Université Paris Cité

23/01/2025

Gestion de Post-its - Architecture Microservices

Réalisé par :

- **Hoang Bao Ngoc Le**
- **Ousmane Doudou Ridouane**

Encadré par :

Benoît CHARROUX

1. Introduction

Ce projet vise à développer une application de gestion de post-its en utilisant une architecture microservices. Les technologies principales incluent **Docker** pour la conteneurisation, **Kubernetes** pour l'orchestration, et **Ingress** pour le routage réseau sécurisé avec HTTPS. Nous avons intégré une base de données relationnelle **MySQL** pour assurer la persistance des données.

Le projet illustre les compétences clés suivantes :

- Mise en œuvre d'architectures microservices.
- Gestion et orchestration des conteneurs avec Kubernetes.
- Sécurisation des communications via HTTPS et RBAC (Role-Based Access Control).
- Configuration et gestion d'une base de données relationnelle.

Le rapport explique les étapes de conception et de mise en œuvre, avec une attention particulière sur la configuration sécurisée, tout en fournissant des captures d'écran pour illustrer le travail réalisé.

2. Objectifs

- Développer une application distribuée composée de trois services : **Frontend**, **Gestion des utilisateurs**, et **Gestion des post-its**.
- Assurer la persistance des données avec une base MySQL.
- Conteneuriser les services pour garantir leur portabilité.
- Orchestrer ces services avec Kubernetes.
- Sécuriser l'application avec HTTPS et des règles d'accès RBAC.

3. Conception et Architecture

1. Architecture de l'application

L'application est composée de :

- **Front-Service** : Fournit une interface utilisateur intuitive pour l'inscription, la connexion, et la gestion des post-its.
- **User-Service** : Gère les opérations liées aux utilisateurs (inscription, connexion).
- **Postit-Service** : Permet la gestion des post-its (création, modification, suppression).
- **MySQL** : Base de données relationnelle utilisée pour stocker les informations des utilisateurs et des post-its.

Les services communiquent via des API REST. Kubernetes orchestre ces services et un **Ingress** gère le routage des requêtes HTTP/HTTPS.

4. Étapes du Projet

4.1. Gestion de la base de données avec MySQL

4.1.1. Conception de la base de données

Une base de données MySQL a été configurée avec trois tables principales :

- **utilisateur** : Contient les informations des utilisateurs (nom, email, mot de passe).
- **postit** : Gère les informations des post-its (titre, contenu, propriétaire).
- **partage** : Permet de partager des post-its entre utilisateurs.

4.1.2. Initialisation et persistance des données

Pour garantir la persistance des données, un fichier **init.sql** a été créé, contenant les instructions SQL pour initialiser la base de données. Les données sont stockées sur un volume persistant (PVC) configuré dans Kubernetes.

```
PS C:\xampp\htdocs\postit-projet> kubectl get pvc -n postit-projet
NAME        STATUS   VOLUME                                     CAPACITY   ACCESS MODES   STORAGECLASS   VOLUMEATTRIBUTESCLASS   AGE
mysql-pvc   Bound    pvc-ac7b4d39-6363-4b0d-bfc4-9e5cd72a33db   1Gi        RWO            standard       <unset>                 79m
```

4.2. Conteneurisation avec Docker

Chaque service a été conteneurisé avec Docker pour garantir sa modularité et faciliter son déploiement. Les images Docker ont été créées localement et publiées sur Docker Hub pour un accès simplifié.

Étapes clés :

1. Création des fichiers Dockerfile pour chaque microservice.
2. Construction des images Docker.
3. Validation des conteneurs localement avant déploiement.
4. Publication des images sur Docker Hub.

```
PS C:\xampp\htdocs\postit-projet> docker ps
CONTAINER ID   IMAGE                              COMMAND                  CREATED         STATUS         PORTS                    NAMES
a604bb7ad881   postit-projet-front-service         "docker-php-entrypoi…"   2 minutes ago   Up 2 minutes   0.0.0.0:8080->80/tcp     front-service
cd8d5f34cc7b   postit-projet-user-service          "docker-php-entrypoi…"   2 minutes ago   Up 2 minutes   0.0.0.0:8001->8001/tcp   user-service
9f021a2bc9c2   postit-projet-postit-service        "docker-php-entrypoi…"   2 minutes ago   Up 2 minutes   0.0.0.0:8002->8002/tcp   postit-service
de8261c34118   rousmane/postit-service:latest     "docker-php-entrypoi…"   5 minutes ago   Up 5 minutes   8002/tcp                 epic_murdock
abbbe7b01879   rousmane/user-service:latest       "docker-php-entrypoi…"   5 minutes ago   Up 5 minutes   8001/tcp                 vibrant_blackwell
c0cff3e6a8ac   rousmane/front-service:latest      "docker-php-entrypoi…"   6 minutes ago   Up 6 minutes   80/tcp                   lucid_clarke
```

```
PS C:\xampp\htdocs\postit-projet> docker images
REPOSITORY                          TAG       IMAGE ID       CREATED         SIZE
rousmane/postit-service             latest    4dfa1057d9c8   4 days ago      724MB
rousmane/user-service               latest    c4c4367bab4b   4 days ago      724MB
rousmane/front-service              latest    68d7193eb7e2   4 days ago      652MB
postit-projet-front-service         latest    b7a719165f2d   4 days ago      652MB
rousmane/postit-front               latest    b7a719165f2d   4 days ago      652MB
postit-projet-postit-service        latest    c80014b14f75   4 days ago      673MB
postit-projet-user-service          latest    36f9829edd69   4 days ago      673MB
rousmane/postit-user                latest    36f9829edd69   4 days ago      673MB
mysql                               8.0       d58ac93387f6   2 months ago    811MB
gcr.io/k8s-minikube/kicbase         v0.0.45   e7c9bc3bc515   4 months ago    1.81GB
gcr.io/k8s-minikube/kicbase         <none>    81df28859520   4 months ago    1.81GB
PS C:\xampp\htdocs\postit-projet>
```

4.3. Orchestration avec Kubernetes

4.3.1. Déploiement des ressources

Kubernetes a été utilisé pour orchestrer les conteneurs et garantir leur disponibilité et leur scalabilité.

Des fichiers YAML ont été créés pour orchestrer les services :

- **Deployments** : Gèrent les pods pour chaque service.
- **Services** : Exposent les services via des ports spécifiques.
- **ConfigMaps** : Centralisent les configurations partagées.
- **Ingress** : Gère le routage HTTP/HTTPS.

4.3.2. Processus suivi

- Déploiement des fichiers YAML
- Vérification des ressources déployées :

```
PS C:\xampp\htdocs\postit-projet> kubectl get all -n postit-projet
NAME                                    READY   STATUS    RESTARTS   AGE
pod/front-service-675987fb7-lbzj8       1/1     Running   0          97m
pod/mysql-7bcd8dbc6b-8xjn5              1/1     Running   0          97m
pod/postit-service-6f96d66c8d-xj42t     1/1     Running   0          97m
pod/user-service-654b48cb9f-dp8np       1/1     Running   0          97m

NAME                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/front-service     NodePort    10.98.158.188    <none>        80:30000/TCP   97m
service/mysql-service     ClusterIP   10.106.187.155   <none>        3306/TCP       97m
service/postit-service    ClusterIP   10.97.110.117    <none>        8002/TCP       97m
service/user-service      ClusterIP   10.111.172.202   <none>        8001/TCP       97m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/front-service       1/1     1            1           97m
deployment.apps/mysql               1/1     1            1           97m
deployment.apps/postit-service      1/1     1            1           97m
deployment.apps/user-service        1/1     1            1           97m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/front-service-675987fb7        1         1         1       97m
replicaset.apps/mysql-7bcd8dbc6b               1         1         1       97m
replicaset.apps/postit-service-6f96d66c8d      1         1         1       97m
replicaset.apps/user-service-654b48cb9f        1         1         1       97m
PS C:\xampp\htdocs\postit-projet>
```

4.4. Sécurisation avec HTTPS

La sécurisation de l'application a été réalisée via HTTPS, garantissant la confidentialité des données échangées. Cela a impliqué la génération d'un certificat SSL et sa configuration dans Kubernetes.

Étapes suivies :

- Génération d'un certificat SSL auto-signé avec OpenSSL.
- Création d'un secret Kubernetes pour stocker le certificat.
- Configuration de l'Ingress pour forcer la redirection HTTPS.

```
PS C:\xampp\htdocs\postit-projet> kubectl describe ingress postit-ingress -n postit-projet
Name:             postit-ingress
Labels:           <none>
Namespace:        postit-projet
Address:          192.168.49.2
Ingress Class:    nginx
Default backend:  <default>
TLS:
  postit-tls terminates postit.local
Rules:
  Host          Path  Backends
  ----          ----  --------
  postit.local
                /   front-service:80 (10.244.0.50:80)
Annotations:    nginx.ingress.kubernetes.io/force-ssl-redirect: true
                nginx.ingress.kubernetes.io/rewrite-target: /
                nginx.ingress.kubernetes.io/ssl-redirect: true
Events:         <none>
PS C:\xampp\htdocs\postit-projet>
```

4.5. Gestion des Permissions avec RBAC

Pour renforcer la sécurité, des règles RBAC ont été mises en place. Cela limite l'accès aux ressources Kubernetes à des utilisateurs spécifiques, garantissant un contrôle précis :

- **Role** : Définition des permissions spécifiques.
- **RoleBinding** : Liaison des rôles aux utilisateurs.

Étapes suivies :

1. Création d'un rôle avec des permissions spécifiques (lecture des pods, accès aux déploiements).
2. Liaison du rôle à un utilisateur ou un service via un RoleBinding.

```
PS C:\xampp\htdocs\postit-projet> kubectl apply -f kubernetes/role.yaml
role.rbac.authorization.k8s.io/developer-role created
PS C:\xampp\htdocs\postit-projet> kubectl apply -f kubernetes/rolebinding.yaml
rolebinding.rbac.authorization.k8s.io/developer-rolebinding created
```

3. Vérifications

```
PS C:\xampp\htdocs\postit-projet> kubectl get roles -n postit-projet
NAME             CREATED AT
developer-role   2025-01-09T16:26:22Z
PS C:\xampp\htdocs\postit-projet> kubectl get rolebindings -n postit-projet
NAME                    ROLE                  AGE
developer-rolebinding   Role/developer-role   12s
```

5. Mise en place du pipeline CI/CD

Pour automatiser les tests, la construction des images Docker et le déploiement sur Kubernetes, un pipeline CI/CD a été mis en place via **GitHub Actions**. Ce pipeline vise à garantir une intégration continue et un déploiement continu des modifications apportées au projet.

1. Objectifs du pipeline

- **Tester** automatiquement le code à chaque modification.
- **Construire** les images Docker pour les microservices (front-service, user-service, postit-service).
- **Pousser** les images Docker sur un registre Docker Hub personnel.
- **Déployer** les modifications sur un cluster Kubernetes local via Minikube.

2. Structure du pipeline

Le pipeline est divisé en trois étapes principales :

1. **Tests :**
    - Vérification du code source.
    - Installation des dépendances et exécution des tests unitaires.
2. **Construction des images Docker :**
    - Connexion au registre Docker Hub via des secrets configurés.
    - Construction et publication des images Docker mises à jour pour chaque service.
3. **Déploiement Kubernetes :**
    - Application des manifests Kubernetes pour synchroniser les déploiements avec les nouvelles images.

3. Configuration des secrets

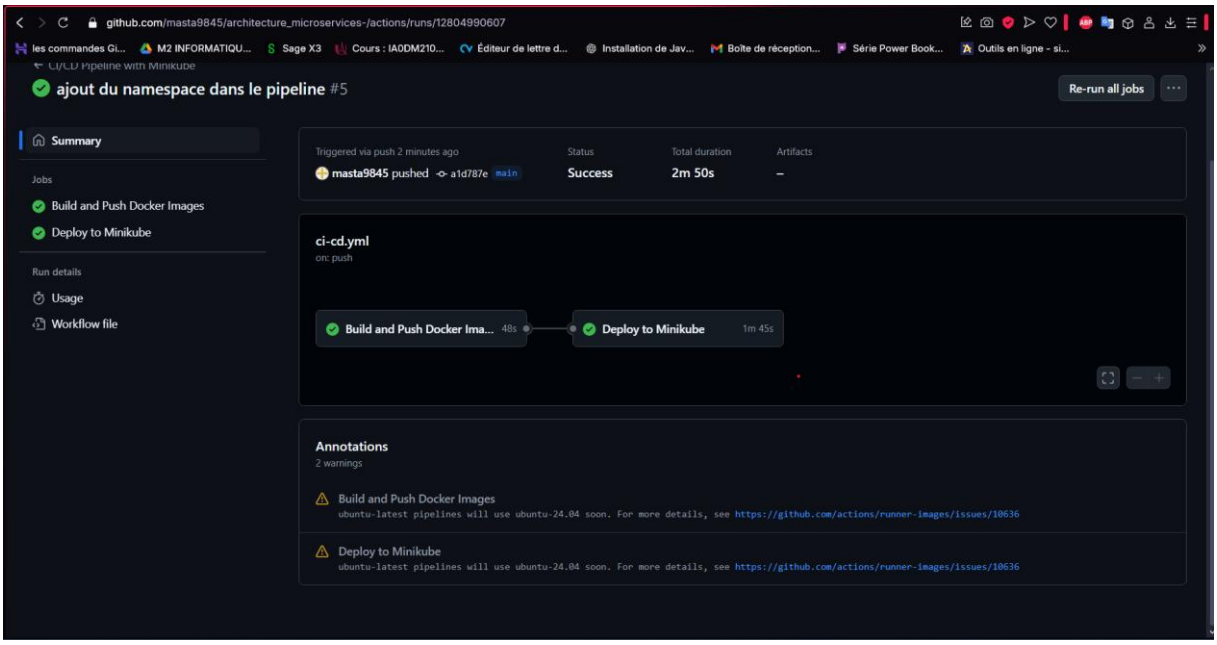
Deux secrets ont été configurés dans le dépôt GitHub pour permettre l'accès sécurisé au registre Docker Hub :

- **ROUSMANE** : Stocke le nom d'utilisateur Docker Hub.
- **DOCKER_PASSWORD** : Stocke le token personnel Docker Hub.

4. Exemple de workflow

Un fichier YAML (.github/workflows/ci-cd.yml) a été ajouté dans le dépôt GitHub pour définir le pipeline. Voici les étapes du workflow :

- **Tests automatisés** : Vérifie que le code est valide.
- **Construction des images Docker** : Crée et pousse les images Docker sur Docker Hub.
- **Déploiement sur Kubernetes** : Met à jour les déploiements Kubernetes avec les nouvelles images.



6. Résultats

L'application est fonctionnelle et répond aux objectifs fixés :

- Les utilisateurs peuvent s'inscrire, se connecter, créer, modifier et supprimer des post-its.
- L'accès à l'application se fait via HTTPS.
- Les données sont sauvegardées de manière persistante dans MySQL.
- Les permissions Kubernetes sont strictement contrôlées grâce à RBAC.

```
PS C:\xampp\htdocs\postit-projet> kubectl get all -n postit-projet
NAME                                    READY   STATUS    RESTARTS   AGE
pod/front-service-675987fb7-lbzj8       1/1     Running   0          108m
pod/mysql-7bcd8dbc6b-8xjn5              1/1     Running   0          108m
pod/postit-service-6f96d66c8d-xj42t     1/1     Running   0          108m
pod/user-service-654b48cb9f-dp8np       1/1     Running   0          108m

NAME                      TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/front-service     NodePort    10.98.158.188    <none>        80:30000/TCP   108m
service/mysql-service     ClusterIP   10.106.187.155   <none>        3306/TCP       108m
service/postit-service    ClusterIP   10.97.110.117    <none>        8002/TCP       108m
service/user-service      ClusterIP   10.111.172.202   <none>        8001/TCP       108m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/front-service       1/1     1            1           108m
deployment.apps/mysql               1/1     1            1           108m
deployment.apps/postit-service      1/1     1            1           108m
deployment.apps/user-service        1/1     1            1           108m

NAME                                           DESIRED   CURRENT   READY   AGE
replicaset.apps/front-service-675987fb7        1         1         1       108m
replicaset.apps/mysql-7bcd8dbc6b               1         1         1       108m
replicaset.apps/postit-service-6f96d66c8d      1         1         1       108m
replicaset.apps/user-service-654b48cb9f        1         1         1       108m
PS C:\xampp\htdocs\postit-projet>
```

7. Défis et Résolutions

1. Redirection **HTTPS non fonctionnelle** :
   - Résolution : Ajout d'annotations spécifiques dans le manifest de l'Ingress.
2. Problème **de connexion à MySQL** :
   - Résolution : Validation des variables d'environnement et redéploiement.
3. Mise **en place de RBAC** :
   - Résolution : Tests avec des permissions limitées pour identifier les besoins exacts.

8. Conclusion

Ce projet a permis de mettre en œuvre une application distribuée en suivant une architecture microservices robuste et scalable. Les concepts clés de conteneurisation, orchestration et sécurisation ont été appliqués, tout en assurant la persistance des données. L'intégration de HTTPS et RBAC a renforcé la sécurité, faisant de cette application un exemple pratique des meilleures pratiques en déploiement moderne.

Activité Labs