

Projet micro-service

SOMMAIRE

I.	Introduction	3
II.	Démarrer Minikube	4
1.	Lancement de minikube avec docker	4
2.	Démarrer le dashboard de minikube	4
III.	Publier l'image de l'application sur Docker Hub	5
1.	Création d'un Dockerfile pour stocker notre application sous forme d'image	5
2.	Construction de notre image	5
3.	Ajout d'un tag à notre image pour l'identifier	6
4.	Publication de l'image sur Docker Hub	6
IV.	Configurer l'application avec Kubernetes	7
1.	Création du fichier web-service.yaml	7
2.	Création du fichier web-deployment.yaml	7
V.	Configurer une base de données PostgreSQL avec Kubernetes	8
1.	Création du fichier db-deployment.yaml	8
2.	Appliquer le fichier YAML	8
3.	Vérifier que le pod est actif	8
4.	Création du fichier db-service.yaml	9
VI.	Configurer pgAdmin avec Kubernetes	10
1.	Création du fichier pgadmin-deployment.yaml	10
2.	Création du fichier pgadmin-service.yaml	10
3.	Accéder à pgAdmin depuis le navigateur	11
4.	Ajouter la base de données dans pgAdmin	11
VII.	Connecter l'application web à la base de données	13
1.	Fichier Database.js	13
VIII.	Déploiement de l'application	14
1.	Appliquer les fichiers de configuration	14
2.	Vérification en ligne de commande	14
3.	Vérification avec Minikube	15

4.	Migration de la base de données.....	15
5.	Accéder à l'application	16
IX.	L'application web	18
1.	La page de connexion	18
2.	La page principale	18

I. Introduction

L'objet de ce projet dans la matière Micro-Service est de créer, déployer et orchestrer une application web conteneurisée. Ce rapport constitue un guide d'utilisation détaillé ainsi qu'un guide de maintenance.

Pour ce projet, nous avons mis en place une application web **ToDoList** qui permet à un utilisateur de gérer des tâches. Sur cette application, on retrouve plusieurs fonctionnalités simples. L'utilisateur doit commencer par se connecter en saisissant son nom et son mot de passe. Ensuite, il arrive sur la page la plus intéressante : celle de la gestion des tâches. L'application permet à l'utilisateur d'ajouter des tâches, de les afficher, de les supprimer et de les marquer par le statut terminé.

II. Démarrer Minikube

1. Lancement de minikube avec docker

Entrer dans un terminal en mode administrateur :

```
minikube start --driver=docker
```

Cela affichera :

```
minikube start --driver=docker
sortie : 💡 Pour extraire de nouvelles images externes, vous devrez peut-être
configurer un proxy :
https://minikube.sigs.k8s.io/docs/reference/networking/proxy/ 🌐
Préparation de Kubernetes v1.33.1 sur Docker 28.1.1... ▪ Génération des
certificats et des clés ▪ Démarrage du plan de contrôle ... ▪ Configuration des
règles RBAC ... 🔗 Configuration de bridge CNI (Container Networking
Interface)... 🔍 Vérification des composants Kubernetes... ▪ Utilisation de
l'image gcr.io/k8s-minikube/storage-provisioner:v5 🌟 Modules activés:
storage-provisioner, default-storageclass ! C:\Program
Files\Docker\Docker\resources\bin\kubectl.exe est la version 1.31.4, qui
peut comporter des incompatibilités avec Kubernetes 1.33.1. ▪ Vous voulez
kubectl v1.33.1 ? Essayez 'minikube kubectl -- get pods -A' 🏁 Terminé !
kubectl est maintenant configuré pour utiliser "minikube" cluster et espace
de noms "default" par défaut.
```

2. Démarrer le dashboard de minikube

Dans le même terminal, entrer :

```
minikube dashboard
```

Cela affichera :

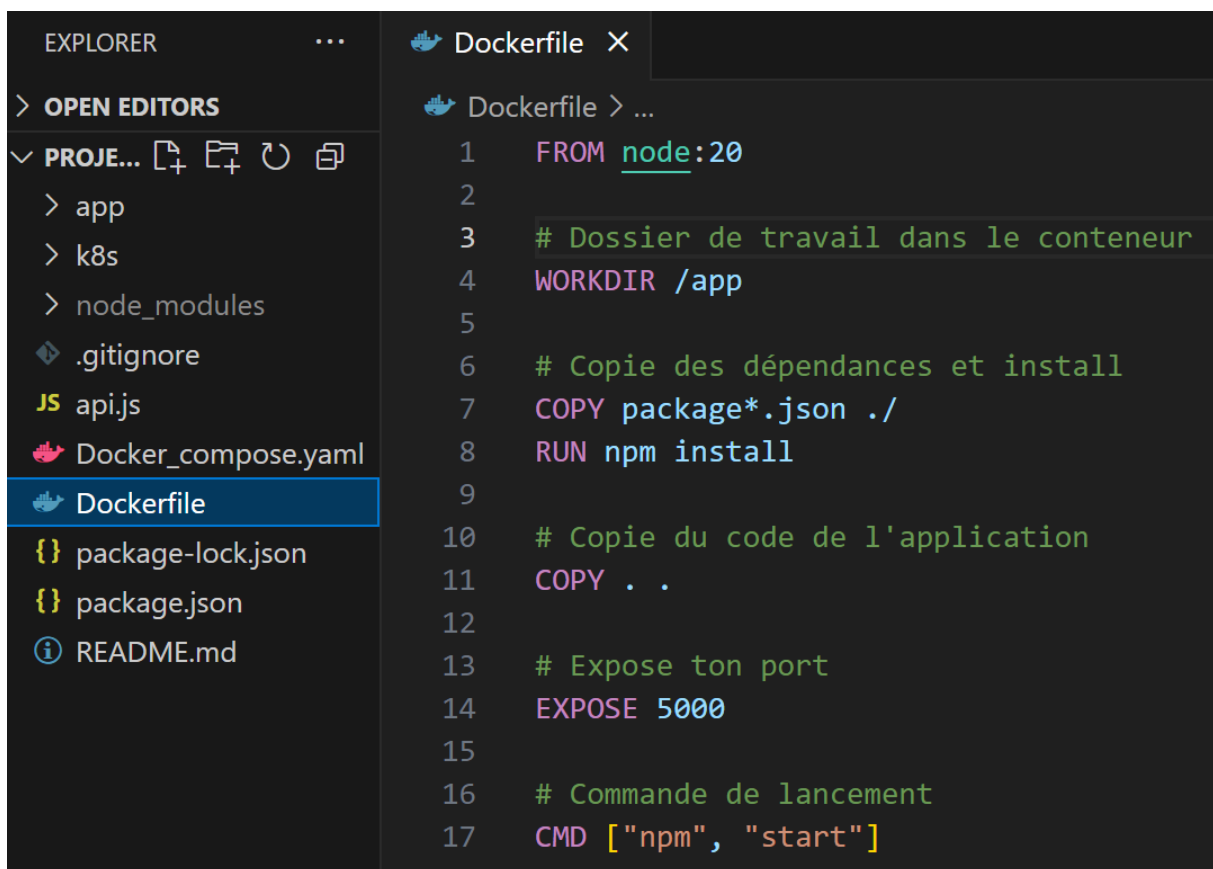
```
👁️ Vérification de l'état du tableau de bord...
🚀 Lancement du proxy...
👁️ Vérification de l'état du proxy...
🌐 Ouverture de http://127.0.0.1:57457/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/ dans votre navigateur par
défaut..
```

III. Publier l'image de l'application sur Docker Hub

Pour déployer notre application, Kubernetes a besoin d'une image Docker qui contient le code de notre application. Grâce à cette image, il ne sera plus nécessaire d'avoir le code de l'application sur les ordinateurs où l'application est déployée.

1. Création d'un Dockerfile pour stocker notre application sous forme d'image

Nous avons donc créé un Docker file afin de pouvoir stocker notre application sous forme d'image Docker :



```
1 FROM node:20
2
3 # Dossier de travail dans le conteneur
4 WORKDIR /app
5
6 # Copie des dépendances et install
7 COPY package*.json ./
8 RUN npm install
9
10 # Copie du code de l'application
11 COPY . .
12
13 # Expose ton port
14 EXPOSE 5000
15
16 # Commande de lancement
17 CMD ["npm", "start"]
```

2. Construction de notre image

Dans un nouveau terminal, entrer :

```
docker build -t todo-list:latest .
```

Cela affichera :

```

PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes> docker build -t todo-list:latest .
[+] Building 2.8s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.1s
=> => transferring dockerfile: 315B                               0.0s
=> [internal] load metadata for docker.io/library/node:20         0.1s
=> [internal] load .dockerignore                                  0.0s
=> => transferring context: 2B                                     0.0s
=> [1/5] FROM docker.io/library/node:20@sha256:7c4cd7c6935554b79c6fffb88e7bde3db0ce25b45d4c634d1fb0 0.2s
=> => resolve docker.io/library/node:20@sha256:7c4cd7c6935554b79c6fffb88e7bde3db0ce25b45d4c634d1fb0 0.2s
=> [internal] load build context                                  1.1s
=> => transferring context: 168.53kB                              1.0s
=> CACHED [2/5] WORKDIR /app                                       0.0s
=> CACHED [3/5] COPY package*.json ./                             0.0s
=> CACHED [4/5] RUN npm install                                    0.0s
=> CACHED [5/5] COPY . .                                          0.0s
=> exporting to image                                             0.5s
=> => exporting layers                                           0.1s
=> => exporting manifest sha256:1a4fb8a0491d352c29eb325de5b4614d9082bd5c21375e8af1c8a7c446573c8f 0.0s
=> => exporting config sha256:e7fca7901ae8efd44791e1e215d5259ffd027e2720ffcb4aaec8cbbf39d493db 0.0s
=> => exporting attestation manifest sha256:6350d9f7f0ee563fd38eb68b5f4452c3b32b857eea471c005100afb9 0.1s
=> => exporting manifest list sha256:fd42be49d3cbefcbb098f3684a56f329292a866938401a7f73f15e53a9d0bf 0.0s
=> => naming to docker.io/library/todo-list:latest              0.0s
=> => unpacking to docker.io/library/todo-list:latest           0.1s
PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes>

```

3. Ajout d'un tag à notre image pour l'identifier

Ensuite, toujours dans le terminal, entrer :

```
docker tag todo-list:latest <dockerhub-username>/todo-list:latest
```

Cela affichera :

```

PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes> docker tag todo-list:latest thisma24/todo-list:latest
PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes>

```

4. Publication de l'image sur Docker Hub

Ensuite, dans le même terminal, entrer:

```
docker push <dockerhub-username>/todo-list:latest
```

Cela affichera :

```

PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes> docker push thisma24/todo-list:latest
The push refers to repository [docker.io/thisma24/todo-list]
e23f099911d6: Layer already exists
3e6b9d1a9511: Layer already exists
c657c59ebca6: Layer already exists
37927ed901b1: Layer already exists
d8df3b059598: Layer already exists
b685d38f6fe4: Layer already exists
3fbcc227ac4b: Layer already exists
79b2f47ad444: Layer already exists
a6810dc34997: Layer already exists
325557f1736c: Layer already exists
88a200f633b8: Pushed
230735c966ae: Pushed
2995dfd1a19c: Layer already exists
latest: digest: sha256:fd42be49d3cbefcbb098f3684a56f329292a866938401a7f73f15e53a9d0bf9d size: 856
PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes>

```

Maintenant que l'image est déposée sur Docker Hub, il ne reste plus qu'à l'utiliser lors du déploiement avec Kubernetes.

IV. Configurer l'application avec Kubernetes

1. Création du fichier web-service.yaml

Ce fichier représente un service de Kubernetes de type NodePort. Il permet aux utilisateurs d'accéder à l'application web depuis l'extérieur en utilisant l'adresse IP du serveur et un port spécial.

```
k8s > ! web-service.yaml
1  apiVersion: v1
2  kind: Service
3  metadata:
4    name: web
5  spec:
6    ports:
7      - port: 8080
8        targetPort: 5000
9        nodePort: 30000
10   selector:
11     app: web
12   type: NodePort
13
```

2. Création du fichier web-deployment.yaml

Ce fichier sert à lancer l'application web **ToDoList** dans Kubernetes, en créant un pod qui contient un conteneur Docker avec l'application.

Kubernetes va utiliser ce fichier pour :

- Télécharger et lancer l'image Docker **thisma24/todo-list:latest**.
Attention : « thisma24 » correspond à votre « dockerhub-username »
- Créer un pod avec un conteneur grâce à **replicas: 1**
- Il va lui fournir des variables d'environnement pour se connecter à la base de données PostgreSQL.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web
spec:
  replicas: 1
  selector:
    matchLabels:
      app: web
  template:
    metadata:
      labels:
        app: web
    spec:
      containers:
        - name: web
          image: thisma24/todo-list:latest
          ports:
            - containerPort: 5000
          env:
            - name: DB_HOST
              value: db
            - name: DB_NAME
              value: postgres
            - name: DB_PASSWORD
              value: password
            - name: DB_PORT
              value: "5432"
            - name: DB_USER
              value: postgres
            - name: NODE_ENV
              value: development
            - name: PGADMIN_URL
              value: http://pgadmin:8888
      workingDir: /app
```

V. Configurer une base de données PostgreSQL avec Kubernetes

1. Création du fichier db-deployment.yaml

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: postgres-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: db
spec:
  replicas: 1
  selector:
    matchLabels:
      app: db
  template:
    metadata:
      labels:
        app: db
    spec:
      containers:
        - env:
            - name: POSTGRES_PASSWORD
              value: password
            - name: POSTGRES_USER
              value: postgres
          image: postgres:latest
          name: db
          ports:
            - containerPort: 5432
          volumeMounts:
            - name: db-storage
              mountPath: /var/lib/postgresql/data
      volumes:
        - name: db-storage
          persistentVolumeClaim:
            claimName: postgres-pvc
          restartPolicy: Always
```

2. Appliquer le fichier YAML

Dans un terminal, entrer :

```
Kubectl apply -f db-deployment.yaml
```

3. Vérifier que le pod est actif

Puis, toujours dans le même terminal, entrer :

```
Kubectl get pods
```


4. Création du fichier db-service.yaml

Ce fichier permet de créer un service Kubernetes qui rend le pod PostgreSQL accessible aux autres pods dans le cluster.

Son rôle consiste à créer un service nommé db, qui agit comme un point d'entrée fixe vers la base. Il permet à l'application web de se connecter à PostgreSQL en utilisant simplement un nom **db** comme hôte.

```
apiVersion: v1
kind: Service
metadata:
  name: db
spec:
  ports:
    - port: 5432
      targetPort: 5432
  selector:
    app: db
```

VI. Configurer pgAdmin avec Kubernetes

1. Création du fichier pgadmin-deployment.yaml

Ce fichier sert à créer et lancer PgAdmin dans Kubernetes, il permet de créer un pod avec l'image PgAdmin4. Il configure également l'adresse email et le mot de passe pour se connecter à PgAdmin. Il indique à Kubernetes que le conteneur de PgAdmin, à l'intérieur du pod, utilise le port 80 pour afficher l'interface web et il permet de garder un seul pod en fonctionnement.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: pgadmin
spec:
  replicas: 1
  selector:
    matchLabels:
      app: pgadmin
  template:
    metadata:
      labels:
        app: pgadmin
    spec:
      containers:
        - env:
            - name: PGADMIN_DEFAULT_EMAIL
              value: louandeshayes@gmail.com
            - name: PGADMIN_DEFAULT_PASSWORD
              value: LouanPgAdmin
          image: dpage/pgadmin4:9.3
          name: pgadmin
          ports:
            - containerPort: 80
```

2. Création du fichier pgadmin-service.yaml

Ce fichier sert à rendre PgAdmin accessible depuis le navigateur. Il crée un service kubernetes qui redirige le port 8888 (sur le pc) vers le port 80 du pod PgAdmin. C'est grâce à cela que nous pouvons ouvrir PgAdmin dans le navigateur en rentrant ce lien <http://localhost:8888>.

```
apiVersion: v1
kind: Service
metadata:
  name: pgadmin
spec:
  ports:
    - port: 8888
      targetPort: 80
  selector:
    app: pgadmin
```

3. Accéder à pgAdmin depuis le navigateur

Pour cela, il faut copier le lien ci-dessous dans un navigateur :

<http://localhost:8888>

Ensuite nous aurons accès à l'interface d'authentification de PgAdmin.

Pour se connecter, il faut saisir les informations suivantes :

Email : **louandeshayes@gmail.com**

Mot de passe : **LouanPgAdmin**

4. Ajouter la base de données dans pgAdmin

Une fois connecté à PgAdmin : clic droit sur « **Server** » puis « **Register** » ensuite « **Server** ».

Entrer les informations suivantes :

Dans General :

Name : <Le nom que vous souhaitez>

Dans Connection :

Host name/address : **db**

Port : **5432**

Maintenance database : **postgres**

Username : **postgres**

Password : **password**

Register - Server

General Connection Parameters SSH Tunnel Advanced Post Connection SQL Tags

Name: Postgres

Server group: Servers

Background: ☐

Foreground: ☐

Connect now?: ☒

Shared?: ☐

Shared Username:

Comments:

Either Host name or Service must be specified.

Close Reset Save

The image shows a 'Register - Server' dialog box with the 'Connection' tab selected. The fields are as follows:

Field	Value
Host name/address	db
Port	5432
Maintenance database	postgres
Username	postgres
Kerberos authentication?	<input type="checkbox"/>
Password	*****
Save password?	<input type="checkbox"/>
Role	
Service	

Buttons at the bottom: Close, Reset, Save.

Nous avons mis en place deux tables : **tasks** qui permet de stocker les différentes tâches et **users** pour la gestion des utilisateurs, ces tables ont été créées via l'interface graphique PgAdmin, en exécutant des requêtes SQL dans l'environnement PostgreSQL du conteneur Docker. Dans le but de faciliter la mise à jour des données, nous avons mis en place des triggers dans les deux tables, cela permet de suivre les modifications apportées à ces dernières.

VII. Connecter l'application web à la base de données

1. Fichier Database.js

Nous avons créé un fichier nommé **database.js** afin de lier la base de données avec l'application web en utilisant comme intermédiaire le Node.js, c'est un serveur dont le rôle est de recevoir les requêtes du navigateur, il utilise **database.js** pour se connecter à PostgreSQL. Ensuite, il fait des requête Sql et renvoie les données au navigateur. le fichier **database.js** s'appuie sur une bibliothèque appelée **pg**, spécialisée dans l'interfaçage entre Node.js et PostgreSQL. Ce fichier permet également d'exporter la connexion à la base de données. Ce qui permet aux autres fichiers du projet de l'utiliser sans devoir refaire la configuration.

```
> backend > db > JS database.js > ...
const {Pool} = require('pg')

module.exports = new Pool({
  host: process.env.DB_HOST,
  database: process.env.DB_NAME,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  port: parseInt(process.env.DB_PORT) || 5432,
});
```

Pool : permet de créer un groupe de connexions réutilisables avec la base de données.

process.env.XXX : Ce sont des variables d'environnement qui contiennent les informations sensibles (identifiant, mot de passe...).

Host : représente l'adresse de la base de données.

Database : c'est le nom de la base à laquelle on se connecte.

User : Le nom de l'utilisateur autorisé à se connecter.

Password : Le mot de passe associé à l'utilisateur.

Port : le port de postgres qui est par défaut 5432.

Pour garantir la sécurité, les variables d'environnement sont définies en dehors du code, notamment via la configuration des conteneurs Docker qui exécutent la base de données et l'application web. Ainsi, les deux conteneurs disposent des mêmes informations de connexion nécessaires pour que l'application puisse accéder à la base PostgreSQL.

VIII. Déploiement de l'application

Maintenant que les fichiers de configuration pour la base de données, PgAdmin et l'application sont fait. Le déploiement de ces trois éléments peut s'effectuer.

1. Appliquer les fichiers de configuration

Entrer dans un terminal :

```
kubectl apply -f .
```

Cette commande permet d'appliquer tous les fichiers de configuration dans Kubernetes.

```
● PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s> kubectl apply -f .
persistentvolumeclaim/postgres-pvc unchanged
deployment.apps/db unchanged
service/db unchanged
deployment.apps/pgadmin unchanged
service/pgadmin unchanged
deployment.apps/web unchanged
service/web unchanged
○ PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s> |
```

2. Vérification en ligne de commande

Pour vérifier si les fichiers de configurations ont bien été appliqués, on exécute ces commandes.

Entrer dans un terminal :

```
kubectl get pods
```

Cette commande permet de voir les déploiements qui sont actifs.

Cela affichera :

```
● PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s> kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
db-5bd59d5795-48qtd                 1/1     Running   1 (27h ago)  33h
pgadmin-65f9d9f7b7-xzb6d            1/1     Running   1 (14m ago)  37h
web-5fdd67fc4c-vvpqk                1/1     Running   1 (27h ago)  34h
○ PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s> |
```

Entrer ensuite dans le même terminal :

```
kubectl get services
```

Cette commande permet de voir quels services sont actifs.

Cela affichera :

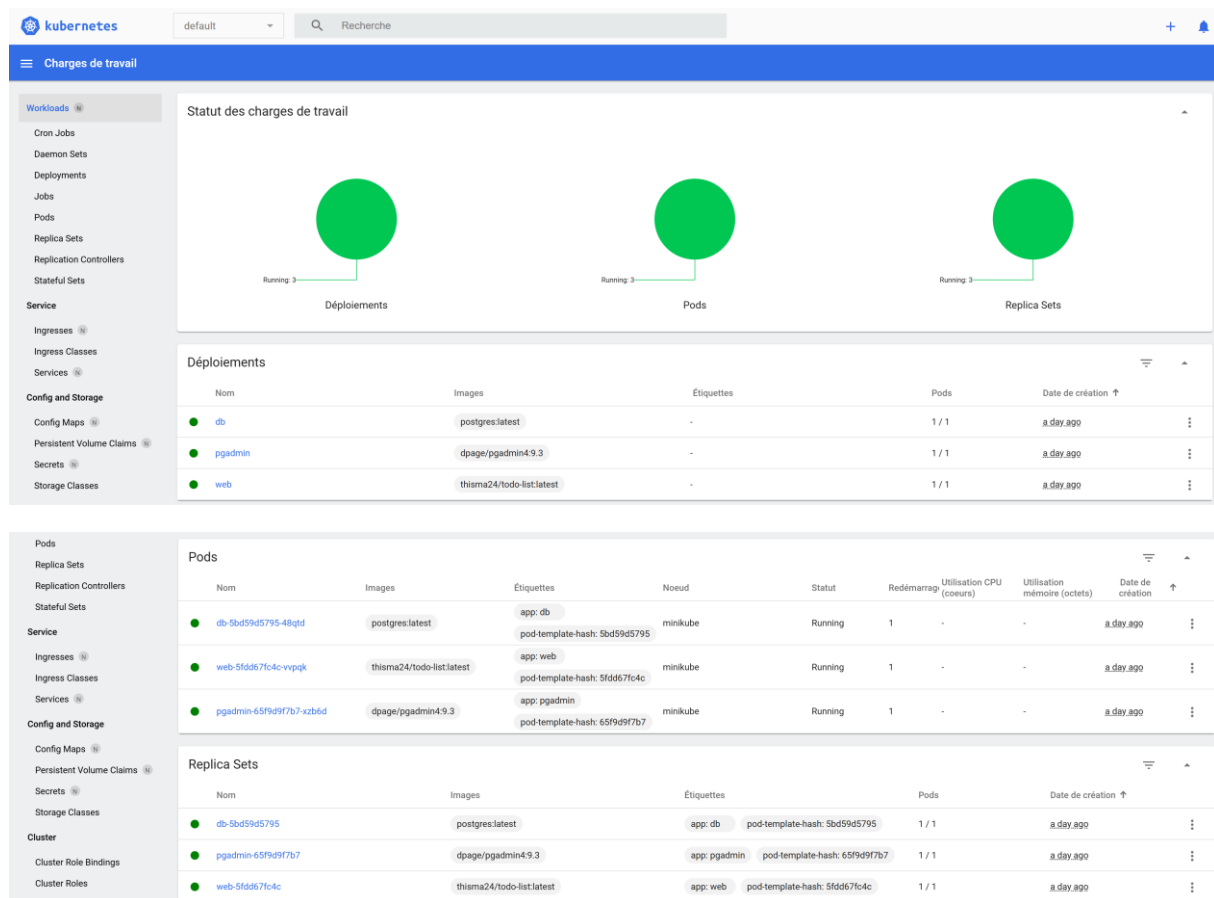
```

PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s> kubectl get services
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
db            ClusterIP     10.106.168.98   <none>           5432/TCP         37h
kubernetes    ClusterIP     10.96.0.1       <none>           443/TCP          37h
pgadmin       ClusterIP     10.109.201.23   <none>           8888/TCP         37h
web           NodePort      10.100.177.148  <none>           8080:30000/TCP   37h
PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s>

```

3. Vérification avec Minikube

La vérification peut aussi être faite depuis le Dashboard de Minikube :

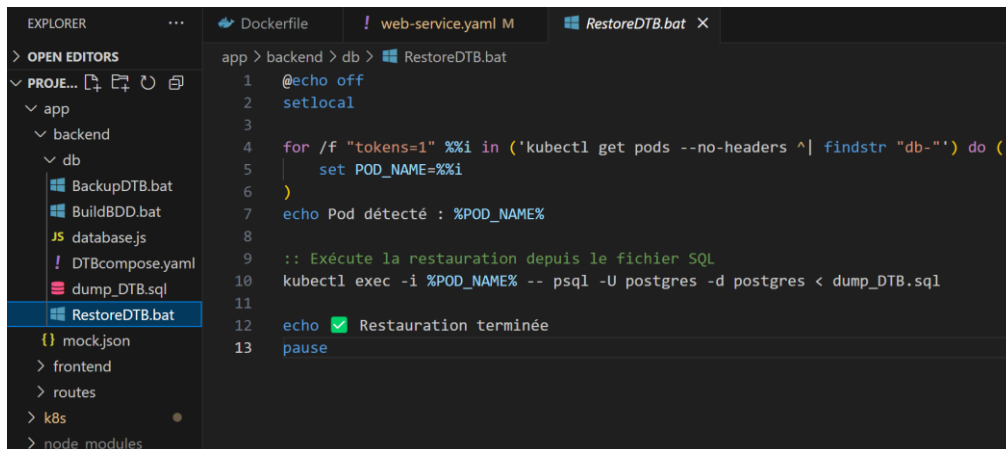


4. Migration de la base de données

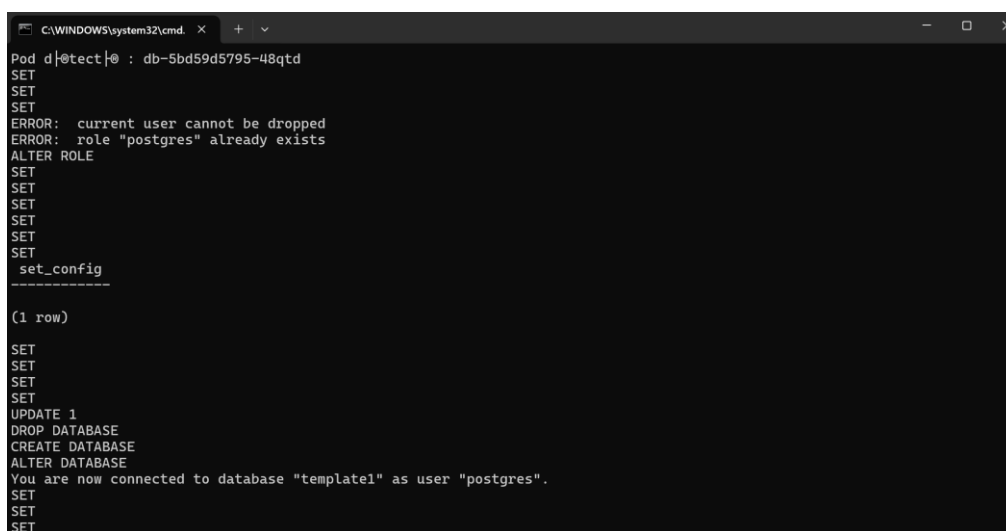
Une fois que tout s'est bien déroulé durant le déploiement, la migration de la base de données peut avoir lieu. Un fichier nommé « **restoreDTB.bat** » permet de restaurer la base de données à partir d'un dump de la base de données (« **dump_DTB.sql** ») que l'on a effectué en amont contenant quelques données déjà pré-enregistrer.

Remarque : Ce dump est généré par un fichier nommé « **backupDTB.bat** » accessible comme « restoreDTB.bat »

On exécute le fichier « restoreDTB.bat » **2 fois** (il est fort possible que la première fois fonctionne partiellement) :



```
1 @echo off
2 setlocal
3
4 for /f "tokens=1" %i in ('kubectl get pods --no-headers ^| findstr "db-"'') do (
5     set POD_NAME=%i
6 )
7 echo Pod détecté : %POD_NAME%
8
9 :: Exécute la restauration depuis le fichier SQL
10 kubectl exec -i %POD_NAME% -- psql -U postgres -d postgres < dump_DTB.sql
11
12 echo ✅ Restauration terminée
13 pause
```



```
Pod db-5bd59d5795-48qtd
SET
SET
SET
ERROR: current user cannot be dropped
ERROR: role "postgres" already exists
ALTER ROLE
SET
SET
SET
SET
SET
SET
set_config
-----
(1 row)
SET
SET
SET
SET
UPDATE 1
DROP DATABASE
CREATE DATABASE
ALTER DATABASE
You are now connected to database "template1" as user "postgres".
SET
SET
SET
```

5. Accéder à l'application

Pour accéder à notre application ToDoList se trouvant dans Kubernetes, on exécute la commande suivante.

Entrer dans un terminal :

```
minikube service web
```

Cette commande nous permet d'accéder à notre application en passant par deux URLs :

- <http://192.168.58.2.30000>
- <http://127.0.0.1:58479>

Attention : Ces deux URLs peuvent changer à chaque exécution de la commande.

Cela affichera :


```
PS E:\Mathis\MASTER\Cours\Micro-services\Projet-docker-kubernetes\k8s> minikube service web
```

NAMESPACE	NAME	TARGET PORT	URL
default	web	8080	http://192.168.58.2:30000

🏃 Starting tunnel for service web.

NAMESPACE	NAME	TARGET PORT	URL
default	web		http://127.0.0.1:58479

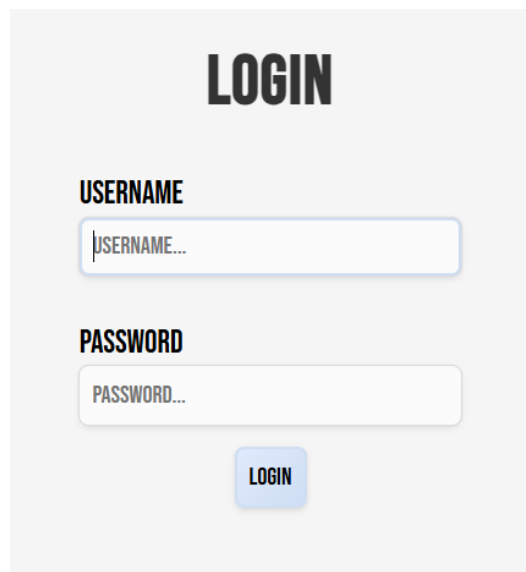
🌐 Opening service default/web in default browser...

! Because you are using a Docker driver on windows, the terminal needs to be open to run it.

IX. L'application web

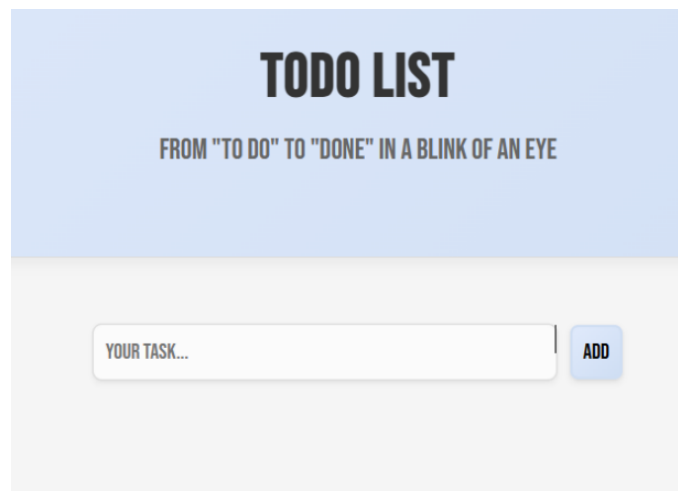
1. La page de connexion

Cette page contient un champ **Username** qui correspond au nom de l'utilisateur, un champ **Password** pour saisir un mot de passe et un bouton **Login** qui va lancer la fonction `login()`. Cette dernière est codée en JavaScript. Elle a pour rôle de récupérer les valeurs saisies par l'utilisateur, d'envoyer une requête au serveur pour vérifier que les informations entrées sont valides. Si l'identifiant de l'utilisateur existe dans la base de données, cela veut dire que l'authentification est réussie et donc l'utilisateur est redirigé vers la page de gestion des tâches.

A login form with a light gray background. At the top, the word "LOGIN" is written in large, bold, black capital letters. Below it, the label "USERNAME" is in bold black capital letters, followed by a white input field with a light blue border and the placeholder text "USERNAME...". Below that, the label "PASSWORD" is in bold black capital letters, followed by a white input field with a light blue border and the placeholder text "PASSWORD...". At the bottom, there is a blue button with the word "LOGIN" in white capital letters.

2. La page principale

La page principale permet d'afficher la liste des tâches. Cette liste est récupérée depuis la base de données. L'utilisateur peut voir ses tâches, les cocher pour indiquer qu'elles sont terminées et il a également l'option de les supprimer.

A form for a TODO list. The top section has a light blue background with the title "TODO LIST" in large, bold, black capital letters. Below the title, the subtitle "FROM 'TO DO' TO 'DONE' IN A BLINK OF AN EYE" is written in smaller, gray capital letters. The bottom section has a light gray background and contains a white input field with a light blue border and the placeholder text "YOUR TASK...". To the right of the input field is a blue button with the word "ADD" in white capital letters.