RAPPORT DU MINI PROJET CONTENEURISATION D'UNE APPLICATION WEB

Réalisé par :
Ilyas DAHHOU

Supervisé par :
M. Driss ALLAKI

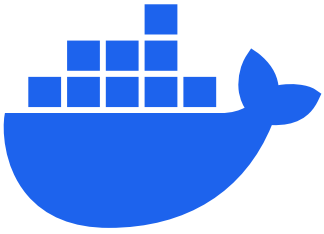


Table de matières

Introduction :

La conteneurisation des applications se positionne comme une révolution majeure dans l'écosystème du déploiement logiciel, apportant une approche novatrice et performante pour l'exécution d'applications. Cette méthode consiste à regrouper de manière cohérente une application avec toutes les ressources nécessaires à son fonctionnement optimal, englobant bibliothèques, fichiers de configuration et dépendances, au sein d'un conteneur autonome. Au cœur de notre projet, nous faisons appel à l'outil Docker, véritable référence en matière de conteneurisation, pour encapsuler avec précision une application web. Pour assurer une gestion orchestrée et dynamique de ces conteneurs, notre choix s'oriente vers la robuste plateforme d'orchestration k8s (Kubernetes). L'orchestration, pièce maîtresse de notre approche, permet une coordination harmonieuse des conteneurs, assurant ainsi une mise en œuvre efficace des applications, une scalabilité fluide et une gestion dynamique des ressources. Ainsi, notre démarche fusionne la puissance de la conteneurisation à l'efficacité de l'orchestration, offrant des solutions logicielles flexibles, évolutives et dotées d'une performance exceptionnelle.

Partie 1 :

1. On pull d'abord l'image mysql du docker hub

```
C:\Users\HP\Desktop\projetconteneurK8s>docker pull mysql:latest
latest: Pulling from library/mysql
8e0176adc18c: Pull complete
2d2c52718f65: Pull complete
d88d03ce139b: Pull complete
4a7d7f11aa1e: Pull complete
ce5949193e4c: Pull complete
f7f024dfb329: Pull complete
5fc3c840facc: Pull complete
509068e49488: Pull complete
cbc847bab598: Pull complete
942bef62a146: Pull complete
Digest: sha256:1773f3c7aa9522f0014d0ad2bbdaf597ea3b1643c64c8ccc2123c64afd8b8
2b1
Status: Downloaded newer image for mysql:latest
docker.io/library/mysql:latest
```

```
C:\Users\HP\Desktop\projetconteneurK8s>docker images
REPOSITORY                    TAG       IMAGE ID       CREATED         SIZE
cassandra                     latest    2c6aaaa0df27   3 days ago      356MB
mysql                         latest    a3b6608898d6   10 days ago     596MB
redis                         latest    e579380d4317   2 weeks ago     138MB
gcr.io/k8s-minikube/kicbase   v0.0.40   c6cc01e60919   3 months ago    1.19GB
```

On instancie maintenant l'image mysql avec un conteneur nommé mysql-container au même temps On crée un « named volume » appelé mysql-data pour assurer la persistance de la base de données. Cette méthode est la plus adéquate puisqu'elle permet la persistance dans un espace de nom accessible seulement par les processus de docker tout en laissant la liberté à docker de choisir l'emplacement exact au sein de l'espace du nom.

```
C:\Users\HP>docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=rootily
as -e MYSQL_DATABASE=testdb -d -v mysql-data:/var/lib/mysql mysql:latest
81dc3883b0448cb59096045e9b4d11d9d7d7a9f8282284b96b4e6fb3dfe78803
```

2. On crée un réseau docker appelé projet_network en précisant un sous-réseau et le driver bridge. On choisit ce driver, car il permet la communication mutuelle entre les conteneurs ainsi que la communication avec la machine hôte.

```
C:\Users\HP>docker network create --driver bridge projet_network
e914426cacb12d138c3c4e1698ebf6aceceae297a0155fa05dda8911c50428af
```

On connecte maintenant le conteneur mysql-container au réseau projet_network

```
C:\Users\HP>docker network connect projet_network mysql-container
```

```
C:\Users\HP>docker inspect projet_network
[
    {
        "Name": "projet_network",
        "Id": "e914426cacb12d138c3c4e1698ebf6aceceae297a0155fa05dda8911c50428af",
        "Created": "2023-11-19T21:20:45.469650011+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "81dc3883b0448cb59096045e9b4d11d9d7d7a9f8282284b96b4e6fb3dfe78803": {
                "Name": "mysql-container",
                "EndpointID": "524f6830a0eb087613bb63907d034742f9f920d0d67e22250ffe53953fc451e6"
```

3. On crée maintenant le fichier Dockerfile

```dockerfile
# utilisation de l'image maven pour la construction
FROM maven:alpine AS 🔨 build

# Définitir la répertoire de travaille
WORKDIR /app/myapp

# Copie du code source dans l'image de construction
COPY . .

# Construction du projet avec Maven
RUN mvn clean package

# Image pour ranner le code java
FROM openjdk:17-alpine

# Définitir la répertoire de travaille
WORKDIR /app

EXPOSE 8080

# Copie du fichier JAR généré depuis l'étape de construction
COPY --from=build /app/myapp/target/spring-boot-data-jpa*.jar app.jar

# Commande à exécuter lors du démarrage du conteneur
ENTRYPOINT ["java","-jar","app.jar"]
```

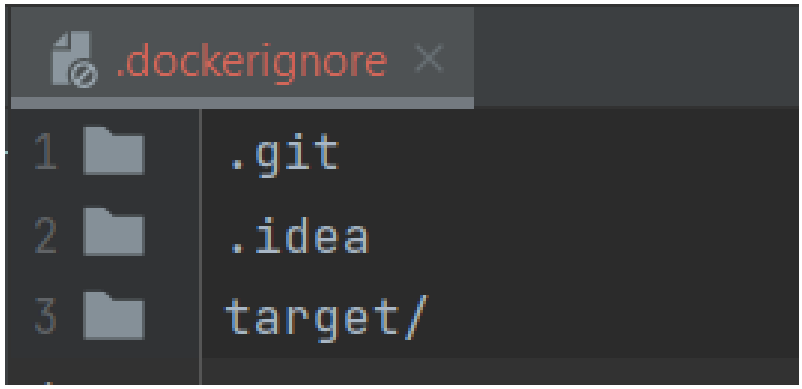
4. Dans l'élaboration de ce fichier Dockerfile, nous avons pris soin de suivre les bonnes pratiques suivantes :

a. L'utilisation d'une image maven et openjdk officielle et vérifiée

b. La spécification d'une version de l'image utilisée

c. L'utilisation du mot-clé alpine qui permet de récupérer une image réduite au fonctionnalités nécessaire ce qui diminue la taille de l'image et par la suite le temps d'import.

d. Utilisation du fichier .dockerignore (pour target/, .git, .idea)



5. Taper les commandes docker nécessaires pour :

a. Avant de créer l'image, on change le lien de la connexion de mysql

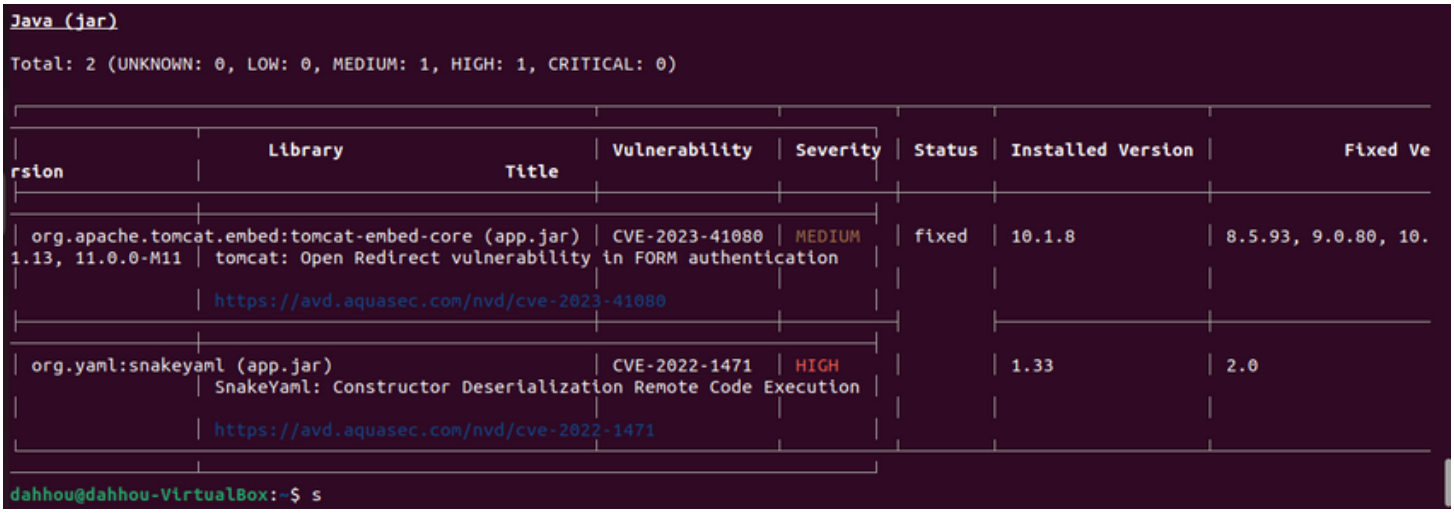
```
spring.datasource.url= jdbc:mysql://mysql-container:3306/testdb?u
spring.datasource.username= root
spring.datasource.password= rootilyas
```

On crée maintenant une image appelée dahhouilyas/backend:1.0.0 à partir du Dockerfile

```
PS C:\Users\HP\Desktop\projetconteneurK8s\spring-boot-data-jpa-mysql> docker build . -t dahhouilyas/backend:1.0.0
DEPRECATED: The legacy builder is deprecated and will be removed in a future release.
            Install the buildx component to build images with BuildKit:
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  111.1kB
Step 1/9 : FROM maven AS build
 ---> 6f90e67c6517
Step 2/9 : WORKDIR /app/myapp
```

b. On scan l'image avec la commande trivy image dahhouilyas/backend:1.0.0



```
Java (jar)

Total: 2 (UNKNOWN: 0, LOW: 0, MEDIUM: 1, HIGH: 1, CRITICAL: 0)

┌──────────────────────────────────────┬────────────────┬──────────┬────────┬───────────────────┬──────────────────────┐
│               Library                │ Vulnerability  │ Severity │ Status │ Installed Version │          Fixed Ve
rsion                │                Title                 │
├──────────────────────────────────────┼────────────────┼──────────┼────────┼───────────────────┼──────────────────────┤
│ org.apache.tomcat.embed:tomcat-embed-core (app.jar) │ CVE-2023-41080 │ MEDIUM   │ fixed  │ 10.1.8            │ 8.5.93, 9.0.80, 10.
1.13, 11.0.0-M11 │ tomcat: Open Redirect vulnerability in FORM authentication │
│                                      │                │          │        │                   │
│                                      │ https://avd.aquasec.com/nvd/cve-2023-41080 │
├──────────────────────────────────────┼────────────────┼──────────┼────────┼───────────────────┼──────────────────────┤
│ org.yaml:snakeyaml (app.jar)         │ CVE-2022-1471  │ HIGH     │        │ 1.33              │ 2.0
                     │ SnakeYaml: Constructor Deserialization Remote Code Execution │
│                                      │                │          │        │                   │
│                                      │ https://avd.aquasec.com/nvd/cve-2022-1471 │
└──────────────────────────────────────┴────────────────┴──────────┴────────┴───────────────────┴──────────────────────┘
dahhou@dahhou-VirtualBox:~$ s
```

c. Pour publier cette image dans le Docker hub, on s'authentifie maintenant avec la commande docker login

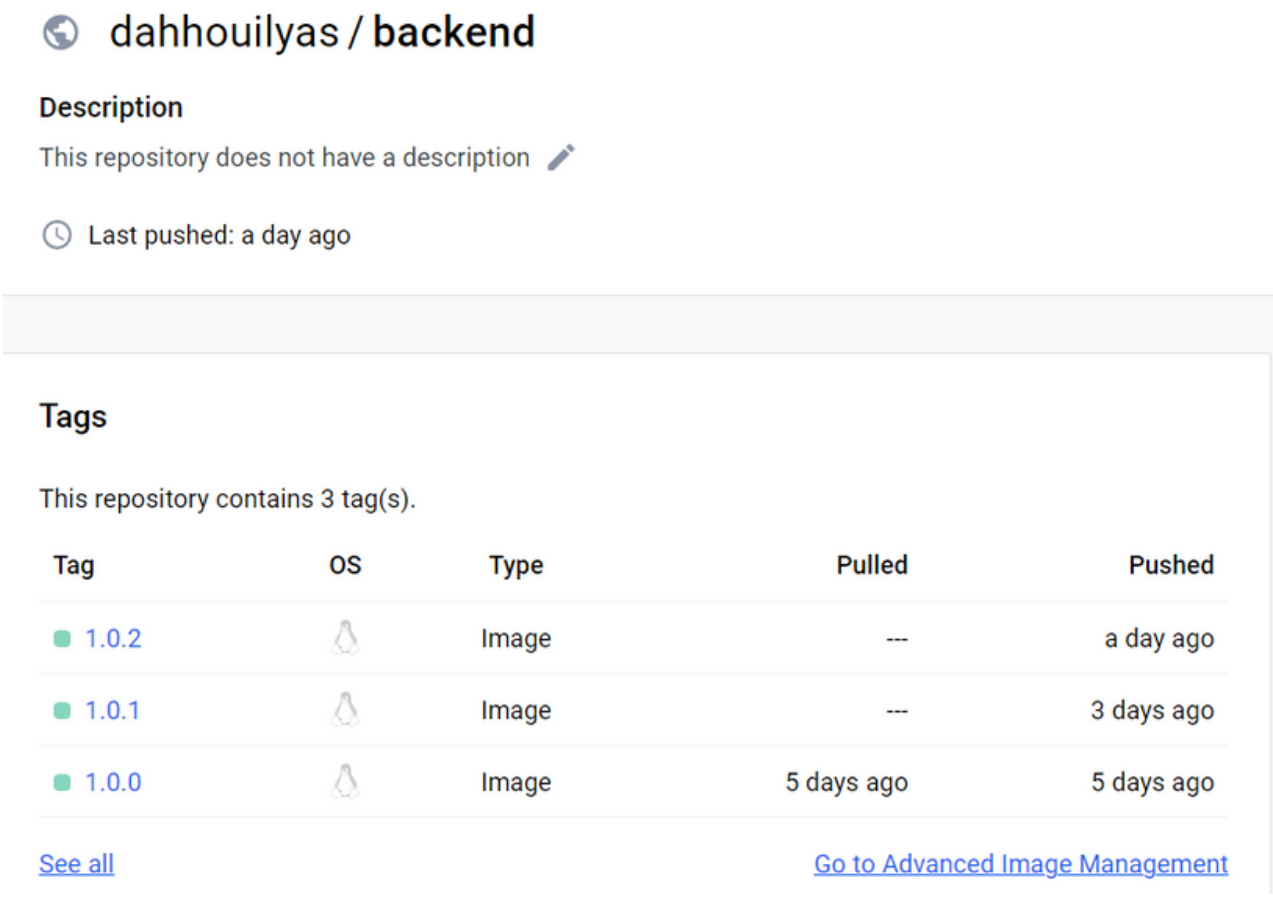
```
C:\Users\HP>docker login
Authenticating with existing credentials...
WARNING! Your password will be stored unencrypted in C:\Users\HP\.docker\config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

par la suite, on pusher l'image

```
C:\Users\HP>docker push dahhouilyas/backend:1.0.0
The push refers to repository [docker.io/dahhouilyas/backend]
ea54596697b8: Pushed
f8b575e19f99: Pushed
34f7184834b2: Mounted from library/openjdk
5836ece05bfd: Mounted from library/openjdk
72e830a4dff5: Mounted from library/openjdk
1.0.0: digest: sha256:5373b74e603709a3055c9cca6aa8c11dd8e4d905dd59c1c89e0597
88bf2f169c size: 1369
```

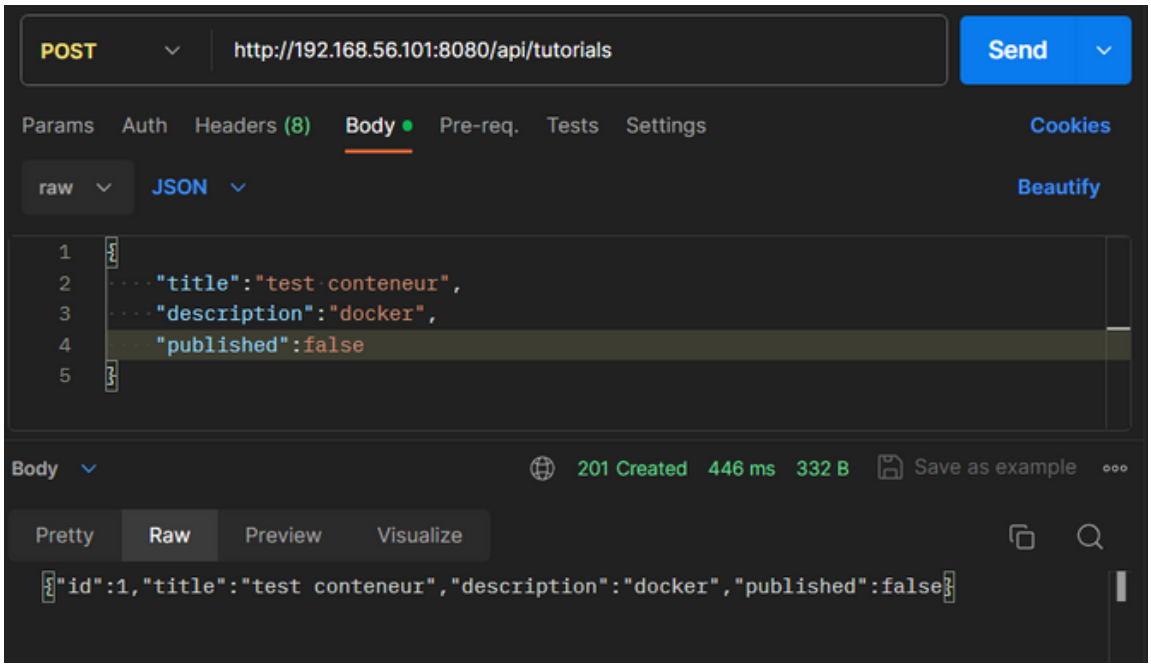
On vérifie maintenant que l'image est disponible dans le repo "backend"

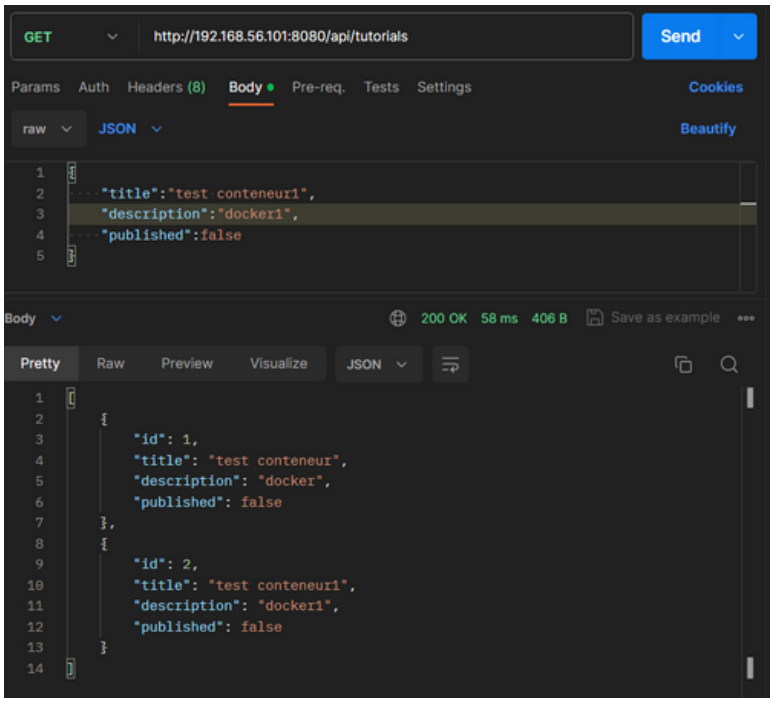


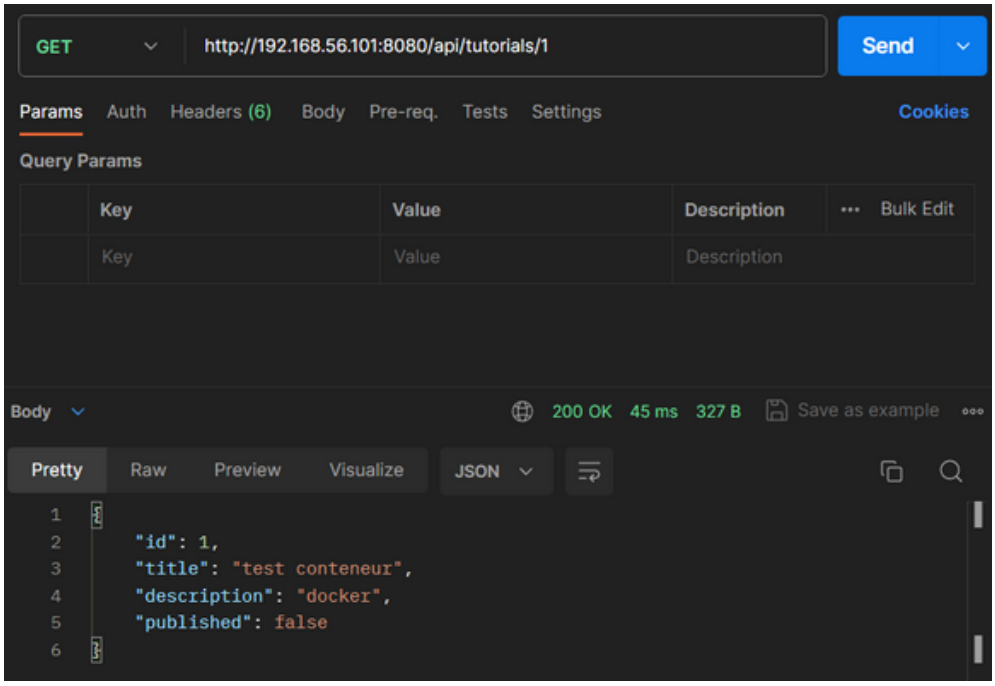
d. On instancie maintenant l'image en créant un conteneur (nommé backend) s'exécutant en local et partageant le même réseau avec le conteneur "mysql-container" de la base de données mysql. On spécifie un mapping 8080 :8080 pour ce conteneur avec l'option -p

```
C:\Users\HP>docker run --name backend --network projet_network -p 8080:8080
-d dahhouilyas/backend:1.0.0
64323133f1855d3f1a777bf697c5af89d0d9e6de1b45648b5b415c0538ebbf0d
```

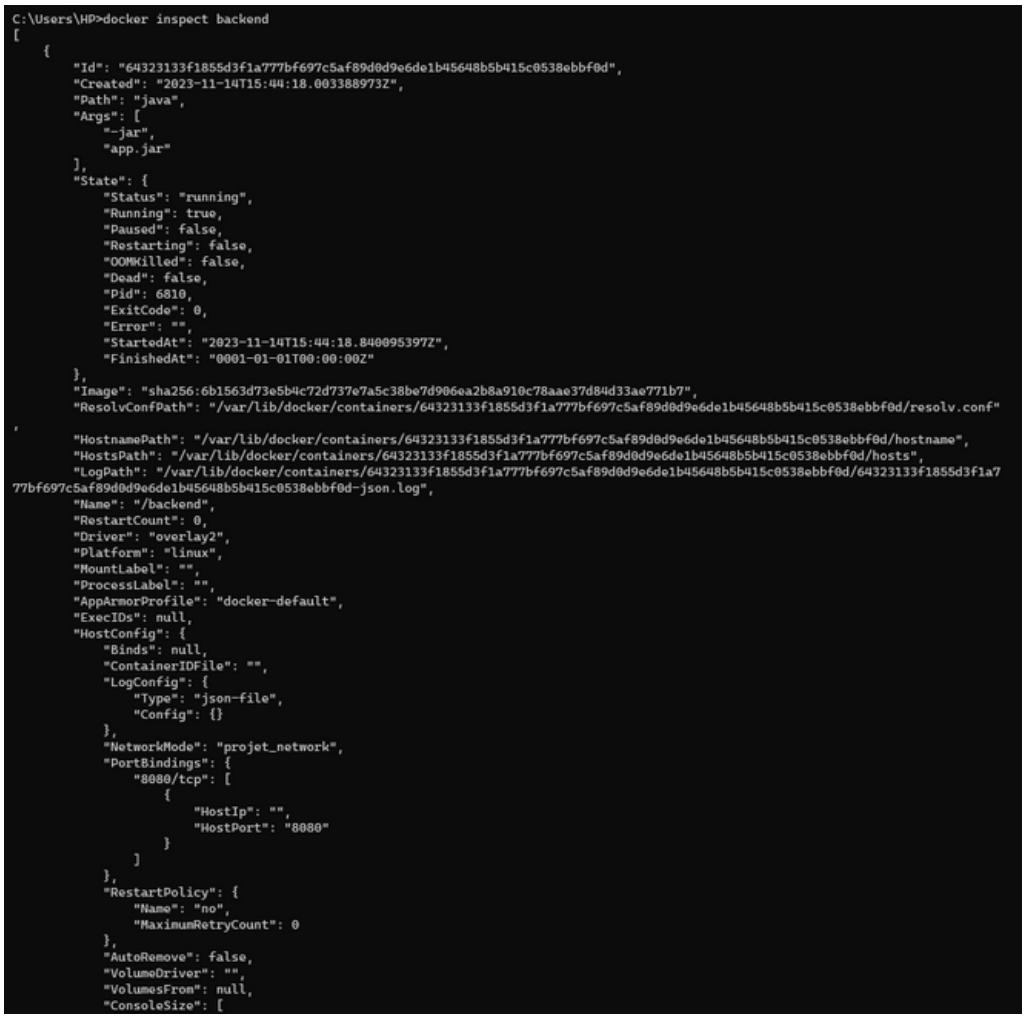
par la suite on fait des test de fonctionnement par postman







e. On inspect le conteneur backend avec la commande docker inspect backend



f. On affiche les logs liés à ce conteneur avec la commande docker logs backend.

6. On refait maintenant les mêmes étapes pour le projet frontend

a. On crée maintenant le fichier Dockerfile

```
# Stage 1: Build Angular app
FROM node:16-alpine as build
WORKDIR /app
COPY . .
RUN npm install
RUN npm run build --prod

# Stage 2: Use Nginx to serve the build
FROM nginx:1.25.3-alpine
COPY --from=build /app/dist/angular-16-crud /usr/share/nginx/html
EXPOSE 80
```

Pendant la création de ce fichier Dockerfile, on a respecté les bonnes pratiques suivantes :

a. L'utilisation des images node et nginx officielles et vérifiées

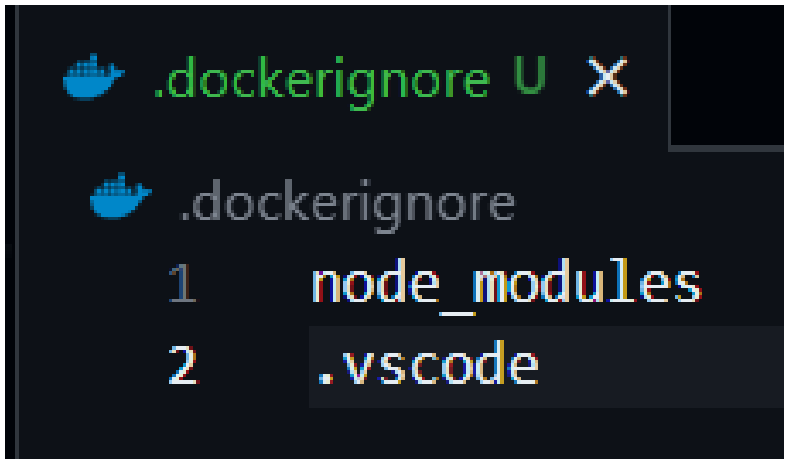
b. La spécification d'une version des image utilisées.

c. L'utilisation du mot-clé alpine qui permet de récupérer une image réduite au fonctionnalités nécessaire ce qui diminue la taille de l'image et par la suite le temps d'import.

On modifie aussi dans le fichier tutorial.service pour assurer la communication entre le frontend et le backend

```
const baseUrl = 'http://192.168.56.101:8080/api/tutorials';
```

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



On crée maintenant une image nommé frontend à partir de ce fichier Dockerfile

```
$ docker build . -t frontend
DEPRECATED: The legacy builder is deprecated and will be remov
            Install the buildx component to build images with
            https://docs.docker.com/go/buildx/

Sending build context to Docker daemon  743.9kB
Step 1/8 : FROM node:16-alpine as build
```

b. On scan l'image avec la commande trivy image dahhouilyas/frontend

dahhouilyas/frontend:1.0.1 (alpine 3.18.4)

Total: 5 (UNKNOWN: 0, LOW: 0, MEDIUM: 4, HIGH: 1, CRITICAL: 0)

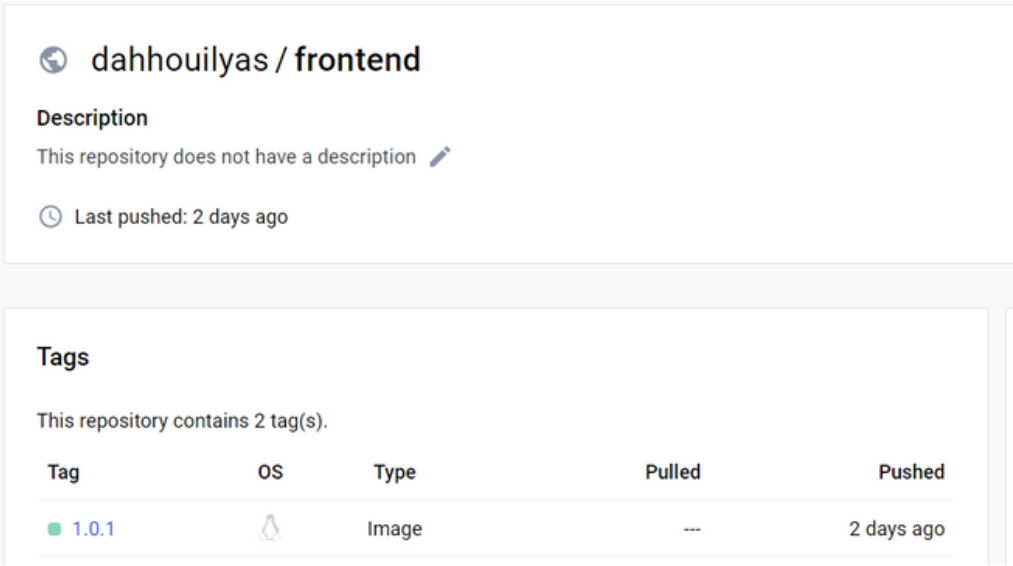
Library	Vulnerability	Severity	Status	Installed Version	Fixed Version	Title
libcrypto3 H keys or	CVE-2023-5678	MEDIUM	fixed	3.1.4-r0	3.1.4-r1	openssl: Generating excessively long X9.4 checking excessively long X9.42... https://avd.aquasec.com/nvd/cve-2023-5678
libssl3						
libx11 o a heap	CVE-2023-43787	HIGH		1.8.4-r4	1.8.7-r0	integer overflow in XCreateImage() leadin overflow https://avd.aquasec.com/nvd/cve-2023-4378
	CVE-2023-43785	MEDIUM				out-of-bounds memory access in _XkbReadKe ms() https://avd.aquasec.com/nvd/cve-2023-4378

On la tag l'image puis on la publie au docker hub

```
C:\Users\HP>docker tag frontend:latest dahhouilyas/frontend:1.0.1

C:\Users\HP>docker push dahhouilyas/frontend:1.0.1
The push refers to repository [docker.io/dahhouilyas/frontend]
cdcdb3da36aa: Pushed
4b701b99fec7: Mounted from library/nginx
01e36c0e0b84: Mounted from library/nginx
901e6dddcc99: Mounted from library/nginx
f126bda54112: Mounted from library/nginx
38067ed663bf: Mounted from library/nginx
854101110f63: Mounted from library/nginx
81fdcc81a9d0: Mounted from library/nginx
cc2447e1835a: Mounted from library/nginx
1.0.1: digest: sha256:bb133ea75e443a85d59604878e7272e3e0c9ecbd3d376467c2d5f998657eab88 size: 2199
```

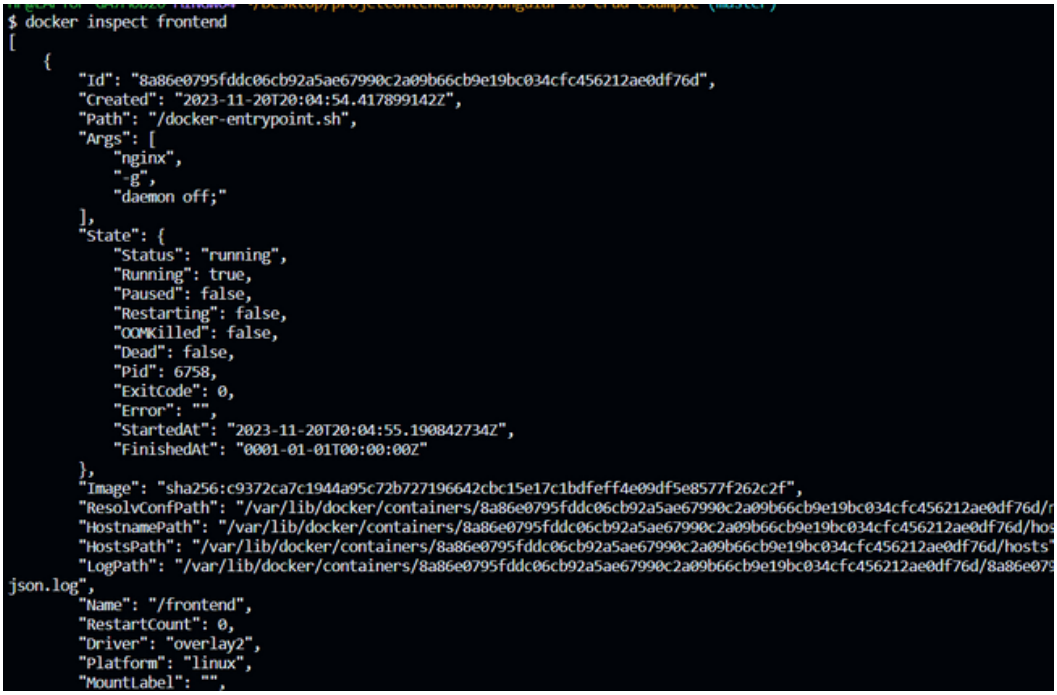
On vérifie maintenant que l'image est disponible dans le repo "frontend"



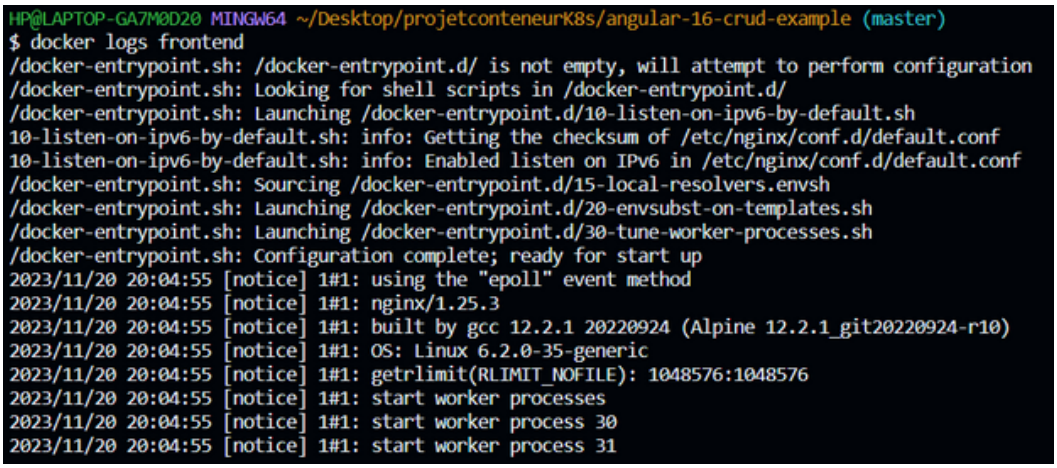
On l'instancie par un conteneur nommé frontend qui partage le même réseau avec les autres conteneurs. On fait le mapping du port 8081:80

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker run --name frontend --network projet_network -p 8081:80 -d frontend:latest
8a86e0795fddc06cb92a5ae67990c2a09b66cb9e19bc034cfc456212ae0df76d
```

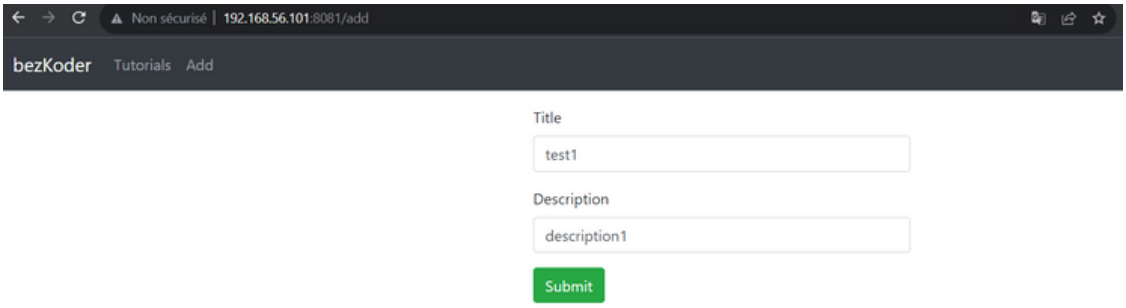
On inspect le conteneur frontend



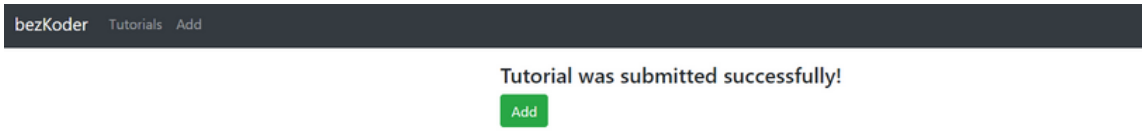
On inspect le conteneur frontend



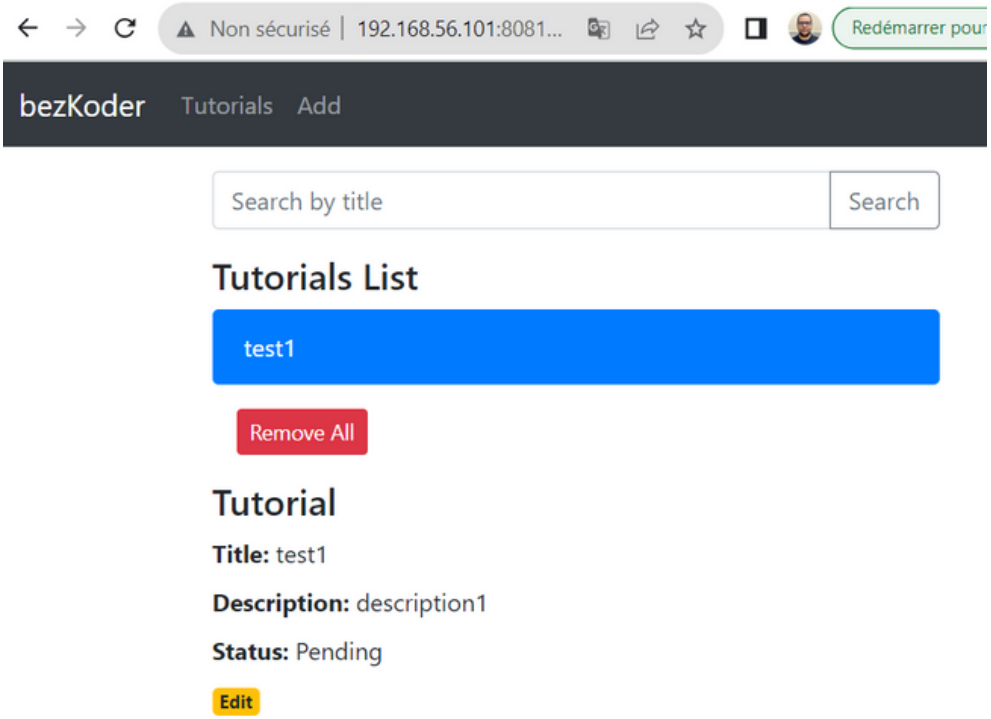
7. On s'assure maintenant que l'application fonctionne bien en essayant de s'enregistrer



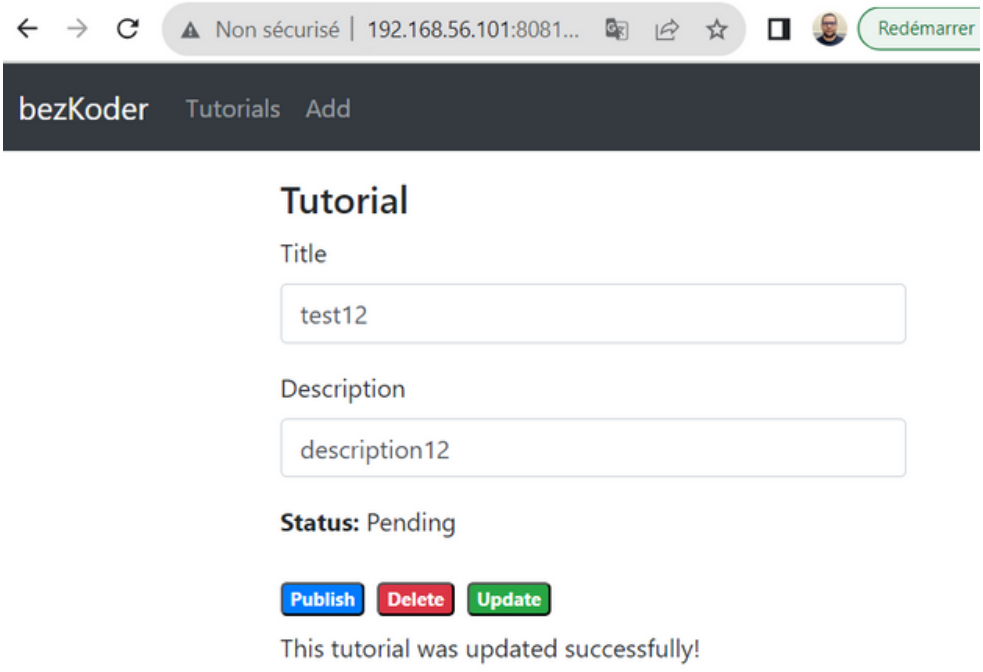
L'enregistrement passe bien et une page s'affiche



L'enregistrement est apparaître dans la page principale



On modifier les données



8. On supprime les trois conteneurs les uns après les autres

```
C:\Users\HP>docker rm frontend --force
frontend

C:\Users\HP>docker rm backend --force
backend

C:\Users\HP>docker rm mysql-container --force
mysql-container
```

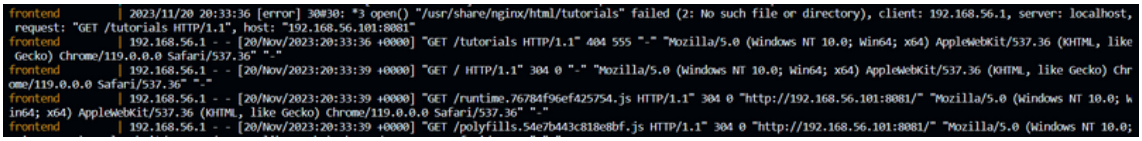
9. On redéploye l'application en utilisant le docker-compose :

   a. On crée le fichier docker compose

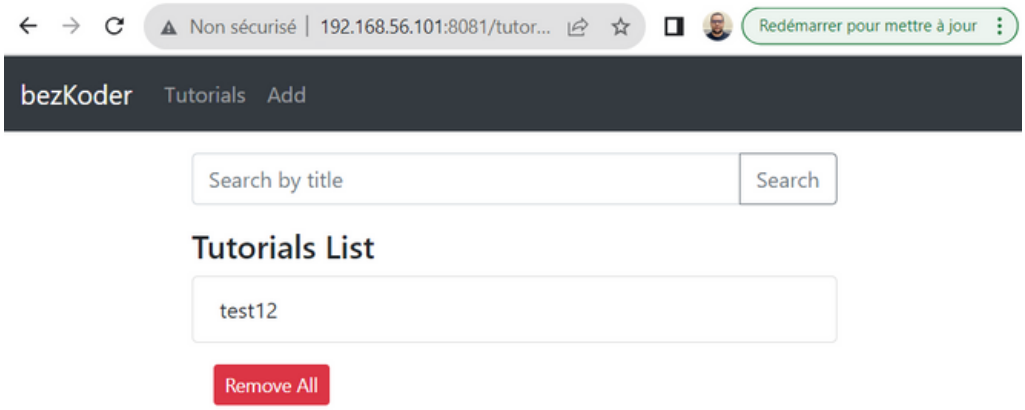
```yaml
version: "3.8"
services:
  mysql-container:
    image: mysql:latest
    container_name: mysql-container
    ports:
      - "3306:3306"
    networks:
      - projet_network
    volumes:
      - mysql-sata:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: "rootilyas"
      MYSQL_DATABASE: "testdb"
  backend:
    image: dahhouilyas/backend:1.0.3
    container_name: backend
    ports:
      - "8080:8080"
    networks:
      - projet_network
    depends_on:
      - mysql-container
  frontend:
    image: dahhouilyas/frontend:1.0.1
    container_name: frontend
    ports:
      - "8081:80"
    networks:
      - projet_network
    depends_on:
      - backend
```

```yaml
volumes:
  mysql-sata:
    driver: local
networks:
  projet_network:
    driver: bridge
```

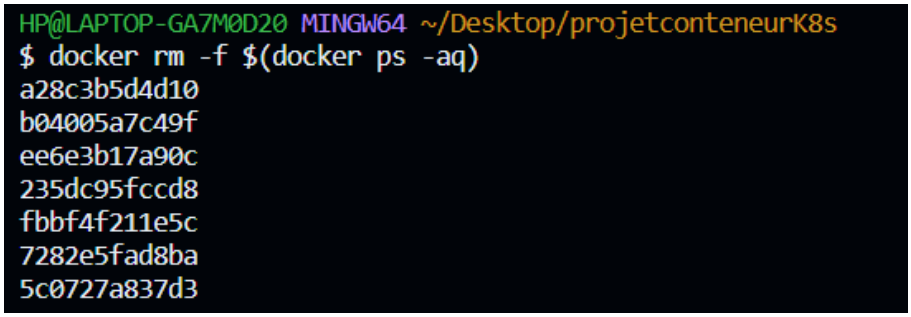
b. On l'exécute avec la commande docker compose up

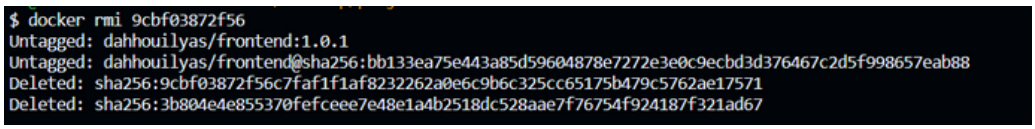


c. L'application fonctionne correctement et donne le même résultat précédent (la persistance des données)



10. On supprime les conteneurs en cours d'exécution ainsi que les images trouvant dans le docker host





```
$ docker rmi a3b6608898d6 --force
Untagged: mysql:latest
Untagged: mysql@sha256:1773f3c7aa9522f0014d0ad2bbdaf597ea3b1643c64c8ccc2123c64afd8b82b1
Deleted: sha256:a3b6608898d600759effd58768b7213bb44a6d58ab3a53495dd88e6b2042a8a4
Deleted: sha256:a6316a7b5ef7408c536c06c5f20e6138cd8725c7ed3fc895ec2540d28f596d87
Deleted: sha256:ce9f13233387d62a08d318ab3756f8acca152ed67e54537ef94ceca1e899fdf4
Deleted: sha256:92ad3da2e834e507138e715ff516025fea7019725b49bf34ff62b97a2cfa8bfd
```

```
$ docker rmi 0fe0557a5348
Untagged: dahhouilyas/backend:1.0.3
Untagged: dahhouilyas/backend@sha256:550f8b8ceb88f056791a85d65d07069e7679f1019a6ae95ddfa284381af20ad7
Deleted: sha256:0fe0557a534814bb6385cb18210a83a57c290a93a7796e756e6f82110e07efc4
Deleted: sha256:2160f56090bec0190b2c375dd491e571e5f699edd6bb5757a424cf5f7b55ddd1
Deleted: sha256:8cd3ba239c1fcb869a61f697139ff4eb594e3a3b5f2f9ba970e5b03b344b1974
Deleted: sha256:46c83b58a93bd744cdb8d0d4a13cf5e50404ee4bca2c8b0c55f46205dfa275d9
Deleted: sha256:707df847c38741d302945e10c6b85c0afed4472f4743ddf9477f9899a985d9dd
Deleted: sha256:0084405f14e42214041013f0360bb7432c0e2a08b84ac48628dd2e5f790af612
```

11. Un registre privé est un conteneur dédié au stockage d'images Docker. Sa création consiste simplement à instancier l'image "registry". Ainsi, la création d'un registre privé se résume à l'instanciation de cette image, que nous nommons ici "registry".

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s
$ docker run -d -p 5000:5000 --restart=always --name registry registry:2
Unable to find image 'registry:2' locally
2: Pulling from library/registry
96526aa774ef: Already exists
834bccaa730c: Pull complete
87a69098c0a9: Pull complete
afc17120a9f7: Pull complete
e5ac04f3acf5: Pull complete
Digest: sha256:8a60daaa55ab0df4607c4d8625b96b97b06fd2e6ca8528275472963c4ae8afa0
Status: Downloaded newer image for registry:2
3f00f3f56229804944427ca2a7f3bd9cab41d4c44ae49730786c1db614204415

HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s
$ docker ps -a
CONTAINER ID   IMAGE        COMMAND                  CREATED         STATUS         PORTS                                       NAMES
3f00f3f56229   registry:2   "/entrypoint.sh /etc…"   3 minutes ago   Up 3 minutes   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp   registry
```

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

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker tag frontend localhost:5000/registry-frontend-image

HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker tag backend localhost:5000/registry-backend-image

HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker tag mysql localhost:5000/registry-mysql-image
```

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker push localhost:5000/registry-frontend-image
Using default tag: latest
The push refers to repository [localhost:5000/registry-frontend-image]
72f5347d2c43: Pushed
4b701b99fec7: Pushed
01e36c0e0b84: Pushed
901e6dddcc99: Pushed
f126bda54112: Pushed
38067ed663bf: Pushed
854101110f63: Pushed
81fdcc81a9d0: Pushed
cc2447e1835a: Pushed
latest: digest: sha256:c180bfa1cb734ce9bcf4ffcfcfd2761bb5b78a398a8cc9bb208c5c414b230c62 size: 2199
```

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker push localhost:5000/registry-backend-image
Using default tag: latest
The push refers to repository [localhost:5000/registry-backend-image]
60370a69a1a5: Pushed
606c8b7b92db: Pushed
34f7184834b2: Pushed
5836ece05bfd: Pushed
72e830a4dff5: Pushed
latest: digest: sha256:83dc1c1757a9de6d39203c791b29d41d7639838d75c016ad199e016711fd80c7 size: 1369
```

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker push localhost:5000/registry-mysql-image
Using default tag: latest
The push refers to repository [localhost:5000/registry-mysql-image]
eff93312e249: Pushed
1af6e691b2c1: Pushed
867dc881a6cf: Pushed
235367ebc71a: Pushed
f159770d104d: Pushed
01cbaaa2e3b9: Pushed
9458eec006d0: Pushed
69a630646f65: Pushed
69ef53c77128: Pushed
d6fe63e8be63: Pushed
latest: digest: sha256:4aa5ed5c96b6f48ffc1ed58360247fcfe3e62fc122e0cabf046df9fec543de22 size: 2411
```

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 on associe d'abord le registre au réseau projet_network.

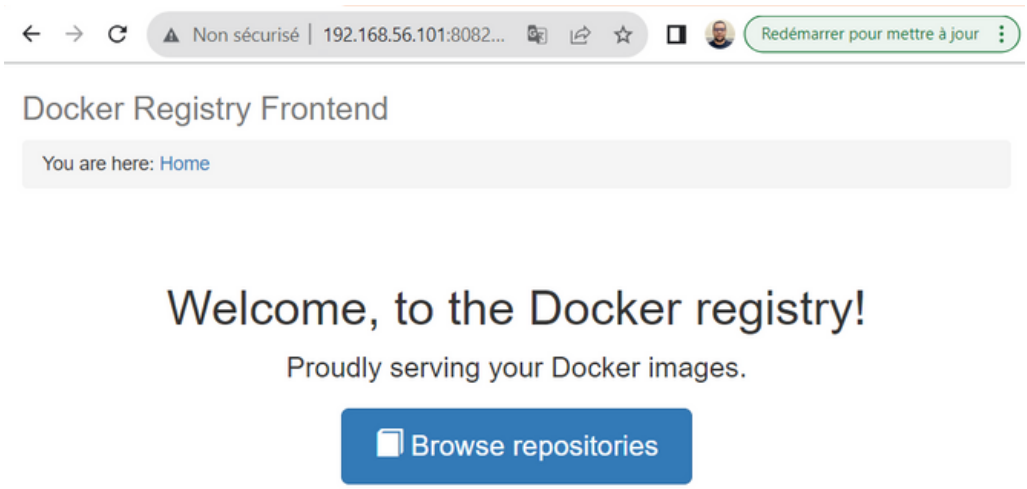
```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker network connect projet_network registry

HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker inspect projet_network
[
    {
        "Name": "projet_network",
        "Id": "e914426cacb12d138c3c4e1698ebf6aceceae297a0155fa05dda8911c50428af",
        "Created": "2023-11-19T21:20:45.469650011+01:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.19.0.0/16",
                    "Gateway": "172.19.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Ingress": false,
        "ConfigFrom": {
            "Network": ""
        },
        "ConfigOnly": false,
        "Containers": {
            "3f00f3f56229804944427ca2a7f3bd9cab41d4c44ae49730786c1db614204415": {
                "Name": "registry",
                "EndpointID": "e2a6d6653577037d8b7de5814b31b3444c593fe60b3a5fdbacf6515a5dc8d3e
4",
                "MacAddress": "02:42:ac:13:00:02",
                "IPv4Address": "172.19.0.2/16",
```

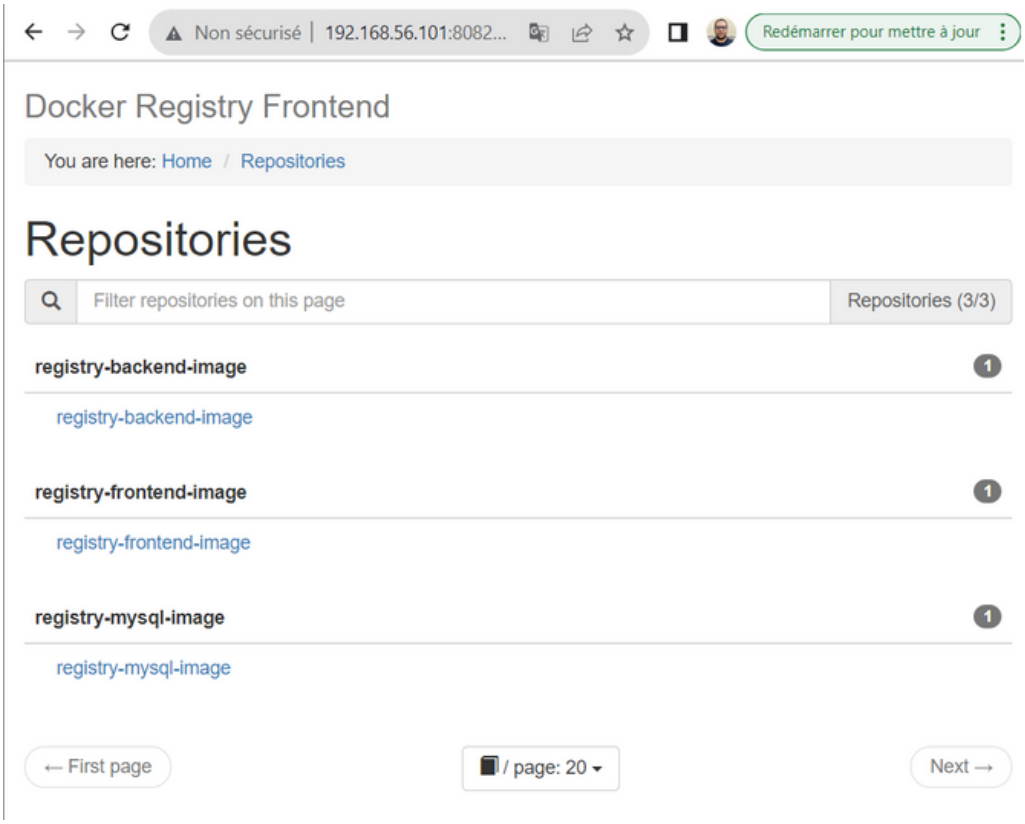
On instancie maintenant l'image konradkleine/docker-registry-frontend :v2 pour créer le conteneur private-registry-ui. Ce registre qu'on met aussi dans le network projet_network et qu'on lui spécifie le registre à visualiser ainsi que son port.

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/angular-16-crud-example (master)
$ docker run --name private-registry-ui --network projet_network -e ENV_DOCKER_REGISTRY_HOST=r
egistry -e ENV_DOCKER_REGISTRY_PORT=5000 -p 8082:80 konradkleine/docker-registry-frontend:v2
Module auth_kerb disabled.
To activate the new configuration, you need to run:
  service apache2 restart
Module ssl already disabled
Enabling module rewrite.
To activate the new configuration, you need to run:
  service apache2 restart
Stopping web server: apache2.
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using
 172.19.0.3. Set the 'ServerName' directive globally to suppress this message
192.168.56.1 - - [20/Nov/2023:21:38:37 +0000] "GET / HTTP/1.1" 200 1153 "-" "Mozilla/5.0 (Wind
```

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



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)

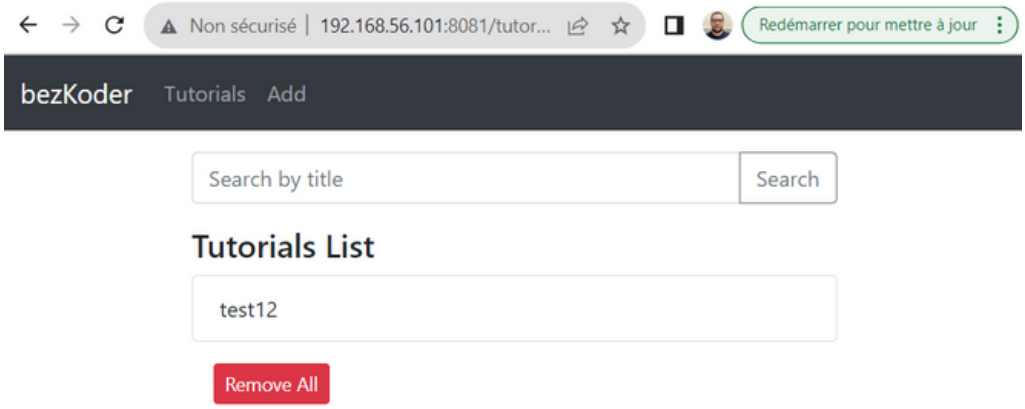


12. On change les images à tirer dans le fichier docker-compose.yaml

```yaml
services:
  mysql-container:
    image: mysql:latest
    container_name: localhost:5000/registry-mysql-image:latest
    ports:
      - "3306:3306"
    networks:
      - projet_network
    volumes:
      - mysql-sata:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: "rootilyas"
      MYSQL_DATABASE: "testdb"
  backend:
    image: localhost:5000/registry-backend-image:latest
    container_name: backend
    ports:
      - "8080:8080"
    networks:
      - projet_network
    depends_on:
      - mysql-container
  frontend:
    image: localhost:5000/registry-frontend-image:latest
    container_name: frontend
    ports:
      - "8081:80"
    networks:
      - projet_network
    depends_on:
      - backend
```

On exécute la commande docker compose up et on constate que les conteneurs se créent

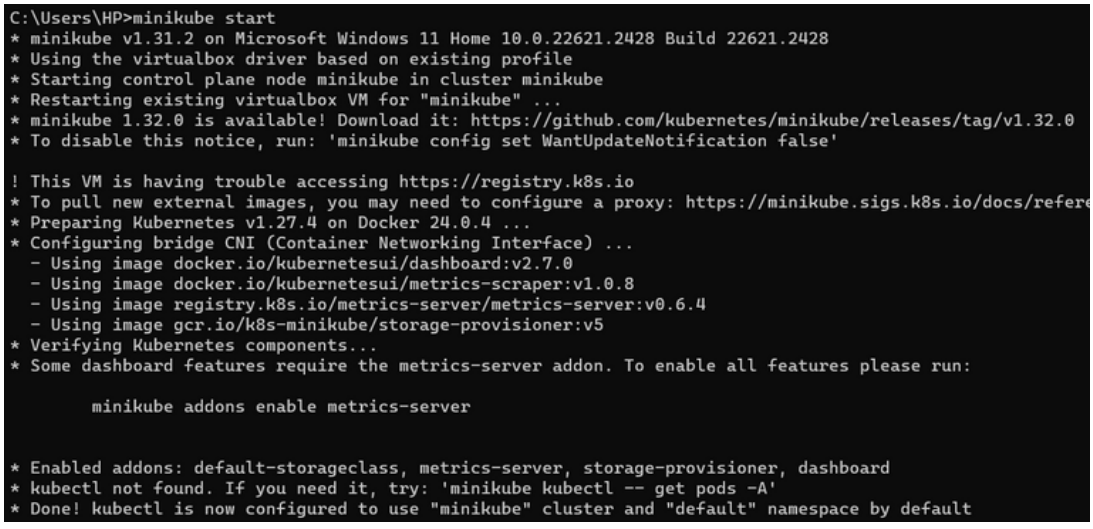
On consulte l'application par le navigateur pour s'assurer de son fonctionnement



on voit que l'application est bien fonctionner avec le même data de début, car on a utilisé la même volume

Partie 2

On lance d'abord un cluster minikube puis on crée un namespace "exam"



```
C:\Users\HP>kubectl create namespace exam
namespace/exam created
```

Tout d'abord, on va déployer la base de données, on va utiliser StatefulSet qui permet de déployer des applications nécessitant des noms de pods stables, une persistance des données, en utilise aussi Secret pour les données confidentiels de l'authentification, en fin on utilise le Service pour permet à les autre pods d'acceder à la base de donné. dans un fichier "database.yaml"

```yaml
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
  namespace: exam
type: Opaque
data:
  MYSQL_ROOT_USERNAME: cm9vdA==
  MYSQL_ROOT_PASSWORD: cm9vdA==
```

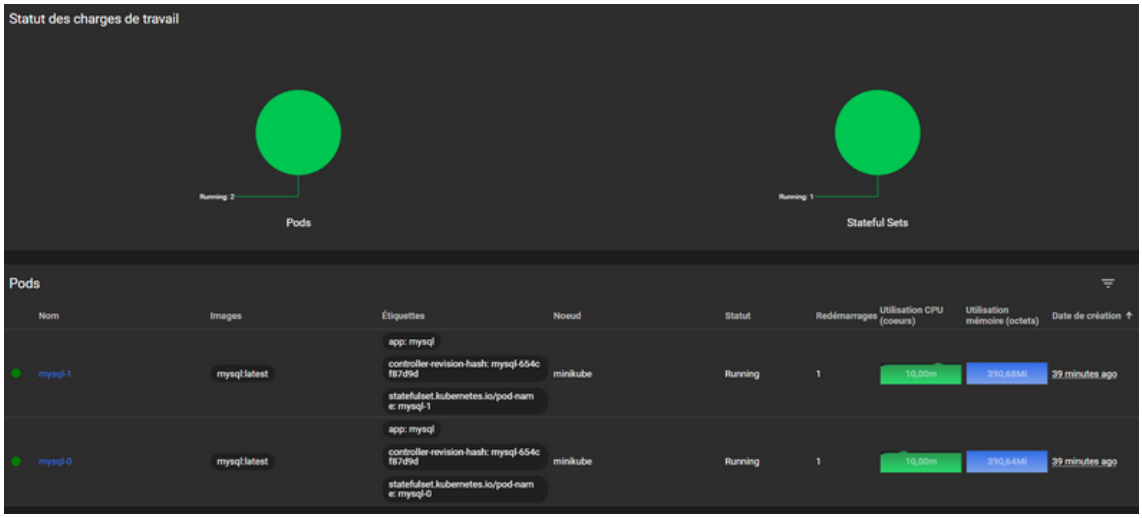
```yaml
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
  namespace: exam
spec:
  selector:
    app: mysql
  ports:
    - protocol: TCP
      port: 3306
      targetPort: 3306
```

```yaml
apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: mysql-statefulset
  namespace: exam
spec:
  serviceName: mysql-service  # Correctio
  replicas: 2
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:latest
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_ROOT_PASSWORD
          - name: MYSQL_DATABASE
            value: testdb
        ports:
          - containerPort: 3306
            name: db-port
        volumeMounts:
        - name: mysql-data
          mountPath: /var/lib/mysql
  volumeClaimTemplates:
  - metadata:
      name: mysql-data
    spec:
      accessModes: ["ReadWriteOnce"]
      resources:
        requests:
          storage: 1Gi
```

en suite on lance la commande kubectl apply –f « database.yaml »

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl.exe apply -f database.yaml
service/mysql-service created
secret/mysql-secret created
statefulset.apps/mysql created
```

on vérifier dans minikube dashboard, on utilise la commande <minikube dashboard>, on voit que les deux pods correspondant à mysql sont bien créés



en suit, pour déployer le backend, on doit faire des petites modifications sur le code source du fichier « application.properties ». On va utiliser des variables d'environnement pour la connexion avec la base de données, cette modification va nous permettre d'utiliser les objets « Configmap » et « Secret » pour le déploiement de notre application

```
spring.datasource.url=${SPRING_DATASOURCE_URL}
spring.datasource.username=${SPRING_DATASOURCE_USERNAME}
spring.datasource.password=${MYSQL_ROOT_PASSWORD}
```

L'image Docker « dahhouilyas/backend:1.0.6 » est celle qui couvre ces changements

Le manifest YAML du backend (contenant : ConfigMap, Service et Deployment)

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-service
  namespace: exam
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
  type: ClusterIP
```

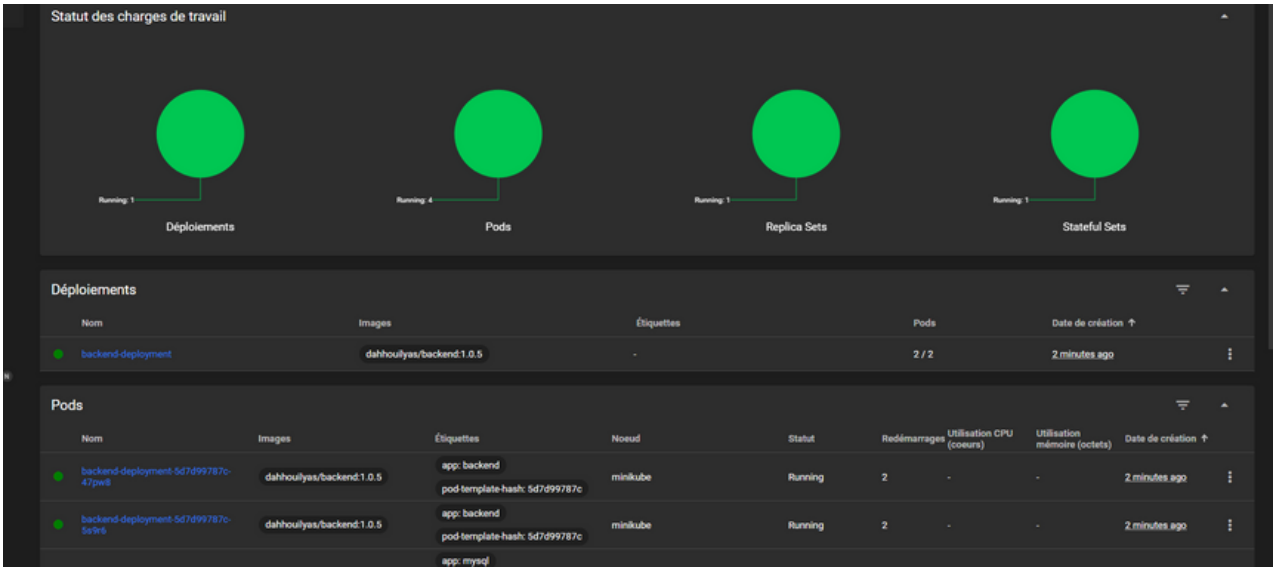
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: backend-config
  namespace: exam
data:
  SPRING_DATASOURCE_URL: jdbc:mysql://mysql-service:3306/testdb?useSSL=fal
  SPRING_DATASOURCE_USERNAME: root
```

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  namespace: exam
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
      - name: backend
        image: dahhouilyas/backend:1.0.6
        ports:
        - containerPort: 8080
        env:
          - name: MYSQL_ROOT_PASSWORD
            valueFrom:
              secretKeyRef:
                name: mysql-secret
                key: MYSQL_ROOT_PASSWORD
          - name: SPRING_DATASOURCE_URL
            valueFrom:
              configMapKeyRef:
                name: backend-config
                key: SPRING_DATASOURCE_URL
          - name: SPRING_DATASOURCE_USERNAME
            valueFrom:
              configMapKeyRef:
                name: backend-config
                key: SPRING_DATASOURCE_USERNAME
```

en suite on lance la commande kubectl apply –f « backend.yaml »

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl.exe apply -f backend.yaml
configmap/backend-config created
service/backend-service created
deployment.apps/backend-deployment created
```

on vérifier dans minikube dashboard, on voit que les deux pods correspondant à backend sont bien créés



en suite, on déploie le frontend. Le manifest YAML du frontend (contenant : Service et un Deployment)

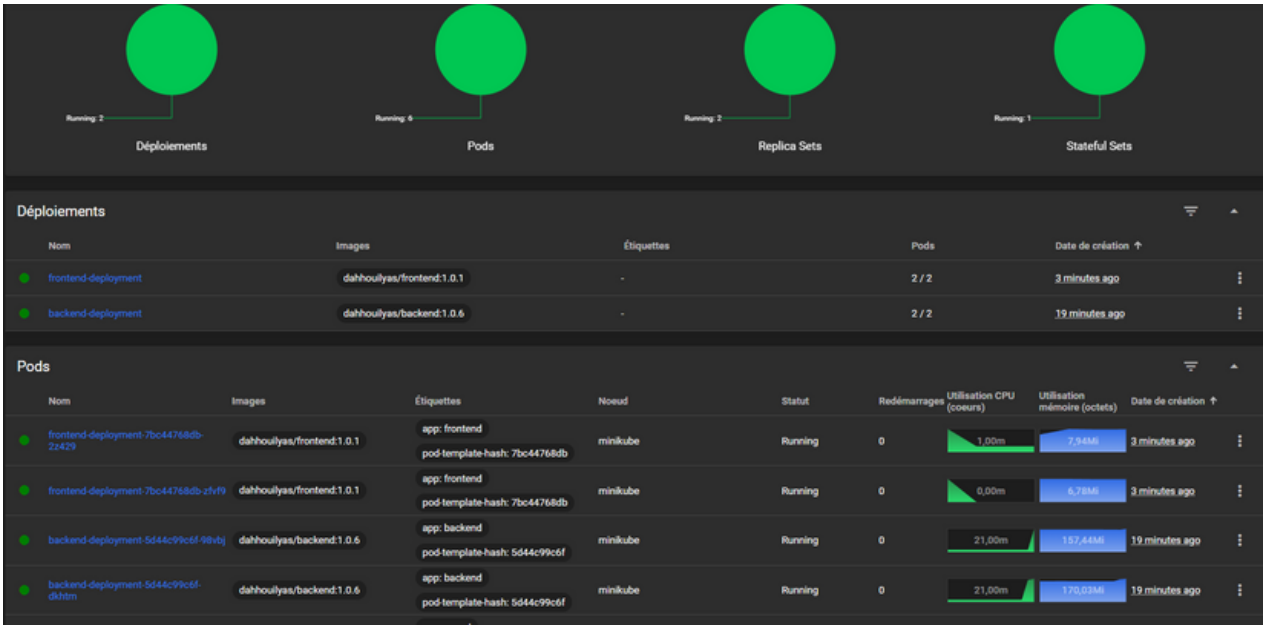
```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
  namespace: exam
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
---
```

```yaml
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  namespace: exam
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: dahhouilyas/frontend:1.0.1
        ports:
        - containerPort: 80
          name: http-port
```

en suite on lance la commande kubectl apply –f « frontend.yaml »

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl.exe apply -f frontend.yaml
service/frontend-service created
deployment.apps/frontend-deployment created
```

on vérifier dans minikube dashboard, on voit que les deux pods correspondant à frontend sont bien créés



2. Pour définir des quotas pour la consommation des ressources mémoire et CPU de chaque Pod exécuté dans le namespace "exam" dans Kubernetes, on va utiliser les objets ResourceQuota. dans un fichier nommé <<ressource-quota.yaml>>

ressource-quota.yaml > {} spec > {} hard

```yaml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: exam-resource-quota
  namespace: exam
spec:
  hard:
    requests.cpu: "1"
    limits.cpu: "2"
    requests.memory: 1Gi
    limits.memory: 2Gi
```

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl.exe apply -f ressource-quota.yaml
resourcequota/exam-resource-quota created
```

On utilisant la commande <<kubectl describe resourcequota –n exam>> pour afficher les informations sur le quota de ressources dans le namespace "exam"

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl.exe describe resourcequota -n exam
Name:            exam-resource-quota
Namespace:       exam
Resource         Used  Hard
--------         ----  ----
limits.cpu       0     2
limits.memory    0     2Gi
requests.cpu     0     1
requests.memory  0     1Gi
```

3. Pour contrôler l'accès aux ressources existantes dans le namespace « exam » par un rôle RBAC, on va utiliser kind "Rôle" et "RoleBinding"

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: exam
  name: exam-role
rules:
  - apiGroups: [""]
    resources: ["pods", "services", "configmaps", "secrets"]
    verbs: ["get", "list", "create", "update", "delete"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: exam-rolebinding
  namespace: exam
subjects:
  - kind: User
    name: user1
    apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: exam-role
  apiGroup: rbac.authorization.k8s.io
```

Ce code Kubernetes établit des règles d'autorisation RBAC dans l'espace de noms "exam". Il crée un rôle (exam-role) qui accorde à l'utilisateur "user1" la capacité de lire, lister, créer, mettre à jour et supprimer des ressources Kubernetes spécifiques telles que les pods, services, configmaps et secrets. Le lien de rôle (exam-rolebinding) attache ce rôle à l'utilisateur "user1" dans le même namespace, définissant ainsi ses autorisations spécifiques.

4. Pour spécifier un budget de perturbation, on crée le fichier yaml qui contient la configuration suivante:

```yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: mysql-pdb
  namespace: exam
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: mysql

---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: backend-pdb
  namespace: exam
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: backend
---
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: frontend-pdb
  namespace: exam
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: frontend
```

Ainsi cette configuration assurent qu'au moins deux pods de l'application restent disponibles en permanence pendant toute perturbation planifiée, ce qui contribue à maintenir la disponibilité et la stabilité de l'application même lors d'opérations de maintenance ou de mise à jour.

Par la suite, on exécute la commande <<kubectl apply –f budget.yaml>>

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube (master)
$ C:/Users/HP/kubectl apply -f budget.yaml
poddisruptionbudget.policy/mysql-pdb created
poddisruptionbudget.policy/backend-pdb created
poddisruptionbudget.policy/frontend-pdb created
```

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube (master)
$ C:/Users/HP/kubectl get poddisruptionbudget -n exam
NAME           MIN AVAILABLE   MAX UNAVAILABLE   ALLOWED DISRUPTIONS   AGE
backend-pdb    2               N/A               0                     113s
frontend-pdb   2               N/A               0                     113s
mysql-pdb      2               N/A               0                     113s
```

5. Pour créer et configurer Liveness Prob, Readiness Prob et un Startup Prob. On ajoutent au fichier frontend.yaml, backend.yaml, database.yaml, dans le section "containers " de la configuration suivant :

```yaml
livenessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 3
  periodSeconds: 5
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 3
  periodSeconds: 5
startupProbe:
  exec:
    command:
    - mkdir
    - /tmp
  failureThreshold: 30
  periodSeconds: 5
```

cette configuration, des sondes de santé sont définies pour surveiller la disponibilité, la vitalité et le démarrage d'un conteneur. readinessProbe utilise la commande "cat /tmp/healthy" toutes les 10 secondes, avec un délai initial de 5 secondes. De même, livenessProbe utilise la même commande toutes les 15 secondes, avec un délai initial de 10 secondes. startupProbe crée le fichier /tmp/healthy avec la commande "touch" lors du démarrage, échouant après 30 tentatives consécutives infructueuses. Ces sondes permettent à Kubernetes de garantir que le conteneur est prêt à recevoir du trafic, reste en cours d'exécution et a démarré correctement

Par la suite on utilise la commande <<describe pod nom_de_pods –n exam>> pour vérifier les événements générés par les différents pots de l'application.

```
$ C:/Users/HP/kubectl describe pod mysql-statefulset-1 -n exam
Name:             mysql-statefulset-1
Namespace:        exam
Priority:         0
Service Account:  default
Node:             minikube/192.168.59.100
Start Time:       Sun, 26 Nov 2023 22:31:45 +0100
Labels:           app=mysql
                  controller-revision-hash=mysql-statefulset-5d94ddbc47
                  statefulset.kubernetes.io/pod-name=mysql-statefulset-1
Annotations:      <none>
Status:           Running
IP:               10.244.0.148
IPs:
  IP:           10.244.0.148
Controlled By:  StatefulSet/mysql-statefulset
Containers:
  mysql:
    Container ID:   docker://509d1aaef814021ed6a4dd385b492e37201536b2c6a2b67e3ac55a9b604f2050
    Image:          mysql:latest
    Image ID:       docker-pullable://mysql@sha256:1773f3c7aa9522f0014d0ad2bbdaf597ea3b1643c64c8ccc21
    Port:           3306/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 26 Nov 2023 22:31:50 +0100
    Ready:          True
    Restart Count:  0
    Liveness:       exec [cat /tmp/healthy] delay=3s timeout=1s period=5s #success=1 #failure=3
    Readiness:      exec [cat /tmp/healthy] delay=3s timeout=1s period=5s #success=1 #failure=3
    Startup:        exec [touch /tmp/healthy] delay=0s timeout=1s period=5s #success=1 #failure=30
    Environment:
      MYSQL_ROOT_PASSWORD:  <set to the key 'MYSQL_ROOT_PASSWORD' in secret 'mysql-secret'>  Optional
      MYSQL_DATABASE:       testdb
```

```
$ C:/Users/HP/kubectl describe pod backend-deployment-d8bdcfbb9-qc7kq -n exam
Name:             backend-deployment-d8bdcfbb9-qc7kq
Namespace:        exam
Priority:         0
Service Account:  default
Node:             minikube/192.168.59.100
Start Time:       Sun, 26 Nov 2023 21:37:11 +0100
Labels:           app=backend
                  pod-template-hash=d8bdcfbb9
Annotations:      <none>
Status:           Running
IP:               10.244.0.136
IPs:
  IP:           10.244.0.136
Controlled By:  ReplicaSet/backend-deployment-d8bdcfbb9
Containers:
  backend:
    Container ID:   docker://5158acb033c6e6d23ddd35e715edd5356e3aba75db86bd2659ae139f780a9128
    Image:          dahhouilyas/backend:1.0.6
    Image ID:       docker-pullable://dahhouilyas/backend@sha256:8b5986c35a5896651f2436f1882ea1bad1b31c538901f637
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 26 Nov 2023 21:37:12 +0100
    Ready:          True
    Restart Count:  0
    Liveness:       exec [cat /tmp/healthy] delay=10s timeout=1s period=15s #success=1 #failure=3
    Readiness:      exec [cat /tmp/healthy] delay=5s timeout=1s period=10s #success=1 #failure=3
    Startup:        exec [touch /tmp/healthy] delay=0s timeout=1s period=10s #success=1 #failure=30
    Environment:
      MYSQL_ROOT_PASSWORD:         <set to the key 'MYSQL_ROOT_PASSWORD' in secret 'mysql-secret'>
      SPRING_DATASOURCE_URL:       <set to the key 'SPRING_DATASOURCE_URL' of config map 'backend-config'>
      SPRING_DATASOURCE_USERNAME:  <set to the key 'SPRING_DATASOURCE_USERNAME' of config map 'backend-config'>
    Mounts:
```

```
$ C:/Users/HP/kubectl describe pod frontend-deployment-547cc6c4f-6ctp8 -n exam
Name:             frontend-deployment-547cc6c4f-6ctp8
Namespace:        exam
Priority:         0
Service Account:  default
Node:             minikube/192.168.59.100
Start Time:       Sun, 26 Nov 2023 21:47:35 +0100
Labels:           app=frontend
                  pod-template-hash=547cc6c4f
Annotations:      <none>
Status:           Running
IP:               10.244.0.139
IPs:
  IP:           10.244.0.139
Controlled By:  ReplicaSet/frontend-deployment-547cc6c4f
Containers:
  frontend:
    Container ID:   docker://a5ce36f662d2f2cb9a4e8657944e62e26eff3f2f9272f85bc8d33a4356229d02
    Image:          dahhouilyas/frontend:1.0.1
    Image ID:       docker-pullable://dahhouilyas/frontend@sha256:bb133ea75e443a85d59604878e7272e3e0c9ecbd3d376467c2d5f998657eab88
    Port:           80/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Sun, 26 Nov 2023 21:47:36 +0100
    Ready:          True
    Restart Count:  0
    Liveness:       exec [cat /tmp/healthy] delay=3s timeout=1s period=5s #success=1 #failure=3
    Readiness:      exec [cat /tmp/healthy] delay=3s timeout=1s period=5s #success=1 #failure=3
    Startup:        exec [touch /tmp/healthy] delay=0s timeout=1s period=5s #success=1 #failure=30
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-96jnh (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
```

6. Avant de créer l'entrée, nous devons créer son contrôleur, nous exécutons donc la commande minikube addonsenable ingress

```
C:\Users\HP>minikube addons enable ingress
* ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
  - Using image registry.k8s.io/ingress-nginx/controller:v1.8.1
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20230407
  - Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20230407
* Verifying ingress addon...
* The 'ingress' addon is enabled
```

par la suite on va créer deux igress un pour le service de frontend et l'autre pour le service backend, pour exposer les deux à l'extérieur

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-api-ingress
  namespace: exam
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/add-base-url: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: backend-api.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: backend-service
                port:
                  number: 8080
```

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: app-ingress
  namespace: exam
  annotations:
    spec.ingressClassName: "nginx"
    nginx.ingress.kubernetes.io/add-base-url: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
    - host: dahhou-ilyas.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: frontend-service
                port:
                  number: 80
```

On doit changer le endpoint qui existe dans le code de frontend, et reconstruire l'image

```
const baseUrl = 'http://backend-api.com/api/tutorials';
```

On doit changer le endpoint qui existe dans le code de frontend, et reconstruire l'image

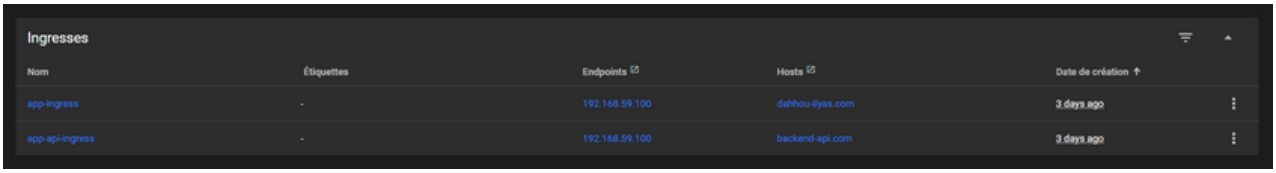
Maintenant pour accéder à notre application depuis le nom de domaine qu'on a défini, on doit modifier le fichier « C:\Windows\System32\drivers\etc\hosts » sur notre machine, ce fichier est utilisé pour stocker les noms d'hôtes et les adresses IP correspondantes, on ajoute les deux lignes suivante

```
192.168.57.99 gateway.docker.internal
192.168.59.100 dahhou-ilyas.com
192.168.59.100 backend-api.com
# To allow the same kube context to work on the host and the container:
127.0.0.1 kubernetes.docker.internal
```

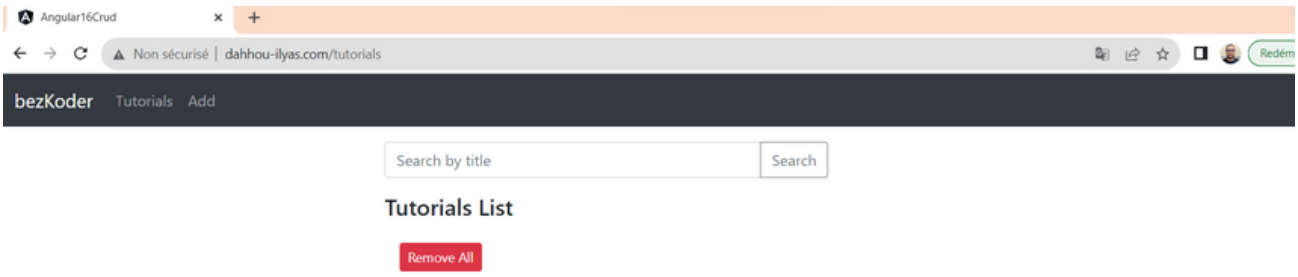
avec 192.168.59.100 est l'adresse IP de minikube

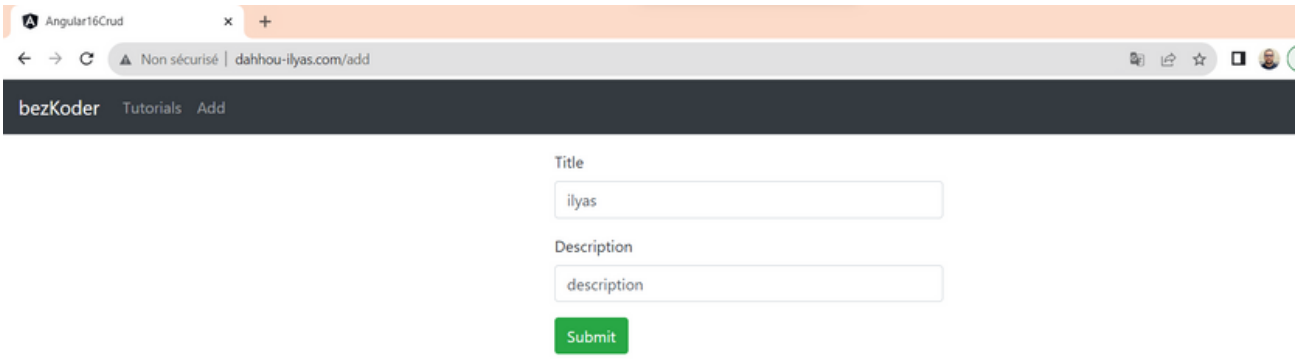
Par suite on exécute la commande ""kubectl apply -f < nom-de-fichier >""

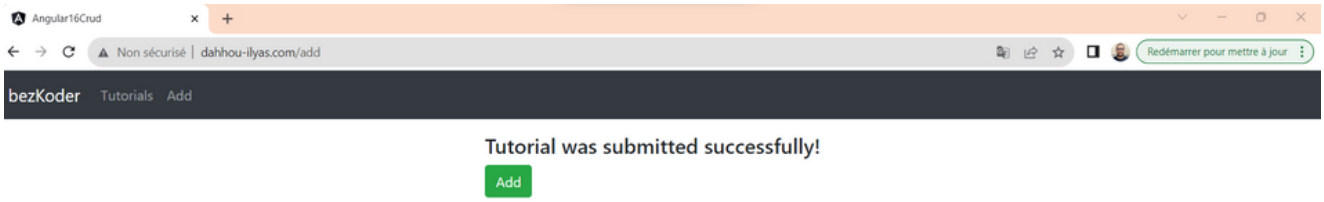
on voit que les deux ingress sont bien créés

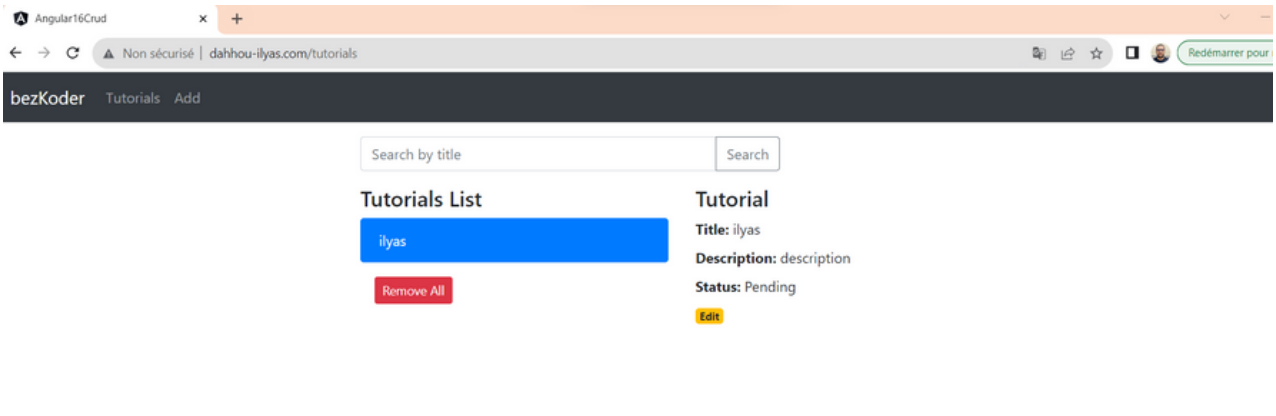


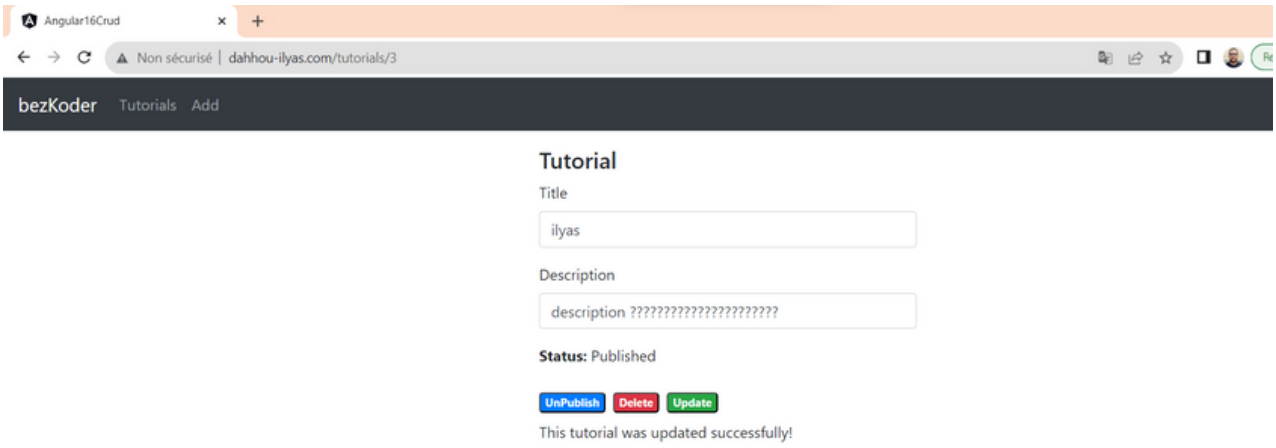
7. Nous testons l'application











8. Pour créer et gérer la partie base de données de l'application en utilisant le concept de : Operator

a. premièrement, on va installer la définition de ressource personnalisée (CRD) utilisée par MySQL Operator pour Kubernetes.

kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-crds.yaml

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-crds.yaml
customresourcedefinition.apiextensions.k8s.io/innodbclusters.mysql.oracle.com configured
customresourcedefinition.apiextensions.k8s.io/mysqlbackups.mysql.oracle.com unchanged
customresourcedefinition.apiextensions.k8s.io/clusterkopfpeerings.zalando.org unchanged
customresourcedefinition.apiextensions.k8s.io/kopfpeerings.zalando.org unchanged
```

b. ensuite, on déploie MySQL Operator pour Kubernetes :

kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/deploy/deploy-operator.yaml

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl apply -f https://raw.githubusercontent.com/mysql/mysql-operator/trunk/de
ploy/deploy-operator.yaml
clusterrole.rbac.authorization.k8s.io/mysql-operator unchanged
clusterrole.rbac.authorization.k8s.io/mysql-sidecar unchanged
clusterrolebinding.rbac.authorization.k8s.io/mysql-operator-rolebinding unchanged
clusterkopfpeering.zalando.org/mysql-operator unchanged
namespace/mysql-operator created
serviceaccount/mysql-operator-sa created
deployment.apps/mysql-operator created
```

on vérifier que l'opérateur s'exécute en vérifiant le déploiement dans l'espace de noms mysql-operator :

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl get deployment -n mysql-operator mysql-operator
NAME             READY   UP-TO-DATE   AVAILABLE   AGE
mysql-operator   1/1     1            1           71s
```

c. Ensuite, pour créer un cluster MySQL InnoDB (est une solution de clustering intégrée pour MySQL, qui facilite la création, la gestion et la maintenance de clusters de bases de données MySQL.), on crée d'abord un secret avec les informations d'identification d'un utilisateur root MySQL utilisé pour effectuer des tâches administratives dans le cluster. Par exemple :

```
kubectl create secret generic mypwds \
      --from-literal=rootUser=root \
      --from-literal=rootHost=% \
      --from-literal=rootPassword="sakila"
```

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl create secret generic mypwds \
>         --from-literal=rootUser=root \
>         --from-literal=rootHost=% \
>         --from-literal=rootPassword="sakila"
error: failed to create secret secrets "mypwds" already exists
```

On définit notre cluster MySQL InnoDB, qui fait référence au secret. Par exemple :

```yaml
apiVersion: mysql.oracle.com/v2
kind: InnoDBCluster
metadata:
  name: mycluster
spec:
  secretName: mypwds
  tlsUseSelfSigned: true
  instances: 2
  router:
    instances: 1
```

en suite, on exécute la commande
kubectl apply –f mycluster.yaml

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl apply -f mycluster.yaml
innodbcluster.mysql.oracle.com/mycluster created
```

on vérifier que le cluster est bien crée

```
HP@LAPTOP-GA7M0D20 MINGW64 ~/Desktop/projetconteneurK8s/minikube
$ C:/Users/HP/kubectl get innodbcluster --watch
NAME      STATUS        ONLINE  INSTANCES  ROUTERS  AGE
mycluster  PENDING       0       1          1        22s
mycluster  PENDING       0       1          1        32s
mycluster  INITIALIZING  0       1          1        32s
mycluster  INITIALIZING  0       1          1        32s
mycluster  INITIALIZING  0       1          1        32s
mycluster  INITIALIZING  0       1          1        32s
mycluster  ONLINE        1       1          1        2m
```

Conclusion

À la clôture de ce projet, nous avons acquis une expertise solide dans la conteneurisation avec Docker et la gestion avancée des conteneurs avec Kubernetes. Cette combinaison offre une solution puissante pour la création, le déploiement et la gestion efficace d'applications dans des environnements dynamiques et évolutifs.

lien github: https://github.com/dahhou-ilyas/codes-projet-docker