Automatiser le redéploiement Kubernetes avec un runner self-hosted GitHub Actions

Guide complet pour configurer un runner self-hosted GitHub Actions dans Kubernetes afin d'automatiser le redéploiement de vos applications lors des push sur la branche main

DevOps Kubernetes 50 min de lecture Niveau Avancé

Document généré le 25/05/2026 à 20h10 · nouvy.fr/wiki/automatiser-redepoiement-kubernetes-github-actions-runner

Sommaire

39 section(s) · 50 min de lecture

- 🎯 Vue d'ensemble
- 1️⃣ Prérequis
- 2️⃣ Créer un ServiceAccount limité pour le runner
  - ↳ Fichier runner-rbac.yaml
  - ↳ Appliquer la configuration
- 3️⃣ Générer un kubeconfig minimal pour le ServiceAccount
  - ↳ Récupérer les informations nécessaires
  - ↳ Générer le fichier kubeconfig
  - ↳ Créer le secret Kubernetes
- 4️⃣ Dockerfile du runner self-hosted
  - ↳ Dockerfile
- 5️⃣ Entrypoint du runner (entrypoint.sh)
- 6️⃣ Build et publication de l'image
  - ↳ Build de l'image
  - ↳ Tag et push sur GHCR
- 7️⃣ Créer le secret pour le token GitHub
  - ↳ Obtenir le token de registration
  - ↳ Créer le secret Kubernetes
- 8️⃣ Deployment Kubernetes du runner
  - ↳ Déployer le runner
  - ↳ Vérifier le déploiement
- 9️⃣ Workflow GitHub Actions
  - ↳ Workflow avancé avec vérification
- 🔟 Tester le workflow
  - ↳ Tester manuellement
  - ↳ Vérifier les logs
- 1️⃣1️⃣ Dépannage
  - ↳ Le runner ne s'enregistre pas

↳ Le workflow ne se déclenche pas
↳ Erreur "kubectl: command not found"
↳ Erreur "permission denied" avec kubectl
↳ Le runner se déconnecte fréquemment
1️2️ Améliorations possibles
↳ Utiliser un token permanent
↳ Ajouter des labels au runner
↳ Monitoring et alertes
↳ Auto-scaling
Résumé
Ressources

Ce guide vous explique comment configurer un runner self-hosted GitHub Actions directement dans votre cluster Kubernetes. Ce runner pourra automatiquement redéployer vos applications lorsque vous poussez du code sur la branche `main`, sans avoir besoin d'un serveur externe.

□ Vue d'ensemble

Dans ce tutoriel, nous allons :

1. □ Créer un ServiceAccount Kubernetes avec des permissions limitées
2. □ Générer un kubeconfig minimal pour le runner
3. □ Créer une image Docker pour le runner GitHub Actions
4. □ Déployer le runner dans Kubernetes
5. □ Configurer un workflow GitHub Actions pour redéployer automatiquement

Avantages :

- □ **Sécurité** : ServiceAccount avec permissions limitées (pas d'accès admin)
- □ **Automatisation** : Redéploiement automatique à chaque push
- □ **Économique** : Pas besoin de serveur externe
- □ **Intégration** : Directement dans votre cluster Kubernetes

---

1□ Prérequis

Avant de commencer, assurez-vous d'avoir :

- □ **Cluster Kubernetes opérationnel** (Master + Workers)
- □ **Docker fonctionnel** sur la machine où vous builderez l'image
- □ **Projet Node.js (backend) déployé** sur Kubernetes
- □ **Compte GitHub** avec un repository pour le backend
- □ **kubectl** configuré et connecté au cluster
- □ **Accès à GHCR** pour publier l'image du runner

2□ Créer un ServiceAccount limité pour le runner

Pour des raisons de sécurité, on ne veut **pas** que le runner utilise le kubeconfig admin. On crée un ServiceAccount avec des permissions limitées au namespace `default`.

Fichier runner-rbac.yaml

Créez le fichier `runner-rbac.yaml` :

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: github-runner
  namespace: default
---
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: github-runner-backend-role
  namespace: default
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get","list","watch"]
  - apiGroups: ["apps"]
    resources: ["deployments"]
    verbs: ["get","update","patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: github-runner-backend-binding
  namespace: default
subjects:
  - kind: ServiceAccount
    name: github-runner
    namespace: default
roleRef:
  kind: Role
  name: github-runner-backend-role
  apiGroup: rbac.authorization.k8s.io
```

Copier

Explications :

- **ServiceAccount** : Compte de service pour le runner
- **Role** : Permissions limitées (lecture des pods, modification des deployments)
- **RoleBinding** : Lie le ServiceAccount au Role

Appliquer la configuration

```
kubectl apply -f runner-rbac.yaml
```

Copier

Vérification :

```
# Vérifier le ServiceAccount
kubectl get sa github-runner -n default

# Vérifier le Role
kubectl get role github-runner-backend-role -n default

# Vérifier le RoleBinding
kubectl get rolebinding github-runner-backend-binding -n default
```

Copier

3️⃣ Générer un kubeconfig minimal pour le ServiceAccount

Le runner a besoin d'un fichier kubeconfig pour se connecter à Kubernetes. On va générer un kubeconfig minimal avec uniquement les permissions nécessaires.

Récupérer les informations nécessaires

```
# Récupérer le nom du secret du ServiceAccount
SECRET_NAME=$(kubectl get sa github-runner -n default -o jsonpath='{.secrets[0].name}')

# Récupérer le token
TOKEN=$(kubectl get secret $SECRET_NAME -n default -o jsonpath='{.data.token}' | base64 --
decode)

# Récupérer le certificat CA
CA=$(kubectl config view --raw --minify -o jsonpath='{.clusters[0].cluster.certificate-
authority-data}')

# Récupérer l'adresse du serveur
SERVER=$(kubectl config view --minify -o jsonpath='{.clusters[0].cluster.server}')
```

Copier

Générer le fichier kubeconfig

```
cat <<EOF > github-runner.kubeconfig
apiVersion: v1
kind: Config
clusters:
- cluster:
    certificate-authority-data: $CA
    server: $SERVER
  name: github-runner-cluster
contexts:
- context:
    cluster: github-runner-cluster
    user: github-runner
  name: github-runner-context
current-context: github-runner-context
users:
- name: github-runner
  user:
    token: $TOKEN
EOF
```

Copier

Vérification :

```
# Tester le kubeconfig
KUBECONFIG=./github-runner.kubeconfig kubectl get pods -n default
```

Copier

Créer le secret Kubernetes

Créez un secret Kubernetes pour monter le kubeconfig dans le pod runner :

```
kubectl create secret generic github-runner-kubeconfig
  --from-file=kubeconfig=github-runner.kubeconfig
  --dry-run=client -o yaml | kubectl apply -f -
```

Copier

Vérification :

```
kubectl get secret github-runner-kubeconfig -n default
```

Copier

---

4 Dockerfile du runner self-hosted

Créez un dossier `runner-backend` avec les fichiers suivants :

```
runner-backend/
├── Dockerfile
└── entrypoint.sh
```

Copier

Dockerfile

Créez le fichier `Dockerfile` :

```
FROM ubuntu:22.04

# Permettre au runner de s'exécuter en root
ENV RUNNER_ALLOW_RUNASROOT=1
ENV RUNNER_VERSION=2.329.0

# Installer dépendances de base + libicu + kubectl
RUN apt-get update && apt-get install -y
    curl
    git
    sudo
    jq
    libicu70
    libkrb5-3
    zlib1g
    libicu-dev
    && rm -rf /var/lib/apt/lists/*

# Installer kubectl ARM64
RUN curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/arm64/kubectl"
    && chmod +x kubectl
    && mv kubectl /usr/local/bin/

WORKDIR /actions-runner

# Télécharger et extraire le runner ARM64
RUN curl -o actions-runner-linux-arm64-${RUNNER_VERSION}.tar.gz -L
https://github.com/actions/runner/releases/download/v${RUNNER_VERSION}/actions-runner-linux-
arm64-${RUNNER_VERSION}.tar.gz
    && tar xzf actions-runner-linux-arm64-${RUNNER_VERSION}.tar.gz

# Installer les dépendances du runner
RUN ./bin/installdependencies.sh

# Copier l'entrypoint
COPY entrypoint.sh /entrypoint.sh
RUN chmod +x /entrypoint.sh

ENTRYPOINT ["/entrypoint.sh"]
```

Copier

Explications :

- **Ubuntu 22.04** : Image de base
- **RUNNER_VERSION** : Version du runner GitHub Actions (2.329.0)
- **kubectl ARM64** : Installation de kubectl pour ARM64 (adaptez si vous utilisez AMD64)
- **Dépendances** : Bibliothèques nécessaires pour le runner
- **Runner GitHub Actions** : Téléchargement et extraction du runner

Note : Si vous utilisez une architecture **AMD64**, modifiez la ligne kubectl :

```
# Pour AMD64
RUN curl -LO "https://dl.k8s.io/release/$(curl -L -s
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
    && chmod +x kubectl
    && mv kubectl /usr/local/bin/

# Et pour le runner
RUN curl -o actions-runner-linux-x64-${RUNNER_VERSION}.tar.gz -L
https://github.com/actions/runner/releases/download/v${RUNNER_VERSION}/actions-runner-linux-
x64-${RUNNER_VERSION}.tar.gz
    && tar xzf actions-runner-linux-x64-${RUNNER_VERSION}.tar.gz
```

Copier

---

5 Entrypoint du runner (entrypoint.sh)

Créez le fichier `entrypoint.sh` :

```
#!/bin/bash
set -e

# Vérification du token
if [ -z "$RUNNER_TOKEN" ]; then
  echo " RUNNER_TOKEN manquant"
  exit 1
fi

# Vérification de l'URL du repo
if [ -z "$RUNNER_REPO_URL" ]; then
  echo " RUNNER_REPO_URL manquant"
  exit 1
fi

# Export GITHUB_REPO pour éviter le message du runner
export GITHUB_REPO="$RUNNER_REPO_URL"

# Configuration du runner
./config.sh
  --url "$RUNNER_REPO_URL"
  --token "$RUNNER_TOKEN"
  --name "$(hostname)"
  --work "_work"
  --unattended

# Lancer le runner
./run.sh
```

Copier

Fonctionnalités :

- Vérifie la présence des variables d'environnement requises
- Configure le runner avec le token et l'URL du repository
- Lance le runner en mode unattended (sans interaction)

Permissions :

```
chmod +x entrypoint.sh
```

Copier

6️⃣ Build et publication de l'image

Build de l'image

```
cd runner-backend

# Build pour ARM64
docker build -t runner-backend:arm64 .

# Ou pour AMD64
docker build -t runner-backend:amd64 .
```

Copier

Tag et push sur GHCR

```
# Authentification avec GHCR
echo $GITHUB_TOKEN | docker login ghcr.io -u <GITHUB_USERNAME> --password-stdin

# Tagger l'image
docker tag runner-backend:arm64 ghcr.io/nouvy/runner-backend:arm64

# Pousser l'image
docker push ghcr.io/nouvy/runner-backend:arm64
```

Copier

Note : Assurez-vous d'avoir un token GitHub avec les permissions `write:packages` et `read:packages`.

7️⃣ Créer le secret pour le token GitHub

Le runner a besoin d'un token GitHub pour s'enregistrer. Créez un secret Kubernetes :

Obtenir le token de registration

1. Allez sur votre repository GitHub
2. **Settings** → **Actions** → **Runners**
3. Cliquez sur **New self-hosted runner**
4. Copiez le token affiché (valide 1 heure)

Ou utilisez un Personal Access Token avec les permissions `repo` et `workflow`.

Créer le secret Kubernetes

```
kubectl create secret generic github-runner-secret
  --from-literal=token=<VOTRE_TOKEN>
  --namespace=default
```

Copier

Note : Pour un token permanent, créez un **Fine-Grained PAT** avec les permissions :

- □ Repository permissions → Actions → Read and write
- □ Repository permissions → Contents → Read

8□ Deployment Kubernetes du runner

Créez le fichier `runner-deployment.yaml` :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: github-runner
  namespace: default
spec:
  replicas: 1
  selector:
    matchLabels:
      app: github-runner
  template:
    metadata:
      labels:
        app: github-runner
    spec:
      serviceAccountName: github-runner
      imagePullSecrets:
        - name: ghcr-secret
      containers:
        - name: runner
          image: ghcr.io/nouvy/runner-backend:arm64
          imagePullPolicy: Always
          env:
            - name: RUNNER_TOKEN
              valueFrom:
                secretKeyRef:
                  name: github-runner-secret
                  key: token
            - name: RUNNER_REPO_URL
              value: "https://github.com/Nouvy/mpi_backend_nodejs"
            - name: GITHUB_REPO
              value: "https://github.com/Nouvy/mpi_backend_nodejs"
            - name: RUNNER_ALLOW_RUNASROOT
              value: "1"
            - name: KUBECONFIG
              value: "/home/runner/.kube/config"
          volumeMounts:
            - name: kubeconfig-volume
              mountPath: /home/runner/.kube
              readOnly: true
          securityContext:
            runAsUser: 0
            runAsGroup: 0
            privileged: false
      volumes:
        - name: kubeconfig-volume
          secret:
            secretName: github-runner-kubeconfig
```

Copier

Points importants :

- **serviceAccountName** : Utilise le ServiceAccount créé précédemment
- **imagePullSecrets** : Secret pour pull l'image depuis GHCR
- **RUNNER_TOKEN** : Token pour enregistrer le runner (depuis le secret)
- **RUNNER_REPO_URL** : URL du repository GitHub
- **KUBECONFIG** : Chemin vers le fichier kubeconfig
- **volumeMounts** : Monte le secret kubeconfig dans le conteneur

Déployer le runner

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

Copier

Vérifier le déploiement

```
# Vérifier le pod
kubectl get pods -l app=github-runner -n default

# Voir les logs
kubectl logs -f deployment/github-runner -n default

# Vérifier sur GitHub
# Allez sur Settings → Actions → Runners
# Vous devriez voir votre runner "self-hosted"
```

Copier

Si le runner ne s'enregistre pas :

- Vérifiez que le token est valide
- Vérifiez les logs du pod
- Vérifiez que l'URL du repository est correcte

9️ Workflow GitHub Actions

Créez le fichier `.github/workflows/restart-backend.yml` dans votre repository :

```
name: Restart Backend on main push

on:
  push:
    branches:
      - main

jobs:
  restart-backend:
    runs-on: self-hosted
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Restart backend deployment
        run: |
          export KUBECONFIG=/home/runner/.kube/config
          kubectl rollout restart deployment/backend
```

Copier

Explications :

- **on: push: branches: main** : Déclenche le workflow à chaque push sur `main`
- **runs-on: self-hosted** : Utilise le runner self-hosted dans Kubernetes

- **kubectl rollout restart** : Redémarre le deployment backend

Workflow avancé avec vérification

Pour un workflow plus robuste :

```
name: Restart Backend on main push

on:
  push:
    branches:
      - main
    paths:
      - 'src/**'
      - 'package.json'
      - 'Dockerfile'

jobs:
  restart-backend:
    runs-on: self-hosted
    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Verify kubectl access
        run: |
          export KUBECONFIG=/home/runner/.kube/config
          kubectl get pods -n default

      - name: Restart backend deployment
        run: |
          export KUBECONFIG=/home/runner/.kube/config
          kubectl rollout restart deployment/backend -n default

      - name: Wait for rollout
        run: |
          export KUBECONFIG=/home/runner/.kube/config
          kubectl rollout status deployment/backend -n default --timeout=5m

      - name: Verify deployment
        run: |
          export KUBECONFIG=/home/runner/.kube/config
          kubectl get pods -l app=backend -n default
```

Copier

Fonctionnalités supplémentaires :

- **paths** : Déclenche uniquement si certains fichiers changent
- **Verify kubectl access** : Vérifie que kubectl fonctionne
- **Wait for rollout** : Attend que le redéploiement soit terminé
- **Verify deployment** : Vérifie que les pods sont prêts

Tester le workflow

Tester manuellement

1. Faites un changement dans votre code backend
2. Committez et poussez sur la branche `main` :

```
git add .
git commit -m "Test: redéploiement automatique"
git push origin main
```

Copier

3. Allez sur **GitHub** → **Actions** dans votre repository
4. Vous devriez voir le workflow s'exécuter
5. Vérifiez que le deployment backend a été redémarré :

```
kubectl get pods -l app=backend -n default
kubectl rollout history deployment/backend -n default
```

Copier

Vérifier les logs

```
# Logs du runner
kubectl logs -f deployment/github-runner -n default

# Logs du workflow (sur GitHub)
# Actions → Votre workflow → Voir les logs
```

Copier

1□1□ Dépannage

Le runner ne s'enregistre pas

Symptômes : Le runner n'apparaît pas dans GitHub Settings → Actions → Runners

Solutions :

```
# Vérifier les logs
kubectl logs -f deployment/github-runner -n default

# Vérifier le token
kubectl get secret github-runner-secret -n default -o jsonpath='{.data.token}' | base64 --
decode

# Vérifier l'URL du repository
kubectl describe deployment github-runner -n default
```

Copier

Le workflow ne se déclenche pas

Symptômes : Aucun workflow ne s'exécute après un push

Solutions :

- Vérifiez que le fichier `.github/workflows/restart-backend.yml` existe
- Vérifiez la syntaxe YAML
- Vérifiez que le runner est bien enregistré et actif
- Vérifiez les permissions du repository

Erreur "kubectl: command not found"

Symptômes : Le workflow échoue avec "kubectl: command not found"

Solutions :

- Vérifiez que kubectl est bien installé dans l'image Docker
- Vérifiez l'architecture (ARM64 vs AMD64)
- Vérifiez les logs du build de l'image

Erreur "permission denied" avec kubectl

Symptômes : Le workflow échoue avec des erreurs de permissions

Solutions :

```
# Vérifier le kubeconfig
KUBECONFIG=./github-runner.kubeconfig kubectl get pods -n default

# Vérifier le Role
kubectl get role github-runner-backend-role -n default -o yaml

# Vérifier le RoleBinding
kubectl get rolebinding github-runner-backend-binding -n default -o yaml
```

Copier

Le runner se déconnecte fréquemment

Symptômes : Le runner disparaît de la liste des runners

Solutions :

- Vérifiez que le pod est toujours en cours d'exécution
- Vérifiez les ressources (CPU/Memory) du pod
- Augmentez les limites de ressources si nécessaire
- Vérifiez les logs pour des erreurs

1□2□ Améliorations possibles

Utiliser un token permanent

Au lieu d'utiliser un token temporaire, créez un **Fine-Grained PAT** :

1. **GitHub → Settings → Developer settings → Personal access tokens → Fine-grained tokens**
2. Créez un token avec :
    - **Repository access** : `Nouvy/mpi_backend_nodejs`
    - **Permissions** :
        - □ Actions → Read and write
        - □ Contents → Read
3. Utilisez ce token dans le secret Kubernetes

Ajouter des labels au runner

Pour utiliser plusieurs runners avec des labels différents :

```
# Dans le deployment
env:
  - name: RUNNER_LABELS
    value: "kubernetes,backend"
```

Copier

Puis dans le workflow :

```
runs-on: [self-hosted, kubernetes, backend]
```

Copier

Monitoring et alertes

Ajoutez des métriques pour surveiller le runner :

```
# Ajouter un sidecar Prometheus ou exporter
# Ou utiliser kubectl top pour surveiller les ressources
```

Copier

Auto-scaling

Pour gérer plusieurs runners automatiquement :

- Utilisez **KEDA** (Kubernetes Event-Driven Autoscaling)
- Ou créez plusieurs deployments avec des labels différents

🎯 Résumé

Ce guide vous a permis de :

✅ Créer un ServiceAccount Kubernetes avec permissions limitées
✅ Générer un kubeconfig minimal pour le runner
✅ Créer une image Docker pour le runner GitHub Actions
✅ Déployer le runner dans Kubernetes
✅ Configurer un workflow pour redéployer automatiquement
✅ Tester et dépanner le système

Avantages obtenus :

- 🔒 Sécurité renforcée avec RBAC
- 🚀 Automatisation complète du redéploiement
- 💰 Pas de coût supplémentaire (runner dans le cluster)
- 🔄 Intégration native avec GitHub

Prochaines étapes :

- Ajouter des tests avant le redéploiement
- Configurer des notifications (Slack, Discord, etc.)
- Mettre en place un système de rollback automatique
- Ajouter du monitoring et des alertes

📚 Ressources

- GitHub Actions Self-Hosted Runners
- Kubernetes RBAC Documentation
- GitHub Actions Runner Releases
- kubectl Documentation