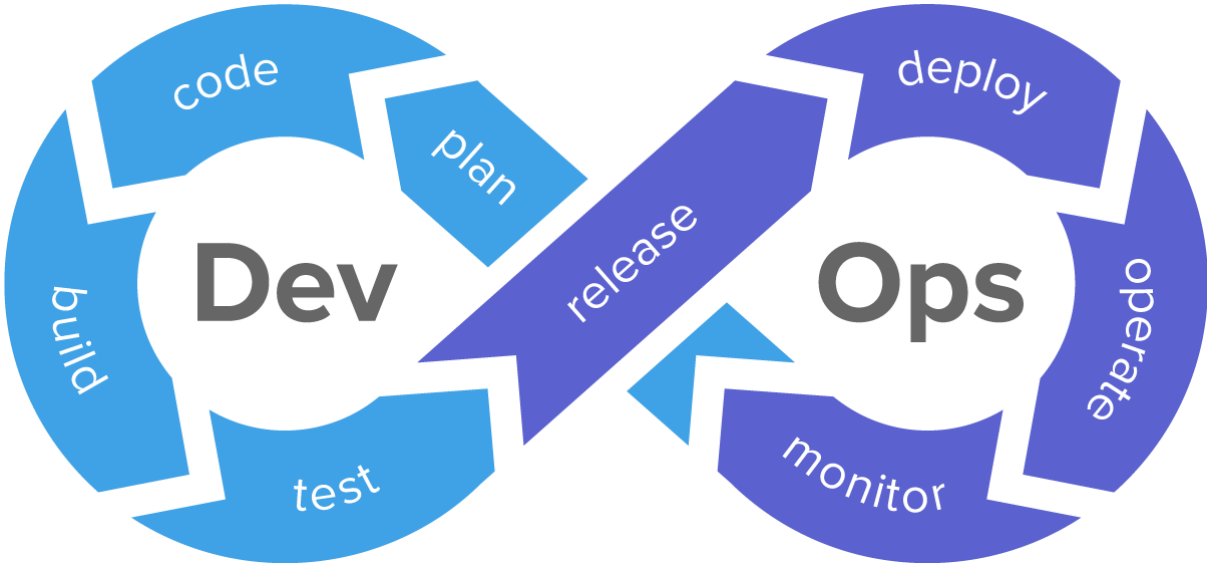
Documentation du projet TRINITY – Partie DevOps

Table des matières



1. Technologies et architecture du projet

Technologies utilisées

Couleur : Bleu foncé pour les titres et sous-titres, noir pour le corps du texte.

1. **Docker**

Conteneurisation des applications pour garantir la cohérence entre les environnements (dev, prod).

Utilisation de docker-compose.yml pour orchestrer les services backend et frontend.

Ce fichier comprend les configurations suivantes :

a. Builds des environnements **dev** et **prod**
b. Configuration des noms de conteneur, ports et variables d'environnement
c. Commandes pour lancer et tester les versions **dev** et **prod** du backend et du frontend.

Note importante :

Utiliser Docker garantit que votre application fonctionne de manière cohérente dans tous les environnements.

2. **GitLab CI/CD**

Automatisation des processus de développement, de tests et de déploiement via des pipelines CI/CD.

Utilisation de **GitLab Runners** pour exécuter des tâches comme les tests, le build et le déploiement.

3. **Backend (Node.js avec Express)**

Utilisation de **Node.js** et **Express** pour gérer les requêtes et les API.

Sécurisation des routes avec **passport.js** pour l'authentification et la gestion des utilisateurs.

Gestion des migrations avec **MongoDB** via migrate-mongo.

4. **Frontend (React avec Vite et Tailwind CSS)**

Utilisation de **React** pour l'interface utilisateur, configuré avec **Vite** pour des builds rapides.

Tailwind CSS pour une gestion flexible du design.

Vitest pour les tests unitaires côté frontend.

2. Différences entre les configurations de développement et de production

Environnement de développement (Backend et Frontend)

Backend Dockerfile (dev)

Couleur : Titre en bleu, texte en noir, et code en **gris clair avec fond bleu clair**.

Pour l'environnement de développement, le Dockerfile backend configure l'application pour un développement fluide avec un maximum de ressources allouées et un démarrage rapide.

```
# Development
FROM node:21.1-alpine AS development

WORKDIR /app

ENV NODE_OPTIONS="--max-old-space-size=2048"

COPY package*.json ./
RUN npm ci

COPY . .

EXPOSE 5001
CMD ["npm", "run", "dev"]
```

Explication :

- WORKDIR /app : Définit le répertoire de travail dans le conteneur.
- npm ci : Installe les dépendances en utilisant un fichier package-lock.json pour garantir des versions fixes.
- CMD ["npm", "run", "dev"] : Lance l'application en mode développement, généralement avec un outil comme nodemon ou un équivalent pour recharger les modifications automatiquement.
- Le port **5001** est exposé pour l'application backend en mode développement.

Environnement de production (Backend et Frontend)

Backend Dockerfile (prod)

```
# Production
FROM node:21.1-alpine AS production

WORKDIR /app

ENV NODE_OPTIONS="--max-old-space-size=2048"

COPY package*.json ./
RUN npm ci --only=production

COPY . .

EXPOSE 5002
CMD ["node", "./server.js"]
```

Explication :

- Les dépendances sont installées avec npm ci --only=production pour s'assurer que seules les dépendances nécessaires pour la production sont installées.
- Le port **5002** est exposé pour l'application backend en production.

3. Pipeline CI/CD

Description des étapes du pipeline

Voici un exemple de pipeline CI/CD basé sur GitLab, qui inclut plusieurs étapes pour construire, tester et déployer l'application dans des environnements de développement et de production.

Étapes principales :

1. **Trigger** : Le pipeline est déclenché automatiquement sur une branche spécifique, ici dockerization.
2. **Build** :
    a. **Backend** : La première étape construit les images Docker pour le backend (dev et prod) via docker-compose build.
    b. **Frontend** : De même, cette étape construit les images Docker pour le frontend (dev et prod).
3. **Test** :
    a. **Backend** : Les tests sont exécutés dans le conteneur backend avec la commande docker exec container_back_test npm run test.
    b. **Frontend** : Les tests sont exécutés dans le conteneur frontend avec docker exec container_front_test npm run test.
4. **Deploy** :
    a. **Déploiement en développement** : Déploie le backend et le frontend dans l'environnement de développement via docker-compose up -d backend-dev frontend-dev.
    b. **Déploiement en production** : Déploie le backend et le frontend dans l'environnement de production via docker-compose up -d backend-prod frontend-prod.

4. Instructions pour installer et configurer les GitLab Runners

Étape 1 : Préparer l'environnement

Assurez-vous d'avoir une machine ou une VM prête, avec Docker installé.

Étape 2 : Installer GitLab Runner

Téléchargez le binaire GitLab Runner
curl -L --output /usr/local/bin/gitlab-runner https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-linux-amd64

```
chmod +x /usr/local/bin/gitlab-runner

# Créez un utilisateur pour GitLab Runner
useradd --comment 'GitLab Runner' --create-home gitlab-runner --shell /bin/bash
```

Étape 3 : Enregistrer le Runner

1. Récupérez votre **token** d'enregistrement depuis GitLab dans **Settings > CI/CD > Runners**.
2. Enregistrez le Runner en exécutant : gitlab-runner register

Étape 4 : Tester le Runner

1. Créez une pipeline simple et assurez-vous que le runner est opérationnel.
2. Vérifiez que les tâches de build, test et déploiement sont exécutées comme prévu.

Conclusion

Cette documentation couvre l'architecture du projet TRINITY et décrit les pratiques DevOps mises en place, de l'automatisation des builds aux déploiements. Assurez-vous d'adapter les configurations et d'ajouter les informations spécifiques de votre projet pour garantir une documentation complète et opérationnelle.