



**Faculté des sciences**

UFR Mathématique & Informatique

# **Rapport Projet Programmation distribuée**

**Présenté par :**

**BOUADI Sabrina (M1 IAD)**

**Année universitaire : 2024/2025**

# Objectif du projet

L'objectif est de concevoir, conteneuriser et déployer une application **moderne**, respectant les standards d'**architecture microservices**, avec des **bonnes pratiques DevOps** (Docker, Kubernetes), et une interface utilisateur fluide. Le tout est réalisé avec une séparation claire entre **frontend** et **backend**, tout en intégrant une API de paiement (Stripe) pour simuler un processus de commande.

---

## Présentation de l'application

Nous avons développé une **application web de gestion de cocktails** en deux parties :

### Frontend – Angular

- Développé avec **Angular**
- Affiche une **liste de cocktails**
- Permet d'**afficher les détails** d'un cocktail
- Interface **moderne, fluide et intuitive**
- Ajout au panier d'un ou plusieurs cocktails
- Intégration avec **Stripe** pour simuler un **paiement**
- Interface **admin** pour :
  - Ajouter de nouveaux cocktails
  - Modifier ou supprimer des cocktails existants

### Backend – Node.js (Express.js)

- Fournit une **API RESTful** pour :
  - Récupérer la liste des cocktails
  - Gérer les opérations CRUD (admin)

- Gérer les paniers
- Intégrer le paiement via **Stripe API**
- Stockage local (en mémoire ou fichier JSON), extensible vers base de données

## Commencer par un seul service en local :

### 1-Créer les images Docker :

#### 1.1.frontend:

Image construite avec succès.

Dockerfile bien détecté.

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/Cocktails (main)
$ docker build -t cocktails-frontend .
[+] Building 8.8s (12/12) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.2s
=> => transferring dockerfile: 535B                               0.1s
=> [internal] load metadata for docker.io/library/node:18-alpine  1.9s
=> [auth] library/node:pull token for registry-1.docker.io        0.0s
=> [internal] load .dockerignore                                   0.3s
=> => transferring context: 2B                                     0.1s
=> [1/6] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d11894558 0.9s
=> => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d11894558 0.8s
=> [internal] load build context                                  0.5s
=> => transferring context: 3.56kB                                  0.1s
=> CACHED [2/6] WORKDIR /app                                       0.0s
=> CACHED [3/6] COPY package*.json ./                             0.0s
=> CACHED [4/6] RUN npm install                                   0.0s
=> CACHED [5/6] RUN npm install -g @angular/cli                  0.0s
```

#### 1.2.backend:

Image construite avec succès.

Dockerfile bien détecté.

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/Cocktails (main)
$ cd ../backend/

Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ docker build -t cocktails-backend .
[+] Building 30.8s (10/10) FINISHED                                docker:desktop-linux
=> [internal] load build definition from Dockerfile                0.5s
=> => transferring dockerfile: 535B                               0.1s
=> [internal] load metadata for docker.io/library/node:18-alpine  0.9s
=> [internal] load .dockerignore                                   0.3s
=> => transferring context: 2B                                     0.1s
=> [1/5] FROM docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d11894558 0.3s
=> => resolve docker.io/library/node:18-alpine@sha256:8d6421d663b4c28fd3ebc498332f249011d11894558 0.3s
=> [internal] load build context                                  16.9s
=> => transferring context: 7.87MB                                  16.6s
=> CACHED [2/5] WORKDIR /usr/src/app                               0.0s
=> CACHED [3/5] COPY package*.json ./                             0.0s
=> CACHED [4/5] RUN npm install                                   0.0s
```

## Connexion et publication sur Docker Hub

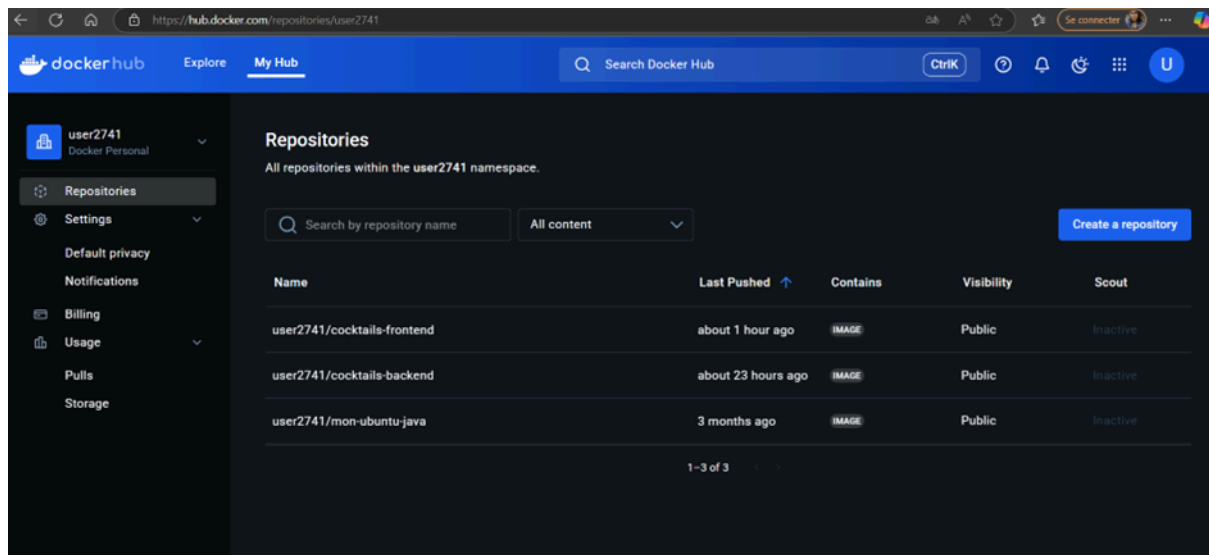
Une fois les images construites localement, j'ai procédé à leur publication sur Docker Hub. Voici les étapes :

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ docker login
Authenticating with existing credentials...
Login Succeeded

Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ docker tag cocktails-backend user2741/cocktails-backend:latest

Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ docker push user2741/cocktails-backend:latest
```

Les deux images sont maintenant accessibles publiquement depuis Docker Hub, ce qui permettra leur déploiement dans tout environnement compatible Docker/Kubernetes.



### 3. Déploiement d'un premier service dans Kubernetes (Nginx):

Nous avons utilisé la commande kubectl pour créer un déploiement basé sur l'image nginx et pour rendre ce service accessible depuis l'extérieur du cluster, nous avons exposé le déploiement via un service de type NodePort

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ kubectl create deployment my-nginx --image nginx
deployment.apps/my-nginx created
```

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ kubectl expose deployment my-nginx --port=80 --type=NodePort
service/my-nginx exposed

Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
my-nginx      1/1     1            1           76s

Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project/backend (main)
$ kubectl get svc
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
kubernetes    ClusterIP      10.96.0.1        <none>           443/TCP          17h
my-nginx      NodePort       10.101.178.148   <none>           80:30536/TCP     18s
```

Cette étape a permis de :

- Vérifier le bon fonctionnement du cluster Kubernetes.
- Tester le déploiement et l'exposition d'un service de manière fiable.
- Préparer l'environnement pour le déploiement des services de notre application web (cocktails).

Une fois les images de nos services (**cocktails-frontend** et **cocktails-backend**) publiées sur Docker Hub, nous avons procédé à leur **déploiement dans le cluster Kubernetes** en appliquant les fichiers de configuration YAML définis dans le répertoire **./kubernetes/**. Ces fichiers décrivent les déploiements, les services, ainsi qu'un Ingress permettant de centraliser l'accès à l'application.

La commande suivante a été exécutée pour déployer l'ensemble

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project (main)
$ kubectl apply -f ./kubernetes/
deployment.apps/backend created
service/backend created
deployment.apps/frontend created
service/frontend created
ingress.networking.k8s.io/example-ingress created
```

## Vérification des déploiements et services Kubernetes

Après avoir appliqué les fichiers de configuration, nous avons vérifié que les **déploiements** et **services Kubernetes** étaient correctement créés et opérationnels. La commande **kubectl get deployment** confirme que les pods des services **backend**, **frontend** et **my-nginx** sont bien déployés et prêts. Ensuite, la commande **kubectl get svc** permet de s'assurer que les services sont accessibles via leur **ClusterIP** ou **NodePort**, notamment

pour le frontend exposé sur le port 30674

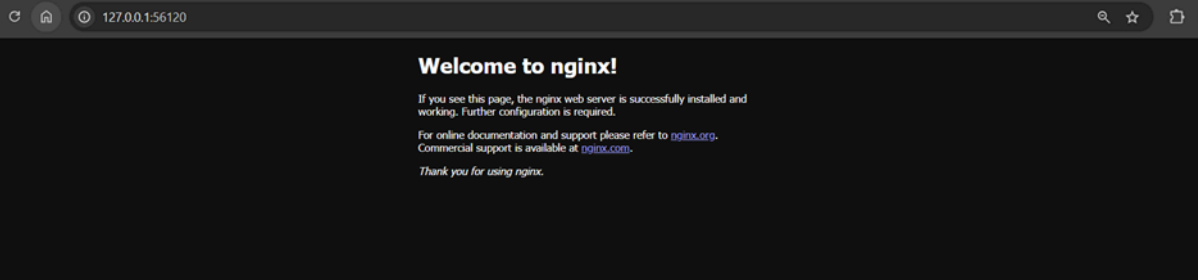
```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project (main)
$ kubectl get deployment
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
backend       1/1     1             1           31s
frontend      1/1     1             1           31s
my-nginx      1/1     1             1           2m46s

Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project (main)
$ kubectl get svc
NAME          TYPE          CLUSTER-IP      EXTERNAL-IP      PORT(S)          AGE
backend       ClusterIP     10.108.87.215   <none>           3000/TCP         43s
frontend      NodePort      10.101.176.252   <none>           4200:30674/TCP   43s
kubernetes    ClusterIP     10.96.0.1       <none>           443/TCP          17h
my-nginx      NodePort      10.101.178.148   <none>           80:30536/TCP     111s
```

## Test de l'accès à l'application via Minikube

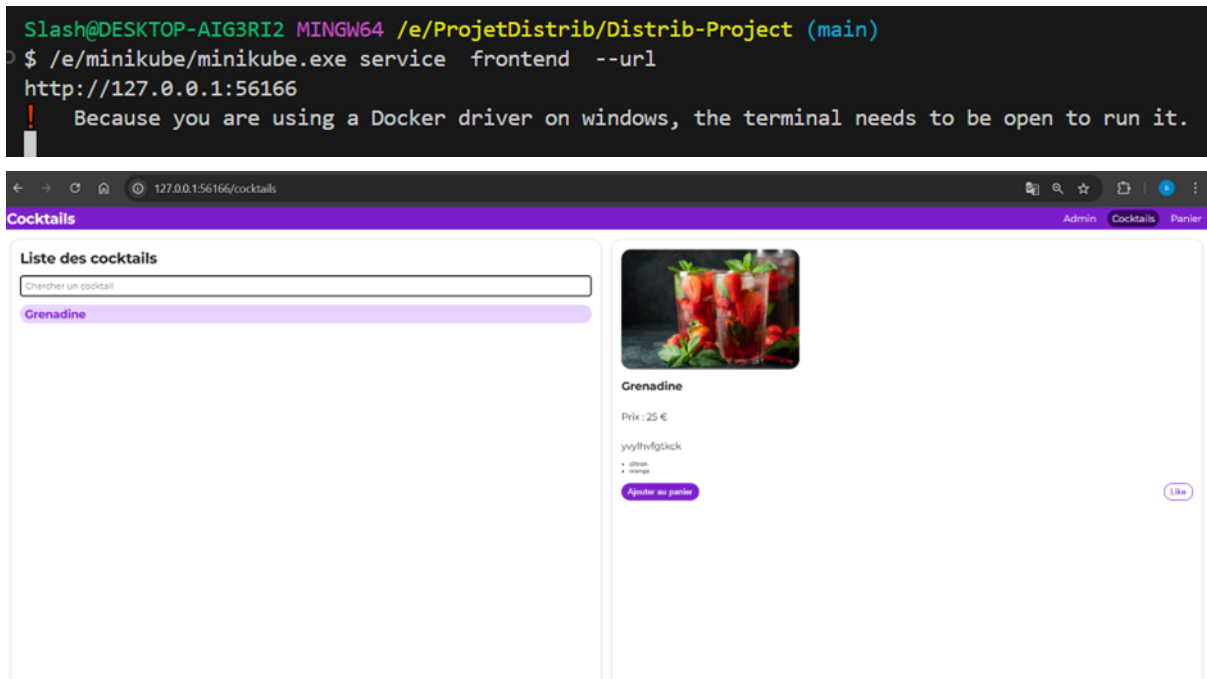
Pour tester que l'application est bien déployée et accessible depuis un navigateur, nous avons utilisé la commande `minikube service` afin de récupérer l'URL publique temporaire exposée par le service `my-nginx`. Cette URL permet d'accéder à l'application depuis l'hôte, ce qui prouve que le routage réseau via Kubernetes fonctionne correctement.

```
Slash@DESKTOP-AIG3RI2 MINGW64 /e/ProjetDistrib/Distrib-Project (main)
$ /e/minikube/minikube.exe service my-nginx --url
http://127.0.0.1:56120
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```



## Vérification de l'accès à l'application via Minikube

Pour valider le bon fonctionnement du service frontend, nous avons utilisé la commande `minikube service frontend --url` afin d'obtenir l'URL locale permettant d'accéder à l'interface web Angular déployée dans Kubernetes. Cette étape confirme que le frontend est bien exposé et accessible via Minikube, assurant ainsi la communication utilisateur avec l'application

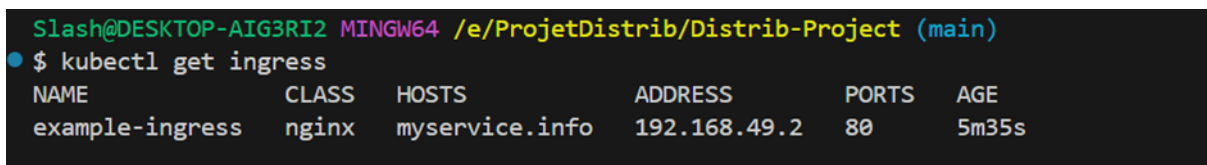


## Configuration et test de l'Ingress Kubernetes

Nous avons configuré un Ingress Kubernetes pour gérer l'accès HTTP centralisé à nos services via un nom de domaine personnalisé `myservice.info`. Après création de l'Ingress, la commande `kubectl get ingress` affiche l'adresse IP assignée par Minikube

- Sur Windows : modifier `C:\Windows\System32\drivers\etc\hosts`

Cependant, malgré cette configuration, le nom de domaine `myservice.info` ne fonctionnait pas correctement dans notre environnement Windows, probablement dû à des restrictions liées à Minikube avec Docker driver ou à la résolution DNS locale.



## **Conclusion**

Ce projet a permis de maîtriser le développement et le déploiement d'une application microservices avec Docker et Kubernetes. L'intégration d'Angular et Node.js, ainsi que la gestion des services et Ingress, ont renforcé nos compétences en architecture cloud native et orchestration.