I. Rapport de petit projet

Sujet :

Déploiement d’une application Flask avec Docker et Kubernetes

Élaboré par :

Jibril Rachedi

Réalisé le 11/08/2024

II. Sommaire

7. Lier le Référentiel Local à GitHub
8. Pousser les Modifications vers GitHub
9. Mettre à Jour le Référentiel Local avec les Modifications Distantes
10. Cloner un Référentiel GitHub
11. Gérer les Branches
    - Créer une Nouvelle Branche
    - Basculer vers une Branche Existante
    - Fusionner une Branche avec la Branche Principale
12. Résolution des Conflits
13. Supprimer une Branche
14. Afficher l'Historique des Commits

IX. Conclusion

1. Bilan du Projet
2. Accomplissements
    - Conteneurisation
    - Déploiement Kubernetes
    - Services et Accessibilité
3. Leçons Apprises
    - Gestion des Conteneurs
    - Orchestration avec Kubernetes
    - Débogage et Résolution de Problèmes
4. Perspectives d'Avenir
    - Automatisation du Déploiement
    - Sécurité
    - Monitoring et Logs

III. Introduction aux Technologies

1. Docker :

Docker est une plateforme qui permet de créer, déployer et exécuter des applications dans des conteneurs. Les conteneurs sont des environnements isolés qui incluent tout ce dont une application a besoin pour fonctionner, ce qui permet une portabilité et une cohérence accrues à travers différents environnements.

2. Kubernetes :

Kubernetes est un système d'orchestration de conteneurs open-source qui facilite le déploiement, la gestion et la mise à l'échelle des applications conteneurisées. Il automatise la gestion des conteneurs dans un cluster, assurant leur haute disponibilité et leur scalabilité.

3. Minikube :

Minikube est un outil qui permet de créer un cluster Kubernetes local sur une machine unique. Il est particulièrement utile pour le développement et les tests avant de déployer sur des environnements de production plus larges.

4. Kubectl :

`kubectl` est l'outil en ligne de commande utilisé pour interagir avec les clusters Kubernetes. Il permet de déployer des applications, gérer les ressources du cluster et obtenir des informations sur l'état des différentes ressources.

IV. Description du Projet

Le projet met en place une application web Flask simple dans un environnement Kubernetes. Voici un aperçu des différents éléments du projet et leur fonction :

1. Contenu du Projet :

Dockerfile :

Fichier de configuration pour créer une image Docker de l'application Flask. Il spécifie les étapes nécessaires pour construire l'image, y compris l'installation des dépendances et la configuration du serveur web.

```
FROM python:3.9-slim
WORKDIR /app
COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "app.py"]
```

deployment.yaml :

Fichier de configuration Kubernetes pour déployer l'application Flask dans le cluster. Ce fichier définit le déploiement, les réplicas, les conteneurs et les ressources nécessaires.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: flask-app
spec:
  replicas: 1
  selector:
    matchLabels:
      app: flask-app
  template:
    metadata:
      labels:
        app: flask-app
    spec:
      containers:
      - name: flask-app
        image: localhost:30336/my-flask-app:latest
        ports:
        - containerPort: 8080
```

service.yaml :

Fichier de configuration Kubernetes pour exposer l'application Flask à l'extérieur du cluster. Ce fichier configure un service de type NodePort pour permettre l'accès à l'application via un port spécifique sur l'hôte.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: flask-app-service
spec:
  selector:
    app: flask-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
      nodePort: 30000
  type: NodePort
```

app.py :

Code source de l'application Flask. Ce fichier contient le code de l'application web, y compris les routes et la logique de traitement des requêtes.

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return "Hello World!"

if __name__ == '__main__':
    app.run(host='0.0.0.0', port=8080)
```

requirements.txt :

Fichier listant les dépendances Python nécessaires à l'application Flask. Il est utilisé pour installer les packages requis dans l'image Docker.

```
Flask==3.0.3
```

README.txt :

Documentation du projet, y compris des instructions pour la construction et le déploiement de l'application.

V. Étapes du Déploiement

1. Construire l'Image Docker :

Utiliser le `Dockerfile` pour créer une image Docker de l'application Flask.

```
docker build -t my-flask-app:latest .
```

2. Démarrer un Registre Docker Local (si nécessaire) :

Démarrer un registre Docker local pour stocker et accéder aux images Docker.

```
docker run -d -p 30336:5000 --name registry registry:2
```

3. Taguer et Pousser l'Image Docker vers le Registre Local :

Taguer l'image Docker et la pousser vers le registre local.

```
docker tag my-flask-app:latest localhost:30336/my-flask-app:latest
docker push localhost:30336/my-flask-app:latest
```

4. Déployer l'Application sur Kubernetes :

Appliquer les fichiers de configuration Kubernetes pour déployer l'application Flask et l'exposer via un service.

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

5. Accéder à l'Application :

Ouvrir un navigateur et accéder à https://localhost:30000 pour voir l'application Flask en fonctionnement.

VI. Détails des Fichiers de Configuration

1. **Dockerfile :**

Configure la construction de l'image Docker avec les dépendances nécessaires et expose le port 8080 pour l'application.

2. **deployment.yaml :**

Définit un déploiement Kubernetes avec une réplique de l'application Flask. Spécifie l'image Docker à utiliser et le port du conteneur.

3. **service.yaml :**

Crée un service NodePort pour exposer l'application sur le port 30000, permettant l'accès externe.

VII. Considérations et Dépannage

1) Problème de Pull d'Image :

Assurez-vous que l'image Docker est correctement poussée vers le registre et que le registre est en cours d'exécution.

2) Erreur de Déploiement :

Vérifiez les logs des pods avec `kubectl logs <pod-name>` pour identifier les erreurs éventuelles.

VIII. Gestion du Code Source avec Git et GitHub

1. Introduction à Git

Git est un système de gestion de versions décentralisé utilisé pour suivre les modifications apportées au code source tout au long du développement. Il permet à plusieurs développeurs de collaborer efficacement sur le même projet en conservant un historique complet des modifications.

2. Introduction à GitHub

GitHub est une plateforme d'hébergement pour les projets utilisant Git. Elle fournit des outils pour la gestion des versions, la collaboration, le suivi des problèmes, et plus encore. GitHub facilite le partage et la révision du code entre développeurs.

3. Configuration Initiale

Avant de commencer à utiliser Git, vous devez configurer votre nom d'utilisateur et votre adresse e-mail. Ces informations seront associées aux commits que vous faites.

```
git config --global user.name "Votre Nom"
git config --global user.email ""
```

4. Initialiser un Référentiel Git

Pour commencer à utiliser Git dans un projet, vous devez initialiser un référentiel Git dans le répertoire de votre projet.

```
git init
```

5. Ajouter des Fichiers au Référentiel

Ajoutez les fichiers que vous souhaitez suivre avec Git.

```
git add .
```

6. Commiter les Modifications

Les modifications ajoutées doivent être enregistrées avec un message de commit.

```
git commit -m "Description des modifications apportées"
```

7. Lier le Référentiel Local à GitHub

Créez un nouveau dépôt sur GitHub. Ensuite, liez votre dépôt local à ce dépôt distant.

```
git remote add origin https://github.com/votre-utilisateur/nom-du-repository.git
```

8. Pousser les Modifications vers GitHub

Envoyez vos commits locaux vers le dépôt distant sur GitHub.

```
git push -u origin main
```

9. Mettre à Jour le Référentiel Local avec les Modifications Distantes

Pour récupérer les modifications effectuées par d'autres collaborateurs, vous devez mettre à jour votre référentiel local.

```
git pull origin main
```

10. Cloner un Référentiel GitHub

Pour créer une copie locale d'un dépôt GitHub existant, utilisez la commande `git clone`.

```
git clone https://github.com/votre-utilisateur/nom-du-repository.git
```

11. Gérer les Branches

Créez et basculez entre les branches pour travailler sur différentes fonctionnalités ou corrections.

- **Créer une Nouvelle Branche**

```
git checkout -b nom-de-branche
```

- **Basculer vers une Branche Existante**

```
git checkout nom-de-branche
```

- **Fusionner une Branche avec la Branche Principale**

```
git checkout main
git merge nom-de-branche
```

12. Résolution des Conflits

Lors de la fusion des branches, des conflits peuvent survenir. Git marque les fichiers en conflit, et vous devez résoudre les conflits manuellement dans ces fichiers avant de compléter la fusion.

13. Supprimer une Branche

Pour supprimer une branche après fusion ou si elle n'est plus nécessaire.

```
git branch -d nom-de-branche
```

14. Afficher l'Historique des Commits

Pour visualiser l'historique des commits dans le dépôt.

```
git log
```

IX. Conclusion

1) Bilan du Projet

Ce projet a permis de démontrer l'intégration et l'utilisation des technologies Docker et Kubernetes dans un environnement de développement. En déployant une application Flask dans un cluster Kubernetes, nous avons pu comprendre les principes fondamentaux de la conteneurisation et de l'orchestration des conteneurs. Docker a facilité la création, le déploiement et l'exécution des conteneurs d'application, tandis que Kubernetes a assuré la gestion, la scalabilité et la résilience du déploiement.

2) Accomplissements

a) Conteneurisation :

L'application Flask a été empaquetée dans un conteneur Docker, simplifiant ainsi le déploiement et la gestion des dépendances.

b) Déploiement Kubernetes :

La configuration et le déploiement de l'application dans Kubernetes ont permis de mettre en place un environnement scalable et résilient.

c) Services et Accessibilité :

La configuration d'un service Kubernetes de type NodePort a rendu l'application accessible via le port 30000 sur localhost, illustrant la capacité de Kubernetes à exposer les services aux utilisateurs finaux.

3) Leçons Apprises

d) Gestion des Conteneurs :

La compréhension approfondie des concepts de Docker, tels que les images, les conteneurs, et les registres, a été essentielle pour le succès du projet.

e) Orchestration avec Kubernetes :

L'apprentissage de Kubernetes a permis de maîtriser la création de déploiements, la gestion des pods, et la configuration des services pour exposer les applications.

f) Débogage et Résolution de Problèmes :

La capacité à résoudre des problèmes complexes liés à la gestion des images Docker et aux configurations Kubernetes a été un aspect clé pour garantir le bon fonctionnement de l'application.

4) Perspectives d'Avenir

Pour améliorer davantage ce projet, il serait pertinent d'explorer les aspects suivants :

g) Automatisation du Déploiement :

Mettre en place des pipelines CI/CD pour automatiser le processus de construction, de test, et de déploiement des applications.

h) Sécurité :

Renforcer la sécurité des déploiements en intégrant des pratiques telles que l'analyse des vulnérabilités des images Docker et la gestion des secrets Kubernetes.

i) Monitoring et Logs :

Intégrer des outils de monitoring et de gestion des logs pour surveiller la performance et les erreurs des applications en production.

En conclusion, ce projet a fourni une base solide pour la compréhension et l'application des technologies modernes de conteneurisation et d'orchestration. Les compétences acquises sont précieuses pour tout environnement de développement et d'opérations, et ouvrent la voie à des améliorations continues et à des défis futurs.