Architecture Kubernetes

Projet : ESP Etudiants

BENCHEKROUN Soufiane
LKOUEN Salah Eddine

2025-2026

Sommaire

Étape 1 – Manipulations Kubernetes avec Minikube

Objectif

L'objectif de cette première étape est de prendre en main l'environnement Kubernetes via **Minikube** et **kubectl**, de vérifier son bon fonctionnement et de manipuler les ressources de base (pods, services, images).

Démarrage de Minikube

Démarrer Minikube

Gestion de Minikube

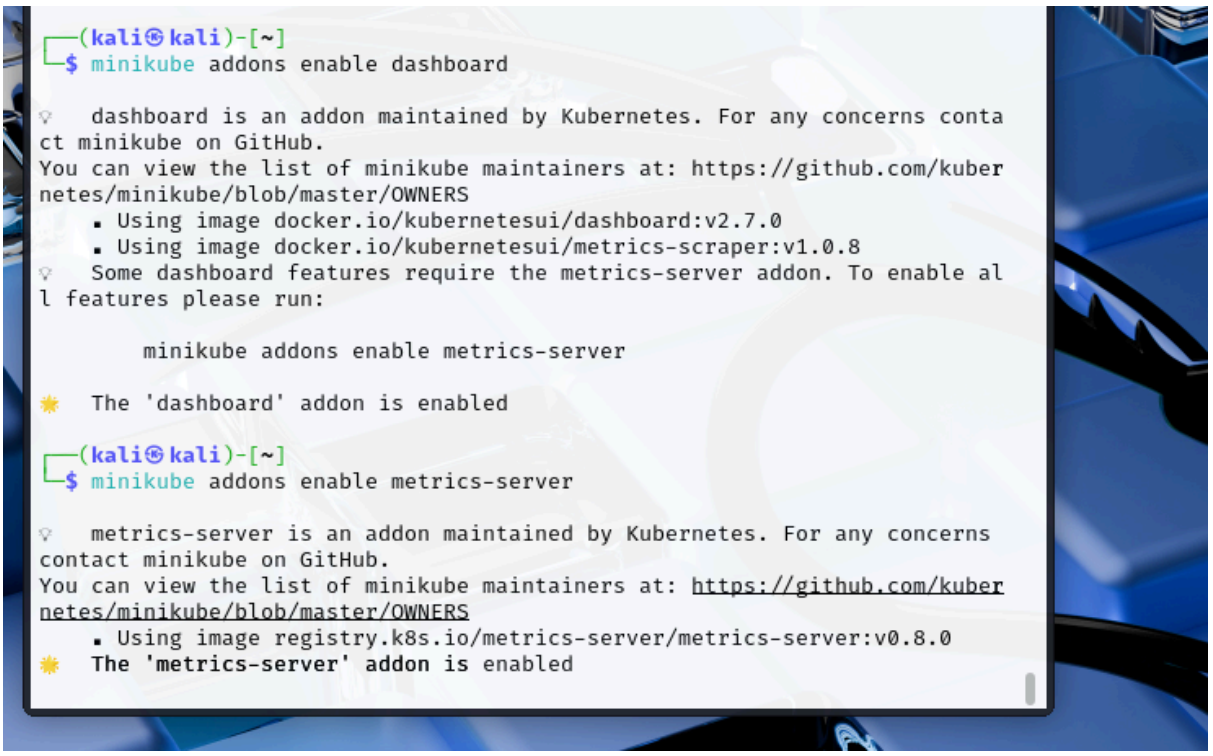
➢ Vérifier que Minikube pointe correctement vers le moteur Docker

```
┌──(kali㉿kali)-[~]
└─$ minikube status
minikube
type: Control Plane
host: InsufficientStorage
kubelet: Stopped
apiserver: Stopped
kubeconfig: Configured
docker-env: in-use
```

➢ Quels sont les addons actuellement installés ?

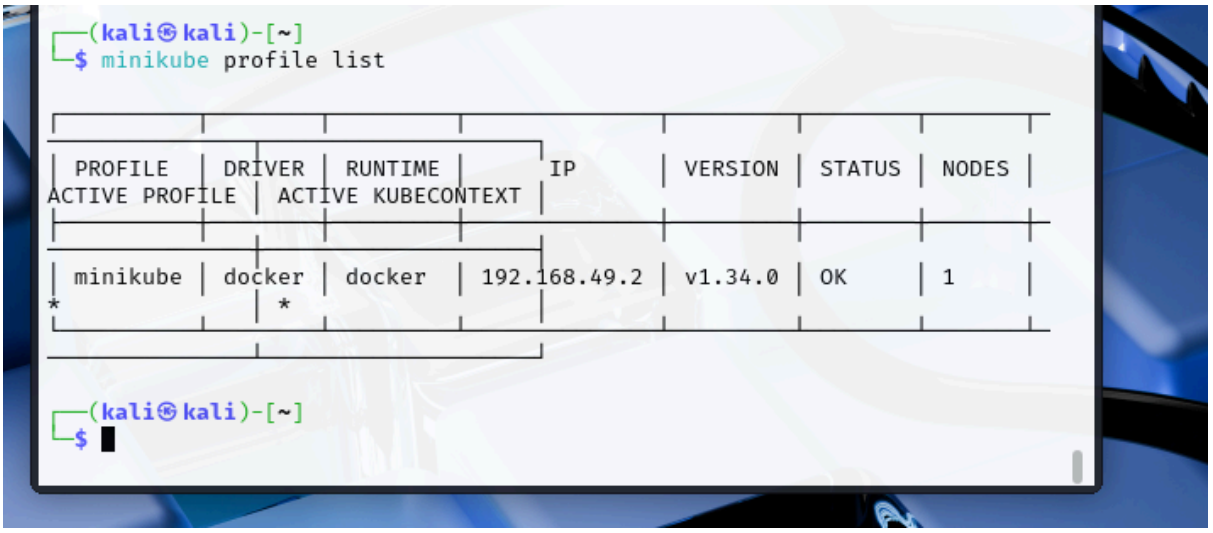
```
└─$ minikube addons list
|-----------------------------|----------|--------------|--------------------------------|
|         ADDON NAME          | PROFILE  |    STATUS    |              MAINTAINER        |
|-----------------------------|----------|--------------|--------------------------------|
| ambassador                  | minikube | disabled     | 3rd party (Ambassador)         |
| amd-gpu-device-plugin       | minikube | disabled     | 3rd party (AMD)                |
| auto-pause                  | minikube | disabled     | minikube                       |
| cloud-spanner               | minikube | disabled     | Google                         |
| csi-hostpath-driver         | minikube | disabled     | Kubernetes                     |
| dashboard                   | minikube | disabled     | Kubernetes                     |
| default-storageclass        | minikube | enabled ✓    | Kubernetes                     |
| efk                         | minikube | disabled     | 3rd party (Elastic)            |
| freshpod                    | minikube | disabled     | Google                         |
| gcp-auth                    | minikube | disabled     | Google                         |
```

➢ Installer un addon intéressant et expliquer pourquoi



Le dashboard permet de visualiser graphiquement les pods, services, deployments et logs.

➢ Lister les profils Minikube avec leurs caractéristiques



➢ Quels sont les profils en cours ?

```
┌──(kali㉿kali)-[~]
└─$ minikube profile
minikube
```

➢ Créer un nouveau profil et expliquer ce qu’est un profil

```
┌──(kali㉿kali)-[~]
└─$ minikube start -p test-profile

😄  [test-profile] minikube v1.37.0 on Debian kali-rolling (vbox/amd64)
    ▪ MINIKUBE_ACTIVE_DOCKERD=minikube
✨  Using the docker driver based on user configuration

❗  The requested memory allocation of 3072MiB does not leave room for sys
tem overhead (total system memory: 3385MiB). You may face stability issues
.
💡  Suggestion: Start minikube with less memory allocated: 'minikube start
 --memory=3072mb'

📌  Using Docker driver with root privileges
👍  Starting "test-profile" primary control-plane node in "test-profile" c
luster
🚜  Pulling base image v0.0.48 ...
🔥  Creating docker container (CPUs=2, Memory=3072MB) ...
```

Un profil permet d'avoir **plusieurs clusters Minikube indépendants** sur une même machine.

➢ Afficher le statut de Minikube

```
┌──(kali㉿kali)-[~]
└─$ minikube status

minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

➢ Accéder au Dashboard Minikube

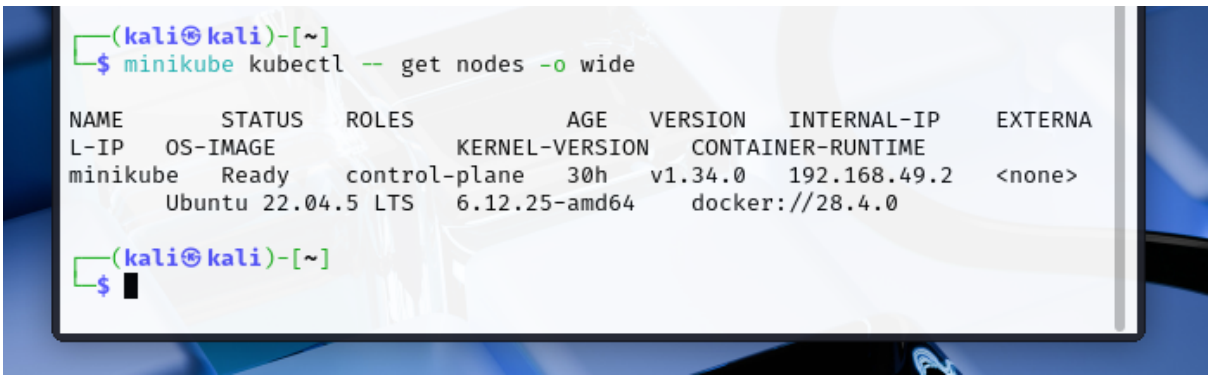
```
┌──(kali㉿kali)-[~]
└─$ minikube dashboard

🤔  Verifying dashboard health ...
🚀  Launching proxy ...
🤔  Verifying proxy health ...
🎉  Opening http://127.0.0.1:38875/api/v1/namespaces/kubernetes-dashboard/
services/http:kubernetes-dashboard:/proxy/ in your default browser...
```

➢ Qu'est-ce que le Dashboard Minikube ?

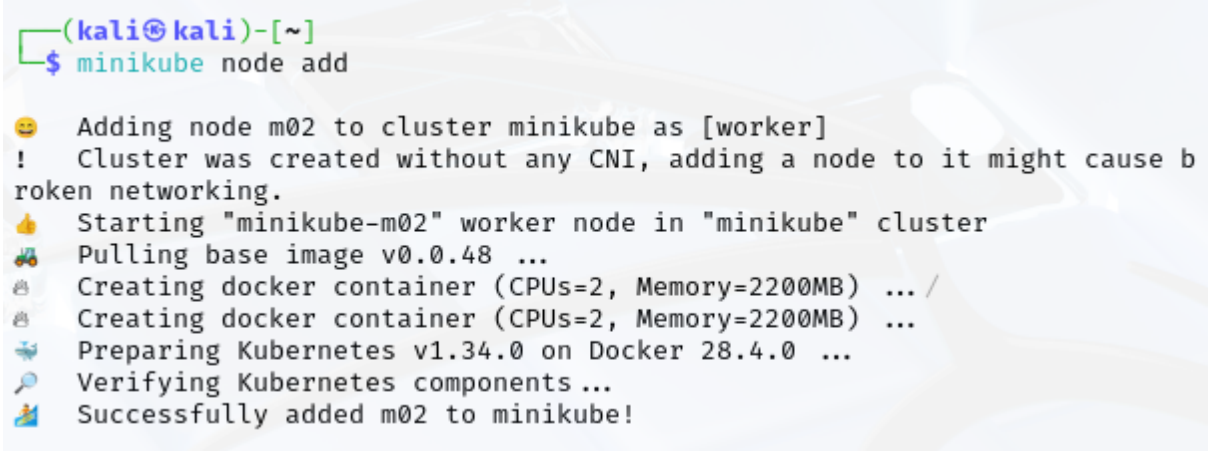
Le dashboard est une interface web Kubernetes permettant :

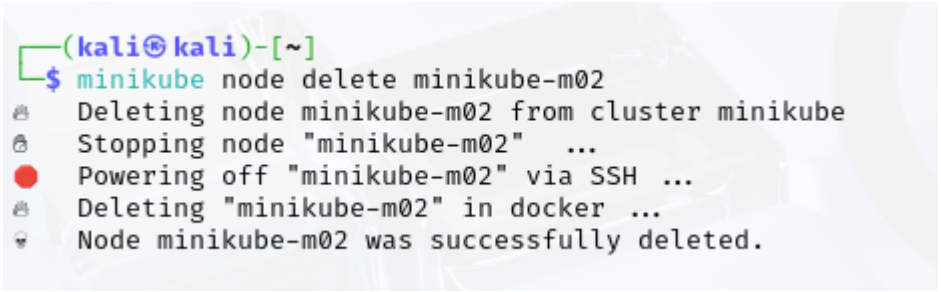
de visualiser les pods, services, deployments, consulter les logs et de surveiller l'état du cluster

➢ Lister les nœuds d'un profil

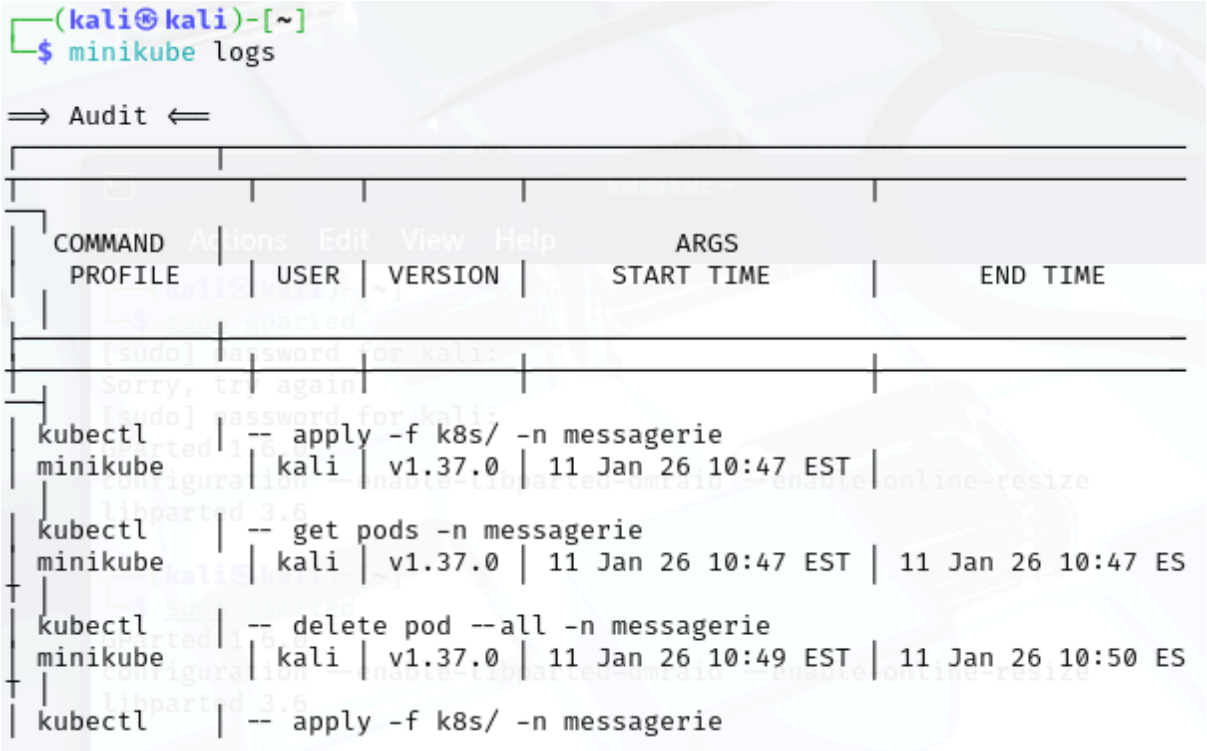


➢ **Ajouter puis supprimer un nœud**





➢ **Consulter les logs de Minikube**



Gestion des Pods et Services Kubernetes

➢ Lister les images/pods en cours d'exécution

```
File  Actions  Edit  View  Help
┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- get pods -A -o wide

NAMESPACE              NAME                                       READY
  STATUS    RESTARTS         AGE   IP             NODE       NOMINATED NODE
   READINESS GATES
kube-system            coredns-66bc5c9577-jfkss                   1/1
  Running   1 (9m56s ago)   30h   10.244.0.9     minikube   <none>
   <none>
kube-system            etcd-minikube                              1/1
  Running   1 (9m56s ago)   30h   192.168.49.2   minikube   <none>
   <none>
kube-system            kube-apiserver-minikube                    1/1
  Running   1 (9m56s ago)   30h   192.168.49.2   minikube   <none>
   <none>
kube-system            kube-controller-manager-minikube           1/1
  Running   1 (9m56s ago)   30h   192.168.49.2   minikube   <none>
   <none>
kube-system            kube-proxy-phk7k                           1/1
  Running   1 (9m56s ago)   30h   192.168.49.2   minikube   <none>
   <none>
kube-system            kube-scheduler-minikube                    1/1
  Running   1 (9m56s ago)   30h   192.168.49.2   minikube   <none>
   <none>
kube-system            metrics-server-85b7d694d7-2pb8s             1/1
  Running   2 (9m6s ago)    30h   10.244.0.12    minikube   <none>
   <none>
```

➢ Lancer une image nginx en mode impératif

```
┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- run nginx-pod --image=nginx:latest --port=80

pod/nginx-pod created

┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- get pod nginx-pod -o wide

NAME        READY   STATUS              RESTARTS   AGE   IP       NODE
   NOMINATED NODE   READINESS GATES
nginx-pod   0/1     ContainerCreating   0          5s    <none>   minikube
   <none>           <none>
```

➢ Créer un service pour nginx (mode impératif)

```
┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- expose pod nginx-pod --type=NodePort --port=80

service/nginx-pod exposed
```

➢ **Visualiser les informations du pod et du service**

```
┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- get pod nginx-pod -o wide

NAME        READY   STATUS              RESTARTS   AGE   IP       NODE
   NOMINATED NODE   READINESS GATES
nginx-pod   0/1     ContainerCreating   0          5s    <none>   minikube
   <none>           <none>

┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- expose pod nginx-pod --type=NodePort --port=80

service/nginx-pod exposed
```

➢ **Obtenir l'URL du service**

```
┌──(kali㉿kali)-[~]
└─$ minikube service nginx-pod --url

http://192.168.49.2:31754
```

➢ **Accéder au service via un navigateur**

➢ **Lancer une commande bash dans le conteneur nginx**

```
┌──(kali㉿kali)-[~]
└─$ minikube kubectl -- exec -it nginx-pod -- /bin/sh

# ls
bin   docker-entrypoint.d   home   media  proc  sbin  tmp
boot  docker-entrypoint.sh  lib    mnt    root  srv   usr
dev   etc                   lib64  opt    run   sys   var
# exit
```

Étape 2 – Présentation d'un projet d'architecture Event Driven Architecture (EDA)

1. Objectif de l'étape

L'objectif de cette étape est de concevoir, déployer et valider une architecture de type Event Driven Architecture (EDA) sous Kubernetes (Minikube).

Cette architecture permet :

- L'ajout et la gestion d'étudiants via une API REST
- La communication asynchrone entre services via Apache Kafka
- L'intégration des données dans une base PostgreSQL
- L'exposition d'une interface web pour l'utilisateur final

2. Description globale de l'architecture

L'architecture repose sur les composants suivants :

Composants techniques

- **Frontend Web (Nginx)** : interface utilisateur
- **Web Backend (API REST)** : reçoit les requêtes HTTP du frontend
- **Backend Producer Kafka** : publie des événements Kafka
- **Service d'intégration (Consumer Kafka)** : consomme les événements et écrit en base
- **Apache Kafka** : bus d'événements (Message Oriented Middleware)
- **Zookeeper** : coordination Kafka
- **PostgreSQL** : base de données relationnelle
- **Kubernetes (Minikube)** : orchestration des services

Principe EDA

1. L'utilisateur ajoute un étudiant via l'interface web
2. Le Web Backend expose un endpoint REST
3. Le Backend publie un événement dans Kafka
4. Le service d'intégration consomme l'événement
5. Les données sont persistées dans PostgreSQL

3. Préparation de l'environnement Kubernetes

Démarrage du cluster Minikube

```
┌──(kali㉿kali)-[~]
└─$ minikube start
😄  minikube v1.37.0 on Debian kali-rolling (vbox/amd64)
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.48 ...
🔄  Restarting existing docker container for "minikube" ...
🧯  Docker is nearly out of disk space, which may cause deployments to fai
l! (93% of capacity). You can pass '--force' to skip this check.
💡  Suggestion:

    Try one or more of the following to free up space on the device:

    1. Run "docker system prune" to remove unused Docker data (optionally
with "-a")
    2. Increase the storage allocated to Docker for Desktop by clicking on
:
    Docker icon > Preferences > Resources > Disk Image Size
    3. Run "minikube ssh -- docker system prune" if using the Docker conta
iner runtime
🍿  Related issue: https://github.com/kubernetes/minikube/issues/9024

```

```
┌──(kali㉿kali)-[~]
└─$ kubectl cluster-info

Kubernetes control plane is running at https://192.168.49.2:8443
CoreDNS is running at https://192.168.49.2:8443/api/v1/namespaces/kube-sys
tem/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info
dump'.
```

Connexion Docker ↔ Minikube

```
┌──(kali㉿kali)-[~]
└─$ eval "$(minikube -p minikube docker-env)
dquote>
dquote> "
```

Cette commande permet de construire des images Docker directement utilisables par Kubernetes sans passer par un registry externe.

4. Construction des images Docker

Chaque composant applicatif dispose de son Dockerfile.

```
┌──(kali㉿kali)-[~]
└─$ docker build -t backend:1.0 ./backend
docker build -t integration:1.0 ./integration
docker build -t web-backend:1.0 ./web-backend
docker build -t frontend:1.0 ./frontend
[+] Building 4.8s (9/10)                                        docker:default
 ⇒ [internal] load build definition from Dockerfile                      0.4s
 ⇒ ⇒ transferring dockerfile: 212B                                       0.1s
 ⇒ [internal] load metadata for docker.io/library/python:3.11-slim       2.4s
 ⇒ [internal] load .dockerignore                                         0.1s
 ⇒ ⇒ transferring context: 2B                                            0.0s
 ⇒ [1/5] FROM docker.io/library/python:3.11-slim@sha256:c24e9effa2       0.2s
 ⇒ ⇒ resolve docker.io/library/python:3.11-slim@sha256:c24e9effa2        0.1s
 ⇒ [internal] load build context                                         0.2s
 ⇒ ⇒ transferring context: 27.14kB                                       0.1s
 ⇒ CACHED [2/5] WORKDIR /app                                             0.0s
 ⇒ CACHED [3/5] COPY requirements.txt .                                  0.0s
 ⇒ CACHED [4/5] RUN pip install --no-cache-dir -r requirements.txt       0.0s
 ⇒ CACHED [5/5] COPY . .                                                 0.0s
 ⇒ exporting to image                                                    0.6s
 ⇒ ⇒ exporting layers                                                    0.0s
 ⇒ ⇒ writing image sha256:1564aed569ba8a8908c7e679a2b0ce736e0d874        0.6s
```

Les images sont construites localement et stockées dans le daemon Docker de Minikube.

5. Déploiement Kubernetes

Création du namespace et déploiement des services :

```
┌──(kali㉿kali)-[~/projet-messagerie-v2]
└─$ kubectl apply -f k8s/etudiants-namespace.yaml

kubectl apply -f k8s/etudiants-postgres.yaml
kubectl apply -f k8s/etudiants-zookeeper.yaml
kubectl apply -f k8s/etudiants-kafka.yaml

kubectl apply -f k8s/etudiants-backend.yaml
kubectl apply -f k8s/etudiants-integration.yaml
kubectl apply -f k8s/etudiants-web-backend.yaml
kubectl apply -f k8s/etudiants-frontend.yaml
namespace/etudiants unchanged
configmap/etudiants-postgres-init unchanged
deployment.apps/postgres unchanged
service/postgres unchanged
deployment.apps/zookeeper unchanged
service/zookeeper unchanged
deployment.apps/kafka unchanged
service/kafka unchanged
deployment.apps/backend unchanged
service/backend unchanged
deployment.apps/integration unchanged
deployment.apps/web-backend unchanged
service/web-backend unchanged
deployment.apps/frontend unchanged
```

6. Vérification du déploiement

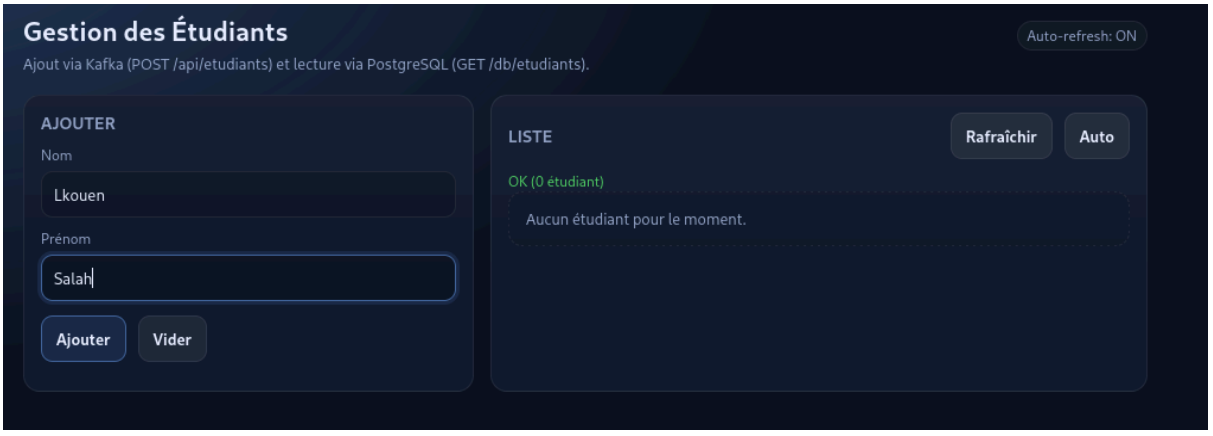
État des services

```
┌──(kali㉿kali)-[~/projet-messagerie-v2]
└─$ kubectl -n etudiants get svc -o wide

NAME          TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AG
E     SELECTOR
backend       ClusterIP   10.101.60.90     <none>        8080/TCP         3d
13h   app=backend
frontend      NodePort    10.103.131.146   <none>        80:30080/TCP     3d
13h   app=frontend
kafka         ClusterIP   10.108.186.28    <none>        9092/TCP         21
h     app=kafka
postgres      ClusterIP   10.100.157.194   <none>        5432/TCP         26
h     app=postgres
web-backend   ClusterIP   10.103.243.254   <none>        5000/TCP         21
h     app=web-backend
zookeeper     ClusterIP   10.109.119.230   <none>        2181/TCP         3d
13h   app=zookeeper
```

7. Accès à l’application Web

Récupération de l’IP Minikube

```
┌──(kali㉿kali)-[~/projet-messagerie-v2]
└─$ MINIKUBE_IP=$(minikube ip)
echo "http://${MINIKUBE_IP}:30080"
http://192.168.49.2:30080
```



8. Problèmes rencontrés

Connexion à la base PostgreSQL

- Difficultés de connexion initiales entre le service d'intégration et PostgreSQL
- Problèmes liés :
    - aux variables d'environnement
    - au nom du service PostgreSQL
    - à l'ordre de démarrage des pods

Ces problèmes ont nécessité plusieurs ajustements dans les fichiers YAML et une analyse des logs Kubernetes.

Gestion des images Docker

- Des erreurs ImagePullBackOff ont été rencontrées
- Résolues en :
    - utilisant eval $(minikube docker-env)
    - forçant imagePullPolicy: IfNotPresent

9. Axes d'amélioration

- Ajouter des readiness & liveness probes
- Externaliser la configuration via ConfigMaps
- Sécuriser Kafka (authentification, ACL)
- Ajouter une gestion des erreurs côté frontend
- Mettre en place un système de logs centralisé
- Ajouter des tests automatisés

10. Conclusion

Ce projet a permis de :

- Mettre en œuvre une architecture Event Driven Architecture
- Déployer une solution complète sous Kubernetes
- Comprendre les problématiques réelles de déploiement distribué
- Manipuler Kafka, PostgreSQL et Kubernetes ensemble

Malgré des difficultés techniques, la solution est fonctionnelle et respecte les principes fondamentaux d'une architecture EDA moderne et scalable.