Déploiement Docker/Kubernetes : React (Vite) + Node.js avec GHCR

Guide complet pour créer des images Docker pour un frontend React/Vite et un backend Node.js, les pousser sur GHCR et les déployer via Kubernetes

Systèmes & Réseaux DevOps Kubernetes 45 min de lecture Niveau Intermédiaire

Document généré le 25/06/2026 à 21h33 · nouvy.fr/wiki/deploiement-docker-kubernetes-react-nodejs-ghcr

Sommaire

34 section(s) · 45 min de lecture

Ce guide vous explique comment créer des images Docker pour une application complète (frontend React/Vite et backend Node.js), les tester localement, les publier sur GitHub Container Registry (GHCR) et les déployer sur un cluster Kubernetes.

🎯 Vue d'ensemble

Dans ce tutoriel, nous allons :

1. 📦 Créer un Dockerfile pour le backend Node.js (clone d'un repo privé)
2. 📦 Créer un Dockerfile pour le frontend React/Vite (clone d'un repo public)
3. 🧪 Tester les images localement avec Docker
4. 🚀 Publier les images sur GHCR
5. ☸️ Déployer l'application sur Kubernetes

Prérequis :

- Docker installé
- Un cluster Kubernetes fonctionnel
- Un compte GitHub
- Accès aux repositories : `Nouvy/mpi_backend_nodejs` (privé) et `Nouvy/mpi_frontend_react` (public)

1️⃣ Configuration du Backend Node.js

Structure du projet

Créez un dossier `back-nodejs` avec les fichiers suivants :

```
back-nodejs/
├── Dockerfile
└── entrypoint.sh
```

Copier

Dockerfile pour le backend

Créez le fichier `Dockerfile` :

```
FROM node:18-alpine

# Installer Git + SSH
RUN apk add --no-cache git openssh

WORKDIR /app

# Copier l'entrypoint
COPY entrypoint.sh .

RUN chmod +x entrypoint.sh

EXPOSE 3000

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

Copier

Explications :

- `node:18-alpine` : Image Node.js 18 basée sur Alpine Linux (légère)
- Installation de Git et SSH pour cloner le repository
- Le script `entrypoint.sh` sera exécuté au démarrage du conteneur

Script entrypoint.sh pour le backend

Créez le fichier `entrypoint.sh` :

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

REPO_URL="https://${GITHUB_TOKEN}@github.com/Nouvy/mpi_backend_nodejs.git"
APP_DIR="/app/mpi_backend_nodejs"

# Cloner ou mettre à jour le dépôt
if [ ! -d "$APP_DIR" ]; then
    echo "🔄 Clonage du dépôt Git..."
    git clone "$REPO_URL" "$APP_DIR"
else
    echo "🔄 Mise à jour du dépôt existant..."
    cd "$APP_DIR"
    git pull
fi

cd "$APP_DIR"

# Créer config.env
echo "📝 Création du fichier config.env..."
cat > config.env <<EOL
DB_HOST=mysql
DB_PORT=3306
DB_USER=root
DB_PASSWORD=secret
DB_NAME=polls_db
EOL

# Installer les dépendances
echo "📦 Installation des dépendances..."
npm install

# Lancer l'application
echo "🚀 Lancement de l'application..."
npm start
```

Copier

Fonctionnalités :

- Clone le repository privé en utilisant un token GitHub
- Crée le fichier de configuration `config.env`
- Installe les dépendances npm
- Lance l'application

Important : Le script nécessite la variable d'environnement `GITHUB_TOKEN` pour cloner le repository privé.

Build et test du backend

```
# Construire l'image
docker build -t back-nodejs .

# Lancer le conteneur (avec lien vers MySQL si nécessaire)
docker run -d -p 3001:3001
  --link db:mysql
  -e GITHUB_TOKEN=<TON_TOKEN>
  --name backend
  back-nodejs
```

Copier

Paramètres :

- `-p 3001:3001` : Mappe le port 3001 du conteneur sur le port 3001 de l'hôte
- `--link db:mysql` : Lie le conteneur à un conteneur MySQL nommé `db`
- `-e GITHUB_TOKEN` : Passe le token GitHub pour cloner le repo privé

---

2 Configuration du Frontend React/Vite

Structure du projet

Créez un dossier `front-react` avec les fichiers suivants :

```
front-react/
├── Dockerfile
└── entrypoint.sh
```

Copier

Dockerfile pour le frontend

Créez le fichier `Dockerfile` :

```
FROM node:18-alpine

# Installer git et bash
RUN apk add --no-cache git bash

WORKDIR /app

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

EXPOSE 4173

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

Copier

Explications :

- `node:18-alpine` : Image Node.js 18

- Installation de Git et Bash pour cloner le repository
- Port 4173 : Port par défaut de Vite Preview

Script entrypoint.sh pour le frontend

Créez le fichier `entrypoint.sh` :

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

REPO_URL="https://github.com/Nouvy/mpi_frontend_react.git"
APP_DIR="/app/mpi_frontend_react"

# Cloner le projet si pas déjà présent
if [ ! -d "$APP_DIR" ]; then
    echo "== Clonage du dépôt =="
    git clone "$REPO_URL" "$APP_DIR"
fi

cd "$APP_DIR"

# .env pour Vite
cat > .env <<EOL
VITE_API_BASE_URL=http://192.168.194.159:3001/api
VITE_SOCKET_URL=http://192.168.194.159:3001
EOL

# Installer dépendances
npm install --silent

# Build Vite
npm run build

# Lancer vite preview
echo "== Lancement du serveur Vite Preview =="
npx vite preview --port 4173 --host
```

Copier

Fonctionnalités :

- Clone le repository public React
- Crée le fichier `.env` avec les URLs de l'API
- Installe les dépendances
- Build l'application Vite
- Lance le serveur de preview sur le port 4173

Note : Adaptez les URLs dans `.env` selon votre configuration réseau.

Build et test du frontend

```
# Construire l'image
docker build -t front-react .

# Lancer le conteneur
docker run -d -p 4173:4173
  --link backend:backend
  --name frontend
  front-react
```

Copier

Paramètres :

- `-p 4173:4173` : Mappe le port 4173
- `--link backend:backend` : Lie le conteneur au backend

---

3 Création des tokens GitHub

Token Fine-Grained PAT pour cloner le repo privé

Pour cloner le repository privé `Nouvy/mpi_backend_nodejs`, vous devez créer un **Fine-Grained Personal Access Token** :

1. Allez sur **GitHub** → **Settings** → **Developer settings**
2. Cliquez sur **Personal access tokens** → **Fine-grained tokens**
3. Cliquez sur **Generate new token**
4. Configurez le token :
    - **Token name** : `MPI Backend Clone Token`
    - **Expiration** : Choisissez une durée appropriée
    - **Repository access** : Sélectionnez `Nouvy/mpi_backend_nodejs`
    - **Repository permissions** → **Contents** → **Read** (obligatoire)
5. Cliquez sur **Generate token**
6. ⚠ **IMPORTANT** : Copiez le token immédiatement !

Ce token sera utilisé dans la variable d'environnement `GITHUB_TOKEN` pour cloner le repository privé.

Token Classic pour GHCR (si pas déjà créé)

Pour publier et lire les packages sur GHCR, créez un **Classic Personal Access Token** :

1. Allez sur **GitHub** → **Settings** → **Developer settings**
2. Cliquez sur **Personal access tokens** → **Tokens (classic)**
3. Cliquez sur **Generate new token (classic)**
4. Configurez le token :
    - **Note** : `GHCR Docker Token`
    - **Scopes** :
        - `write:packages` : Pour publier des images
        - `read:packages` : Pour télécharger des images

5. Cliquez sur **Generate token**
6. ⚠ **IMPORTANT** : Copiez le token immédiatement !

---

4 Publication sur GHCR

Authentification avec GHCR

```
# Se connecter à GHCR
echo $GITHUB_TOKEN | docker login ghcr.io -u <GITHUB_USERNAME> --password-stdin
```

Copier

Exemple :

```
export GITHUB_TOKEN=ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
echo $GITHUB_TOKEN | docker login ghcr.io -u nouvy --password-stdin
```

Copier

Tag et push du backend

```
# Tagger l'image
docker tag back-nodejs ghcr.io/nouvy/back-nodejs:latest

# Pousser l'image
docker push ghcr.io/nouvy/back-nodejs:latest
```

Copier

Tag et push du frontend

```
# Tagger l'image
docker tag front-react ghcr.io/nouvy/front-react:latest

# Pousser l'image
docker push ghcr.io/nouvy/front-react:latest
```

Copier

Vérification sur GitHub

1. Allez sur votre profil GitHub
2. Cliquez sur **Packages** (à droite de votre avatar)
3. Vous devriez voir vos packages `back-nodejs` et `front-react`

Note : Par défaut, les packages sont privés. Pour les rendre publics :

- Allez dans **Package settings** → **Change visibility** → **Public**

5⃣ Configuration Kubernetes

Créer le secret pour GHCR

Kubernetes a besoin d'un secret pour s'authentifier auprès de GHCR et pull les images :

```
# Créer le secret Docker registry
kubectl create secret docker-registry ghcr-secret
  --docker-server=ghcr.io
  --docker-username=<GITHUB_USERNAME>
  --docker-password=<GITHUB_TOKEN>
  --docker-email=<VOTRE_EMAIL>
```

Copier

Exemple :

```
kubectl create secret docker-registry ghcr-secret
  --docker-server=ghcr.io
  --docker-username=nouvy
  --docker-password=ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
  --docker-email=your.email@example.com
```

Copier

Créer le secret pour le token GitHub

Pour que le backend puisse cloner le repository privé, créez un secret Kubernetes :

```
# Créer le secret pour le token GitHub
kubectl create secret generic github-token
  --from-literal=TOKEN=<VOTRE_FINE_GRAINED_PAT>
```

Copier

Exemple :

```
kubectl create secret generic github-token
  --from-literal=TOKEN=github_pat_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Copier

Important : Utilisez le **Fine-Grained PAT** avec la permission `Contents: Read` sur le repository privé.

6⃣ Déploiement Kubernetes

Deployment et Service pour le Backend

Créez le fichier `backend-deployment.yaml` :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
  labels:
    app: backend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      imagePullSecrets:
        - name: ghcr-secret
      containers:
        - name: backend
          image: ghcr.io/nouvy/back-nodejs:latest
          ports:
            - containerPort: 3001
          env:
            - name: GITHUB_TOKEN
              valueFrom:
                secretKeyRef:
                  name: github-token
                  key: TOKEN
            - name: DB_HOST
              value: "mysql"
            - name: DB_PORT
              value: "3306"
            - name: DB_USER
              value: "root"
            - name: DB_PASSWORD
              value: "secret"
            - name: DB_NAME
              value: "polls_db"
---
apiVersion: v1
kind: Service
metadata:
  name: backend
  labels:
    app: backend
spec:
  type: NodePort
  selector:
    app: backend
  ports:
    - name: http
      port: 3001        # port interne du cluster
      targetPort: 3001  # port sur lequel le container écoute
      nodePort: 30301   # port exposé sur les nœuds du cluster
```

Copier

Points importants :

- `imagePullSecrets` : Utilise le secret `ghcr-secret` pour pull l'image
- `GITHUB_TOKEN` : Récupère le token depuis le secret `github-token`
- `NodePort: 30301` : Expose le service sur le port 30301 des nœuds

Deployment et Service pour le Frontend

Créez le fichier `frontend-deployment.yaml` :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  replicas: 1
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      imagePullSecrets:
        - name: ghcr-secret
      containers:
        - name: frontend
          image: ghcr.io/nouvy/front-react:latest
          ports:
            - containerPort: 4173
          env:
            - name: VITE_API_BASE_URL
              value: "http://192.168.194.100:30301/api"
            - name: VITE_SOCKET_URL
              value: "http://192.168.194.100:30301"
---
apiVersion: v1
kind: Service
metadata:
  name: frontend
  labels:
    app: frontend
spec:
  type: NodePort
  selector:
    app: frontend
  ports:
    - name: http
      port: 4173
      targetPort: 4173
      nodePort: 30173
```

Copier

Points importants :

- `VITE_API_BASE_URL` : URL de l'API backend (adaptez l'IP selon votre cluster)
- `VITE_SOCKET_URL` : URL pour les WebSockets
- `NodePort: 30173` : Expose le frontend sur le port 30173

Note : Adaptez l'IP `192.168.194.100` selon l'adresse IP de votre nœud Kubernetes.

Déployer les applications

```
# Déployer le backend
kubectl apply -f backend-deployment.yaml

# Déployer le frontend
kubectl apply -f frontend-deployment.yaml
```

Copier

Vérifier le déploiement

```
# Vérifier les pods
kubectl get pods

# Vérifier les services
kubectl get services

# Voir les logs du backend
kubectl logs -f deployment/backend

# Voir les logs du frontend
kubectl logs -f deployment/frontend
```

Copier

Accéder à l'application

Une fois déployée, vous pouvez accéder à l'application :

- **Frontend** : `http://<NODE_IP>:30173`
- **Backend API** : `http://<NODE_IP>:30301`

---

7 Mise à jour des images

Workflow de mise à jour

Lorsque vous modifiez le code et souhaitez mettre à jour le déploiement :

1. **Rebuild les images** :

   ```
   docker build -t back-nodejs .
   docker build -t front-react .
   ```

   Copier

2. **Tag et push** :

```
docker tag back-nodejs ghcr.io/nouvy/back-nodejs:latest
docker tag front-react ghcr.io/nouvy/front-react:latest
docker push ghcr.io/nouvy/back-nodejs:latest
docker push ghcr.io/nouvy/front-react:latest
```

Copier

3. **Redémarrer les pods Kubernetes** :

```
kubectl rollout restart deployment/backend
kubectl rollout restart deployment/frontend
```

Copier

Ou supprimer les pods pour forcer le redémarrage :

```
kubectl delete pod -l app=backend
kubectl delete pod -l app=frontend
```

Copier

---

8 Dépannage

Problèmes courants

Erreur : "Failed to pull image"

Cause : Le secret `ghcr-secret` n'est pas correctement configuré.

Solution :

```
# Vérifier le secret
kubectl get secret ghcr-secret

# Recréer le secret si nécessaire
kubectl delete secret ghcr-secret
kubectl create secret docker-registry ghcr-secret
  --docker-server=ghcr.io
  --docker-username=<USERNAME>
  --docker-password=<TOKEN>
  --docker-email=<EMAIL>
```

Copier

Erreur : "Repository not found" lors du clone

Cause : Le token GitHub n'a pas les bonnes permissions.

Solution : Vérifiez que votre Fine-Grained PAT a bien la permission `Contents: Read` sur le repository privé.

Le frontend ne peut pas se connecter au backend

Cause : Les URLs dans `.env` ne sont pas correctes.

Solution : Vérifiez l'IP du nœud Kubernetes et mettez à jour les variables d'environnement dans le deployment.

Les pods sont en état "ImagePullBackOff"

Cause : L'image n'existe pas sur GHCR ou le secret est incorrect.

Solution :

```
# Vérifier les événements
kubectl describe pod <POD_NAME>

# Vérifier que l'image existe
docker pull ghcr.io/nouvy/back-nodejs:latest
```

Copier

Résumé

Ce guide vous a permis de :

Créer des images Docker pour React/Vite et Node.js
Cloner des repositories publics et privés dans les conteneurs
Tester les images localement
Publier les images sur GHCR
Configurer les secrets Kubernetes
Déployer l'application sur Kubernetes
Exposer les services via NodePort

Prochaines étapes :

- Configurer un Ingress pour une meilleure gestion du trafic
- Ajouter un service MySQL dans Kubernetes
- Mettre en place un CI/CD avec GitHub Actions
- Utiliser des ConfigMaps pour la configuration

Ressources

- Documentation Docker
- Documentation Kubernetes
- GitHub Container Registry
- Vite Documentation