Kubernetes Project

1. Presentation

Application web multi-tiers deployee sur Kubernetes (Minikube) : un frontend Nginx, un backend Spring Boot avec JPA, et une base PostgreSQL. Le routing est gere par un Ingress Nginx (/ -> frontend, /api -> backend). PostgreSQL est accessible uniquement en interne (ClusterIP).

2. Travail realise

PostgreSQL dans le cluster

- Secret Kubernetes pour les credentials (base64)
- PersistentVolumeClaim 1Gi + Deployment postgres:16-alpine + Service ClusterIP

Backend Spring Boot + JPA

- Dependances : spring-boot-starter-data-jpa, postgresql driver
- Entite JPA Message (id, content, createdAt) + MessageRepository
- API REST : GET /api (liste messages), POST /api (creation)
- Dockerfile securise avec utilisateur non-root (appuser)

Consolidation des manifests

- 1 fichier par composant : backend.yml, frontend.yml (Deployment + Service)
- Suppression des anciens fichiers separes

Securisation du cluster

- ServiceAccount dedie (backend-sa, automountServiceAccountToken: false)
- NetworkPolicy default-deny + regles explicites backend et postgres
- SecurityContext : runAsNonRoot, drop ALL capabilities
- Resources requests/limits sur chaque container

3. Fichiers du projet

```
postgres-secret.yml          Secret credentials PostgreSQL
postgres.yml                 PVC + Deployment + Service PostgreSQL
backend-serviceaccount.yml   ServiceAccount backend
backend.yml                  Deployment + Service backend
frontend.yml                 Deployment + Service frontend
ingress.yml                  Ingress Nginx (routing)
networkpolicy-*.yml          NetworkPolicies (deny + allow)
backend/                     Code Spring Boot (JPA, REST, Dockerfile)
frontend/                    Nginx + index.html
```

4. Deploiement

```
minikube start && minikube addons enable ingress
eval $(minikube docker-env)
docker build -t sneakyrat/backend:v3 ./backend
kubectl apply -f postgres-secret.yml -f postgres.yml
kubectl apply -f backend-serviceaccount.yml -f backend.yml
kubectl apply -f frontend.yml -f ingress.yml
kubectl apply -f networkpolicy-default-deny.yml
kubectl apply -f networkpolicy-backend.yml -f networkpolicy-postgres.yml
```

5. Ce qui est attendu lors du test

A. Deploiement

- Tous les pods Running (kubectl get pods)
- Services frontend, backend, postgres presents
- Ingress route / et /api correctement

B. Test fonctionnel

- GET /api retourne la liste des messages (JSON)
- POST /api avec {"content": "Hello!"} cree un message
- Un second GET confirme la persistance
- Frontend accessible sur /

C. Persistance

- PVC status Bound (kubectl get pvc)
- Supprimer le pod postgres, attendre recreate, verifier que les donnees sont conservees

D. Securite

- Credentials dans un Secret (pas en clair)
- NetworkPolicies appliquees (kubectl get networkpolicies)
- Backend non-root (kubectl exec deploy/backend -- whoami)
- ServiceAccount backend-sa sans token monte
- Resources limits definis

E. Comprehension des concepts

- Role de chaque fichier YAML
- Difference ClusterIP vs NodePort vs LoadBalancer
- Pourquoi NetworkPolicies (default deny + allow explicite)
- Fonctionnement Ingress et routing
- Role du PVC pour la persistance

6. Commandes utiles pour la demo

```
# Tester l'API
curl -X POST http://<IP>/api -H "Content-Type: application/json" -d '{"content":"Hello!"}'
curl http://<IP>/api
```

```
# Verifier persistance
kubectl delete pod -l app=postgres
kubectl wait --for=condition=ready pod -l app=postgres --timeout=60s
curl http://<IP>/api
```

```
# Verifier securite
kubectl exec deploy/backend -- whoami
kubectl describe networkpolicy default-deny-ingress
kubectl get secret postgres-secret -o yaml
kubectl get pod -l app=backend -o jsonpath='{.items[0].spec.serviceAccountName}'
```