



## Rapport projet ProgWeb

M1 2024/2025

Alex JIN

Avi GILARDI

Lien github du projet :

<https://github.com/alex1350/ProjetWebM1>

## Introduction

Dans le cadre de ce projet, nous avons développé une architecture basée sur des microservices en utilisant Spring Boot. Notre objectif était de concevoir et déployer une application conteneurisée en exploitant Docker et Kubernetes via Minikube.

Nous avons suivi une approche progressive en créant d'abord un service principal, que nous avons conteneurisé et publié sur Docker Hub. Ensuite, nous avons mis en place un déploiement Minikube, accompagné d'une gateway locale pour gérer le routage des requêtes. Enfin, nous avons travaillé sur l'aspect sécurisation, en veillant à la protection de l'image Docker et en intégrant HTTPS pour garantir des échanges sécurisés.

Ce projet nous a permis d'explorer les principes fondamentaux de Kubernetes, tout en appliquant des bonnes pratiques de conteneurisation, de gestion du réseau et de sécurité dans un environnement distribué.

### **Commencer par un seul service en local (10/20)**

- Coder une mini application dans le langage que vous voulez
- Créer une image Docker => faire un Dockerfile
- Publier l'image Docker sur le Docker Hub
- Créer un déploiement Kubernetes
- Créer un service Kubernetes

Pour notre application nous avons décidé d'utiliser Spring Boot qui est une framework de développement web en java.

### **`./gradlew build`**

Pour compiler le projet java.

## Dockerfile

Pour créer une image docker de l'application nous créons tout d'abord un Dockerfile :

```
FROM openjdk:21-oracle
VOLUME /tmp
EXPOSE 8080
ADD ./build/libs/HouseRental-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Ce **Dockerfile** permet de conteneuriser l'application **HouseRental**, développée en Java avec le framework Spring Boot, afin de la rendre portable et facilement exécutable dans un environnement isolé. Il utilise comme base l'image **OpenJDK 21 (Oracle)**, garantissant un environnement d'exécution compatible avec la version du JDK nécessaire à l'application.

Tout d'abord, un volume Docker est créé sur le répertoire /tmp, ce qui permet d'y stocker temporairement des fichiers lors de l'exécution du conteneur. Ensuite, le port **8080** est exposé, correspondant au port par défaut sur lequel Spring Boot exécute les applications web. L'étape suivante consiste à ajouter le fichier **JAR** généré par la compilation du projet (HouseRental-0.0.1-SNAPSHOT.jar) à l'intérieur de l'image Docker sous le nom app.jar.

Enfin, le point d'entrée du conteneur est défini afin d'exécuter l'application. La commande utilise java -jar /app.jar, permettant ainsi de lancer le fichier JAR. Une option spécifique (-Djava.security.egd=file:/dev/./urandom) est ajoutée pour optimiser l'accès à la génération de nombres aléatoires, ce qui améliore légèrement le temps de démarrage de l'application.

Ce **Dockerfile** constitue une solution efficace pour déployer l'application **HouseRental** dans un conteneur Docker. Il facilite l'exécution de l'application sur différents environnements sans nécessiter d'installation locale de Java ou de Spring Boot, améliorant ainsi la portabilité et la gestion du déploiement.

## Construction de l'image Docker

Tout d'abord, nous construisons l'image Docker à partir du **Dockerfile** en utilisant la commande suivante :

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ docker build -t houserental .
```

Cette commande crée une image nommée **houserental**, qui contient l'application et son environnement d'exécution.

Ensuite, nous vérifions que l'image a bien été créée en listant les images Docker disponibles :

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ docker images
REPOSITORY          TAG         IMAGE ID      CREATED        SIZE
houserental         latest     aa6e30776685  5 minutes ago  525MB
```

Cela affiche toutes les images Docker stockées localement, y compris **houserental**.

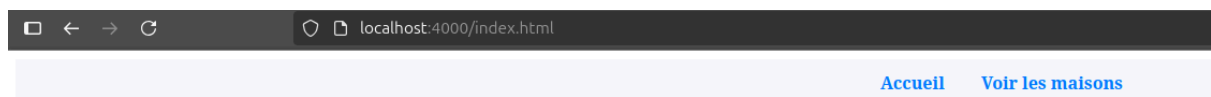
Une fois l'image prête, nous lançons un conteneur basé sur cette image avec la commande :

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ docker run -p 4000:8080 -t houserental
```

Ici, nous **associons le port 8080 du conteneur** (où l'application s'exécute) au **port 4000 de la machine hôte**, permettant d'accéder au service via `http://localhost:4000`.

Enfin, pour tester si le service fonctionne correctement, il suffit d'ouvrir un navigateur et de visiter l'URL :

Si tout est correctement configuré, l'application web **HouseRental** s'affiche à l'écran.



## Bienvenue sur notre service de location de maisons !

Explorez nos maisons disponibles à la location.

[Voir les maisons](#)

## Publier l'image Docker sur DockerHub

Avant de publier notre image sur Docker Hub, nous lui ajoutons un **tag** pour l'identifier clairement en utilisant la commande :

```
alex@alex-ThinkPad-T14-Gen-1:~$ docker tag 802b4460960e alexballe/houserental:1
```

Ensuite, nous envoyons l'image vers Docker Hub avec la commande :

```
alex@alex-ThinkPad-T14-Gen-1:~$ docker push alexballe/houserental:1
The push refers to repository [docker.io/alexballe/houserental]
66cfee172c21: Pushed
```

Cela permet de rendre l'image accessible en ligne pour être utilisée sur d'autres machines ou serveurs.

## Créer un déploiement Kubernetes

Pour déployer une application et exposer un service Kubernetes, on commence par créer un fichier YAML contenant la configuration nécessaire.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: houserental
spec:
  replicas: 1
  selector:
    matchLabels:
      app: houserental
  template:
    metadata:
      labels:
        app: houserental
    spec:
      containers:
        - image: alexballe/houserental:1
          imagePullPolicy: IfNotPresent
          name: houserental
          restartPolicy: Always
```

La commande **minikube start** sert à démarrer un cluster Kubernetes local sur votre machine.

Ensuite, on exécute la commande :

- pour appliquer cette configuration et déployer l'application.

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl apply -f deployment.yml
deployment.apps/houserental created
```

- **kubectl get pods** pour vérifier l'état des pods et s'assurer qu'ils sont bien en cours d'exécution.

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
houserental-848fddf94f-q9k89        1/1     Running   0           108s
```

- Ensuite on crée le service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: houserental
spec:
  ports:
    - nodePort: 31280
      port: 8080
      protocol: TCP
      targetPort: 8080
  selector:
    app: houserental
  type: NodePort
```

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl apply -f service.yml
service/houserental created
```

- On obtient l'url du service avec :

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ minikube service houserental --url
http://192.168.49.2:31280
```



## Ajouter une gateway en local

Pour attribuer un nom de domaine personnalisé à mon service, j'ai modifié le fichier **/etc/hosts** sur ma machine. J'y ai ajouté l'adresse IP récupérée via `kubectl get ingress`, suivie du nom de domaine souhaité, par exemple `houserental.info`. Cette modification permet d'accéder au service via ce nom dans un navigateur. Ainsi, la requête est correctement redirigée vers l'Ingress de Kubernetes.

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ minikube addons enable ingress
```

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: houserental-ingress
  annotations:
#    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
    - host: houserental.info
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: houserental
                port:
                  number: 8080

```

```



alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl apply -f ingress.yml
ingress.networking.k8s.io/example-ingress created

```

```

alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl get ingress
NAME                CLASS    HOSTS                ADDRESS          PORTS    AGE
houserental-ingress  nginx    houserental.info     192.168.49.2    80      16s

```


 Not Secure houserental.info/index.html

[Accueil](#)
[Voir les maisons](#)

## Bienvenue sur notre service de location de maisons !

Explorez nos maisons disponibles à la location.

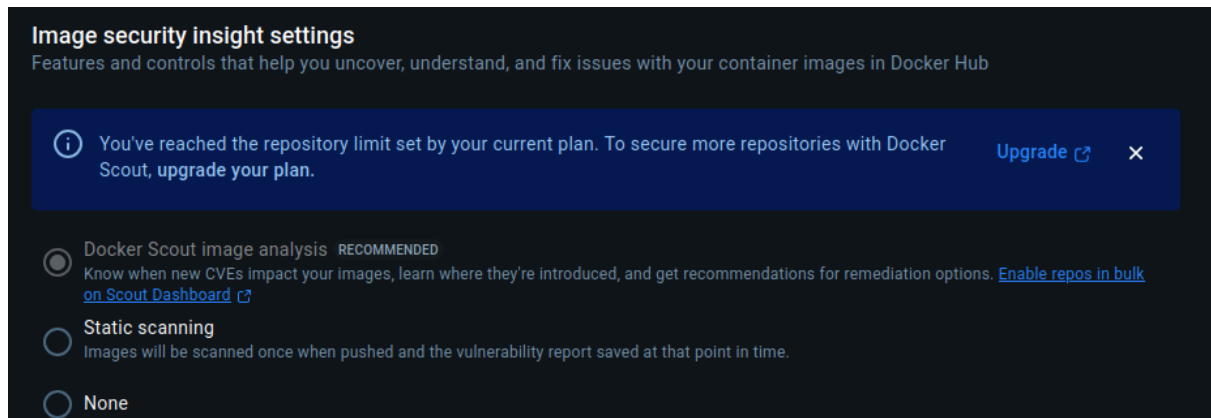
[Voir les maisons](#)



# Sécuriser le cluster

On sécurise l'image Docker.

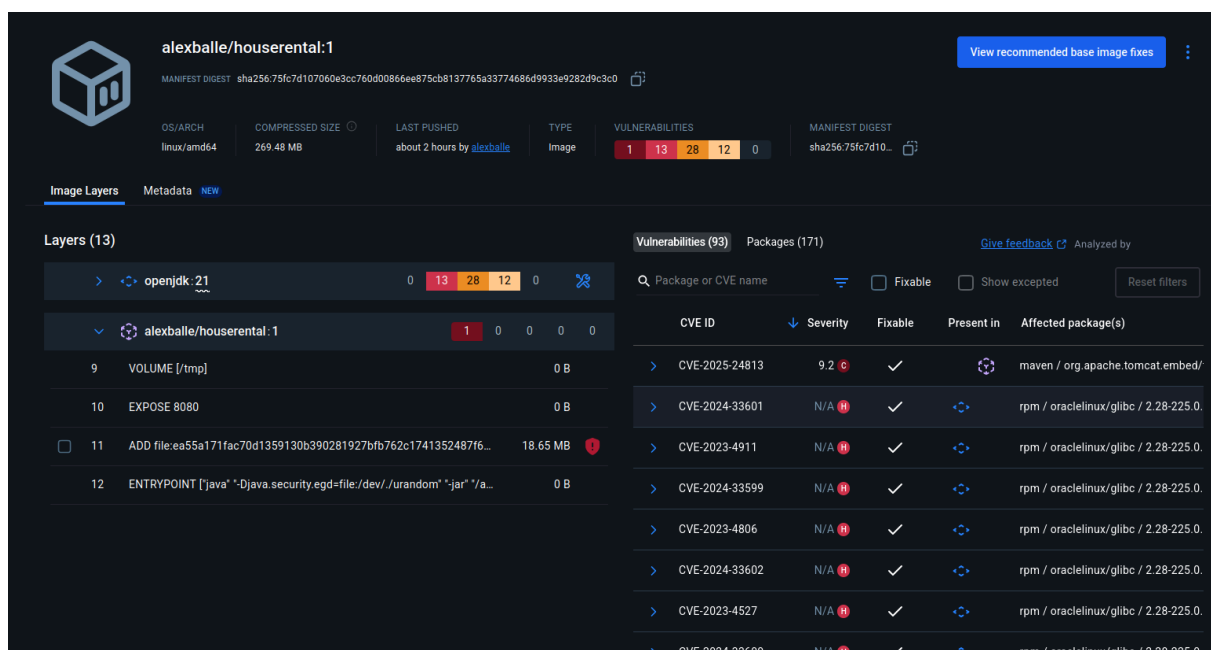
Dockerhub security



Ensuite, je me redirige vers:

<https://hub.docker.com/repository/docker/alexballe/houserental/tags/1/sha256-75fc7d107060e3cc760d00866ee875cb8137765a33774686d9933e9282d9c3c0>

pour voir le rapport associé.



Grace à ce rapport nous pouvons améliorer la sécurité de notre infrastructure en upgradant nos paquets.

# HTTPS

## génération d'un certificat TLS

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ openssl req -x509 -newkey rsa:4096 -keyout tls.key -out tls.crt -days 365 -nodes -subj "/CN=houserental.info"
```

## génération d'un secret kubernetes

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl create secret tls houseentalsec --cert=tls.crt --key=tls.key
secret/housetentalsec created
```

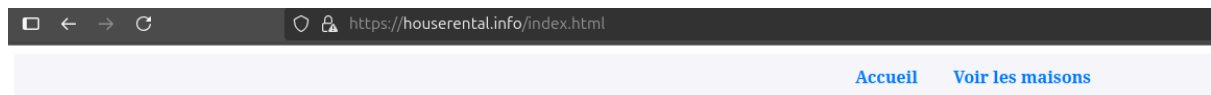
## httpsingress.yml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: houseentalsec-https-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  tls:
  - hosts:
    - houseentalsec.info
    secretName: houseentalsec # Matches the secret name
  rules:
  - host: houseentalsec.info
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: houseentalsec
            port:
              number: 8080
```

On supprime l'ancien ingress.

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl get ingress
NAME                CLASS  HOSTS                ADDRESS        PORTS    AGE
houseentalsec-ingress  nginx  houseentalsec.info  192.168.49.2   80       54m
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl delete ingress houseentalsec-ingress
ingress.networking.k8s.io "houseentalsec-ingress" deleted
```

```
alex@alex-ThinkPad-T14-Gen-1:~/ProjetWebM1$ kubectl apply -f httpsingress.yml
ingress.networking.k8s.io/housetentalsec-https-ingress created
```

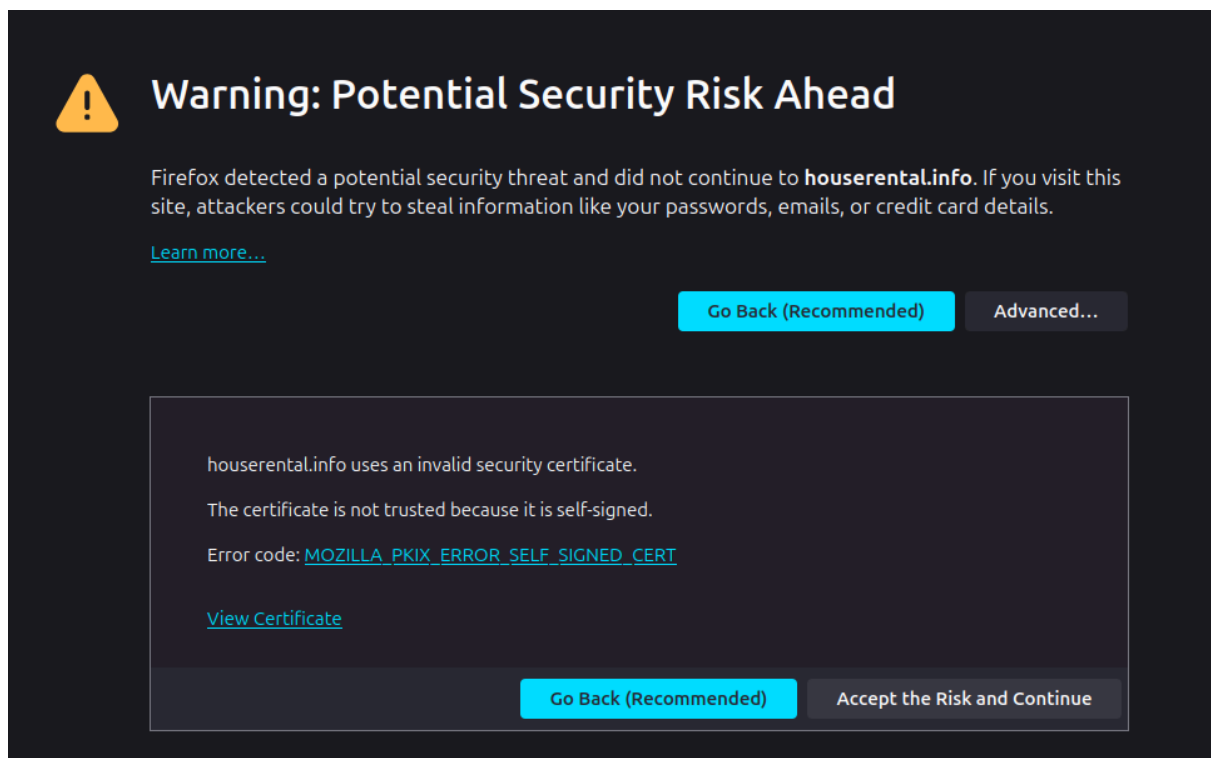


## Bienvenue sur notre service de location de maisons !

Explorez nos maisons disponibles à la location.

[Voir les maisons](#)

Mais puisque le certificat n'est pas authentifié par une CA le message suivant apparaît.



# Labs Certification

Google Cloud Skills Boost

[← Back to Dashboard](#)

## Progress

Track your learning activities, pick up where you left off, and celebrate what you've completed

Cours

Atelier

Quiz

Jeu

Learning path

Classroom

En cours

Terminée

Activité	Type	Date de début	Date de fin	Score	R
<a href="#">Infrastructure as Code avec Terraform</a>	Atelier	16 déc. 2024	16 déc. 2024	Assessment: 100%	✓
<a href="#">A Tour of Google Cloud Hands-on Labs</a>	Atelier	7 oct. 2024	7 oct. 2024	Assessment: 100%	✓

Alex Jin

Date d'abonnement : 2024

0 crédit

Buy credits or subscription

Dashboard

Activités

Paramètres

Se déconnecter

Vie privée

Conditions d'utilisation

Google Cloud

Tableau de bord

Explorer

Parcours

Abonnements

Google Cloud Skills Boost

[← Back to Dashboard](#)

## Progress

Track your learning activities, pick up where you left off, and celebrate what you've completed

Cours

Atelier

Quiz

Jeu

Learning path

Classroom

En cours

Terminée

Activité	Type	Date de début	Date de fin	Score	R
<a href="#">Infrastructure as Code avec Terraform</a>	Atelier	16 déc. 2024	16 déc. 2024	Assessment: 100%	✓
<a href="#">A Tour of Google Cloud Hands-on Labs</a>	Atelier	7 oct. 2024	7 oct. 2024	Assessment: 100%	✓

Avi Gilardi

Date d'abonnement : 2024

0 crédit

Buy credits or subscription

Dashboard

Activités

Paramètres

Se déconnecter

Vie privée

Conditions d'utilisation