Rapport de Projet

API Joueurs NBA – DevOps

Auteurs :

- PHAM DANG Son Alain
- HERNU Charlotte

1. Introduction

Ce rapport présente l'ensemble du projet réalisé dans le cadre du cours *DevOps Data*. Le projet consiste à développer et déployer une application web simple qui permet de rechercher et d'afficher des statistiques de joueurs NBA. La solution comporte une API REST (backend), une interface web en Html (frontend) et une base de données PostgreSQL. Un pipeline CI/CD a été mis en place pour automatiser la construction, les tests et le déploiement, même si le déploiement final est effectué localement sur un cluster Kubernetes (Minikube).

2. Objectifs

- **Développer une API REST** permettant de récupérer des statistiques sur les joueurs NBA depuis une base de données(fichier csv).

- **Construire une interface web** minimaliste pour interroger cette API.

- **Conteneuriser l'application** (backend et frontend) via Docker.

- **Déployer l'application sur Kubernetes** en utilisant des manifestes YAML.

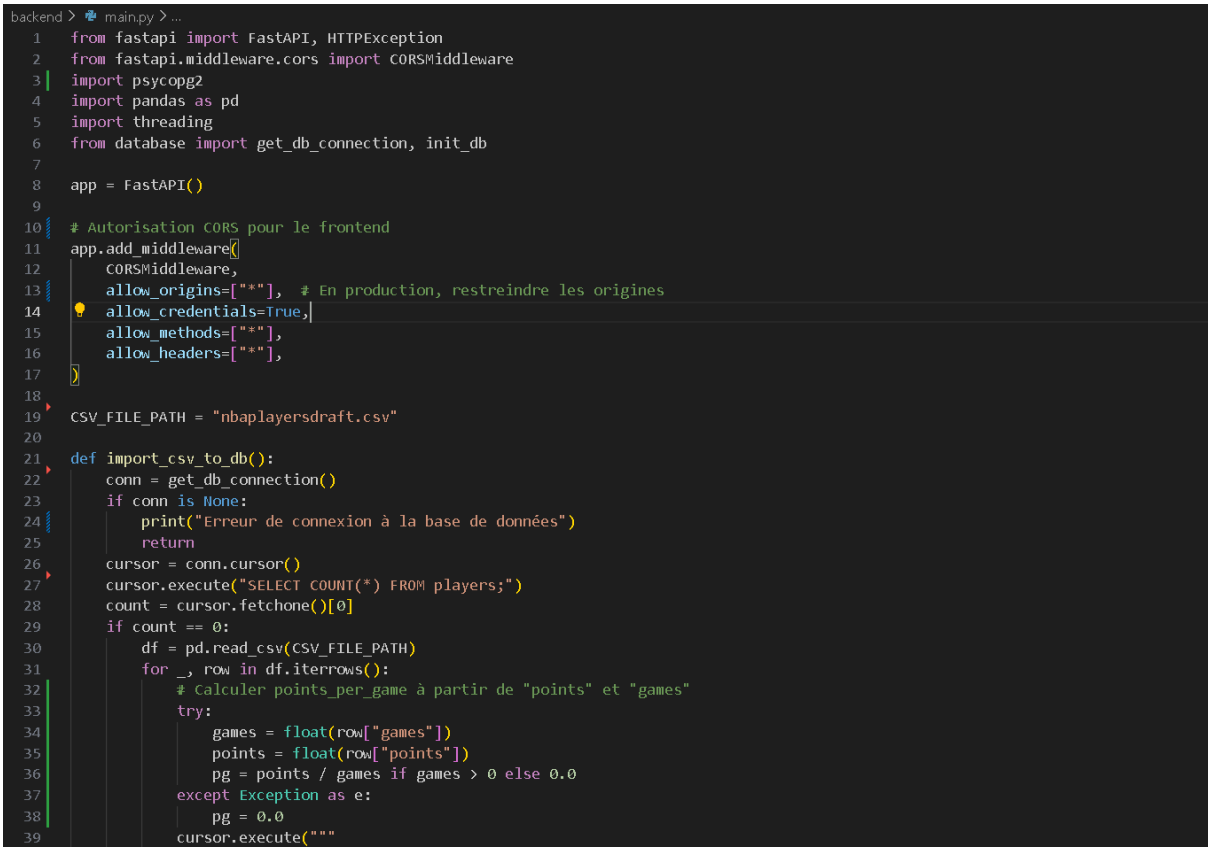
- **Mettre en place un pipeline CI/CD** via GitHub Actions pour automatiser la construction et les tests, avec un déploiement manuel sur le cluster local.

3. Architecture du Projet

Le projet est organisé en plusieurs composants :

- **Backend :**
  Réalisé en Python avec FastAPI, il gère l'import des données depuis un fichier CSV dans une base PostgreSQL et expose l'endpoint `/players/{name}`.
  Le fichier `database.py` assure la connexion à la base et l'initialisation de la table.
  Le fichier `main.py` inclut la logique d'import (avec recalcul de "points_per_game") et de renvoi des statistiques.

main.py :



```python
@app.get("/players/{name}")
def get_player(name: str):
    conn = get_db_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM players WHERE LOWER(player) LIKE LOWER(%s)", (f"%{name}%",))
    player = cursor.fetchone()
    conn.close()

    if player:
        return {
            "id": player[0],
            "name": player[5] if player[5] else "Inconnu",
            "team": player[4] if player[4] else "Inconnu",
            "field_goal_percentage": float(player[10]) if player[10] is not None else 0.0,
            "three_point_percentage": float(player[11]) if player[11] is not None else 0.0,
            "free_throw_percentage": float(player[12]) if player[12] is not None else 0.0,
            "average_minutes_played": float(player[13]) if player[13] is not None else 0.0,
            "points_per_game": float(player[14]) if player[14] is not None else 0.0,
            "average_total_rebounds": float(player[15]) if player[15] is not None else 0.0,
            "average_assists": float(player[16]) if player[16] is not None else 0.0,
            "win_shares": float(player[17]) if player[17] is not None else 0.0,
            "win_shares_per_48_minutes": float(player[18]) if player[18] is not None else 0.0,
            "box_plus_minus": float(player[19]) if player[19] is not None else 0.0,
            "value_over_replacement": float(player[20]) if player[20] is not None else 0.0,
        }
    else:
        raise HTTPException(status_code=404, detail="Player not found")

if __name__ == "__main__":
    init_db()
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

database.py :

```python
DATABASE_URL = "postgresql://user:password@localhost:5432/basket"
def get_db_connection(retries=5, delay=5):
    """
    Tente de se connecter à la base de données plusieurs fois avant d'abandonner.
    """
    for attempt in range(retries):
        try:
            conn = psycopg2.connect(DATABASE_URL)
            print("Connexion à la base réussie à la tentative", attempt + 1)
            return conn
        except psycopg2.OperationalError as e:
            print(f"Tentative {attempt + 1} échouée : {e}. Nouvelle tentative dans {delay} secondes...")
            time.sleep(delay)
    print("Impossible de se connecter à la base après plusieurs tentatives.")
    sys.exit(1)
def init_db():
    """
    Initialise la table players dans la base de données si elle n'existe pas.
    """
    conn = get_db_connection()
    if conn is None:
        print("Échec de connexion lors de l'initialisation de la base.")
        return
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS players (
            id SERIAL PRIMARY KEY,
            year INT,
            rank INT,
            overall_pick INT,
            team TEXT,
            player TEXT,
            college TEXT,
            years_active FLOAT,
            games FLOAT,
            minutes_played FLOAT,
            field_goal_percentage FLOAT,
            three_point_percentage FLOAT,
```

```
        three_point_percentage FLOAT,
        free_throw_percentage FLOAT,
        average_minutes_played FLOAT,
        points_per_game FLOAT,
        average_total_rebounds FLOAT,
        average_assists FLOAT,
        win_shares FLOAT,
        win_shares_per_48_minutes FLOAT,
        box_plus_minus FLOAT,
        value_over_replacement FLOAT
    )
""")
conn.commit()
cursor.close()
conn.close()
print("Base de données initialisée avec succès.")
__name__ == "__main__":
init_db()
```

- **Frontend :**
  Une interface web simple (HTML, CSS, JavaScript) qui permet à l'utilisateur de saisir le nom d'un joueur et d'afficher les statistiques récupérées via l'API.

app.js :

```
frontend > JS app.js > searchPlayer
const API_URL = "http://localhost:8000/players/";
async function searchPlayer() {
    let playerName = document.getElementById("playerName").value;

    if (playerName.trim() === "") {
        alert("Veuillez entrer un nom de joueur !");
        return;
    }
    try {
        let response = await fetch(API_URL + encodeURIComponent(playerName));

        console.log("Réponse API:", response);

        if (response.ok) {
            let player = await response.json();
            console.log("Données joueur:", player);

            document.getElementById("playerInfo").innerHTML = `
                <h2>${player.name} (${player.team})</h2>
                <p>Points par match : ${player.points_per_game}</p>
                <p>Rebonds par match : ${player.average_total_rebounds}</p>
                <p>Passes par match : ${player.average_assists}</p>
                <p>Field Goal % : ${player.field_goal_percentage}</p>
                <p>Three Point % : ${player.three_point_percentage}</p>
                <p>Free Throw % : ${player.free_throw_percentage}</p>
                <p>Moyenne minutes jouées : ${player.average_minutes_played}</p>
                <p>Win Shares : ${player.win_shares}</p>
                <p>Win Shares per 48 min : ${player.win_shares_per_48_minutes}</p>
                <p>Box Plus Minus : ${player.box_plus_minus}</p>
                <p>Value Over Replacement : ${player.value_over_replacement}</p>
            `;
        } else {
            console.log("Erreur API :", response.statusText);
            document.getElementById("playerInfo").innerHTML = "<p>Joueur introuvable.</p>";
        }
    } catch (error) {
        console.error("Erreur lors de la requête :", error);
        document.getElementById("playerInfo").innerHTML = "<p>Erreur de connexion.</p>";
    }
}
```

index.html :

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Recherche de Joueur NBA</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>🏀 Recherche de Joueur NBA</h1>
        <input type="text" id="playerName" placeholder="Entrez un nom de joueur...">
        <button onclick="searchPlayer()">Rechercher</button>

        <div id="playerInfo"></div>
    </div>

    <script src="app.js"></script>
</body>
</html>
```

- **Base de données :**
  Un conteneur PostgreSQL qui stocke les données importées depuis le fichier `nbaplayersdraft.csv`.
- **Déploiement Kubernetes :**
  Les ressources (Deployments, Services, PersistentVolumeClaim) sont décrites dans des manifestes YAML situés dans le dossier `k8s/`.

- backend-deployment :

k8s > ! backend-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      containers:
        - name: backend
          image: sonalain/projet-backend:latest
          ports:
            - containerPort: 8000
          env:
            - name: DATABASE_URL
              value: "postgresql://user:password@database:5432/basket"

---
apiVersion: v1
kind: Service
metadata:
  name: backend
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 8000
      targetPort: 8000
  type: NodePort
```

frontend-deployment.yaml :

k8s > ! frontend-deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image: sonalain/projet-frontend:latest
          ports:
            - containerPort: 80

---
apiVersion: v1
kind: Service
metadata:
  name: frontend
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

- **CI/CD :**
  Un pipeline CI est mis en œuvre via GitHub Actions (workflow `ci.yaml`) qui construit les images Docker et exécute des tests unitaires. Un workflow CD (`cd-pipeline.yaml`) est présent pour automatiser le déploiement, mais le déploiement est finalement réalisé manuellement sur le cluster local en raison des limitations d'accès des runners GitHub aux clusters locaux.

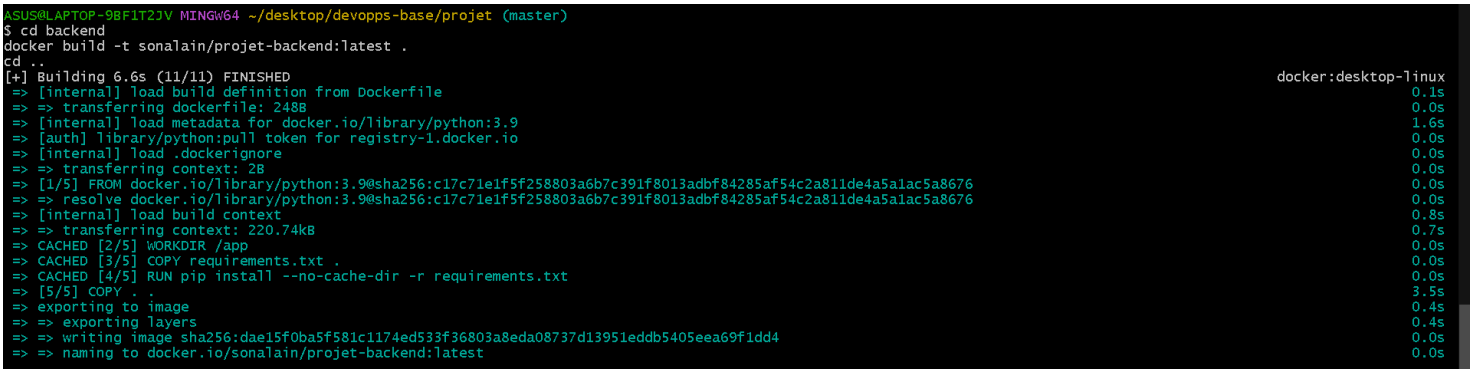
4. Déploiement Local et Procédure de Test

4.1 Construction des Images Docker

Backend :

Commande pour construire l'image Docker du backend :

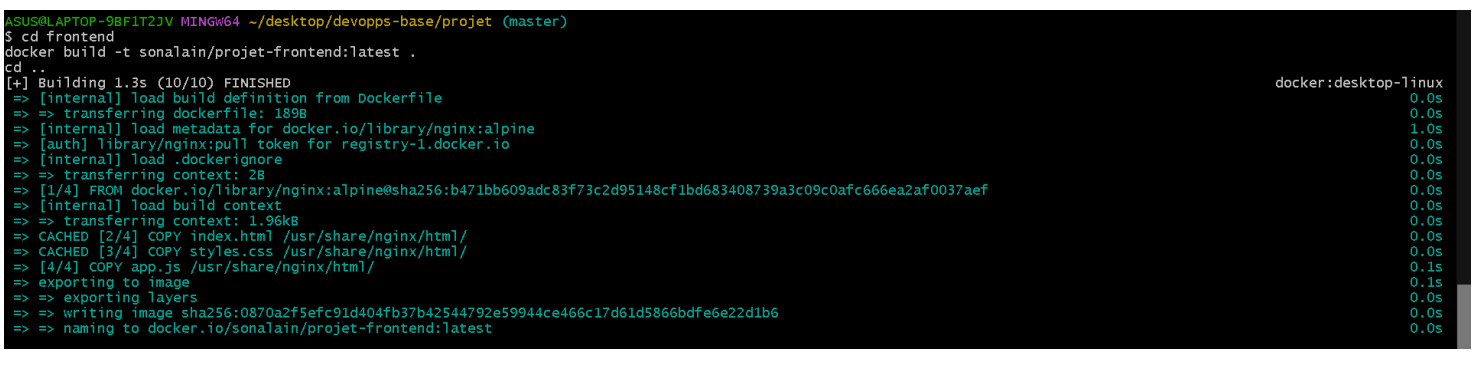
```
cd backend
docker build -t sonalain/projet-backend:latest .
cd ..
```



Frontend :

Commande pour construire l'image Docker du frontend :

```
cd frontend
docker build -t sonalain/projet-frontend:latest .
cd ..
```



4.2 Configuration de l'Environnement Docker pour Minikube

Pour construire les images dans l'environnement Docker de Minikube

```
eval $(minikube docker-env)
```

4.3 Déploiement sur Kubernetes

Appliquez les manifestes Kubernetes depuis le répertoire racine du projet :

```
kubectl apply -f k8s/
```

```
ASUS@LAPTOP-9BF1T2JV MINGW64 ~/desktop/devopps-base/projet (master)
$ kubectl apply -f k8s/
deployment.apps/backend unchanged
service/backend unchanged
deployment.apps/database unchanged
service/database unchanged
persistentvolumeclaim/postgres-pvc unchanged
deployment.apps/frontend unchanged
service/frontend unchanged
```

4.4 Vérification du Déploiement

Listez les pods pour vérifier leur état :

```
kubectl get pods -o wide
```

```
ASUS@LAPTOP-9BF1T2JV MINGW64 ~/desktop/devopps-base/projet (master)
$ kubectl get pods -o wide
NAME                        READY   STATUS    RESTARTS      AGE    IP            NODE       NOMINATED NODE   READINESS GATES
backend-7d5d45565c-2zv6z    1/1     Running   4 (80m ago)   2d1h   10.244.0.92   minikube   <none>           <none>
backend-7d5d45565c-wxtxd    1/1     Running   4 (80m ago)   2d1h   10.244.0.88   minikube   <none>           <none>
database-5ff54df5f9-5hsjc   1/1     Running   2 (80m ago)   2d1h   10.244.0.90   minikube   <none>           <none>
frontend-7d7bbc556d-8cx2v   1/1     Running   2 (80m ago)   2d1h   10.244.0.89   minikube   <none>           <none>
frontend-7d7bbc556d-9mbx5   1/1     Running   2 (80m ago)   2d1h   10.244.0.93   minikube   <none>           <none>
```

4.5 Test de l'API du Backend

Pour tester l'API, on utilise la commande suivante avec le nom du pod remplacer par le nom réel, par exemple ici `backend-7d5d45565c-2zv6z`) :

```
kubectl exec -it backend-7d5d45565c-2zv6z -- curl -s
http://localhost:8000/players/LeBron
```

```
ASUS@LAPTOP-9BF1T2JV MINGW64 ~/desktop/devopps-base/projet (master)
$ kubectl exec -it backend-7d5d45565c-2zv6z -- curl -s http://localhost:8000/players/LeBron
{"id":786,"name":"LeBron James","team":"CLE","points_per_game":0.0,"assists_per_game":7.5,"rebounds_per_game":27.1}
```

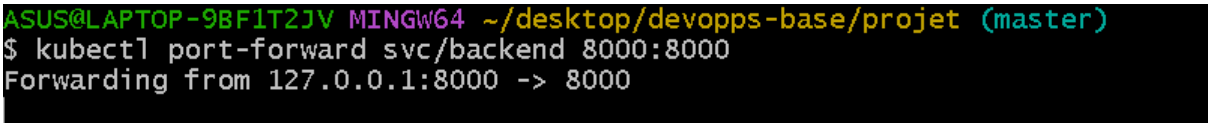
4.6 Accès Externe aux Services (Port-forwarding)

Le port-forwarding est utilisé pour contourner les limitations d'accès direct aux services déployés dans le cluster Minikube (notamment en raison des limitations du driver Docker sur Windows)

Pour le Backend :

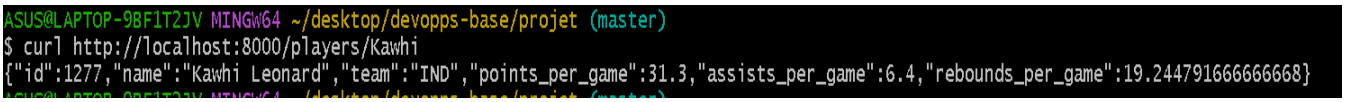
Lancement du port-forwarding :

```
kubectl port-forward svc/backend 8000:8000
```



Le port 8000 du service `backend` dans le cluster est redirigé vers le port 8000 de ma machine. Ainsi, en accédant à `http://localhost:8000` dans mon navigateur, nous atteignons le service backend déployé dans Kubernetes.

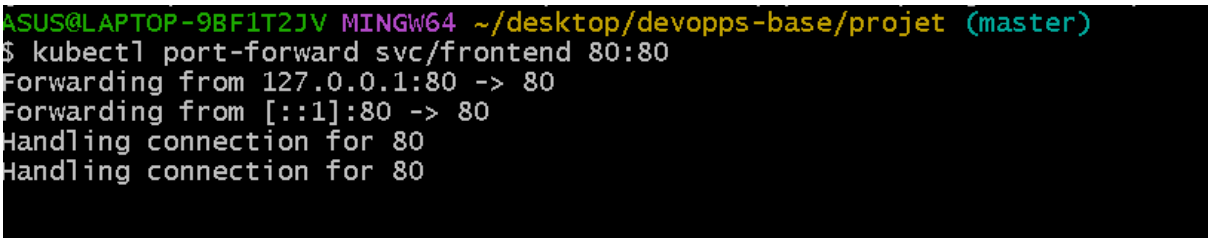
```
curl http://localhost:8000/players/LeBron
```



Pour le Frontend :

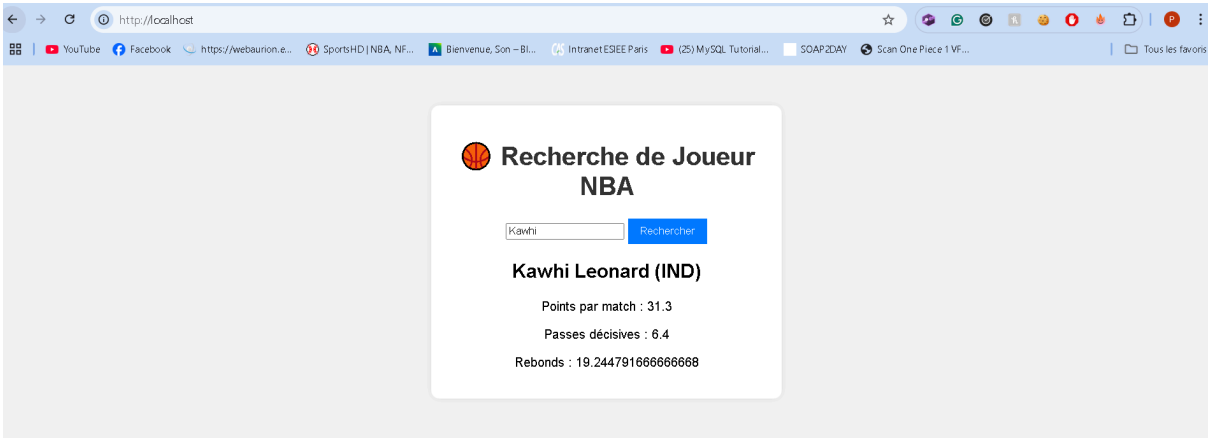
Lancement le port-forwarding pour le frontend :

```
kubectl port-forward svc/frontend 80:80
```



On peut donc acceder sur le navigateur avec cette adresse :

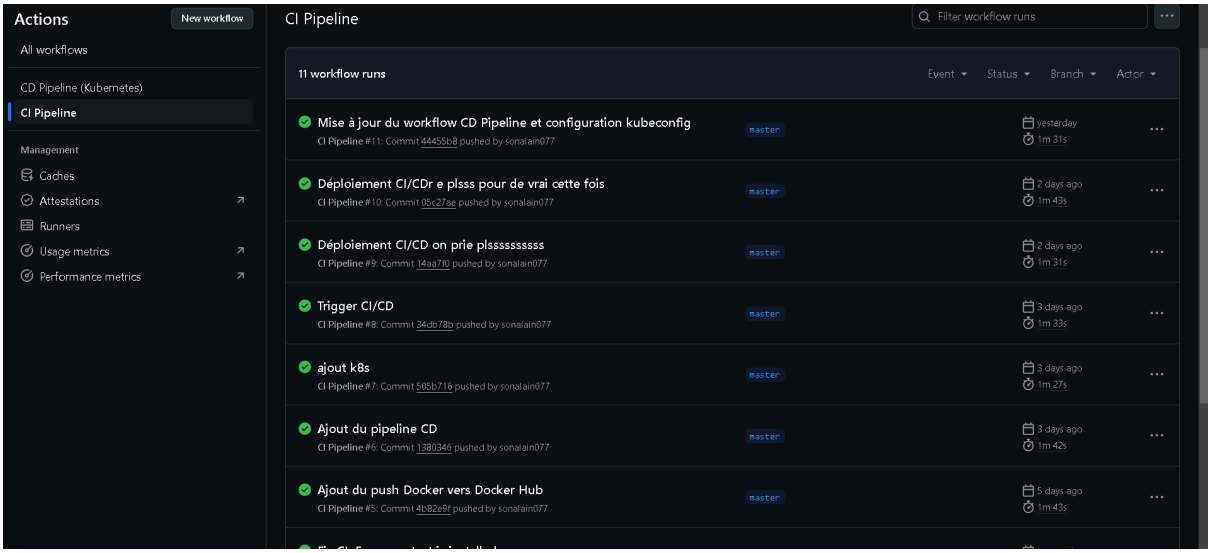
```
http://localhost
```



5. Déploiement et CI/CD

CI Pipeline via GitHub Actions

- Le workflow **ci.yaml** s'occupe de la construction des images et de l'exécution des tests unitaires. Il se déclenche automatiquement à chaque push ou pull request sur la branche master



ci.yaml

CD Pipeline

- Bien qu'un workflow CD soit défini dans `cd-pipeline.yaml`, le déploiement final se fait manuellement sur le cluster local. Cela est dû aux limitations d'accès des runners GitHub aux clusters locaux. La partie CD dans le pipeline permet néanmoins de documenter le processus de déploiement.

6. Problèmes Rencontrés et Solutions

Accès aux Services via NodePort

Problème :
L'accès direct via `curl http://$(minikube ip):30166/players/LeBron` échoue en raison des limitations du driver Docker sur Windows.

Solution :
Utilisation du port-forwarding pour rediriger les ports du cluster vers `localhost`.

- Backend : `kubectl port-forward svc/backend 8000:8000`
- Frontend : `kubectl port-forward svc/frontend 80:80`

Mise à Jour du Schéma de la Base de Données

Problème :
Le schéma de la table `players` n'était pas mis à jour automatiquement (commande `CREATE TABLE IF NOT EXISTS` ne modifie pas une table existante).

Solution :
Supprimer la table existante via :

```
DROP TABLE IF EXISTS players;
```

Puis on a redémarrer l'application pour recréer la table avec le nouveau schéma et réimporter les données.

```
ASUS@LAPTOP-9BF1T2JV MINGW64 ~/desktop/devopps-base/projet (master)
$ pgcli -h localhost -U user -d basket
Password for user:
Server: PostgreSQL 17.2 (Debian 17.2-1.pgdg120+1)
Version: 4.1.0
Home: http://pgcli.com
user@localhost:basket> DROP TABLE IF EXISTS players;
You're about to run a destructive command.
Do you want to proceed? [y/N]: y
Your call!
DROP TABLE
Time: 0.037s
user@localhost:basket> exit
Goodbye!
```

7. Conclusion

Ce projet démontre la mise en œuvre d'un pipeline DevOps complet pour une application web simple :

- **Intégration Continue (CI) :**
  Les images Docker sont construites et les tests unitaires sont exécutés via GitHub Actions.
- **Déploiement Local (CD) :**
  L'application est déployée sur un cluster Kubernetes local (Minikube) et accessible via port-forwarding.

Le déploiement s'effectue entièrement en local, répondant ainsi aux exigences du projet. Ce rapport détaille toutes les étapes du projet, les commandes utilisées et présente les solutions apportées aux différents problèmes rencontrés.

Video de démonstration :

https://drive.google.com/file/d/1TtlXBUQBGesSO9yNdEP9pu0hRc6d8p5u/view?usp=drive_link