**RAPPORT DE PROJET – DÉVELOPPEMENT ET DÉPLOIEMENT D'UNE APPLICATION WEB FULLSTACK AVEC DOCKER ET KUBERNETES**

💬 **YasChat** – Application de messagerie temps réel fullstack (Go + React)


## 👥 Membres du binôme

- BLIBEK Rania–
- ZAHEM Yasmine –


---

## 🏫 Formation & Encadrant

- Master cybersécurité et e-santé
- Université Paris Cité
- Projet encadré par **M. Benoît Charroux**

# Présentation du Projet YasChat

Dans un monde où la communication en temps réel est devenue indispensable, les applications de messagerie sont au cœur de notre quotidien, tant pour les échanges personnels que professionnels. Face à la croissance rapide des besoins en matière de performance, de sécurité et de scalabilité, il est crucial de concevoir des solutions qui répondent à ces enjeux avec efficacité.

C'est dans cette optique que nous avons lancé le projet **YasChat**, une application de messagerie moderne qui allie simplicité, performance et évolutivité. En choisissant des technologies robustes et populaires telles que **Go** pour le back-end, **React** pour le front-end et **Kubernetes** pour le déploiement, nous avons souhaité créer une application non seulement performante, mais également facile à maintenir et évolutive.

L'objectif principal de ce projet était de développer une application **full-stack**, capable de gérer une communication en temps réel entre plusieurs utilisateurs, tout en respectant des contraintes techniques précises : utilisation de **conteneurs Docker**, orchestration via **Kubernetes**, et déploiement dans un environnement cloud pour assurer la **scalabilité**. À travers ce projet, nous avons également exploré des pratiques DevOps modernes, en intégrant des concepts tels que **l'intégration continue** et **le déploiement continu** (CI/CD).

Ce rapport vous emmène à travers les différentes étapes de la conception et du développement de **YasChat**, en mettant en avant nos choix technologiques, les défis rencontrés et les solutions mises en place pour réaliser cette application de messagerie performante et évolutive. Chaque étape a été l'occasion de mettre en œuvre des concepts avancés tout en apprenant et en explorant les meilleures pratiques du développement moderne.

## 1. Architecture du Projet

L'architecture de **YasChat** repose sur une approche moderne, fondée sur **une série de microservices** interconnectés, déployés dans un environnement Kubernetes. Cette architecture découpe l'application en plusieurs services autonomes et spécialisés, chacun responsable d'une fonction spécifique. Voici les principaux services qui composent l'application :

1. **Frontend (React)**
   Le service Frontend est responsable de l'interface utilisateur de l'application. Développé avec **React**, ce service génère une interface dynamique et réactive permettant aux utilisateurs de communiquer facilement en temps réel. C'est le point d'entrée visible de l'utilisateur, où il peut envoyer des messages, consulter des conversations et interagir avec les autres fonctionnalités de l'application.

2. **Backend (Go)**
   Le service Backend, quant à lui, est développé en **Go (Golang)**. Il gère toutes les interactions principales de l'application, notamment l'envoi et la réception de messages, ainsi que la gestion des conversations. Ce service expose des API REST pour permettre la communication avec le Frontend et d'autres services, garantissant

ainsi la fluidité et la performance de l'application, notamment grâce à la gestion efficace des requêtes concurrentes offerte par Go.

3. **Users  (Java Spring Boot)**

Le service **Users**  est un microservice développé en **Java Spring Boot**. Ce service gère la liste des utilisateurs de l'application et leurs interactions. Il permet d'ajouter de nouveaux utilisateurs à l'application et de récupérer la liste actuelle des utilisateurs. Ce service expose une API REST avec les routes suivantes :

- **GET /users** : Cette route retourne la liste des utilisateurs inscrits dans l'application. Elle permet au front-end d'afficher les utilisateurs disponibles.

- **POST /users** : Cette route permet d'ajouter un nouvel utilisateur à la liste des utilisateurs. Le nom de l'utilisateur est envoyé dans le corps de la requête.

Le service **Users**  est essentiel pour la gestion des utilisateurs dans l'application YasChat, et il est conçu pour être extensible et facilement intégré avec d'autres services de l'application (comme le back-end et le front-end).

# Instructions de Lancement en Local

Avant de passer à la phase de conteneurisation avec Docker ou de déploiement sur Kubernetes, nous avons d'abord développé et testé l'application YasChat en local. Cette étape permet de s'assurer que chaque composant fonctionne correctement de manière autonome. Pour simplifier l'exécution de l'application pendant le développement, nous avons utilisé des scripts Make qui automatisent le lancement des services front-end et back-end, éliminant ainsi la nécessité de saisir manuellement des commandes complexes.

## Lancer le back-end (Go)

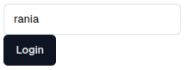
```
make start-backend
```

Cette commande démarre le **serveur Go**, qui expose une **API REST** dédiée à la gestion des messages, des utilisateurs, et des interactions principales de l'application. Le serveur Go est le cœur du back-end et assure la communication avec le front-end via des requêtes HTTP.

## Lancer le front-end (React)

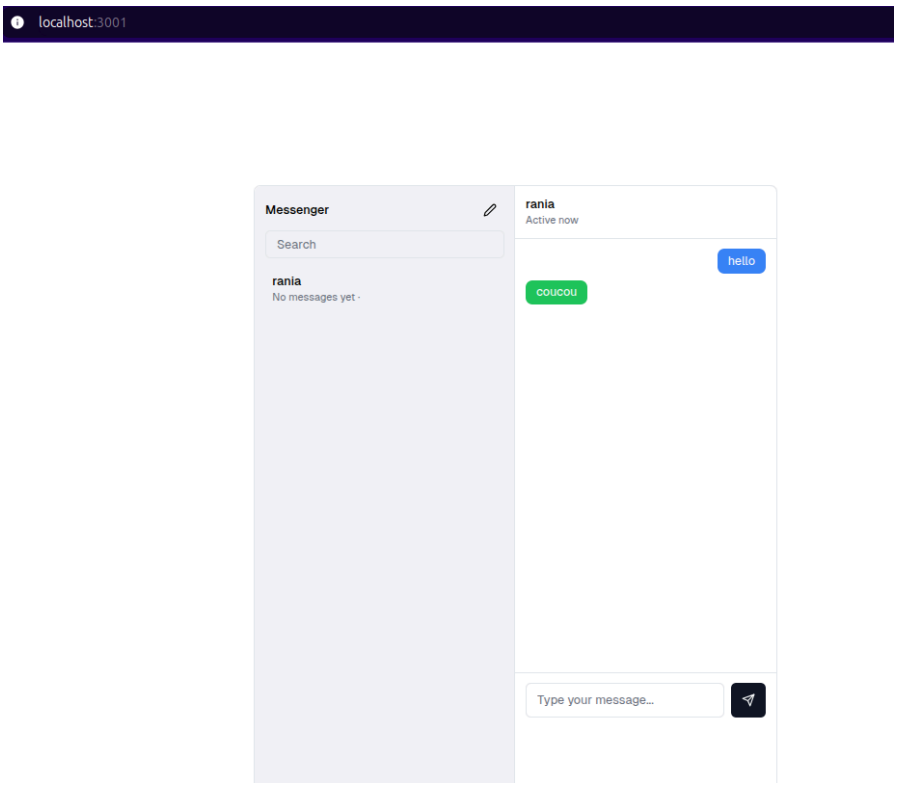
```
make start-frontend
```

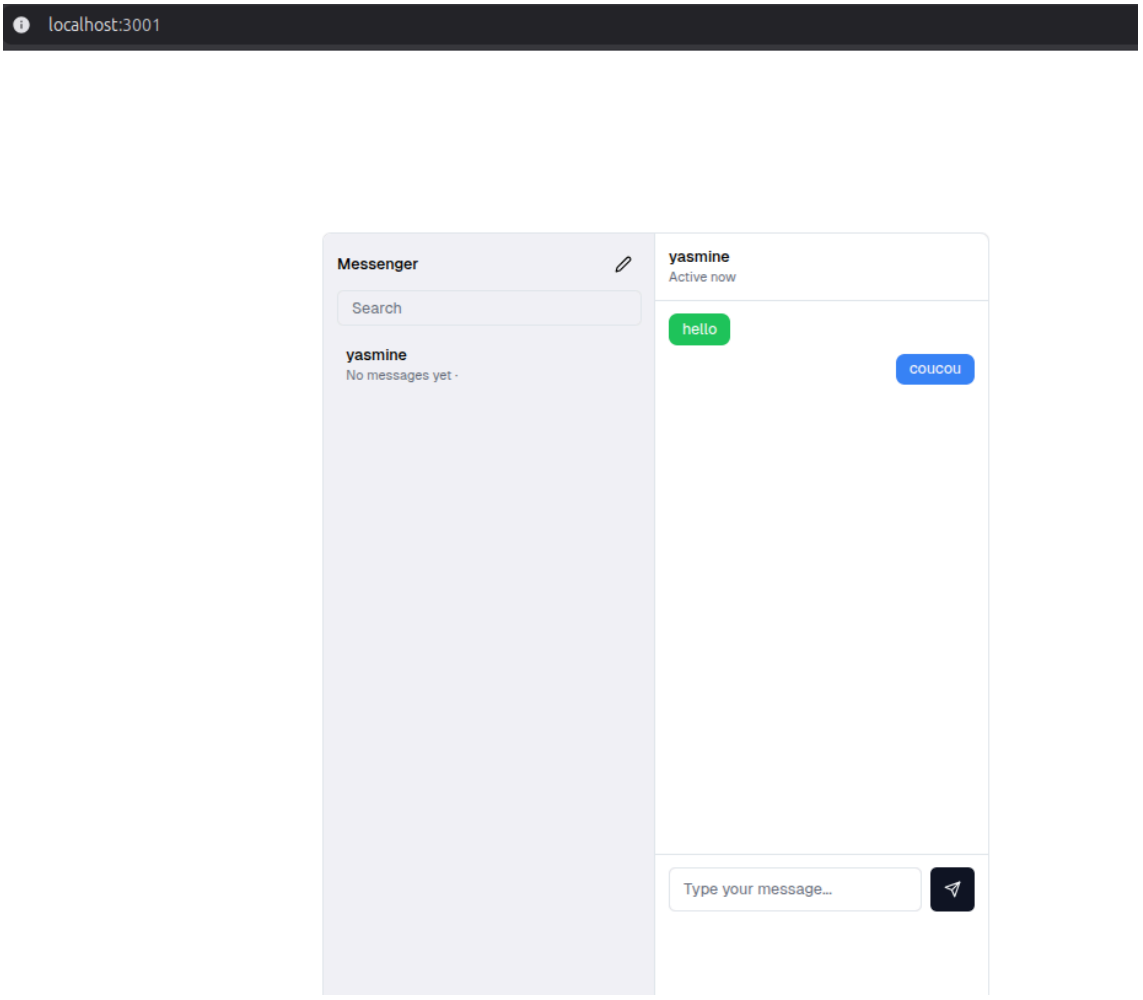
Cette commande lance le **client React**, qui constitue l'interface utilisateur de l'application. Il se connecte automatiquement à l'API du back-end et permet à l'utilisateur d'interagir avec l'application de messagerie, en envoyant des messages et en visualisant les interactions en temps réel.


```
paris-cite@paris-cite:~/web/yaschat$ make start-backend
cd backend && go run cmd/messaging-app/main.go
2025/04/15 15:28:37 Starting server on port 8080
2025/04/15 15:28:37 CSV file initialized successfully
```

```
paris-cite@paris-cite:~/web/yaschat$ make start-frontend
cd frontend && npm run dev

> new-frontend@0.1.0 dev
> next dev

  ▲ Next.js 14.2.8
  - Local:        http://localhost:3000

 ✓ Starting...
 ✓ Ready in 1435ms
 ○ Compiling / ...
 ✓ Compiled / in 1891ms (575 modules)
 GET / 200 in 2083ms
 ✓ Compiled in 163ms (274 modules)
 GET / 200 in 45ms
 ✓ Compiled /favicon.ico in 222ms (314 modules)
 GET /favicon.ico 200 in 280ms
 GET / 200 in 79ms
 GET /favicon.ico 200 in 8ms
 GET / 200 in 27ms
 GET /favicon.ico 200 in 4ms
 GET / 200 in 15ms
 GET /favicon.ico 200 in 11ms
```


localhost:3001
Messenger
Search
rania
No messages yet ·
rania
Active now
hello
coucou
Type your message...


localhost:3001
Messenger
Search
yasmine
No messages yet ·
yasmine
Active now
hello
coucou
Type your message...

# 🧱 Partie 1 : Conteneurisation avec Docker

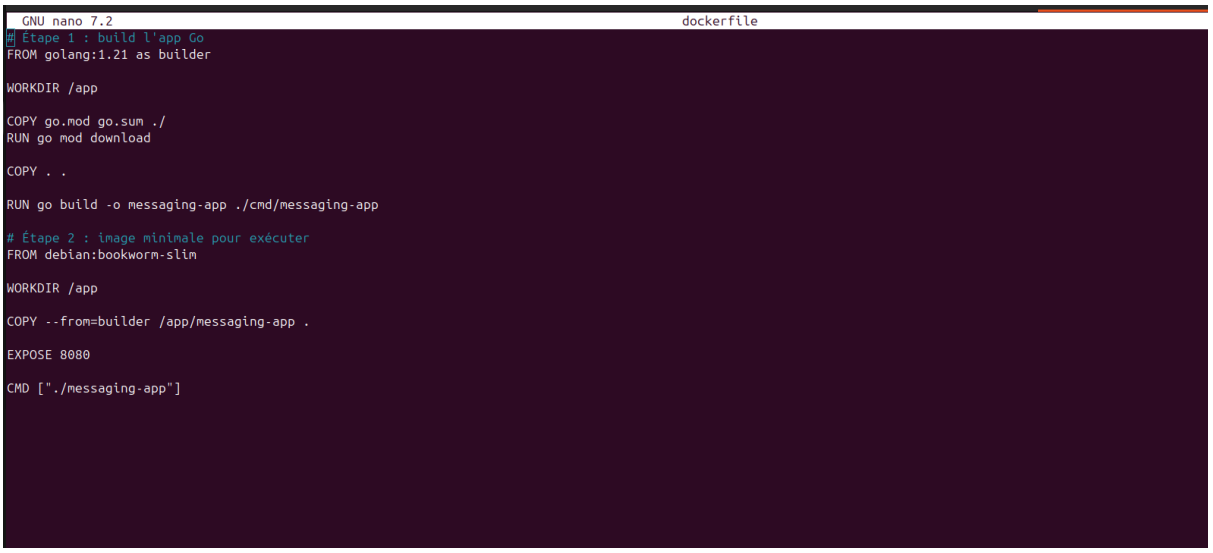
## Objectif

La première étape de la mise en production de l'application *YasChat* a consisté à conteneuriser les deux composants principaux de l'application : le **back-end en Go** et le **front-end en React**. L'utilisation de Docker permet d'assurer une portabilité, une reproductibilité et une facilité de déploiement, quel que soit l'environnement hôte.

## 📦 Création des Dockerfiles

Pour chaque composant, un `Dockerfile` a été créé :

### Backend (`backend/Dockerfile`)

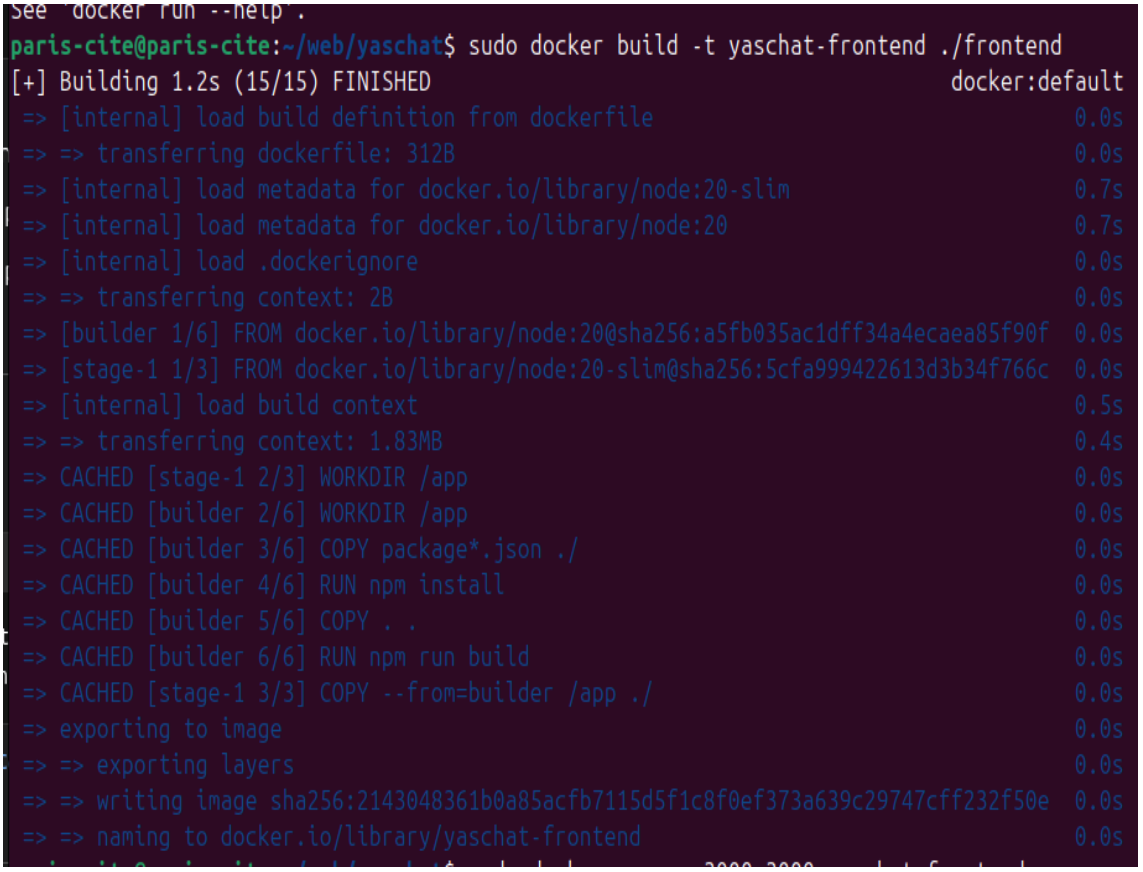


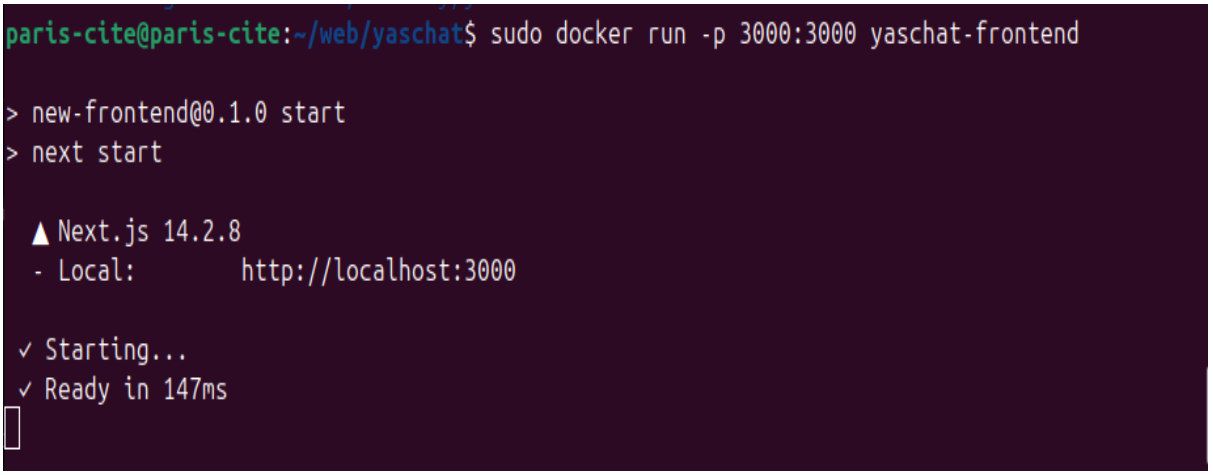
```
 GNU nano 7.2                                                dockerfile
# Étape 1 : build l'app Go
FROM golang:1.21 as builder

WORKDIR /app

COPY go.mod go.sum ./
RUN go mod download

COPY . .

RUN go build -o messaging-app ./cmd/messaging-app

# Étape 2 : image minimale pour exécuter
FROM debian:bookworm-slim

WORKDIR /app

COPY --from=builder /app/messaging-app .

EXPOSE 8080

CMD ["./messaging-app"]
```

### Frontend (`frontend/Dockerfile`)

```
 GNU nano 7.2                                                dockerfile
# Étape 1: build Next.js
FROM node:20 AS builder

WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build

# Étape 2: image finale avec app en prod
FROM node:20-slim

WORKDIR /app

COPY --from=builder /app ./

EXPOSE 3000

CMD ["npm", "start"]
```

Ces fichiers définissent comment construire les images à partir du code source, installer les dépendances, compiler (si besoin), exposer les bons ports, et lancer l'application.



```
paris-cite@paris-cite:~/web/yaschat$ sudo docker build -t yaschat-backend ./backend
[+] Building 51.8s (14/14) FINISHED                                    docker:default
 => [internal] load build definition from dockerfile                           0.0s
 => => transferring dockerfile: 357B                                           0.0s
 => [internal] load metadata for docker.io/library/alpine:latest               1.3s
 => [internal] load metadata for docker.io/library/golang:1.18                 1.3s
 => [internal] load .dockerignore                                              0.0s
 => => transferring context: 2B                                                0.0s
 => [stage-1 1/3] FROM docker.io/library/alpine:latest@sha256:a8560b36e8b8210634f77d  0.0s
 => => resolve docker.io/library/alpine:latest@sha256:a8560b36e8b8210634f77d9f7f9efd  0.0s
 => => sha256:a8560b36e8b8210634f77d9f7f9efd7ffa463e380b75e2e74aff45 9.22kB / 9.22kB  0.0s
 => => sha256:1c4eef651f65e2f7daee7ee785882ac164b02b78fb74503052a26d 1.02kB / 1.02kB  0.0s
 => => sha256:aded1e1a5b3705116fa0a92ba074a5e0b0031647d9c315983ccba2ee54 581B / 581B  0.0s
 => [builder 1/5] FROM docker.io/library/golang:1.18@sha256:50c889275d26f816b5314fc  49.0s
 => => resolve docker.io/library/golang:1.18@sha256:50c889275d26f816b5314fc99f55425f  0.0s
 => => sha256:740324e52de766f230ad7113fac9028399d6e03af34883de625dc2 1.80kB / 1.80kB  0.0s
 => => sha256:c37a56a6d65476eabfb50e74421f16f415093e2d1bdd7f83e8bbb4 7.12kB / 7.12kB  0.0s
 => => sha256:f049f75f014ee8fec2d4728b203c9cbee0502ce142aec030f874aa 5.16MB / 5.16MB  3.3s
 => => sha256:56261d0e6b05ece42650b14830960db5b42a9f23479d868256f9 10.88MB / 10.88MB  6.1s
 => => sha256:50c889275d26f816b5314fc99f55425fa76b18fcaf16af255f5d57 2.36kB / 2.36kB  0.0s
 => => sha256:bbeef03cda1f5d6c9e20c310c1c91382a6b0a1a2501c3436b28 55.03MB / 55.03MB  25.8s
 => => sha256:9bd150679dbdb02d9d4df4457d54211d6ee719ca7bc77747a7b 54.58MB / 54.58MB  24.7s
 => => sha256:bfcb68b5bd105d3f88a2c15354cff6c253bedc41d83c1da28b3 85.98MB / 85.98MB  33.9s
 => => sha256:06d0c5d18ef41fa1c2382bd2afd189a01ebfff4910b868879 141.98MB / 141.98MB  46.2s
 => => extracting sha256:bbeef03cda1f5d6c9e20c310c1c91382a6b0a1a2501c3436b28152f1389  1.1s
 => => sha256:cc7973a07a5b4a44399c5d36fa142f37bb343bb123a3736357365fd90 156B / 156B  25.9s
 => => extracting sha256:f049f75f014ee8fec2d4728b203c9cbee0502ce142aec030f874aa28359  0.1s
 => => extracting sha256:56261d0e6b05ece42650b14830960db5b42a9f23479d868256f91d96869  0.1s
 => => extracting sha256:9bd150679dbdb02d9d4df4457d54211d6ee719ca7bc77747a7be4cd99ae  1.1s
 => => extracting sha256:bfcb68b5bd105d3f88a2c15354cff6c253bedc41d83c1da28b3d686c37c  1.0s
 => => extracting sha256:06d0c5d18ef41fa1c2382bd2afd189a01ebfff4910b868879b6dcfeef46  2.6s
 => => extracting sha256:cc7973a07a5b4a44399c5d36fa142f37bb343bb123a3736357365fd9040  0.0s
 => [internal] load build context                                              0.0s
 => => transferring context: 6.96MB                                            0.0s
 => [stage-1 2/3] WORKDIR /root/                                               0.0s
 => [builder 2/5] WORKDIR /app                                                 0.2s
 => [builder 3/5] COPY . .                                                     0.0s
 => [builder 4/5] RUN go mod tidy                                              0.4s
 => [builder 5/5] RUN go build -o messaging-app cmd/messaging-app/main.go      0.7s
 => [stage-1 3/3] COPY --from=builder /app/messaging-app .                     0.0s
 => exporting to image                                                         0.0s
 => => exporting layers                                                        0.0s
 => => writing image sha256:0f83604c72506f7e7c8d4834b321864032d6447b2bd6ff26497ec5c8  0.0s
 => => naming to docker.io/library/yaschat-backend                             0.0s
```

```
paris-cite@paris-cite:~/web/yaschat$ sudo docker run -p 8080:8080 yaschat-backend

> new-frontend@0.1.0 start
> next start

  ▲ Next.js 14.2.8
  - Local:        http://localhost:3000

 ✓ Starting...
 ✓ Ready in 148ms
```

## 🧪 Construction des images

Les images Docker ont été construites localement à l'aide des commandes :

```
docker build -t yaschat-backend ./backend
docker build -t yaschat-frontend ./frontend
```

## 📤 Publication sur Docker Hub

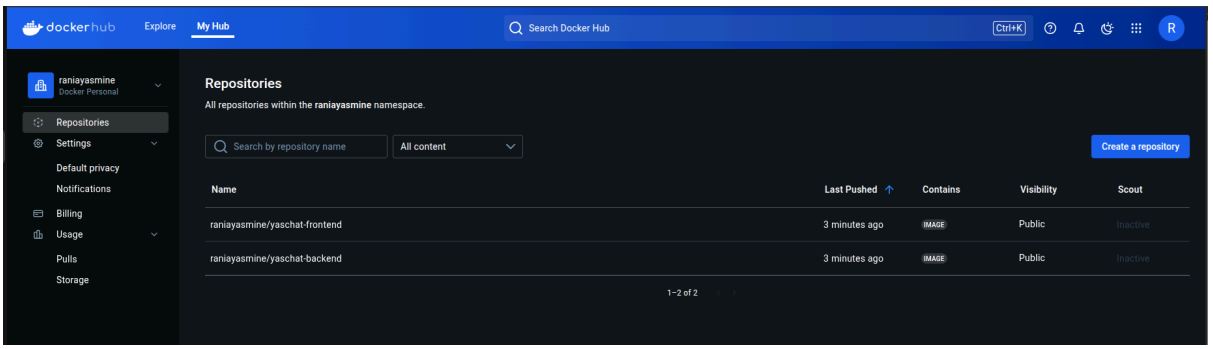
Pour faciliter le déploiement, les images ont été publiées sur Docker Hub :

```
paris-cite@paris-cite:~/web/yaschat$ sudo docker login

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

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

Waiting for authentication in the browser…



WARNING! Your password will be stored unencrypted in /root/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credential-stores

Login Succeeded
```

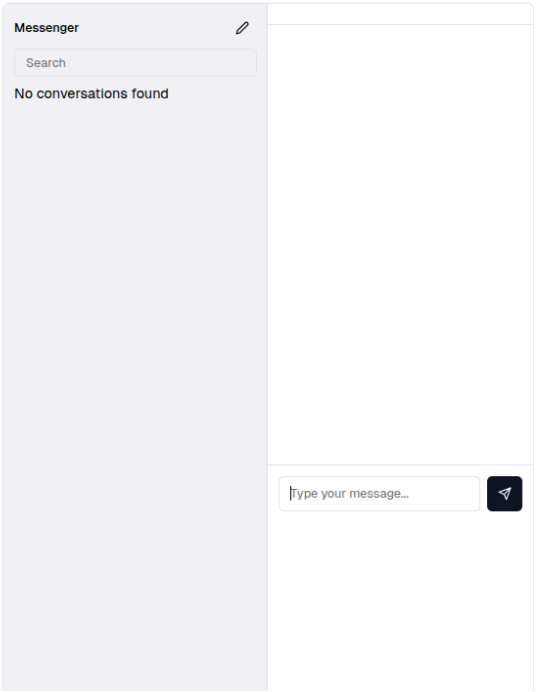
```
paris-cite@paris-cite:~/web/yaschat$ sudo docker info | grep Username
 Username: raniayasmine
WARNING: bridge-nf-call-iptables is disabled
WARNING: bridge-nf-call-ip6tables is disabled
```

```
paris-cite@paris-cite:~/web/yaschat$ sudo docker tag yaschat-backend raniayasmine/yaschat-backend:latest
[sudo] password for paris-cite:
paris-cite@paris-cite:~/web/yaschat$ sudo docker tag yaschat-frontend raniayasmine/yaschat-frontend:latest
```

```
paris-cite@paris-cite:~/web/yaschat$ sudo docker push raniayasmine/yaschat-backend:latest
The push refers to repository [docker.io/raniayasmine/yaschat-backend]
b097a6f50a0f: Pushed
2440079e03db: Pushed
6f2931d8686b: Mounted from library/node
1757b9451247: Mounted from library/node
a4323cc5b526: Mounted from library/node
b00e5af731b0: Mounted from library/node
ea680fbff095: Mounted from library/node
latest: digest: sha256:9f9509b09101f45e6dfb6bf18f0c90e9cebdbcc1300a7af85eda80baa55dbbb6 size: 1786
paris-cite@paris-cite:~/web/yaschat$ sudo docker push raniayasmine/yaschat-frontend:latest
The push refers to repository [docker.io/raniayasmine/yaschat-frontend]
b097a6f50a0f: Mounted from raniayasmine/yaschat-backend
2440079e03db: Mounted from raniayasmine/yaschat-backend
6f2931d8686b: Mounted from raniayasmine/yaschat-backend
1757b9451247: Mounted from raniayasmine/yaschat-backend
a4323cc5b526: Mounted from raniayasmine/yaschat-backend
b00e5af731b0: Mounted from raniayasmine/yaschat-backend
ea680fbff095: Mounted from raniayasmine/yaschat-backend
latest: digest: sha256:9f9509b09101f45e6dfb6bf18f0c90e9cebdbcc1300a7af85eda80baa55dbbb6 size: 1786
paris-cite@paris-cite:~/web/yaschat$
```








## ✅ Résultat

Une fois les conteneurs en cours d'exécution, l'application est accessible sur :

- `http://localhost:3000` (interface React)
- `http://localhost:8080` (API Go)

# ☸️ Déploiement avec Kubernetes

## 🎯 Objectif

Après avoir conteneurisé les deux services de *YasChat*, nous avons mis en place un déploiement via Kubernetes, en utilisant Minikube comme cluster local. Cette étape permet de tester le comportement de l'application dans un environnement similaire à celui d'un cloud réel, avec gestion des pods, services, redémarrages automatiques, etc.

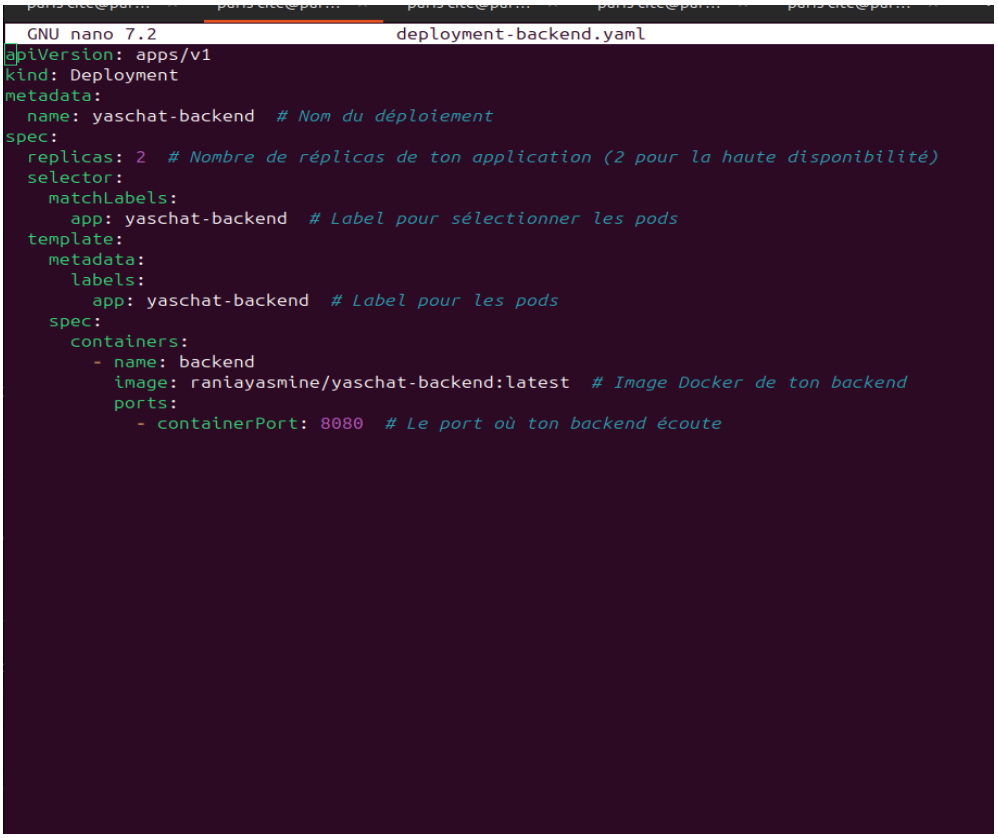
## 📄 Configuration Kubernetes

### 📁 Organisation des fichiers

Nous avons créé les fichiers YAML suivants dans un dossier `k8s/` :

- `deployment-backend.yaml`
- `service-backend.yaml`
- `deployment-frontend.yaml`
- `service-frontend.yaml`

✍️ **`deployment-backend.yaml`**



## ✍️ service-backend.yaml

```
  GNU nano 7.2                          service-backend.yaml
apiVersion: v1
kind: Service
metadata:
  name: yaschat-backend
spec:
  type: NodePort
  selector:
    app: yaschat-backend
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 30001  # tu peux choisir une valeur entre 30000 et 32767
```

## ✍️ deployment-frontend.yaml

```
  GNU nano 7.2                        deployment-frontend.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yaschat-frontend  # Nom du déploiement
spec:
  replicas: 2  # Nombre de réplicas de ton application
  selector:
    matchLabels:
      app: yaschat-frontend  # Label pour sélectionner les pods
  template:
    metadata:
      labels:
        app: yaschat-frontend  # Label pour les pods
    spec:
      containers:
        - name: frontend
          image: raniayasmine/yaschat-frontend:latest  # Image Docker du frontend
          ports:
            - containerPort: 3000  # Le port où ton frontend écoute
```

## ✍️ service-frontend.yaml

```
  GNU nano 7.2                    service-frontend.yaml
apiVersion: v1
kind: Service
metadata:
  name: yaschat-frontend
spec:
  type: NodePort
  selector:
    app: yaschat-frontend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
      nodePort: 30002
```

## 🚀 Déploiement sur Minikube

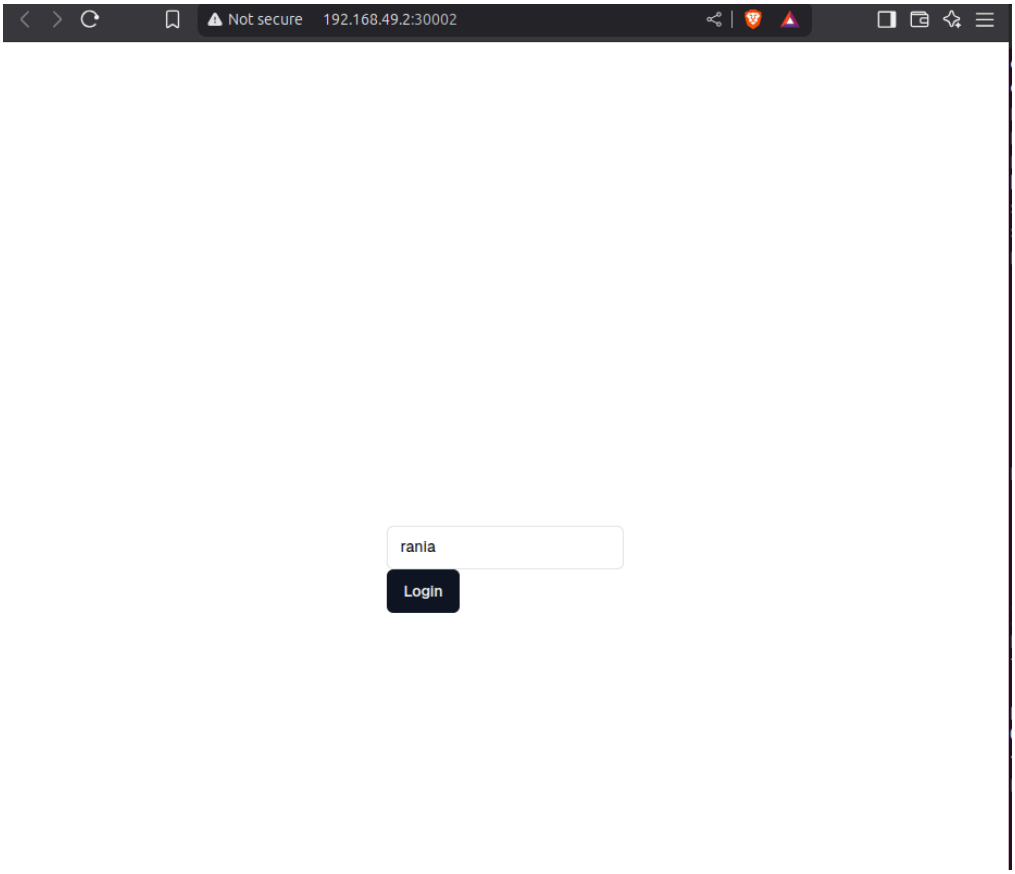
```
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl apply -f deployment-backend.yaml
deployment.apps/yaschat-backend created
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl apply -f service-backend.yaml
service/yaschat-backend created
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl apply -f deployment-frontend.yaml
deployment.apps/yaschat-frontend created
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl apply -f service-frontend.yaml
service/yaschat-frontend created
```

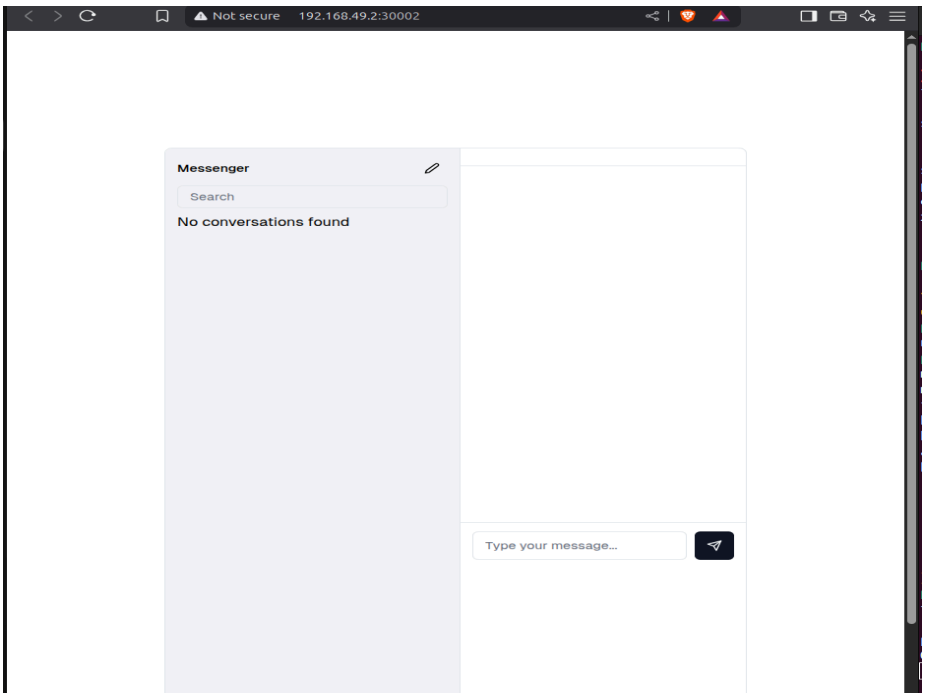
```
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
yaschat-backend-665dcc6669-8w5d5    1/1     Running   0          66s
yaschat-backend-665dcc6669-xx44f    1/1     Running   0          66s
yaschat-frontend-7c4d4d6ddd-7ljm6   1/1     Running   0          39s
yaschat-frontend-7c4d4d6ddd-stw9k   1/1     Running   0          39s
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl get services
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
```

```
yaschat-frontend-7c4d4d6ddd-7ljm6   1/1     Running   0          39s
yaschat-frontend-7c4d4d6ddd-stw9k   1/1     Running   0          39s
paris-cite@paris-cite:~/web/yaschat/k8s$ kubectl get services
NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
kubernetes         ClusterIP   10.96.0.1        <none>        443/TCP    2m11s
yaschat-backend    ClusterIP   10.102.148.190   <none>        80/TCP     59s
yaschat-frontend   ClusterIP   10.101.133.183   <none>        80/TCP     37s
paris-cite@paris-cite:~/web/yaschat/k8s$
```

```
paris-cite@paris-cite:~/web/yaschat/k8s$ minikube service list
|-------------|-----------------|-------------|---------------------------|
|  NAMESPACE  |       NAME      | TARGET PORT |            URL            |
|-------------|-----------------|-------------|---------------------------|
| default     | kubernetes      | No node port |                           |
| default     | yaschat-backend |         8080 | http://192.168.49.2:30001 |
| default     | yaschat-frontend |         3000 | http://192.168.49.2:30002 |
| kube-system | kube-dns        | No node port |                           |
|-------------|-----------------|-------------|---------------------------|
```

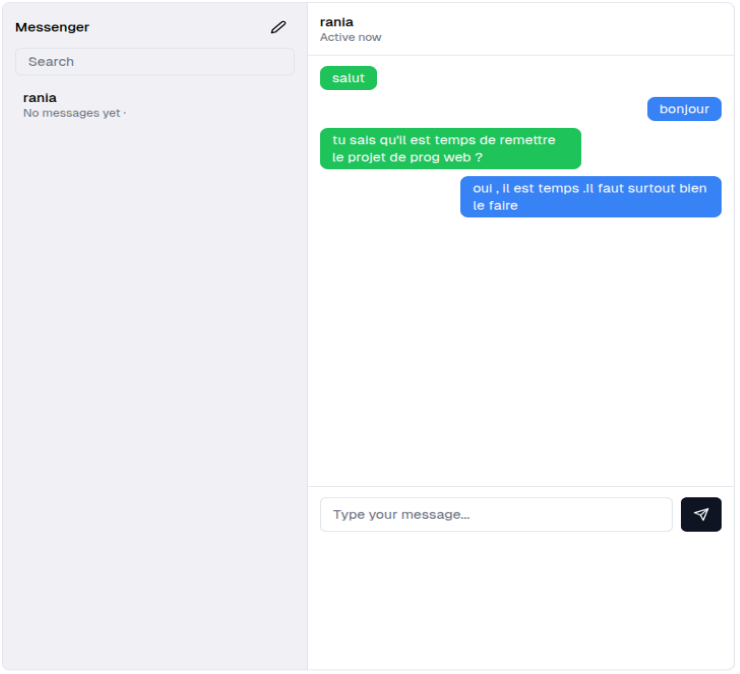
```
paris-cite@paris-cite:~/web/yaschat/k8s$ minikube service yaschat-frontend
|-----------|-----------------|-------------|---------------------------|
| NAMESPACE |       NAME      | TARGET PORT |            URL            |
|-----------|-----------------|-------------|---------------------------|
| default   | yaschat-frontend |        3000 | http://192.168.49.2:30002 |
|-----------|-----------------|-------------|---------------------------|
🎉  Opening service default/yaschat-frontend in default browser...
paris-cite@paris-cite:~/web/yaschat/k8s$ update.go:85: cannot change mount namespace acco
ing to change mount (/run/user/1000/doc/by-app/snap.brave /run/user/1000/doc none bind,rw
-snapd.ignore-missing 0 0): cannot inspect "/run/user/1000/doc": lstat /run/user/1000/doc
permission denied
Opening in existing browser session.
```



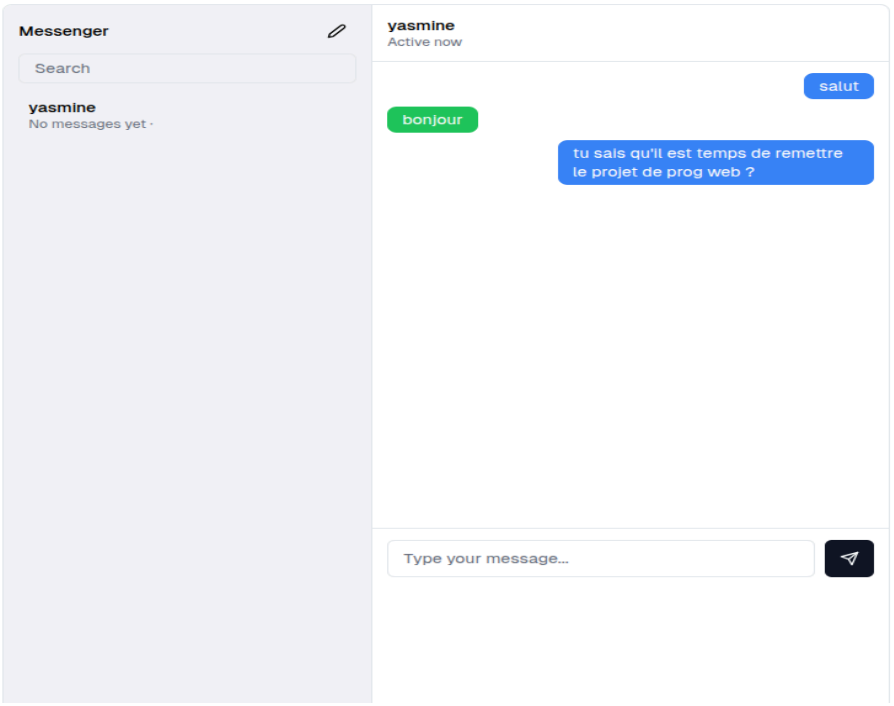



```
paris-cite@paris-cite:~/web/yaschat/k8s$ minikube status
minikube service yaschat-backend
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

|-----------|-----------------|-------------|--------------------------|
| NAMESPACE |      NAME       | TARGET PORT |            URL           |
|-----------|-----------------|-------------|--------------------------|
| default   | yaschat-backend |        8080 | http://192.168.49.2:30001 |
|-----------|-----------------|-------------|--------------------------|
🎉  Opening service default/yaschat-backend in default browser...
paris-cite@paris-cite:~/web/yaschat/k8s$ update.go:85: cannot change mount namespace accord
ing to change mount (/run/user/1000/doc/by-app/snap.brave /run/user/1000/doc none bind,rw,x
-snapd.ignore-missing 0 0): cannot inspect "/run/user/1000/doc": lstat /run/user/1000/doc:
permission denied
Opening in existing browser session.
```


Not secure
192.168.49.2:30002


Messenger
Search
rania
No messages yet
rania
Active now
salut
bonjour
tu sais qu'il est temps de remettre le projet de prog web ?
oui , il est temps .Il faut surtout bien le faire
Type your message...


Not secure
192.168.49.2:30002


Messenger
Search
yasmine
No messages yet
yasmine
Active now
salut
bonjour
tu sais qu'il est temps de remettre le projet de prog web ?
Type your message...

### 🌐 Accès à l'application

Minikube expose les services localement. Pour accéder à l'interface front-end, on peut utiliser :

```
minikube service yaschat-frontend
```

Cela ouvre automatiquement l'URL du service dans le navigateur.

# 🔁 Mise en place d'une Gateway Ingress pour exposer les services YasChat

### 📌 Objectif

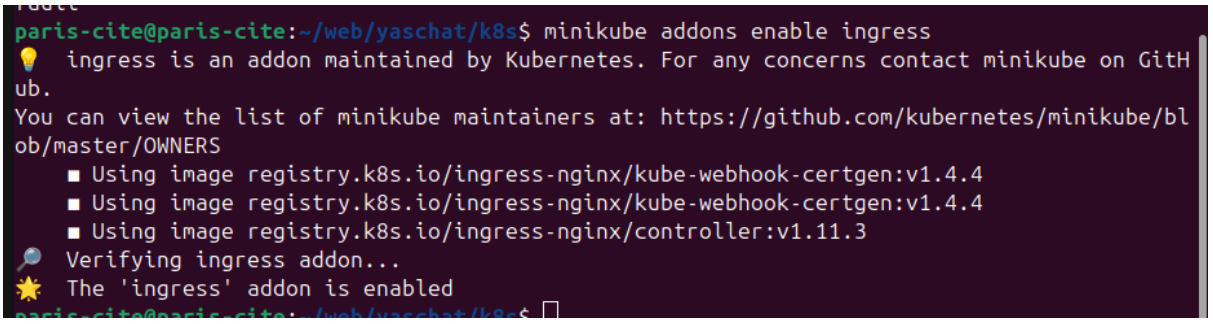
L'objectif de cette étape est de configurer une gateway HTTP en local à l'aide d'un Ingress Controller dans Kubernetes, afin de permettre un accès via une URL conviviale (`http://yaschat.local`) aux différents services de l'application YasChat (frontend et backend).

### ⚙️ Activation du contrôleur Ingress dans Minikube

Pour utiliser un Ingress Controller dans un environnement local basé sur Minikube, il est nécessaire d'activer l'addon `ingress` :

```
minikube addons enable ingress
```

Cette commande déploie automatiquement un Ingress Controller NGINX dans le cluster.



## 🛠️ Définition de l’Ingress

Une ressource `Ingress` a été créée afin de router les requêtes HTTP vers les bons services internes. Le fichier suivant (`yaschat-ingress.yaml`) définit les routes suivantes :

- **`/` redirige vers le frontend React (port 3000)**
- **`/api` redirige vers le backend Go (port 8080)**

```
  GNU nano 7.2                          yaschat-ingress.yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: yaschat-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
  - host: yaschat.local
    http:
      paths:
      - path: /(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: yaschat-frontend
            port:
              number: 3000
      - path: /api(/|$)(.*)
        pathType: ImplementationSpecific
        backend:
          service:
            name: yaschat-backend
            port:
              number: 8080
```

## 🗂️ Ajout du nom de domaine local

Pour que l'URL `http://yaschat.local` soit reconnue par le système, une entrée a été ajoutée dans le fichier `/etc/hosts` :

```
echo "$(minikube ip) yaschat.local" | sudo tee -a /etc/hosts
```

Cela permet à l’environnement local de résoudre ce nom vers l’adresse IP du cluster Minikube.

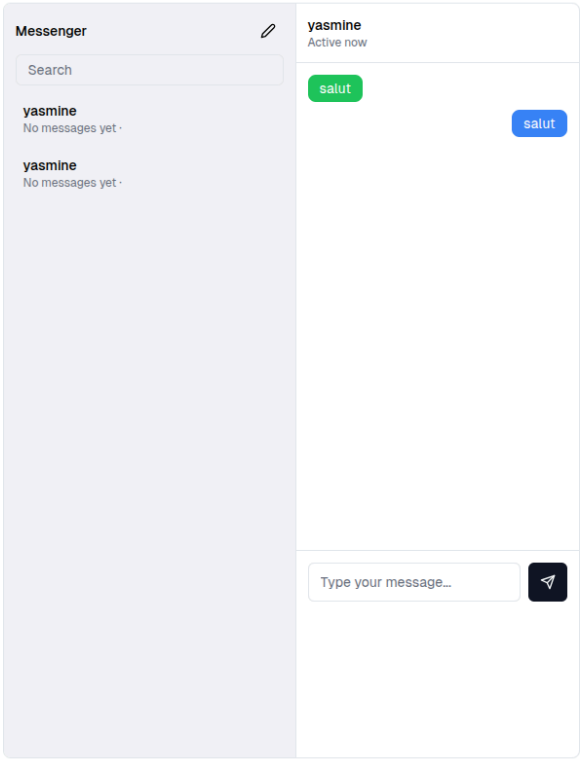
## ✅ Déploiement et test

Une fois la configuration appliquée :

```
kubectl apply -f yaschat-ingress.yaml
```

Les services ont été exposés et peuvent être testés via un navigateur :

- **Interface utilisateur (frontend) :** `http://yaschat.local`
- **API backend :** `http://yaschat.local/api`

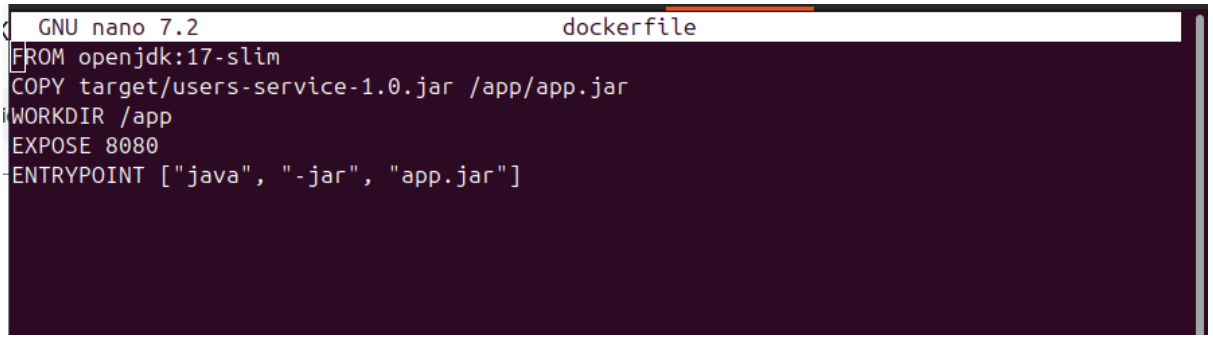
# 🔧 Ajout du service users

### 📌 Objectif

Dans le cadre du projet **YasChat**, j'ai conçu un second microservice appelé users-service, en complément du service initial . Ce nouveau service permet de gérer les utilisateurs, et il est destiné à interagir avec les autres composants de l'application via des appels HTTP REST.

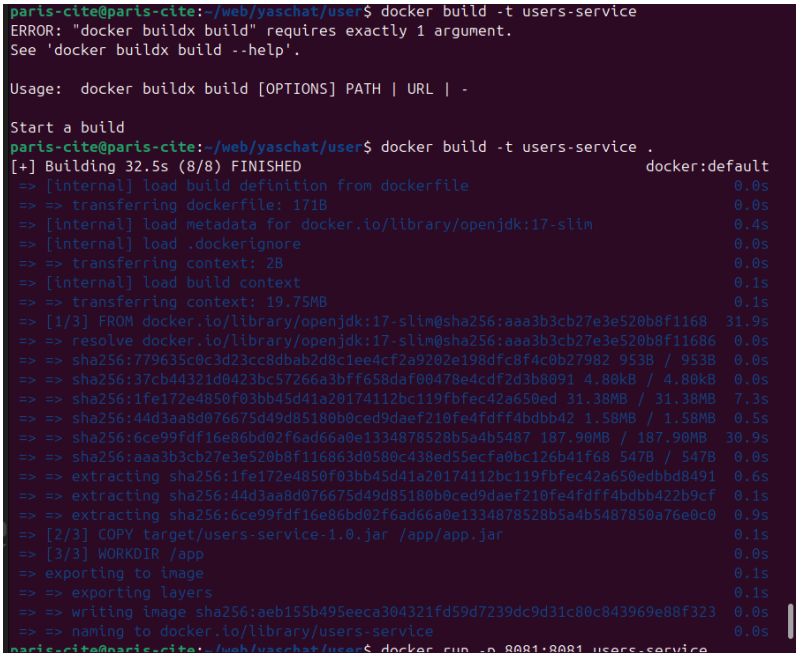
## ⚙️ Dockerisation

Comme pour les autres services de l'application, le service **Users** a été conteneurisé à l'aide de Docker. Un fichier Dockerfile a été créé pour définir le processus de construction de l'image Docker du service. Ce fichier spécifie comment construire l'image, installer les dépendances nécessaires et exposer le port adéquat pour que le service soit accessible.



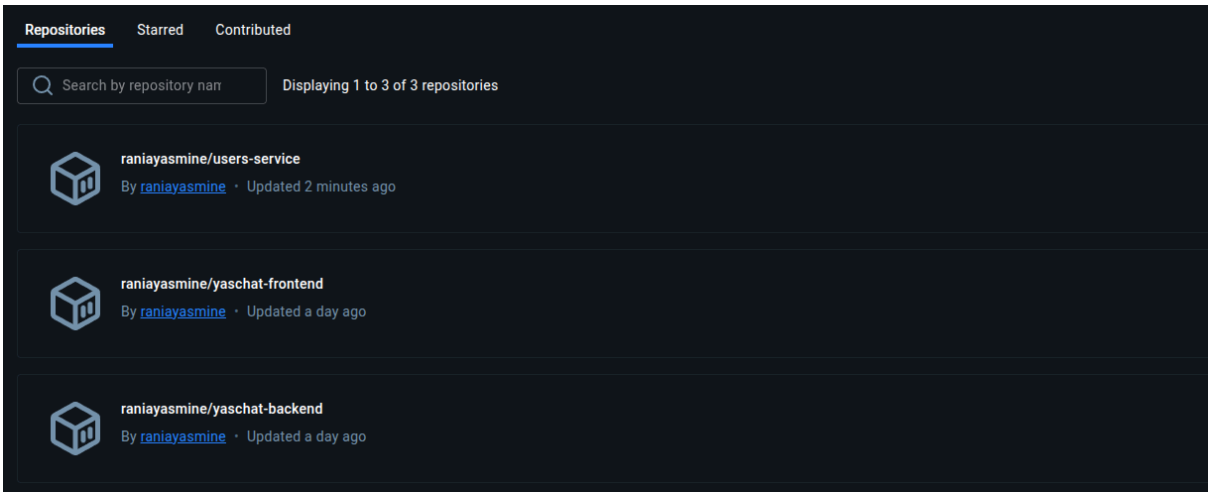

L'image a été construite avec :

```
docker build -t users-service .
```



```
paris-cite@paris-cite:~/web/yaschat/user$ docker run -p 8081:8081 users-service

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.3)

2025-04-17T18:00:02.838Z  INFO 1 --- [           main] com.yaschat.UsersService
        : Starting UsersService v1.0 using Java 17.0.2 with PID 1 (/app/app.jar starte
d by root in /app)
2025-04-17T18:00:02.841Z  INFO 1 --- [           main] com.yaschat.UsersService
        : No active profile set, falling back to 1 default profile: "default"
2025-04-17T18:00:03.286Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWe
bServer  : Tomcat initialized with port 8080 (http)
2025-04-17T18:00:03.294Z  INFO 1 --- [           main] o.apache.catalina.core.StandardS
ervice   : Starting service [Tomcat]
2025-04-17T18:00:03.294Z  INFO 1 --- [           main] o.apache.catalina.core.StandardE
ngine    : Starting Servlet engine: [Apache Tomcat/10.1.19]
2025-04-17T18:00:03.312Z  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[
/]       : Initializing Spring embedded WebApplicationContext
2025-04-17T18:00:03.312Z  INFO 1 --- [           main] w.s.c.ServletWebServerApplicatio
nContext : Root WebApplicationContext: initialization completed in 417 ms
2025-04-17T18:00:03.483Z  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWe
bServer  : Tomcat started on port 8080 (http) with context path ''
2025-04-17T18:00:03.490Z  INFO 1 --- [           main] com.yaschat.UsersService
        : Started UsersService in 0.828 seconds (process running for 1.013)
```

```
paris-cite@paris-cite:~/web/yaschat/user$ docker tag users-service raniayasmine/users-s
ervice:latest
paris-cite@paris-cite:~/web/yaschat/user$ docker push raniayasmine/users-service:latest

The push refers to repository [docker.io/raniayasmine/users-service]
5f70bf18a086: Pushed
f846506c0db5: Pushed
6be690267e47: Mounted from library/openjdk
13a34b6fff78: Mounted from library/openjdk
9c1b6dd6c1e6: Mounted from library/openjdk
latest: digest: sha256:5f5aea0c1d456ffaf0a8c0b41a508e2a645de1d94198157d105f57a05e5c082d
 size: 1371
paris-cite@paris-cite:~/web/yaschat/user$
```



## ☸️ Déploiement Kubernetes - Service Users

## 🎯 Objectif

Le service Users est une composante clé de l'application YasChat, permettant de gérer les utilisateurs, d'en ajouter de nouveaux et de récupérer des informations utilisateur existantes. Le déploiement de ce service dans Kubernetes vise à intégrer ce composant de manière fluide dans l'architecture existante.

## 📂 Structure des fichiers

Le déploiement est organisé à travers les fichiers suivants :

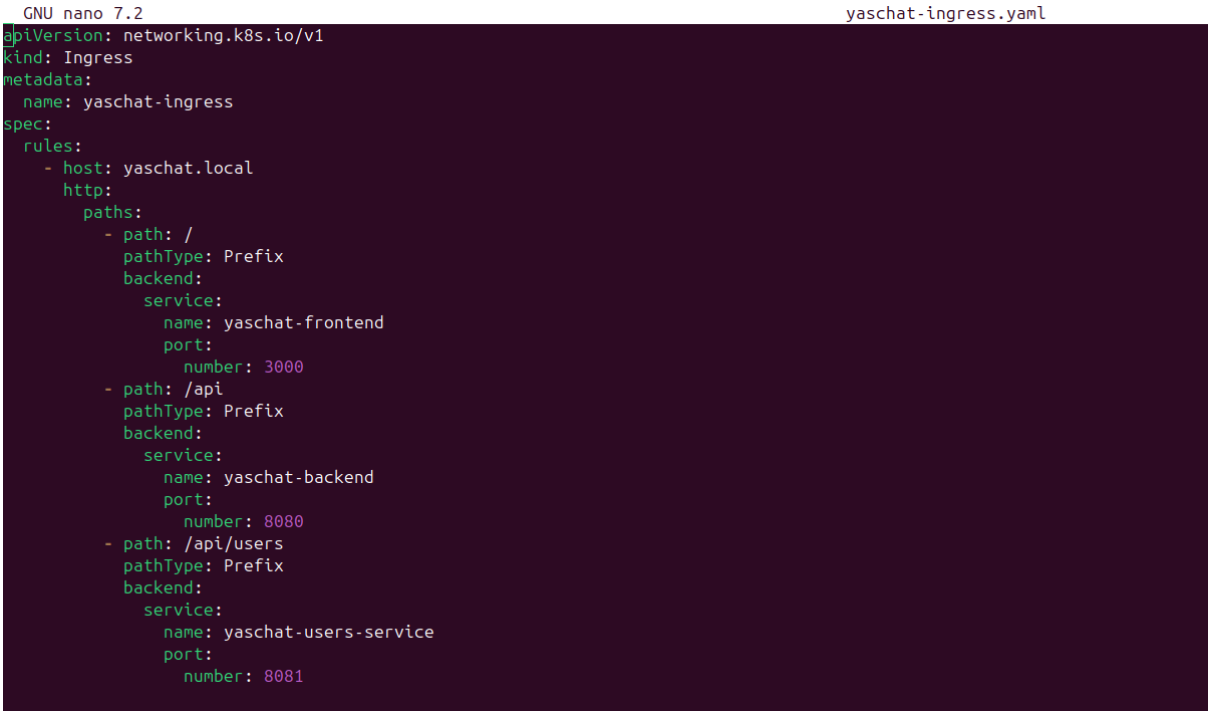
- `deployment-users.yaml` : Définit le déploiement du service Users, spécifiant le conteneur Docker, le nombre de réplicas et le port exposé.
- `service-users.yaml` : Crée un service Kubernetes pour permettre l'accès au service Users à partir des autres composants.
- `yaschat-ingress.yaml` : Met à jour l'Ingress pour router les requêtes vers le service Users via `/api/users`.

GNU nano 7.2 service-users.yaml

```yaml
apiVersion: v1
kind: Service
metadata:
  name: yaschat-users-service
spec:
  selector:
    app: yaschat-users-service
  ports:
    - protocol: TCP
      port: 8081
      targetPort: 8081
  type: ClusterIP
```

GNU nano 7.2 deployment-users.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: yaschat-users-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: yaschat-users-service
  template:
    metadata:
      labels:
        app: yaschat-users-service
    spec:
      containers:
        - name: yaschat-users-service
          image: raniayasmine/users-service:latest  # Remplacer par ton image Docker
          ports:
            - containerPort: 8081
```




## ✅ Déploiement

Les commandes pour appliquer les configurations :

kubectl apply -f deployment-users.yaml

kubectl apply -f service-users.yaml

kubectl apply -f yaschat-ingress.yaml

## 🔍 Vérification

Après le déploiement, vérifier le statut des composants :

kubectl get pods

kubectl get services

kubectl get ingress

```
paris-cite@paris-cite:~/R/yaschat/k8s$ kubectl get pods
kubectl get services
kubectl get ingress
NAME                                  READY   STATUS             RESTARTS        AGE
users-deployment-7d7cbbf9cf-h6qbx     0/1     ImagePullBackOff   0               10d
users-deployment-859d9c845c-q8h4s     1/1     Running            2 (7d21h ago)   19d
yaschat-backend-665dcc6669-k65p7      1/1     Running            2 (7d21h ago)   19d
yaschat-backend-665dcc6669-q6pxg      1/1     Running            2 (7d21h ago)   19d
yaschat-frontend-7c4d4d6ddd-7kvr2     1/1     Running            2 (7d21h ago)   19d
yaschat-frontend-7c4d4d6ddd-svddn     1/1     Running            2 (7d21h ago)   19d
yaschat-users-service-758f54b598-sjjpb   1/1     Running            1 (7d21h ago)   10d
NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes             ClusterIP   10.96.0.1        <none>        443/TCP          19d
users-service          ClusterIP   10.101.44.211    <none>        8080/TCP         19d
yaschat-backend        NodePort    10.102.210.211   <none>        8080:30001/TCP   19d
yaschat-frontend       NodePort    10.107.248.205   <none>        3000:30002/TCP   19d
yaschat-users-service  ClusterIP   10.99.100.42     <none>        8081/TCP         19d
NAME             CLASS   HOSTS           ADDRESS        PORTS   AGE
yaschat-ingress  nginx   yaschat.local   192.168.49.2   80      19d
```

## 📊 Résultats des Labs Google Cloud

Voici les résultats obtenus par moi et mon binôme à l'issue des différents labs réalisés sur la plateforme Google Cloud.