Projet Final : Pipeline CI/CD orienté données sur Kubernetes

Author : Badr TAJINI - DevOps for SWE - ESIEE 2025

Résumé du Projet

Le projet final du cours **Cloud Native DevOps** consiste à concevoir et déployer une application web simple (frontend, backend, base de données) à l'aide d'un **pipeline CI/CD** - (**cf. Chapitre 5/Lab 5**). L'objectif est de se familiariser avec les fondamentaux de l'intégration et de la livraison continues CI/CD, en privilégiant la simplicité : **peu de sécurité avancée**, **peu de configurations complexes**, mais une **mise en pratique concrète du déploiement sur Kubernetes**, **en local** ou **dans le cloud**.

Vous pouvez développer votre application web dans le langage de votre choix (**Python**, **JavaScript**, **Go**, etc.) et choisir la plateforme qui vous convient le mieux (ex. : **Minikube** en local, ou un cluster managé comme **EKS** sur AWS ou **AKS** sur Azure). L'essentiel est de mettre en place un pipeline automatisé qui assure la construction, le test et le déploiement de l'application, tout en gérant la base de données et quelques opérations de lecture/écriture simples (**CRUD : Create-Read-Update-Delete**).

Objectifs Pédagogiques

1. **Intégration Continue (CI)** :
    - Mettre en place des tests (unitaires ou simples) pour le backend et éventuellement le frontend.
    - Automatiser la génération d'images Docker.
2. **Livraison Continue (CD)** :
    - Déployer automatiquement l'application sur un cluster Kubernetes (local ou cloud).
    - Gérer la mise à jour continue à chaque commit ou fusion de code.
3. **Gestion de la Base de Données** :
    - Implémenter une base de données conteneurisée ou un service managé (MySQL, PostgreSQL, MongoDB, etc.).
    - Vérifier la connectivité avec le backend et effectuer des opérations de lecture/écriture.
4. **Culture DevOps Simplifiée** :

Travailler avec Git et un outil CI/CD (GitHub Actions, GitLab CI, ou Azure DevOps).
Comprendre les notions de pipeline (build, test, déploiement) et de collaboration basique (pull requests, merges).

5. **Documentation et Présentation** :
    - Rédiger un rapport technique (**ReadME**) concis et réaliser, si possible, une courte démonstration vidéo.
    - Expliquer les étapes du déploiement et les principaux choix techniques.

Conditions du Projet

1. **Choix du domaine** : Vous pouvez choisir n'importe quel thème motivant (ex. : blog personnel, gestion de tâches, notes de cours, etc.).
2. **Choix de la plateforme** :
    - **Implémentation locale** : Minikube (recommandée pour débuter).
    - **Implémentation cloud** : AWS EKS ou Azure AKS (pour ceux qui disposent de crédits étudiants). Utilisation de **Azure AKS** avec le **free tier** serait plus judicieux pour respecter la condition de la gratuité du projet.
3. **Travail collaboratif** : Binômes ou équipes restreintes (**jusqu'à un maximum de 4, mais la traçabilité de qui a fait quoi dans le projet est absolument nécessaire**) sont encouragées pour favoriser l'échange d'idées (le projet en monôme reste acceptable).
4. **Livrables** : Rendu incluant un **rapport technique**, le **code source** (backend, frontend, config Kubernetes) et une **vidéo démonstrative**.

Phases du Projet

Phase 1 : Initialisation et déploiement de l'application

- **Objectif** : Créer la structure initiale (frontend, backend, base de données) et configurer Kubernetes.
- **Tâches** :
    1. Installer et configurer **Minikube** en local (ou un cluster **AKS (AZURE)** / **EKS (AWS)**).
    2. Créer un simple backend (API REST, par exemple) et un frontend minimal.
    3. Conteneuriser ces deux composants (Dockerfiles).
    4. Déployer la base de données (conteneur local ou service managé).

Phase 2 : Mise en place de l'intégration continue (CI)

- **Objectif** : Automatiser la construction et les tests du code applicatif.
- **Tâches** :
    1. Configurer un pipeline CI (GitHub Actions, GitLab CI, ou Azure DevOps) :
        - Compilation et/ou build Docker.
        - Lancement des tests unitaires pour le backend (et éventuellement le frontend).
    2. Gérer le push des images Docker vers un registre (Docker Hub ou équivalent).
    3. Mettre en place des badges de statut (build, tests) dans le README (facultatif).

Phase 3 : Mise en place de la livraison continue (CD)

- **Objectif** : Déployer automatiquement sur Kubernetes après validation du code.
- **Tâches** :
    1. Rédiger des manifestes Kubernetes ou un Helm Chart simple (Deployments, Services) pour le frontend, le backend et la base de données.
    2. Configurer le pipeline CD (ou GitOps) pour **kubectl apply** ou **helm upgrade** à chaque merge ou tag.
    3. Vérifier la connectivité entre le backend et la base de données (via variables d'environnement).

Phase 4 : Validation et démonstration de l'application

- **Objectif** : Tester le bon fonctionnement et présenter le résultat final.
- **Tâches** :
    1. Réaliser des scénarios de tests simples : créer, modifier, supprimer des enregistrements en base et vérifier l'affichage dans le frontend.
    2. Prendre des captures d'écran ou enregistrer une courte vidéo montrant le pipeline en action et l'application en fonctionnement.
    3. Rédiger un rapport ou **README** expliquant :
        - L'architecture générale (comment tout s'articule).
        - Les étapes pour lancer le projet (installation, commandes, déploiement).

Critères d'Évaluation

1. **Technique** :
    - Qualité du pipeline CI/CD (build, test, déploiement).
    - Bonne structure du code et clarté des Dockerfiles.
2. **Gestion des données** :
    - Mise en place correcte de la base de données.
    - Fonctions de lecture/écriture minimales et validation du fonctionnement.
3. **Simplicité et lisibilité** :
    - Fichiers Kubernetes/Helm clairs et faciles à répliquer.
    - Documentation (**README**, **rapport**) suffisamment détaillée pour qu'un tiers puisse déployer l'application.
4. **Présentation** :
    - Pertinence du rapport technique (ou **README**) et éventuellement de la vidéo.
    - Explication claire du pipeline et démonstration concrète (screenshots ou vidéo).

Livrables

1. **Rapport Technique / README** : Documentation de l'architecture, des étapes d'installation et des résultats.
2. **Code Source** : Contenu du backend, du frontend et fichiers de configuration Kubernetes (YAML ou Helm Chart).
3. **Fichiers de Pipeline CI/CD** : Script(s) YAML pour GitHub Actions/GitLab CI/Azure DevOps.
4. **Vidéo Démonstrative** : Courte présentation illustrant le pipeline et l'application en fonctionnement.

Deadline

- **Rendu Final du projet (Site web personnel)** : 01/02/2026 à 23h59 (heure de Paris).
    - Vous devez uploader votre site web personnel avec toutes les ressources nécessaires permettant sa validation finale.
- **Rendu Final des labs (Site web personnel)** : 01/02/2026 à 23h59 (heure de Paris).
    - **Lab 5** : obligatoire à rendre.
    - **Les autres labs** : optionnel, cependant, ils seront comptés commes des bonus dans la note final des labs.

Pondération de l'évaluation

- **Projet** : 30%
- **Documentation :** 20%
- **Labs** : 40%
- **Participation** : 10%

Identification

P.S. : N'oubliez pas de mentionner : votre nom - filière - année académique - école (pour les crédits sur votre Github).

Livraison (traçabilité)

Afin d'assurer la traçabilité de votre travail (**pour le projet et les Labs**), veuillez indiquer votre nom complet et celui de vos coéquipiers dans le formulaire Google suivant : Lien (TBA)

Références :

- Deploying FastAPI Microservices on Azure Kubernetes Service (AKS) using Terraform
- How to set up a Kubernetes cluster with minikube and then with Amazon EKS
- Deploying a Python FastAPI application with high availability and a database - (a series of 4 Parts)
- How to Build a Full-Stack CRUD Application with FastAPI and Streamlit, and Deploy on Kubernetes with Google Cloud Platform - (a series of 3 Parts)

Annexe 1

> Note rapide : Ce projet donne un aperçu concret des **pratiques DevOps** et de la **gestion des données** sur Kubernetes, en évitant les configurations complexes. Il représente une première expérience intégrant l'automatisation (CI/CD), le déploiement sur Kubernetes, et la manipulation de données au sein d'une application distribuée.

Annexe 2 : Proposition d'un projet avancé – Web App NLP avec BERT (Hugging Face)

Pour aller plus loin, vous pouvez envisager une **roadmap évolutive** afin de transformer votre projet initial en une **application orientée NLP**.

- **Illustration :**

1. **Backend (Python + BERT pour Analyse de Sentiments)**
    - Utiliser **Hugging Face Transformers** pour charger un modèle BERT spécialisé dans l'analyse de sentiments.
    - Créer un endpoint (ex. : POST /analyze) permettant de soumettre un texte et de retourner la polarité (positif, neutre, négatif).
    - Effectuer des opérations CRUD dans **PostgreSQL** : par exemple, stocker l'historique des textes analysés et leurs scores de sentiment.
2. **Frontend (Streamlit ou React)**

- **Streamlit** : Simple pour prototyper rapidement une interface où l'utilisateur peut saisir du texte, voir le résultat d'analyse et consulter l'historique des données.
- **React** : Plus personnalisable pour créer une interface plus riche, avec un tableau des analyses passées, des graphiques, etc.

3. **Base de Données PostgreSQL**

   - Installer PostgreSQL dans votre cluster Kubernetes (via un chart Helm existant) ou utiliser un service managé (Azure Database for PostgreSQL, Amazon RDS).
   - Mettre en place les tables de suivi (ex. : texts, analysis_results) et assurer la connexion avec le backend.

4. **Pipeline CI/CD Évolutif**

   - **CI** :
     - Tests unitaires Python (vérifier l'API d'analyse), tests sur la connexion à la base.
     - Linting et vérification que la dépendance Hugging Face est correctement installée.
   - **CD** :
     - Déploiement automatique sur Kubernetes (backend, frontend et base de données).
     - Possibilité d'utiliser des volumes persistants pour PostgreSQL.

5. **Résumé de la Roadmap**

   - **Étape 1** : Ajouter l'API d'analyse de sentiments au backend, tester localement.
   - **Étape 2** : Intégrer PostgreSQL (local ou managé) et implémenter les opérations CRUD (enregistrer les textes et les résultats).
   - **Étape 3** : Mettre à jour le pipeline CI/CD pour automatiser les tests et le déploiement.
   - **Étape 4** : Développer l'interface Streamlit ou React pour interagir avec l'API, visualiser les résultats et l'historique.
   - **Étape 5** : Déployer le tout sur votre cluster Kubernetes et présenter la solution (capture vidéo ou démonstration en direct).

Cette **Annexe 2** vous propose donc une vision élargie du projet, qui couvre un **cas d'usage NLP** plus concret, tout en s'appuyant sur les connaissances acquises lors du module CI/CD. C'est une occasion d'appréhender la **Data Science** et le **Machine Learning** dans un environnement DevOps, tout en enrichissant vos compétences en développement web et en bases de données.