2024/12/13 21:35:00 [notice] 1#1: start worker process 42
PS C:\Users\Elias\OneDrive\Bureau\tp\frontend>
```

7. On s'assure maintenant que l'application fonctionne bien en essayant d'ajouter un étudiant



A screenshot of a web browser window titled "Liste des Etudiants". The address bar shows "localhost:3000". The page contains a table with four columns: "Id", "Nom", "Date de création", and "Moyenne". There is one row with data: Id 1, Nom Ahmed, Date de création 2024-12-13T22:21:09.508+00:00, and Moyenne 15.166666666666666. A search input field contains "Anouar" and a blue button labeled "Ajouter étudiant" is visible.

Id	Nom	Date de création	Moyenne
1	Ahmed	2024-12-13T22:21:09.508+00:00	15.166666666666666



A screenshot of a web browser window titled "Liste des Etudiants". The address bar shows "localhost:3000". The page contains a table with four columns: "Id", "Nom", "Date de création", and "Moyenne". There are two rows of data: one for "Ahmed" with a green background and one for "Anouar" with a pink background. The "Nom de l'étudiant" search input field is empty, and the "Ajouter étudiant" button is blue.

Id	Nom	Date de création	Moyenne
1	Ahmed	2024-12-13T22:21:09.508+00:00	15.166666666666666
2	Anouar	2024-12-13T22:23:37.749+00:00	0

On teste maintenant l'ajout d'une nouvelle notes pour un étudiant:

A screenshot of a web browser window. The address bar shows the URL `localhost:3000/etudiants/1`. The main content area has a title **Liste des notes de Ahmed**. Below the title is a horizontal row of input fields: "Nom du cours" (Chimie), "Note" (12.25), and a blue button "Ajouter la note". Below this row is a table with three rows, each representing a course and its note value.

Nom du cours	Note
Chimie	12.25
Math	14.25
Physique	19

A screenshot of a web browser window, identical to the one above but with a new row added to the table. The table now has four rows, showing the newly added note for Chimie.

Nom du cours	Note
Chimie	12.25
Math	14.25
Physique	19

8. On arrête et supprime les conteneurs en exécutions

```
PS C:\Users\Elias\OneDrive\Bureau\tp\backend> docker rm -f backend-springboot postgres frontend-etudiants  
backend-springboot  
postgres  
frontend-etudiants  
PS C:\Users\Elias\OneDrive\Bureau\tp\backend>
```

9. On redéploie l'application en utilisant le docker-compose :

a-

```
docker-compose.yml  
1  services:  
2    postgres:  
3      image: postgres:latest  
4      container_name: postgres  
5      environment:  
6        - POSTGRES_DB=etudiants  
7        - POSTGRES_USER=postgres  
8        - POSTGRES_PASSWORD=root  
9      ports:  
10     - "5432:5432"  
11     volumes:  
12     - db_data:/var/lib/postgresql/data  
13     networks:  
14     - mon_reseau  
15  
16   frontend:  
17     image: elias007/mini-projet-conteneurisation:react-image  
18     container_name: frontend  
19     ports:  
20     - "3000:80"  
21     networks:  
22     - mon_reseau  
23  
24   backend:  
25     image: elias007/mini-projet-conteneurisation:springboot-image  
26     container_name: backend
```

```
ESHOW docker-compose.yml
 1   services:
16     frontend:
17       ...
21     networks:
22       - mon_reseau
23
24   backend:
25     image: elias007/mini-projet-conteneurisation:springboot-image
26     container_name: backend
27     ports:
28       - "8080:8080"
29     networks:
30       - mon_reseau
31
32   volumes:
33     db_data:
34       driver: local
35
36   networks:
37     mon_reseau:
38       driver: bridge
39
```

b -

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker-compose up -d
● [+] Running 3/3
  ✓ Container postgres    Started
  ✓ Container backend     Started
  ✓ Container frontend    Started
○ PS C:\Users\Elias\OneDrive\Bureau\tp>
```

c - L'application fonctionne correctement



## Liste des notes de Ayman

Nom du cours	Note
<input type="text"/>	<input type="text"/>

Ajouter la note

Nom du cours	Note
Math	16

## 10 - Arrêter et supprimer les conteneurs en cours

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker ps
● CONTAINER ID   IMAGE                               COMMAND             CREATED            STATUS              NAMES
      STATUS          PORTS     NAMES
13654f2caf7a   postgres:latest                      "docker-entrypoint.s..."   3 minutes ago
○ Up 3 minutes   0.0.0.0:5432->5432/tcp    postgres
8de9aa4efe9a   elias007/mini-projet-conteneurisation:springboot-image   "java -jar app.jar"   3 minutes ago
○ Up 3 minutes   0.0.0.0:8080->8080/tcp    backend
6cc779c05fd2   elias007/mini-projet-conteneurisation:react-image        "/docker-entrypoint...." 3 minutes ago
○ Up 3 minutes   0.0.0.0:3000->80/tcp      frontend
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker stop $(docker ps -q)
13654f2caf7a
8de9aa4efe9a
6cc779c05fd2
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker rm $(docker ps -a -q)
13654f2caf7a
8de9aa4efe9a
6cc779c05fd2
○ PS C:\Users\Elias\OneDrive\Bureau\tp>
```

## Supprimer les images Docker

```
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker rmi -f $(docker images -q)
Untagged: elias007/mini-projet-conteneurisation:springboot-image
Untagged: elias007/mini-projet-conteneurisation@sha256:ce254aeeb85effbceff773faf5a967ae92d8bcb2e3c52b71c2b6b3
baa36f946d
Untagged: backend2-image:latest
Deleted: sha256:66d2e7bf7099a9e4359936efdf472fe8b074c577df1eb8948750021eaf4c0f46
Untagged: elias007/mini-projet-conteneurisation:react-image
Untagged: elias007/mini-projet-conteneurisation@sha256:c30086e39bb1d8851b0060d66dc47405cc40b022ac73a4570ac1cb
0a829354c6
Untagged: frontend-image:latest
Deleted: sha256:28573b1bde5b0cba939a656d26a9ae1439060bd4fe43cdfb5505c0d30e70c050
Error response from daemon: No such image: 66d2e7bf7099:latest
Error response from daemon: No such image: 28573b1bde5b:latest
○ PS C:\Users\Elias\OneDrive\Bureau\tp> █
```

11 - Un registre privé est lui-même un conteneur qui permet de stocker des images. Pour le créer, il suffit d'instancier l'image registry. On crée donc un registre privé appelé private-registry

```
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker run -d -p 5000:5000 --name private-registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
dc0decf4841d: Pull complete
6cb0aa443e23: Pull complete
813676e291ef: Pull complete
dc2fb7dcec61: Pull complete
916205650bfe: Pull complete
Digest: sha256:543dade69668e02e5768d7ea2b0aa4fae6aa7384c9a5a8dbecc2be5136079ddb
Status: Downloaded newer image for registry:2
4162a8707e0adbddd9ec46f9b089c26d215e7e682f777c174c53855c38d74fc5
█
```

a - On recrée les images supprimées, on les tag par localhost :5000/registry-image-name puis on les push au registre

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker tag elias007/mini-projet-conteneurisation:react-image localhost:5000/registry-frontend-image
PS C:\Users\Elias\OneDrive\Bureau\tp> docker tag elias007/mini-projet-conteneurisation:springboot-image localhost:5000/registry-backend-image
PS C:\Users\Elias\OneDrive\Bureau\tp> docker tag postgres:latest localhost:5000/registry-postgres
█
```

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker push localhost:5000/registry-frontend-image
Using default tag: latest
● The push refers to repository [localhost:5000/registry-frontend-image]
5e301605250f: Pushed
bdfaf1e3a3b6: Pushed
2a9ed7a63dd9: Pushed
5203171b6392: Pushed
adae2ffee4c3: Pushed
34dcab1046f4: Pushed
38efb9d2ae82: Pushed
75654b8eeebd: Pushed
latest: digest: sha256:c30086e39bb1d8851b0060d66dc47405cc40b022ac73a4570ac1cb0a829354c6 size: 1985
█
```

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker push localhost:5000/registry-backend-image
Using default tag: latest
The push refers to repository [localhost:5000/registry-backend-image]
789d5265492f: Pushed
1144d2f5b330: Pushed
8fba85eb2e69: Pushed
21cf7d294a3a: Pushed
a5729e07101c: Pushed
dfc91b7f7620: Pushed
75654b8eeeebd: Mounted from registry-frontend-image
latest: digest: sha256:ce254aeeb85effbceff773faf5a967ae92d8bcb2e3c52b71c2b6b3baa36f946d size: 1785
PS C:\Users\Elias\OneDrive\Bureau\tp> docker push localhost:5000/registry-postgres
Using default tag: latest
The push refers to repository [localhost:5000/registry-postgres]
97d0da1ce974: Pushed
0b8ece3d93b7: Pushed
f1f11505df06: Pushed
88e07dfeb8ef: Pushed
bf0c9622ebbe: Pushed
4d96fc7cd897: Pushed
7b9b519c5526: Pushed
7635ed1a550f: Pushed
9d9b007dba3e: Pushed
4759ed09c338: Pushed
5e84af0b2f05: Pushed
2ee17811b681: Pushed
37430cc3024b: Pushed
c0f1022b22a9: Pushed
latest: digest: sha256:d6322be4dc8b238e0636f1a934c52078cfa1a6bfaa6818df0e12e3eb4166640d size: 3247
```

b- Pour visualiser via une UI les images contenues dans ce registre, on crée un autre conteneur qui s'occupe de cette mission. Mais avant, il faut que les deux conteneurs soient dans le même réseau,

c'est pour cela qu' on associe d'abord le registre au réseau mon\_reseau.

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker network connect mon_reseau privaye-registry
PS C:\Users\Elias\OneDrive\Bureau\tp> docker inspect privaye-registry
● [{"Id": "4162a8707e0adbddd9ec46f9b089c26d215e7e682f777c174c53855c38d74fc5", "Created": "2024-12-14T22:06:56.169269257Z", "Path": "/entrypoint.sh", "Args": ["/etc/docker/registry/config.yml"], "State": { "Status": "running", "Running": true, "Paused": false, "Restarting": false, "OOMKilled": false, "Dead": false, "Pid": 527, "ExitCode": 0, "Error": "", "StartedAt": "2024-12-14T22:06:56.678906763Z", "FinishedAt": "0001-01-01T00:00:00Z" }, "Image": "sha256:c18a86d35e983225b84aae3bb15c84bd9e47506dd0102975c0ed97958703bb73", "ResolvConfPath": "/var/lib/docker/containers/4162a8707e0adbddd9ec46f9b089c26d215e7e682f777c174c53855c38d74fc5/resolv.conf", "HostnamePath": "/var/lib/docker/containers/4162a8707e0adbddd9ec46f9b089c26d215e7e682f777c174c53855c38d74fc5/h_
```

On instancie maintenant l'image konradkleine/docker-registry-frontend:v2 pour créer le conteneur private-registry-ui. Ce registre qu'on met aussi dans le network mon\_reseau et qu'on lui spécifie le registre à visualiser ainsi que son port.

```
● PS C:\Users\Elias\OneDrive\Bureau\tp> docker run -d --name private-registry-ui --network mon_reseau -e ENV_DOCKER_REGISTRY_HOST=privaye-registry -e ENV_DOCKER_REGISTRY_PORT=5000 -p 8082:80 konradkleine/docker-registry-frontend:v2
Unable to find image 'konradkleine/docker-registry-frontend:v2' locally
v2: Pulling from konradkleine/docker-registry-frontend
85b1f47fbfa49: Pull complete
e3c64813de17: Pull complete
6e61107884ac: Pull complete
411f14e0e0fd: Pull complete
987d1071cd71: Pull complete
95913db6ef30: Pull complete
1eb7ee3fbde2: Pull complete
9b6f26b1b1a1: Pull complete
daa6941a3108: Pull complete
86cc842193a6: Pull complete
024ab6890532: Pull complete
af9b7d0cb338: Pull complete
02f33fb0dcad: Pull complete
e8275670ee05: Pull complete
1c1a56903b01: Pull complete
afc4e94602b9: Pull complete
df1a95efa681: Pull complete
```

PS C:\Users\Elias\OneDrive\Bureau\tp> docker ps -a		COMMAND	CREATED	STATUS	PO
● CONTAINER ID	IMAGE	NAMES			
a9780ea61581	konradkleine/docker-registry-frontend:v2	/bin/sh -c \$START_S..."	42 seconds ago	Up 38 seconds	44
3/tcp, 0.0.0.0:8082->80/tcp	private-registry-ui				
4162a8707e0a	registry:2	"/entrypoint.sh /etc..."	24 minutes ago	Up 24 minutes	0.
0.0.0.0:5000->5000/tcp	privaye-registry				
○ PS C:\Users\Elias\OneDrive\Bureau\tp>					

On accède maintenant à ce conteneur via le port 8082 où on trouve le UI

Docker Registry Frontend

You are here: Home

# Welcome, to the Docker registry!

Proudly serving your Docker images.

[Browse repositories](#)

[Report a bug](#) [Fork me on GitHub](#) [About](#) This is git revision: acd3f1f

On clique sur Browse repositories pour voir les images disponibles dans l'ensemble des registres associés au conteneur (dans ce cas il y'a un seul)

The screenshot shows a web browser window displaying the Docker Registry Frontend at [localhost:3082/repositories/20](http://localhost:3082/repositories/20). The page title is "Docker Registry Frontend". A breadcrumb navigation bar indicates the user is at "Home / Repositories". The main heading is "Repositories". A search bar with placeholder text "Filter repositories on this page" and a button labeled "Repositories (3/3)" are visible. Three repository entries are listed:

- registry-backend-image**: One item.
- registry-frontend-image**: One item.
- registry-postgres**: One item.

Pagination controls at the bottom include "First page" (left arrow), a dropdown for "page: 20", and "Next" (right arrow). Footer links include "Report a bug", "Fork me on GitHub", "About", and a git revision message "This is git revision: acd3ff1".

12 - On change le fichier docker-compose pour s'adapter au nouvelle image à tirer

```
docker-compose.yml
1   services:
2     postgres:
3       image: localhost:5000/registry-postgres
4       container_name: postgres
5       environment:
6         - POSTGRES_DB=etudiants
7         - POSTGRES_USER=postgres
8         - POSTGRES_PASSWORD=root
9       ports:
10      - "5432:5432"
11     volumes:
12     - db_data:/var/lib/postgresql/data
13     networks:
14     - mon_reseau
15
16   frontend:
17     image: localhost:5000/registry-frontend-image
18     container_name: frontend
19     ports:
20     - "3000:80"
21     networks:
22     - mon_reseau
23
24   backend:
25     image: localhost:5000/registry-backend-image
26     container_name: backend
```

```

○ PS C:\Users\Elias\OneDrive\Bureau\tp> docker compose up -d
[+] Running 3/3
  ✓ Container backend    Started
  ✓ Container postgres   Started
[+] Running 3/3
  ✓ Container backend    Started
  ✓ Container postgres   Started
  ✓ Container backend    Started
  ✓ Container postgres   Started
  ✓ Container postgres   Started
  ✓ Container frontend   Started
PS C:\Users\Elias\OneDrive\Bureau\tp>

```

Pour s'assurer que la création des conteneurs a été faite à base des image du registre on les consulte dans docker desktop

The screenshot shows the Docker Desktop interface with the 'Containers' tab selected. A single container named 'frontend' is listed. The container's status is 'Running (2 minutes ago)'. The 'Inspect' tab is active, showing the container's configuration details. The configuration file (docker-compose.yml) includes the following configuration:

```

164 v      "Cmd": [
165        "nginx",
166        "-g",
167        "daemon off;";
168    ],
169    "Image": "localhost:5000/registry-frontend-image",
170    "Volumes": null,
171    "WorkingDir": "/",
172    "Entrypoint": [
173        "/docker-entrypoint.sh"
174    ],
175    "OnBuild": null,
176    "Labels": {
177        "com.docker.compose.config-hash": "06a82fa5c814991a85b91b60a7f6f619b476602a91774f27eb09b8208733cd00",
178        "com.docker.compose.container-number": "1",
179        "com.docker.compose.depends_on": "",
180        "com.docker.compose.image": "sha256:28573b1bde5b0cba939a656d26a9ae1439060bd4fe43cdfb5505c0d30e70c050",
181        "com.docker.compose.oneoff": "False",
182        "com.docker.compose.project": "tp",
183        "com.docker.compose.project.config_files": "C:\\\\Users\\\\Elias\\\\OneDrive\\\\Bureau\\\\tp\\\\docker-compose.yml",
184        "com.docker.compose.project.working_dir": "C:\\\\Users\\\\Elias\\\\OneDrive\\\\Bureau\\\\tp",
185        "com.docker.compose.service": "frontend",

```

[Containers](#) / postgres

### postgres

 1e58c589ba03 ⚡  810c36706d00 (was registry-postgres:<none>)  
5432:5432 ⚡

STATUS  
Running (2 minutes ago)

[Logs](#) **Inspect** [Bind mounts](#) [Exec](#) [Files](#) [Stats](#)

[Platform](#) [Cmd](#) [State](#) [Image](#) [PortBindings](#) [Runtime](#) [Mounts](#) [Volumes](#) [Env](#) [Labels](#) [Networks](#) [Raw JSON](#)

```

171      },
172      "Tty": false,
173      "OpenStdin": false,
174      "StdinOnce": false,
175      "Env": [
176          "POSTGRES_PASSWORD=root",
177          "POSTGRES_DB=etudiants",
178          "POSTGRES_USER=postgres",
179          "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/usr/lib/postgresql/17/bin",
180          "GOSU_VERSION=1.17",
181          "LANG=en_US.utf8",
182          "PG_MAJOR=17",
183          "PG_VERSION=17.2-1.pgdg120+1",
184          "PGDATA=/var/lib/postgresql/data"
185      ],
186      "Cmd": [
187          "postgres"
188      ],
189      "Image": "localhost:5000/registry-postgres",
190      "Volumes": [
191          "/var/lib/postgresql/data"
192      ]
  
```

[image](#) [next](#) [previous](#) [all](#)  match case  regexp  by word

---

[Containers](#) / backend

### backend

 3dd10ebd2cb6 ⚡  66d2e7bf7099 (was registry-backend-image:<none>)  
8080:8080 ⚡

STATUS  
Running (3 minutes ago)

[Logs](#) **Inspect** [Bind mounts](#) [Exec](#) [Files](#) [Stats](#)

[Platform](#) [Cmd](#) [State](#) [Image](#) [PortBindings](#) [Runtime](#) [Mounts](#) [Volumes](#) [Env](#) [Labels](#) [Networks](#) [Raw JSON](#)

```

148      "AttachStdin": false,
149      "AttachStdout": true,
150      "AttachStderr": true,
151      "ExposedPorts": {
152          "8080/tcp": {}
153      },
154      "Tty": false,
155      "OpenStdin": false,
156      "StdinOnce": false,
157      "Env": [
158          "PATH=/opt/java/openjdk/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
159          "JAVA_HOME=/opt/java/openjdk",
160          "LANG=en_US.UTF-8",
161          "LANGUAGE=en_US:en",
162          "LC_ALL=en_US.UTF-8",
163          "JAVA_VERSION=jdk-17.0.13+11"
164      ],
165      "Cmd": null,
166      "Image": "localhost:5000/registry-backend-image",
167      "Volumes": null,
168      "Mounts": null
  
```

[image](#) [next](#) [previous](#) [all](#)  match case  regexp  by word

On consulte l'application par le navigateur pour s'assurer de son fonctionnement

The screenshot shows a web browser window with the URL `localhost:3000/etudiants/2`. The page title is "Liste des notes de Ayman". On the right, there is a form with fields for "Nom du cours" (Physical) and "Note" (9), with a blue "Ajouter la note" button. Below the form is a table with two rows:

Nom du cours	Note
Math	16

The screenshot shows a web browser window with the URL `localhost:3000/etudiants/2`. The page title is "Liste des notes de Ayman". On the right, there is a form with fields for "Nom du cours" and "Note", with a blue "Ajouter la note" button. Below the form is a table with three rows:

Nom du cours	Note
Math	16
Physique	9

Tout fonctionne, c'est OK!

## Partie 2

1. On crée notre fichier docker-compose.yml compatible avec Docker Swarm

```
📦 docker-compose.yml
1   services:
2     frontend:
3       image: elias007/mini-projet-conteneurisation:react-image
4       ports:
5         - "80:3000"
6       deploy:
7         replicas: 2
8         resources:
9           limits:
10          cpus: "0.5"
11          memory: "256M"
12         restart_policy:
13           condition: on-failure
14         networks:
15           - app-network
16
17     backend:
18       image: elias007/mini-projet-conteneurisation:springboot-image
```

```
17   backend:
18     image: elias007/mini-projet-conteneurisation:springboot-image
19   ports:
20     - "5000:5000"
21   deploy:
22     replicas: 2
23     resources:
24       limits:
25         cpus: "0.5"
26         memory: "256M"
27     restart_policy:
28       condition: on-failure
29   networks:
30     - app-network
31
```

Le réseau `overlay` permet aux services de communiquer entre eux dans le cluster.

```
32   postgres:
33     image: postgres:latest
34   ports:
35     - "5432:5432"
36   deploy:
37     replicas: 2
38     resources:
39       limits:
40         cpus: "0.5"
41         memory: "512M"
42     restart_policy:
43       condition: on-failure
44   networks:
45     - app-network
46
47   networks:
48     app-network:
49       driver: overlay
50
```

2. Dans la section `deploy` de chaque service, le champ `replicas: 2` spécifie le nombre de réplicas pour assurer la haute disponibilité. Swarm gère automatiquement l'équilibrage de charge.
3. La section `deploy.resources.limits` est utilisée pour contrôler les ressources CPU et mémoire pour chaque instance de service. Par exemple :

```
deploy:  
  replicas: 2  
  resources:  
    limits:  
      cpus: "0.5"  
      memory: "512M"
```

4. La clé `restart_policy` permet de spécifier les conditions de redémarrage des conteneurs en cas d'échec

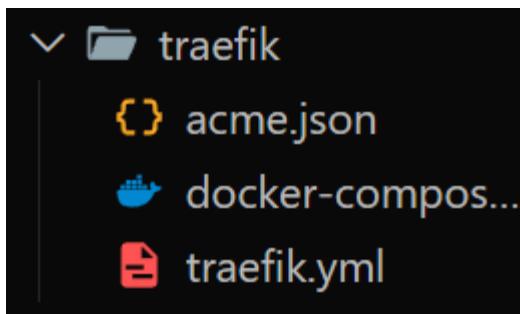
```
  restart_policy:  
    condition: on-failure  
  networks:
```

## 5. Configuration d'un accès avec un Reverse Proxy Traefik

Traefik est un reverse proxy moderne et un load balancer qui gère la distribution du trafic HTTP et HTTPS entre différents services dans une infrastructure containerisée. Il fonctionne en se basant sur des labels définis dans le fichier `docker-compose.yml` pour déterminer comment router les requêtes HTTP vers les services correspondants.

**A -** On crée un dossier nommé `traefik` ou dedans on a les fichiers suivants:

- **`docker-compose.yml`** : Fichier de configuration pour définir et déployer les services Docker de Traefik, notamment le reverse proxy et les règles de routage.
- **`traefik.yml`** : Fichier de configuration principal de Traefik, où sont définis les paramètres comme les entry points, les providers, et les règles de routage.
- **`acme.json`** : Fichier utilisé par Traefik pour stocker les certificats SSL générés via ACME (Let's Encrypt)



- docker-compose.yml

```
traefik > 🐳 docker-compose.yml
1   services:
2     reverse-proxy:
3       image: traefik:v2.11
4       command:
5         - "--api.insecure=true"
6         - "--providers.docker=true"
7         - "--providers.docker.swarmMode=true"
8         - "--providers.docker.exposedByDefault=false"
9         - "--providers.docker.network=traefik"
10        - "--entrypoints.web.address=:80"
11        - "--entrypoints.websecure.address=:443"
12        - "--accesslog=true"
13        - "--log.level=DEBUG"
14       ports:
15         - target: 80
16           published: 80
17           mode: host
18         - target: 443
19           published: 443
20           mode: host
21         - target: 8080
22           published: 8080
23           mode: host
24       volumes:
25         - /var/run/docker.sock:/var/run/docker.sock:ro
26         - ./acme.json:/acme.json
```

```
traefik > ⚡ docker-compose.yml
 1   services:
 2     reverse-proxy:
14       ports:
24         volumes:
25           - /var/run/docker.sock:/var/run/docker.sock:ro
26           - ./acme.json:/acme.json
27         networks:
28           - traefik
29       deploy:
30         mode: replicated
31         replicas: 1
32         placement:
33           constraints:
34             - node.role == manager
35         labels:
36           - "traefik.enable=true"
37           - "traefik.http.routers.dashboard.rule=Host(`traefik.aseds.inpt.com`)"
38           - "traefik.http.routers.dashboard.service=api@internal"
39           - "traefik.http.routers.dashboard.entrypoints=web"
40           - "traefik.http.services.dashboard.loadbalancer.server.port=8080"
41
42     networks:
43       traefik:
44         external: true
45         name: traefik
```

- traefik.yml

```
traefik > ⌂ traefik.yml
 1   entryPoints:
 2     web:
 3       address: ":80"
 4       http:
 5         redirections:
 6           entryPoint:
 7             to: websecure
 8             scheme: https
 9       websecure:
10         address: ":443"
11
12   certificatesResolvers:
13     myresolver:
14       acme:
15         email: smooth.criminal5698@gmail.com
16         storage: acme.json
17         httpChallenge:
18           entryPoint: web
19   api:
20     dashboard: true
21     debug: false
22
23   log:
24     level: INFO
25
26   accessLog: {}
```

```
traefik > traefik.yml
12   certificatesResolvers:
13     myresolver:
14       acme.
19   api:
20     dashboard: true
21     debug: false
22
23   log:
24     level: INFO
25
26   accessLog: {}
27
28   providers:
29     docker:
30       swarmMode: true
31       exposedByDefault: false
32       network: traefik
```

#### acme.json:

On doit s'assurer que le fichier acme.json possède les permission lecture et écriture par Traefik, équivalent à chmod 600. Pour ce faire, on fait les changement nécessaire avec Windows Powershell

```
PS C:\Windows\system32> icacls "C:\Users\Elias\OneDrive\Bureau\tp\traefik\acme.json" /grant "Elias:(R,W)"
fichier traité : C:\Users\Elias\OneDrive\Bureau\tp\traefik\acme.json
1 fichiers correctement traités ; échec du traitement de 0 fichiers
PS C:\Windows\system32>
```

## B - On modifie notre fichier docker-compose.yml global:

Les ajouts Traefik dans ce fichier docker-compose.yml permettent de configurer Traefik comme reverse proxy pour les services frontend et backend. Ils définissent des règles de routage pour que Traefik dirige les requêtes vers les services en fonction du nom d'hôte, ainsi que des configurations pour activer le SSL via Traefik et lier les services au réseau traefik. Cela permet à Traefik de gérer automatiquement la distribution du trafic et de sécuriser les connexions avec des certificats SSL. Dans ce cas, grâce aux labels ajoutés dans le fichier docker-compose.yml de l'application, Traefik prend en charge le routage des requêtes vers les services frontend et backend. Par exemple, la règle `traefik.http.routers.frontend.rule=Host('aseds.inpt.com')` indique à Traefik que, lorsque le domaine `aseds.inpt.com` est appelé, il doit acheminer la requête vers le service frontend. De plus, avec la configuration `traefik.http.routers.your-router.tls.certresolver=myresolver``, Traefik gère automatiquement les certificats SSL, ce qui permet de sécuriser les connexions HTTPS pour le domaine.

Docker-compose.yml:

```
↳ docker-compose.yml
1   services:
2     postgres:
3       image: postgres:latest
4       environment:
5         - POSTGRES_DB=etudiants
6         - POSTGRES_USER=postgres
7         - POSTGRES_PASSWORD=root
8       ports:
9         - "5432:5432"
10      deploy:
11        replicas: 1
12        resources:
13          limits:
14            cpus: '0.5'
15            memory: 1G
16          restart_policy:
17            condition: on-failure
18        volumes:
19          - db_data:/var/lib/postgresql/data
20        networks:
21          - traefik
22
23      frontend:
24        image: elias007/mini-projet-conteneurisation:react-image
25        deploy:
26          replicas: 1
```

```

23   frontend:
24     image: elias007/mini-projet-conteneurisation:react-image
25     deploy:
26       replicas: 1
27       resources:
28         limits:
29           cpus: '0.5'
30           memory: 512M
31       restart_policy:
32         condition: on-failure
33       labels:
34         - "traefik.enable=true"
35         - "traefik.http.routers.frontend.rule=Host(`aseds.inpt.com`)"
36         - "traefik.http.services.frontend.loadbalancer.server.port=80"
37         - "traefik.http.routers.frontend.entrypoints=web"
38         - "traefik.http.routers.your-router.tls.certresolver=myresolver"
39     networks:
40       - traefik
41
42   backend:
43     image: elias007/mini-projet-conteneurisation:springboot-image
44     deploy:
45       replicas: 1
46       resources:
47         limits:
48           cpus: '1.0'
49           memory: 1G
50       restart_policy:
51         condition: on-failure
52       labels:
53         - "traefik.enable=true"
54         - "traefik.docker.network=traefik"
55         - "traefik.http.routers.backend.rule=Host(`api.aseds.inpt.com`)"
56         - "traefik.http.services.backend.loadbalancer.server.port=8080"
57         - "traefik.http.routers.your-router.tls.certresolver=myresolver"
58     networks:
59       - traefik
60
61   volumes:
62     db_data:
63       driver: local
64
65   networks:
66     traefik:
67       external: true
68       name: traefik
69

```

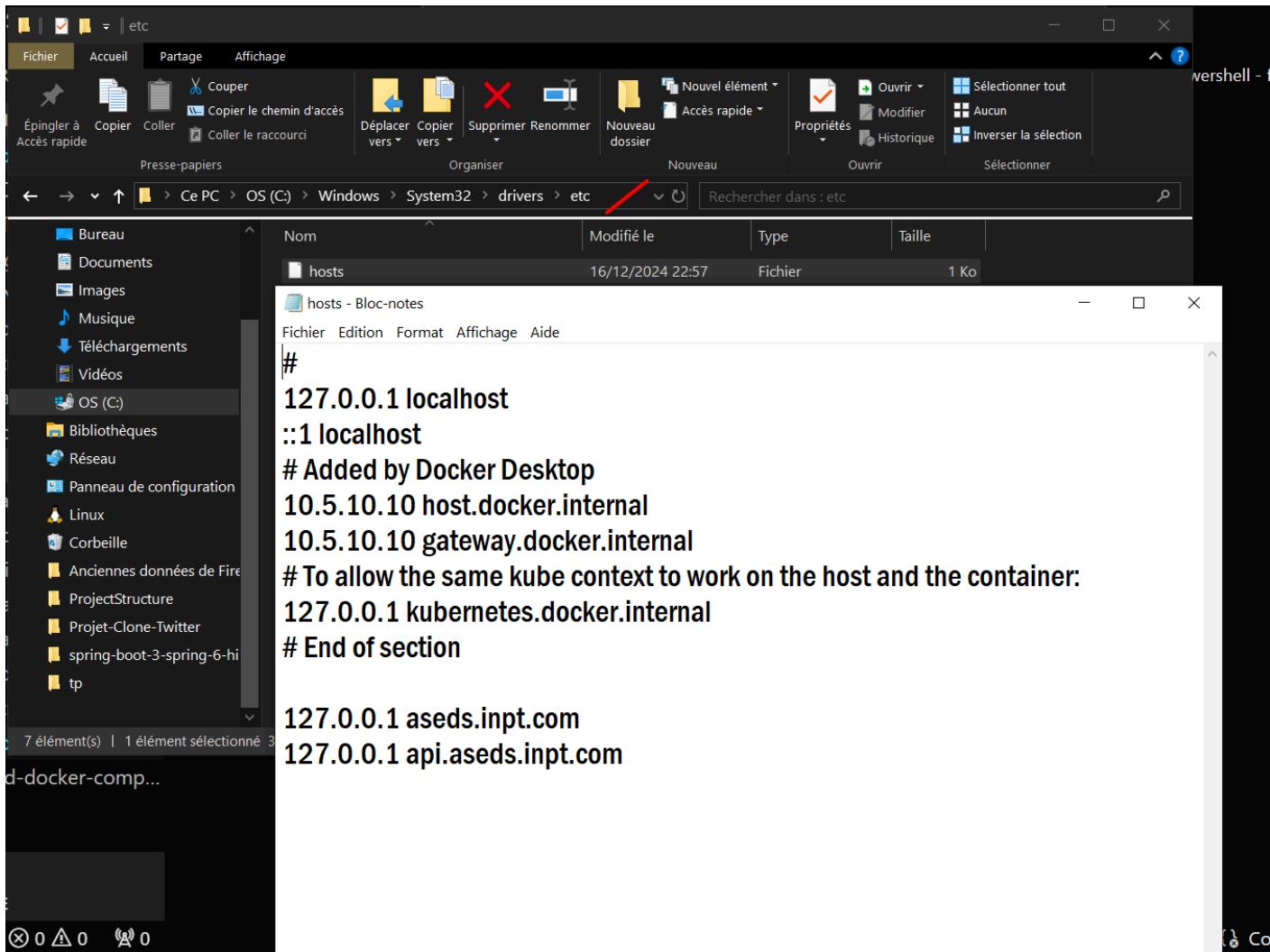
Ensuite, on crée notre réseau overlay traefik

```
PS C:\Users\Elias\OneDrive\Bureau\tp\traefik> docker network create -d overlay traefik
nun7sm8k3yfke495txlu5mmsd
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating service traefik_reverse-proxy
```

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker network inspect traefik
```

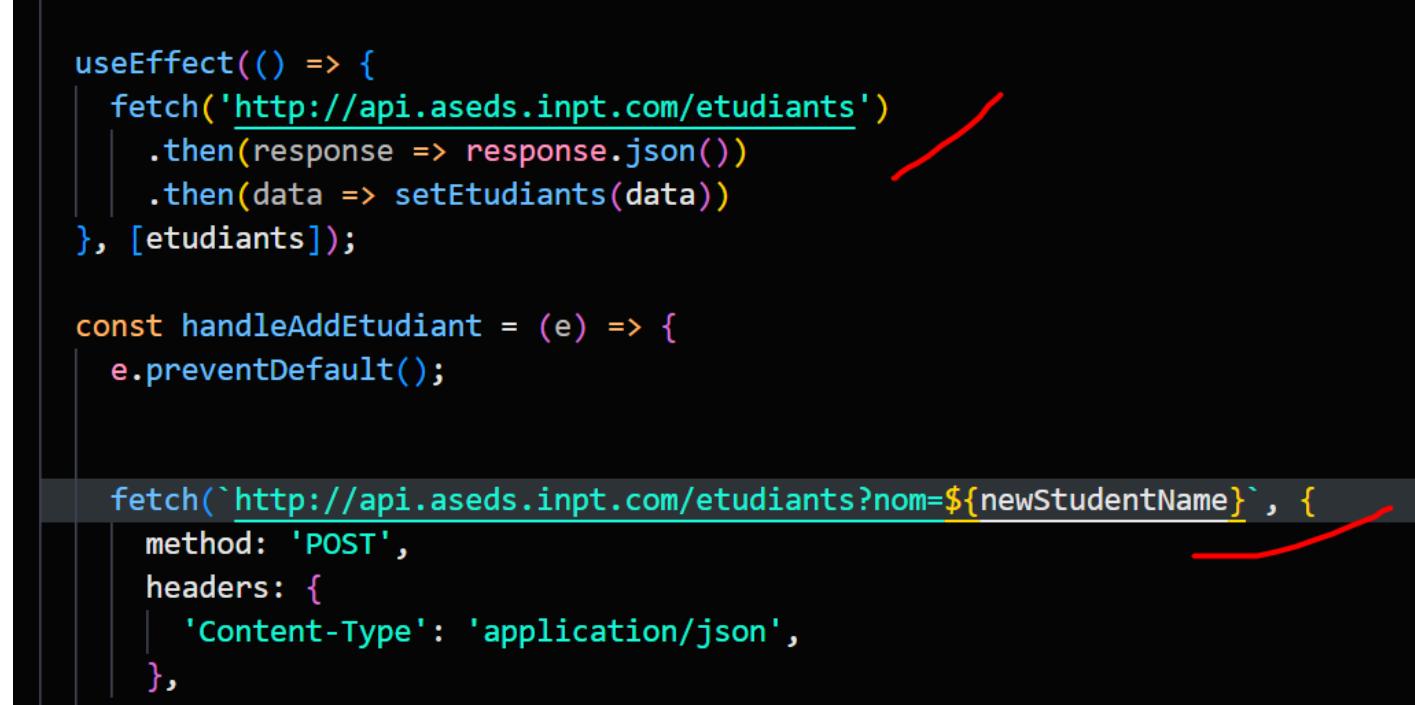
```
● [
  {
    "Name": "traefik",
    "Id": "nun7sm8k3yfke495txlu5mmsd",
    "Created": "2024-12-16T21:54:53.274741169Z",
    "Scope": "swarm",
    "Driver": "overlay",
    "EnableIPv6": false,
    "IPAM": {
      "Driver": "default",
      "Options": null,
      "Config": [
        {
          "Subnet": "10.0.1.0/24",
          "Gateway": "10.0.1.1"
        }
      ]
    },
    "Internal": false,
    "Attachable": false,
    "Ingress": false,
    "ConfigFrom": {
      "Network": ""
    },
    "ConfigOnly": false,
    "Containers": {
      "124b9ef2723932beb57fcb8945129b26908e10e351221b364ddd607183186dc7": {
        "Ports": [
          {
            "HostPort": 80
          }
        ],
        "HostConfig": {
          "Binds": [
            "/var/run/docker.sock:/var/run/docker.sock"
          ],
          "Mounts": [
            {
              "HostPath": "/var/run/docker.sock",
              "ContainerPath": "/var/run/docker.sock"
            }
          ],
          "CgroupParent": ""
        }
      }
    }
  }
]
```

Ensuite, nous avons modifié le fichier `/etc/hosts` pour associer un domaine personnalisé à l'adresse IP locale de notre serveur Traefik, afin de pouvoir tester en local



Puis, nous avons modifié les points de terminaison des requêtes `fetch` pour les adapter aux nouveaux, à savoir `api.inpt.aseds.com`. De plus, nous avons effectué les changements nécessaires dans Docker,

notamment la création de nouvelles images et leur déploiement



```
useEffect(() => {
  fetch('http://api.aseds.inpt.com/etudiants')
    .then(response => response.json())
    .then(data => setEtudiants(data))
}, [etudiants]);

const handleAddEtudiant = (e) => {
  e.preventDefault();

  fetch(`http://api.aseds.inpt.com/etudiants?nom=${newStudentName}`, {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
  });
}
```

Après avoir effectué toutes les étapes nécessaires, nous avons commencé par initialiser le cluster Docker Swarm avec la commande `docker swarm init`. Ensuite, nous avons déployé le stack Traefik en utilisant la commande `docker stack deploy -c docker-compose.yml traefik`, ce qui permet de configurer Traefik comme reverse proxy et d'assurer le routage des requêtes vers les bons services. Enfin, nous avons déployé notre application principale (frontend, backend, etc.) avec la commande `docker stack deploy -c docker-compose.yml app-stack`. Cela a permis de mettre en place l'ensemble de l'infrastructure, avec Traefik en gestion du trafic, et de lancer les différents services dans le cluster Docker Swarm.

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker swarm init
Swarm initialized: current node (za1mho4r3henecjklu0z4fotz) is now a manager.

To add a worker to this swarm, run the following command:

  docker swarm join --token SWMTKN-1-1vswzpr1sfry1zy741mezt89hifses12pkmj4xswa8dk8cpcrl-dysfed19p1nq2ha1affniph0q 19
2.168.65.3:2377
To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions.
```

```
PS C:\Users\Elias\OneDrive\Bureau\tp\traefik> docker stack deploy -c docker-compose.yml traefik
Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating service traefik_reverse-proxy
```

```
PS C:\Users\Elias\OneDrive\Bureau\tp> docker stack deploy -c docker-compose.yml app-stack
● Since --detach=false was not specified, tasks will be created in the background.
In a future release, --detach=false will become the default.
Creating service app-stack_frontend
Creating service app-stack_backend
Creating service app-stack_postgres
```

7 - Voilà, après avoir effectué toutes ces étapes, notre application est maintenant fonctionnelle et accessible sur le lien **aseds.inpt.com**.

The screenshot shows a web browser window with the following details:

- Address bar: aseds.inpt.com
- Page title: Liste des Etudiants
- Search bar: Nom de l'étudiant
- Action button: Ajouter étudiant
- Table header (thead): Id, Nom, Date de création, Moyenne

On peut aussi vérifier avec Postman que les requêtes vers notre backend accessible via `api.aseds.inpt.com` passe (200 OK)

The screenshot shows the Postman application interface. At the top, there is a header bar with a blue bar on the left, followed by a dropdown menu, the URL `http://api.aseds.inpt.com/etudiants`, a **Send** button, and a copy icon. Below the header, there are tabs for **Params**, **Authorization**, **Headers (6)**, **Body**, **Scripts**, and **Settings**. The **Cookies** tab is also visible. Under the **Params** tab, there is a table titled "Query Params" with columns for Key, Value, Description, and Bulk Edit. There is one row with "Key" and "Value" fields. On the right side of the interface, there are several icons for file operations like copy, paste, and search. In the bottom left, there are tabs for **Body**, **Cookies**, **Headers (6)**, and **Test Results**. The **Test Results** tab is active, showing a green **200 OK** status, response time of **21 ms**, and size of **207 B**. Below the status, there are buttons for **Pretty**, **Raw**, **Preview**, and **Visualize**, and a dropdown for **JSON**. The preview area shows a single item with index **1** and value **[]**. At the bottom of the interface, there are links for **Postbot**, **Runner**, **Start Proxy**, **Cookies**, **Vault**, **Trash**, and a help icon.

## Partie 3

1 - On crée notre namespace via un fichier YAML avec la commande kubectl apply -f /Namespace

```
exam-namespace.yml X
mini-projet-conteneurisation > K8s > Namespace > exam-namespace.yml
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4    name: exam
5    labels:
6      name: exam
7      environment: dev
8  spec:
9    finalizers:
10   - kubernetes.io/metadata.deletion
```

A- **Deployment** : Gère le déploiement et la scalabilité des conteneurs.

backend-deployment.yml:

```
📄 backend-deployment.yml ✘  
mini-projet-conteneurisation > K8s > Deployment > 📄 backend-deployment.yml  
1   apiVersion: apps/v1  
2   kind: Deployment  
3   metadata:  
4     name: backend  
5     namespace: exam  
6     labels:  
7       app: backend  
8   spec:  
9     replicas: 2  
10    selector:  
11      matchLabels:  
12        app: backend  
13    template:  
14      metadata:  
15        labels:  
16          app: backend  
17    spec:  
18      containers:  
19        - name: backend  
20          image: elias007/mini-projet-conteneurisation:backend-image  
21          imagePullPolicy: Always  
22          ports:  
23            - containerPort: 8080  
24              name: http  
25          envFrom:  
26            - configMapRef:
```

## backend-deployment.yml X

mini-projet-conteneurisation > K8s > Deployment > backend-deployment.yml

```
 8   spec:
13     template:
17       spec:
19         - name: backend
25           envFrom:
26             - configMapRef:
27               name: backend-config
28             - secretRef:
29               name: db-credentials
30             resources:
31               requests:
32                 cpu: "250m"
33                 memory: "512Mi"
34               limits:
35                 cpu: "500m"
36                 memory: "1Gi"
37             livenessProbe:
38               httpGet:
39                 path: /actuator/health/liveness
40                 port: 8080
41               initialDelaySeconds: 60
42               periodSeconds: 10
43               timeoutSeconds: 5
44               failureThreshold: 3
45             readinessProbe:
46               httpGet:
```

## backend-deployment.yml X

mini-projet-conteneurisation > K8s > Deployment > backend-deployment.yml

```
8   spec:
13     template:
17       spec:
19         - name: backend
37           livenessProbe:
41             initialDelaySeconds: 60
42             periodSeconds: 10
43             timeoutSeconds: 5
44             failureThreshold: 3
45           readinessProbe:
46             httpGet:
47               path: /actuator/health/readiness
48               port: 8080
49             initialDelaySeconds: 30
50             periodSeconds: 5
51             timeoutSeconds: 3
52             failureThreshold: 3
53           startupProbe:
54             httpGet:
55               path: /actuator/health
56               port: 8080
57             initialDelaySeconds: 30
58             periodSeconds: 10
59             failureThreshold: 30
```

## frontend-deployment.yml

```
frontend-deployment.yml X
mini-projet-conteneurisation > K8s > Deployment > frontend-deployment.yml
1   apiVersion: apps/v1
2   kind: Deployment
3   metadata:
4     name: frontend
5     namespace: exam
6     labels:
7       app: frontend
8   spec:
9     replicas: 2
10    selector:
11      matchLabels:
12        app: frontend
13    template:
14      metadata:
15        labels:
16          app: frontend
17    spec:
18      containers:
19        - name: frontend
20          image: elias007/mini-projet-conteneurisation:frontend-image
21          ports:
22            - containerPort: 80
23              name: http
24          resources:
25            requests:
26              cpu: "200m"
```

## 📄 *frontend-deployment.yml* ✘

mini-projet-conteneurisation > K8s > Deployment > 📄 *frontend-deployment.yml*

```
 8   spec:  
13     template:  
17       spec:  
19         - name: frontend  
24           resources:  
26             limits:  
27               cpu: "200m"  
28               memory: "256Mi"  
29             requests:  
30               cpu: "400m"  
31               memory: "512Mi"  
31             livenessProbe:  
32               httpGet:  
33                 path: /  
34                 port: 80  
35               initialDelaySeconds: 30  
36               periodSeconds: 10  
37               timeoutSeconds: 5  
38               failureThreshold: 3  
39             readinessProbe:  
40               httpGet:  
41                 path: /  
42                 port: 80  
43               initialDelaySeconds: 15  
44               periodSeconds: 5  
45               timeoutSeconds: 3  
46               failureThreshold: 3
```

## 📄 *frontend-deployment.yml* ✎

mini-projet-conteneurisation > K8s > Deployment > 📄 frontend-deployment.yml

```
 8   spec:
13     template:
17       spec:
19         - name: frontend
31           livenessProbe:
35             initialDelaySeconds: 30
36             periodSeconds: 10
37             timeoutSeconds: 5
38             failureThreshold: 3
39           readinessProbe:
40             httpGet:
41               path: /
42               port: 80
43             initialDelaySeconds: 15
44             periodSeconds: 5
45             timeoutSeconds: 3
46             failureThreshold: 3
47           startupProbe:
48             httpGet:
49               path: /
50               port: 80
51             initialDelaySeconds: 30
52             periodSeconds: 10
53             failureThreshold: 30
```

B- **Service** : Expose les pods pour les rendre accessibles.

backend-service.yml:

## backend-service.yml X

mini-projet-conteneurisation > K8s > Services > [backend-service.yml](#)

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: backend-service
5    namespace: exam
6    labels:
7      app: backend
8  spec:
9    type: NodePort
10   selector:
11     app: backend
12   ports:
13     - name: http
14       port: 8080
15       targetPort: 8080
16       protocol: TCP
```

frontend-service.yml:

## 📄 *frontend-service.yml* ✎

mini-projet-conteneurisation > K8s > Services > 📄 *frontend-service.yml*

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: frontend-service
5    namespace: exam
6  spec:
7    type: NodePort
8    selector:
9      app: frontend
10   ports:
11     - protocol: TCP
12       port: 80
13       targetPort: 80
14       nodePort: 30005
```

postgres-service.yml:

```
postgres-service.yml X
mini-projet-conteneurisation > K8s > Services > postgres-service.yml
1 apiVersion: v1
2 kind: Service
3 metadata:
4   name: postgres
5   namespace: exam
6   labels:
7     app: postgres
8 spec:
9   clusterIP: None
10  selector:
11    app: postgres
12  ports:
13    - port: 5432
14      targetPort: 5432
15      name: postgres
```

C - **ConfigMap** : Stocke les configurations de l'application.

## backend-ConfigMap.yml

```
backend-ConfigMap.yml X
mini-projet-conteneurisation > K8s > ConfigMaps > backend-ConfigMap.yml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: backend-config
5    namespace: exam
6    labels:
7      app: backend
8      component: config
9  data:
10   DB_HOST: postgres
11   DB_PORT: "5432"
12   DB_NAME: etudiants
13   DB_USER: postgres
14   SPRING_PROFILES_ACTIVE: prod
15   SPRING_DATASOURCE_URL: "jdbc:postgresql://postgres:5432/etudiants"
16   SPRING_DATASOURCE_USERNAME: "postgres"
17   SPRING_DATASOURCE_DRIVER_CLASS_NAME: "org.postgresql.Driver"
18   SPRING_JPA_HIBERNATE_DDL_AUTO: "update"
19   SPRING_JPA_DATABASE_PLATFORM: "org.hibernate.dialect.PostgreSQLDialect"
```

db-ConfigMap.yml

### db-ConfigMap.yml ×

mini-projet-conteneurisation > K8s > ConfigMaps > db-ConfigMap.yml

```
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: db-configmap
5    namespace: exam
6    labels:
7      app: postgres
8      component: config
9  data:
10   POSTGRES_DB: etudiants
11   POSTGRES_USER: postgres
```

D - **Secret** : Stocke les données sensibles (comme mots de passe).

Mais avant, il faut mettre en place en local d'un moyen permettant d'encoder des mots de passe en base 64.

Pour notre cas, on va installer via le package manager de windows « chocolatey » l'utilitaire « base64 ». Puis l'utiliser pour afficher une chaîne de caractères encodée en base64 qu'on va utiliser après dans notre fichier Secret.

```
● PS C:\Users\Elias\OneDrive\Bureau\tp> choco install base64
Chocolatey v2.2.2
Installing the following packages:
base64
By installing, you accept licenses for the packages.
Progress: Downloading base64 1.0.0... 100%

base64 v1.0.0 [Approved]
base64 package files install completed. Performing other installation steps.
The package base64 wants to run 'chocolateyinstall.ps1'.
Note: If you don't run this script, the installation will fail.
Note: To confirm automatically next time, use '-y' or consider:
choco feature enable -n allowGlobalConfirmation
Do you want to run the script?([Y]es/[A]ll - yes to all/[N)o/[P]rint): y
```

```
● PS C:\Users\Elias\OneDrive\Bureau\tp\mini-projet-conteneurisation\K8s> echo -n "root" | base64
cm9vdA0K
```

app-secrets.yml

```
📄 app-secrets.yml M ✘
mini-projet-conteneurisation > K8s > Secrets > 📄 app-secrets.yml
 1   apiVersion: v1
 2   kind: Secret
 3   metadata:
 4     name: db-credentials
 5     namespace: exam
 6     labels:
 7       app: database
 8       component: credentials
 9     data:
10       DB_PASSWORD: cm9vdA== #root (base64 encoded)
11       POSTGRES_PASSWORD: cm9vdA==
```

E - **StatefulSet** : Gère les applications nécessitant un état persistant (comme une base de données).

postgres-statefulset.yaml

```
mini-projet-conteneurisation > K8s > StatefulSets > postgres-StatefulSet.yml
1  apiVersion: apps/v1
2  kind: StatefulSet
3  metadata:
4    name: db-statefulset
5    namespace: exam
6    labels:
7      app: postgres
8  spec:
9    serviceName: postgres
10   replicas: 1
11   selector:
12     matchLabels:
13       app: postgres
14   template:
15     metadata:
16       labels:
17         app: postgres
18     spec:
19       containers:
20         - name: postgres
21           image: postgres:17-alpine
22           ports:
23             - containerPort: 5432
24               name: postgres
25             envFrom:
26               - configMapRef:
```

mini-projet-conteneurisation > K8s > StatefulSets >  postgres-StatefulSet.yml

```
8   spec:
14     template:
18       spec:
20         - name: postgres
25           envFrom:
26             - configMapRef:
27               name: db-configmap
28             - secretRef:
29               name: db-credentials
30             env:
31               - name: PGDATA
32                 value: /var/lib/postgresql/data/pgdata
33             resources:
34               requests:
35                 memory: "256Mi"
36                 cpu: "250m"
37               limits:
38                 memory: "512Mi"
39                 cpu: "500m"
40             volumeMounts:
41               - name: postgres-storage
42                 mountPath: /var/lib/postgresql/data
43             volumeClaimTemplates:
44               - metadata:
45                 name: postgres-storage
46             spec:
```

mini-projet-conteneurisation > K8s > StatefulSets >  postgres-StatefulSet.yml

```
8   spec:
14     template:
18       spec:
20         - name: postgres
23           resources:
39             |   cpu: "500m"
40             |   volumeMounts:
41             |     - name: postgres-storage
42             |       mountPath: /var/lib/postgresql/data
43             volumeClaimTemplates:
44               - metadata:
45                 name: postgres-storage
46               spec:
47                 accessModes: [ "ReadWriteOnce" ]
48                 resources:
49                   requests:
50                     storage: 1Gi
```

## 2- Définir des quotas pour la consommation des ressources :

- **Objectif** : Limiter et standardiser la consommation de CPU et de mémoire par les pods dans le namespace "exam".

## quotas.yml

```
mini-projet-conteneurisation > K8s > Quotas > quotas.yml
1  apiVersion: v1
2  kind: ResourceQuota
3  metadata:
4    name: quotas
5    namespace: exam
6  spec:
7    hard:
8      requests.cpu: "4"
9      requests.memory: 6Gi
10     limits.cpu: "6"
11     limits.memory: 8Gi
12     pods: "20"
```

Par ailleurs, on crée un fichier Limit Range qui permet de définir des limites par défaut et des requêtes par défaut pour les ressources CPU et mémoire utilisées par les conteneurs dans le namespace exam. Cela assure que chaque pod dans ce namespace consomme une quantité raisonnable de ressources.

## exam-limit-range.yml

```
mini-projet-conteneurisation > K8s > limit-range > exam-limit-range.yml
1  apiVersion: v1
2  kind: LimitRange
3  metadata:
4    name: exam-limits
5    namespace: exam
6  spec:
7    limits:
8      - default:
9        cpu: 500m
10       memory: 512Mi
11      defaultRequest:
12        cpu: 200m
13        memory: 256Mi
14      type: Container
```

### 3 - Contrôler l'accès avec un rôle RBAC :

- **Objectif** : Mettre en place un système d'autorisation basé sur les rôles pour restreindre l'accès aux ressources du namespace "exam".

exam-role.yml

```
mini-projet-conteneurisation > K8s > RBAC > exam-role.yml
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: Role
3  metadata:
4    name: exam-role
5    namespace: exam
6  rules:
7    - apiGroups: [""]
8      resources: ["pods", "services", "configmaps", "secrets"]
9      verbs: ["get", "list", "watch"]
10   - apiGroups: ["apps"]
11     resources: ["deployments", "statefulsets"]
12     verbs: ["get", "list", "watch"]
```

exam-roleBinding.yml

```
mini-projet-conteneurisation > K8s > RBAC > exam-roleBinding.yml
1  apiVersion: rbac.authorization.k8s.io/v1
2  kind: RoleBinding
3  metadata:
4    name: exam-rolebinding
5    namespace: exam
6  subjects:
7    - kind: ServiceAccount
8      name: exam-user
9      namespace: exam
10  roleRef:
11    kind: Role
12    name: exam-role
13    apiGroup: rbac.authorization.k8s.io
```

#### 4 - Spécifier un budget de perturbation (Disruption Budget) :

- **Objectif** : Garantir une haute disponibilité en définissant combien de pods peuvent être arrêtés pour maintenance ou mise à jour sans perturber le service.

pdb-backend.yml

```
mini-projet-conteneurisation > K8s > PDB > pdb-backend.yml
```

```
1  apiVersion: policy/v1
2  kind: PodDisruptionBudget
3  metadata:
4    name: backend-pdb
5    namespace: exam
6  spec:
7    minAvailable: 1
8    selector:
9      matchLabels:
10     app: backend
```

pdb-db.yml

```
mini-projet-conteneurisation > K8s > PDB > pdb-db.yml
```

```
1  apiVersion: policy/v1
2  kind: PodDisruptionBudget
3  metadata:
4    name: postgres-pdb
5    namespace: exam
6  spec:
7    minAvailable: 1      # At least 1 database pod must be available
8    selector:
9      matchLabels:
10     app: postgres
```

pdb-frontend.yml

```
mini-projet-conteneurisation > K8s > PDB > pdb-frontend.yml
1  apiVersion: policy/v1
2  kind: PodDisruptionBudget
3  metadata:
4    name: frontend-pdb
5    namespace: exam
6  spec:
7    minAvailable: 1      # At least 1 pod must be available
8    selector:
9      matchLabels:
10     app: frontend
```

##### 5 - Créer des probes (Liveness, Readiness, Startup) :

- **Objectif** : Configurer des vérifications automatiques pour :
  - **Liveness** : S'assurer que les pods sont vivants et fonctionnent.
  - **Readiness** : Vérifier que les pods sont prêts à recevoir du trafic.
  - **Startup** : S'assurer que les pods démarrent correctement.

Pour le backend, on ajoute dans notre fichier application.properties les configurations suivantes pour exposer nos probes

```
11 management.endpoints.web.exposure.include=health
12 management.endpoint.health.probes.enabled=true
13 management.health.livenessState.enabled=true
14 management.health.readinessState.enabled=true
15
```

Ensuite, on ajoute les configurations suivantes dans notre deployment:

```
mini-projet-conteneurisation > K8s > Deployment > 📄 backend-deployment.yml
 8   spec:
13     template:
17       spec:
19         - name: backend
30           resources:
31             memory: 1Gi
37             livenessProbe:
38               httpGet:
39                 path: /actuator/health/liveness
40                 port: 8080
41                 initialDelaySeconds: 60
42                 periodSeconds: 10
43                 timeoutSeconds: 5
44                 failureThreshold: 3
45             readinessProbe:
46               httpGet:
47                 path: /actuator/health/readiness
48                 port: 8080
49                 initialDelaySeconds: 30
50                 periodSeconds: 5
51                 timeoutSeconds: 3
52                 failureThreshold: 3
53             startupProbe:
54               httpGet:
55                 path: /actuator/health
56                 port: 8080
57                 initialDelaySeconds: 30
58                 periodSeconds: 10
59                 failureThreshold: 30
```

De même, pour le frontend:

mini-projet-conteneurisation > K8s > Deployment >  frontend-deployment.yml

```
8   spec:
13     template:
17       spec:
19         - name: frontend
24           resources:
30             limits:
31               memory: "512Mi"
32             livenessProbe:
33               httpGet:
34                 path: /
35                 port: 80
36               initialDelaySeconds: 30
37               periodSeconds: 10
38               timeoutSeconds: 5
39               failureThreshold: 3
40             readinessProbe:
41               httpGet:
42                 path: /
43                 port: 80
44               initialDelaySeconds: 15
45               periodSeconds: 5
46               timeoutSeconds: 3
47               failureThreshold: 3
48             startupProbe:
49               httpGet:
50                 path: /
51                 port: 80
52               initialDelaySeconds: 30
53               periodSeconds: 10
54               failureThreshold: 30
```

Et on peut accéder à ces endpoints pour vérifier l'état de notre application

The image contains three separate browser windows, each displaying a JSON response from the `/actuator/health` endpoint of a service named `api.aseds.inpt.com`.

- Screenshot 1:** Shows the overall health status. The `status` field is `"UP"`. The `groups` section contains two items: `0: "liveness"` and `1: "readiness"`, both of which also have `status: "UP"`.
- Screenshot 2:** Shows the `liveness` group status. The `status` field is `"UP"`.
- Screenshot 3:** Shows the `readiness` group status. The `status` field is `"UP"`.

## 6- Créez un Ingress pour accéder à l'application avec un nom de domaine :

- **Objectif :** Configurer un point d'entrée unique pour accéder à l'application via un nom de domaine, sans avoir à manipuler directement les ports des services. Dans notre cas, on choisit d'exposer notre frontend via le domaine `aseds.inpt.com` et le backend via `api.aseds.inpt.com`

backend-ingress.yml

```
mini-projet-conteneurisation > K8s > Ingress > backend-ingress.yml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: backend-ingress
5    namespace: exam
6    annotations:
7      | kubernetes.io/ingress.class: "nginx"
8  spec:
9    rules:
10      - host: api.aseds.inpt.com
11        http:
12          paths:
13            - path: /
14              pathType: Prefix
15              backend:
16                service:
17                  name: backend-service
18                  port:
19                    number: 8080
```

## frontend-ingress.yml

```
mini-projet-conteneurisation > K8s > Ingress > 📄 frontend-ingress.yml
1  apiVersion: networking.k8s.io/v1
2  kind: Ingress
3  metadata:
4    name: frontend-ingress
5    namespace: exam
6    annotations:
7      | kubernetes.io/ingress.class: "nginx"
8  spec:
9    rules:
10   - host: aseds.inpt.com
11     http:
12       paths:
13         - path: /
14           pathType: Prefix
15           backend:
16             service:
17               name: frontend-service
18               port:
19                 number: 80
```

nginx-ConfigMap.yaml:

```
mini-projet-conteneurisation > K8s > ConfigMaps > nginx-ConfigMap.yml
1  apiVersion: v1
2  kind: ConfigMap
3  metadata:
4    name: custom-nginx-conf
5    namespace: exam
6  data:
7    default.conf: |
8      server {
9        listen 80;
10       root /usr/share/nginx/html;
11       index index.html;
12
13       location / {
14         try_files $uri $uri/ /index.html;
15       }
16
17       location /api {
18         proxy_pass http://backend-service:8080;
19         proxy_set_header Host $host;
20         proxy_set_header X-Real-IP $remote_addr;
21       }
22     }
```

7 - Tests de l'application:

Pour chaque fichier YAML de la structure (ConfigMaps, Deployment, Ingress, Namespace, PDB, Quotas, RBAC, Secrets, Services, StatefulSets), les commandes suivantes ont été utilisées pour appliquer la configuration dans le cluster :

```
kubectl apply -f <nom-du-fichier>.yml
```

Une fois les manifests appliqués, nous avons utilisé la commande suivante pour vérifier que les pods, services, ingress et autres ressources sont correctement créés et fonctionnent :

```
kubectl get all -n exam
```

```
● PS C:\Users\Elias\OneDrive\Bureau\tp\mini-projet-conteneurisation\K8s> kubectl get all -n exam
  NAME          READY   STATUS    RESTARTS   AGE
  pod/backend-5bd6687d9c-bcp9h  1/1     Running   1 (26m ago)  23h
  pod/backend-5bd6687d9c-xvzx4  1/1     Running   1 (23h ago)  23h
  pod/db-statefulset-0          1/1     Running   3 (23h ago)  24h
  pod/frontend-96b969f48-rp5gx  1/1     Running   1 (23h ago)  23h
  pod/frontend-96b969f48-t5rpg   1/1     Running   1 (23h ago)  23h

  NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)      AGE
  service/backend-service  NodePort    10.103.116.12  <none>       8080:30635/TCP  24h
  service/db-service     ClusterIP   None          <none>       5432/TCP      3d23h
  service/frontend-service  NodePort    10.103.50.255  <none>       80:30005/TCP  24h
  service/postgres       ClusterIP   None          <none>       5432/TCP      3d23h

  NAME          READY   UP-TO-DATE   AVAILABLE   AGE
  deployment.apps/backend  2/2      2           2           24h
  deployment.apps/frontend 2/2      2           2           24h

  NAME          DESIRED  CURRENT  READY   AGE
  replicaset.apps/backend-5bd6687d9c  2         2         2       24h
  replicaset.apps/backend-6b58879fb  0         0         0       24h
  replicaset.apps/frontend-769b8dc5cb  0         0         0       24h
  replicaset.apps/frontend-96b969f48  2         2         2       24h

  NAME          READY   AGE
  statefulset.apps/db-statefulset  1/1     24h
  ● PS C:\Users\Elias\OneDrive\Bureau\tp\mini-projet-conteneurisation\K8s>
```

### Vérifier le bon fonctionnement de l'application :

- **Objectif** : Accéder à l'application via le navigateur pour confirmer que tous les composants (frontend, backend, database) fonctionnent ensemble comme attendu.

Ajout d'un nouveau etudiant:

Navigation bar



## Liste des Etudiants

Rachid

Ajouter étudiant

Id	Nom	Date de création	Moyenne
1	Bismillah	2024-12-27T00:10:56.366+00:00	20
2	Ahmed	2024-12-27T00:27:10.434+00:00	12.875



## Liste des Etudiants

Nom de l'étudiant

Ajouter étudiant

Id	Nom	Date de création	Moyenne
1	Bismillah	2024-12-27T00:10:56.366+00:00	20
2	Ahmed	2024-12-27T00:27:10.434+00:00	13.7
3	Rachid	2024-12-27T00:29:06.225+00:00	0



Ajout d'une nouvelle note pour un étudiant:



## Liste des notes de Ahmed

Nom du cours

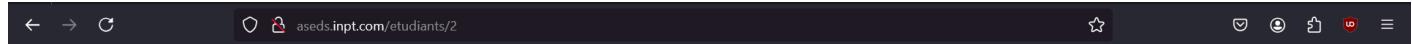
Sport

Note

17

Ajouter la note

Nom du cours	Note
Math	15
Physique	17.5
Chimie	10
Langues	9



## Liste des notes de Ahmed

Nom du cours

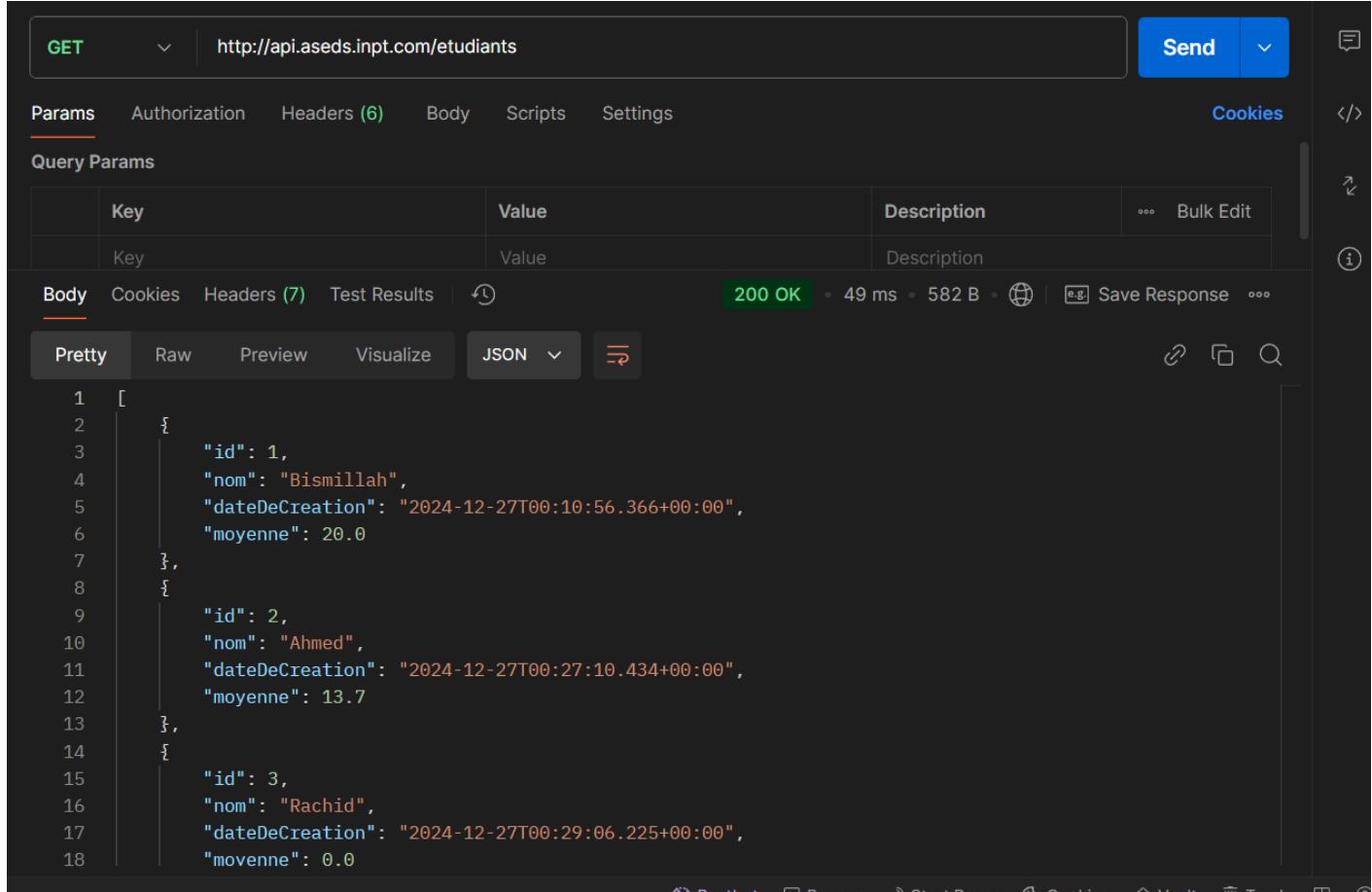
Note

Ajouter la note

Nom du cours	Note
Math	15
Physique	17.5
Chimie	10
Langues	9
Sport	17



Test avec Postman:



The screenshot shows the Postman interface with a successful API call. The URL is `http://api.aseds.inpt.com/etudiants`. The response status is `200 OK` with a duration of `49 ms` and a size of `582 B`. The response body is a JSON array containing three student records:

```
1 [  
2 {  
3   "id": 1,  
4   "nom": "Bismillah",  
5   "dateDeCreation": "2024-12-27T00:10:56.366+00:00",  
6   "moyenne": 20.0  
7 },  
8 {  
9   "id": 2,  
10  "nom": "Ahmed",  
11  "dateDeCreation": "2024-12-27T00:27:10.434+00:00",  
12  "moyenne": 13.7  
13 },  
14 {  
15  "id": 3,  
16  "nom": "Rachid",  
17  "dateDeCreation": "2024-12-27T00:29:06.225+00:00",  
18  "moyenne": 0.0
```

## Conclusion

Ce projet nous a permis d'acquérir des compétences pratiques dans le développement d'applications web 3-tiers tout en explorant des outils modernes de déploiement et de gestion d'infrastructures. Nous avons appris à utiliser Docker pour conteneurisé les composants frontend, backend et base de données, Docker Compose pour orchestrer ces conteneurs, et Docker Swarm pour assurer la haute disponibilité des services. De plus, nous avons approfondi notre compréhension de Kubernetes en créant des manifestes pour gérer le déploiement, la scalabilité et la résilience de l'application. Ces apprentissages renforcent notre maîtrise des technologies essentielles à la gestion d'applications modernes dans des environnements distribués.

