Pratique

Voici un exemple complet d'un déploiement Kubernetes pour un serveur Nginx avec un conteneur sidecar Git Sync pour synchroniser le contenu d'un dépôt Git, le tout exposé via un Ingress pour l'URL docs.k8s.local.

1. Deployment avec Nginx et Git Sync sidecar

Le sidecar git-sync clone un dépôt Git et synchronise les fichiers dans un volume partagé avec Nginx. Cela permet à Nginx de servir le contenu à jour du dépôt.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-git-sync
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        ports:
        - containerPort: 80
        volumeMounts:
        - name: git-content
          mountPath: /usr/share/nginx/html  # Nginx sert les fichiers clonés

      - name: git-sync
        image: k8s.gcr.io/git-sync/git-sync:v3.3.1
        env:
        - name: GIT_SYNC_REPO
          value: "https://github.com/<your-username>/<your-repo>.git" # Remplacez par l'URL de votre dépôt
        - name: GIT_SYNC_BRANCH
          value: "main"  # Branche que vous souhaitez synchroniser
        - name: GIT_SYNC_ROOT
          value: "/git"
        - name: GIT_SYNC_DEST
          value: "repo"  # Nom du dossier dans lequel le dépôt sera cloné
```

```
        - name: GIT_SYNC_WAIT
          value: "30"  # Synchronisation toutes les 30 secondes
        volumeMounts:
        - name: git-content
          mountPath: /git/repo  # Le sidecar synchronise dans ce ré
              pertoire partagé

      volumes:
      - name: git-content
        emptyDir: {}
```

Dans cet exemple :

- **nginx** : Ce conteneur sert les fichiers synchronisés depuis le dépôt Git.
- **git-sync** : Ce conteneur clone et met à jour les fichiers depuis le dépôt Git à intervalles réguliers.
- **volume** : emptyDir est utilisé pour partager les fichiers entre Nginx et Git Sync.

2. Service pour exposer le déploiement en interne

Un service de type ClusterIP expose le déploiement en interne au cluster, sur le port 80.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

3. Ingress pour exposer le service via l'URL docs.k8s.local

L'Ingress utilise un hôte virtuel (docs.k8s.local) pour exposer Nginx. Assurez-vous que votre Ingress Controller est correctement configuré dans votre cluster.

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: nginx-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
```

```
  - host: docs.k8s.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: nginx-service
            port:
              number: 80
```

Dans cet exemple :

- **Ingress** : Définit une règle pour docs.k8s.local qui redirige les requêtes vers le service nginx-service.
- **Rewrite Target** : L'annotation nginx.ingress.kubernetes.io/rewrite-target: / redirige les requêtes à la racine (/) pour simplifier l'URL.

Résumé des étapes

- Déploiement de Nginx avec un sidecar Git Sync pour la mise à jour des fichiers statiques.
- Service pour exposer le déploiement au sein du cluster.
- Ingress pour exposer le service via l'URL docs.k8s.local.

Assurez-vous d'ajouter docs.k8s.local au fichier /etc/hosts de votre machine (ou configurez un DNS pour le domaine) pour pouvoir y accéder depuis un navigateur, si vous travaillez localement.