

# Rapport de projet DevOps

## 1. Introduction

Ce projet a pour objectif de mettre en pratique les concepts DevOps de bout en bout à travers la conception, le développement et le déploiement d'un service backend léger sous forme d'API REST. Le projet couvre l'ensemble du cycle de vie logiciel : gestion du code source, intégration continue, livraison continue, observabilité, sécurité et déploiement conteneurisé.

L'API développée est volontairement simple (moins de 150 lignes de code) afin de se concentrer sur les bonnes pratiques DevOps plutôt que sur la complexité fonctionnelle.

---

## 2. Description générale du projet

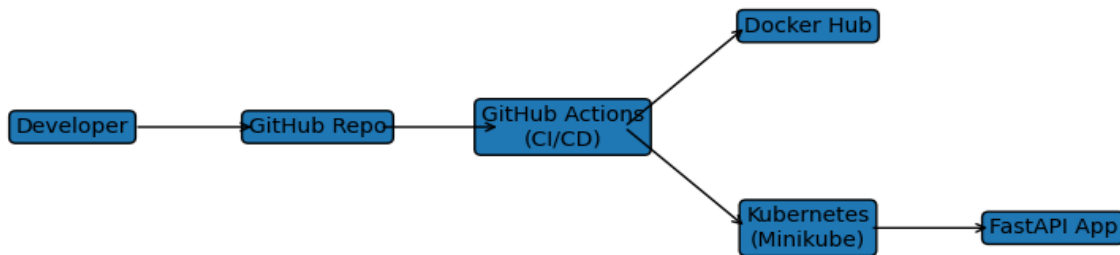
- **Type de service** : API REST
  - **Framework backend** : FastAPI (Python)
  - **Contrôle de version** : Git et GitHub
  - **Méthodologie** : GitHub Issues et Pull Requests avec revue par les pairs
  - **Objectif principal** : démontrer une chaîne DevOps complète et fonctionnelle
- 

## 3. Architecture du système

### 3.1 Vue d'ensemble

L'architecture du projet repose sur une API FastAPI conteneurisée avec Docker, intégrée dans un pipeline CI/CD via GitHub Actions, et déployée dans un cluster Kubernetes local.

## 3.2 Diagramme d'architecture



### Description textuelle du diagramme :

1. Le développeur pousse le code sur GitHub.
2. GitHub Actions déclenche automatiquement le pipeline CI/CD.
3. Les étapes CI incluent : build, tests, analyse SAST (Bandit).
4. L'image Docker est construite et publiée sur Docker Hub.
5. L'application est déployée sur Kubernetes (Minikube / Kind).
6. Des outils d'observabilité collectent métriques, logs et traces.
7. Une analyse DAST (OWASP ZAP) est exécutée sur l'API déployée.

---

## 4. Gestion du code et collaboration

### 4.1 GitHub Workflow

- Utilisation de **GitHub Issues** pour découper le projet en tâches claires.
- Création de **branches dédiées** pour chaque fonctionnalité.
- Ouverture de **Pull Requests** pour chaque étape importante.
- Réalisation d'au moins une **peer review** avec commentaires constructifs.

## 4.2 Revue par les pairs

La revue de code a permis :

- D'améliorer la lisibilité du code.
  - De vérifier le respect des bonnes pratiques DevOps.
  - D'identifier des améliorations en matière de sécurité et de documentation.
- 

## 5. CI/CD Pipeline

Le pipeline CI/CD est implémenté avec **GitHub Actions**.

Étapes principales :

1. Récupération du code source.
  2. Installation des dépendances.
  3. Lancement des tests unitaires.
  4. Analyse SAST avec **Bandit**.
  5. Build de l'image Docker.
  6. Push de l'image vers Docker Hub.
  7. (Optionnel) Déploiement automatique sur Kubernetes.
- 

## 6. Conteneurisation

L'application est conteneurisée avec **Docker**.

- **Dockerfile** optimisé et léger.
- Image publiée sur **Docker Hub**.
- Possibilité d'exécution locale via `docker run`.

Avantages : - Portabilité - Reproductibilité - Isolation des dépendances

---

## 7. Observabilité

### 7.1 Métriques

- Endpoint `/metrics` exposé.
- Mesures simples : nombre de requêtes, temps de réponse.

### 7.2 Logs

- Logs structurés pour chaque requête entrante.
- Logs d'erreurs clairs et exploitables.

## 7.3 Tracing

- Traçage basique des requêtes pour suivre le flux d'exécution.
- 

## 8. Sécurité

### 8.1 SAST – Analyse statique

- Outil utilisé : **Bandit**
- Analyse du code source Python.
- Rapport généré et archivé dans le dépôt.

### 8.2 DAST – Analyse dynamique

- Outil utilisé : **OWASP ZAP (baseline scan)**
  - Scan de l'API en cours d'exécution.
  - Rapport HTML généré (zap-report.html).
- 

## 9. Déploiement Kubernetes

- Utilisation de **Minikube / Kind**.
  - Définition des manifests Kubernetes :
    - Deployment
    - Service
  - Application accessible localement via un service exposé.
- 

## 10. Documentation

Le dépôt GitHub contient : - Un README.md clair. - Instructions d'installation locale. - Guide d'utilisation Docker. - Exemples d'appels API.

---

## 11. Difficultés rencontrées

- Problèmes de configuration Docker et PowerShell.
- Gestion des scans de sécurité dans l'environnement local.
- Problèmes système (boot Windows) nécessitant des solutions temporaires.

Ces difficultés ont permis de renforcer la capacité de diagnostic et de résolution de problèmes.

---

## 12. Leçons apprises

- Importance de l'automatisation dans le cycle DevOps.

- Valeur ajoutée des revues de code.
  - Intégration précoce de la sécurité (DevSecOps).
  - Vision globale du cycle de vie d'une application moderne.
- 

## 13. Conclusion

Ce projet a permis de mettre en œuvre une chaîne DevOps complète et cohérente, en respectant les exigences pédagogiques. Il démontre la capacité à concevoir, sécuriser, observer et déployer une application backend moderne selon les standards actuels de l'ingénierie logicielle.

---