Documentation du Projet DevOps en Autonomie

Table des matières

Introduction

Ce document présente la mise en œuvre d'un pipeline DevOps complet pour une application Flask simple. Le projet couvre l'ensemble de la chaîne DevOps :

- Développement d'une application Flask
- Conteneurisation avec Docker
- Mise en place d'une pipeline CI/CD avec GitHub Actions
- Déploiement sur un cluster Kubernetes géré (AKS) créé avec Terraform
- Utilisation de Helm pour la gestion des applications sur Kubernetes

Structure du projet

Le projet est organisé selon la structure suivante :

```
.
├── Dockerfile                  # Configuration pour la
création de l'image Docker
├── README.md                   # Documentation principale du
projet
├── helm.md                     # Commandes utilisées pour
installer les charts Helm
├── mon-app/                    # Dossier contenant le chart
Helm personnalisé
│   ├── Chart.yaml              # Métadonnées du chart Helm
│   ├── templates/              # Templates Kubernetes
│   │   ├── deployment.yaml     # Configuration du déploiement
│   │   ├── ingress.yaml        # Configuration de l'ingress
│   │   └── service.yaml        # Configuration du service
│   └── values.yaml             # Valeurs par défaut du chart
├── .github/workflows/          # Fichiers de workflows GitHub
Actions
│   ├── ci.yml                  # Pipeline CI
│   └── cd.yml                  # Pipeline CD
├── requirements.txt            # Dépendances Python de
l'application
└── src/                        # Code source de l'application
    └── app.py                  # Application Flask principale
```

L'application

L'application est une API web simple développée avec Flask qui renvoie un message au format "Hello <IP_adresse>, votre user-agent est : <user_agent>" lors d'une requête GET.

Caractéristiques techniques

- **Langage** : Python (compatible avec 3.13.2 et autres versions)
- **Framework** : Flask

- **Port d'écoute** : 8080
- **Dépendances** : Définies dans requirements.txt

Dockerisation

L'application est conteneurisée à l'aide d'un Dockerfile :

```
FROM python:3.9
WORKDIR /src
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY . .
CMD ["python", "/src/app.py"]
EXPOSE 8080
```

Ce Dockerfile :

- Utilise Python 3.9 comme image de base (compatible avec l'application)
- Configure `/src` comme répertoire de travail
- Installe les dépendances depuis requirements.txt
- Copie le code source
- Expose le port 8080 sur lequel l'application écoute
- Définit la commande de démarrage

L'image Docker est publiée sur Docker Hub sous le nom `clementgarcia/mon-app:latest`.

CI/CD

Intégration Continue (CI)

Le pipeline CI est implémenté avec GitHub Actions et défini dans le fichier `.github/workflows/ci.yml` :

```
name: CI Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  lint-and-test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.9'

      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          pip install flake8 bandit
          if [ -f requirements.txt ]; then pip install -r
requirements.txt; fi

      - name: Lint with flake8
        run: |
          flake8 src/ --count --select=E9,F63,F7,F82 --show-
source --statistics

      - name: Security check with bandit
        run: |
          bandit -r src/ --severity-level high

  build-and-push:
    needs: lint-and-test
    runs-on: ubuntu-latest
```

```
    steps:
      - uses: actions/checkout@v3

      - name: Login to Docker Hub
        uses: docker/login-action@v2
        with:
          username: ${{ secrets.DOCKER_HUB_USERNAME }}
          password: ${{ secrets.DOCKER_HUB_TOKEN }}

      - name: Build and push Docker image
        uses: docker/build-push-action@v4
        with:
          context: .
          push: true
          tags: clementgarcia/mon-app:latest

      - name: Run Docker Scout
        run: |
          docker scout cves clementgarcia/mon-app:latest
```

Ce pipeline CI :

1. Vérifie le code avec Flake8 pour s'assurer qu'il respecte les bonnes pratiques
2. Exécute des tests de sécurité avec Bandit, en filtrant uniquement les problèmes de haute sévérité
3. Construit l'image Docker si les tests précédents sont réussis
4. Analyse l'image avec Docker Scout pour détecter les vulnérabilités
5. Pousse l'image vers Docker Hub

Déploiement Continu (CD)

Le pipeline CD est également implémenté avec GitHub Actions et défini dans `.github/workflows/cd.yml` :

```
name: CD Pipeline

on:
  workflow_run:
    workflows: ["CI Pipeline"]
    branches: [main]
    types:
      - completed

jobs:
  deploy:
    runs-on: ubuntu-latest
    if: ${{ github.event.workflow_run.conclusion == 'success'
}}
```

```
    steps:
      - uses: actions/checkout@v3

      - name: Install Helm
        uses: azure/setup-helm@v3
        with:
          version: 'latest'

      - name: Set up kubectl
        uses: azure/setup-kubectl@v3

      - name: Get kubeconfig
        run: |
          echo "${{ secrets.KUBE_CONFIG }}" > kubeconfig
          chmod 600 kubeconfig

      - name: Deploy with Helm
        run: |
          export KUBECONFIG=./kubeconfig
          helm upgrade --install mon-app ./mon-app
```

Ce pipeline CD :

1. Se déclenche automatiquement après un CI réussi
2. Installe Helm et kubectl
3. Récupère la configuration Kubernetes depuis les secrets GitHub
4. Déploie l'application sur le cluster Kubernetes avec Helm

Kubernetes et Helm

Déploiement d'AKS avec Terraform

Le cluster AKS a été déployé avec Terraform en utilisant les commandes suivantes :

```
terraform init
terraform apply
```

Installation des charts Helm

Les charts Helm suivants ont été installés selon le fichier `helm.md` :

```
# Commandes utilisées

## Installation NGINX Ingress
helm repo add ingress-nginx https://kubernetes.github.io/
ingress-nginx
helm repo update
helm install ingress-nginx ingress-nginx/ingress-nginx

## Installation Kubecost
helm repo add kubecost https://kubecost.github.io/cost-
analyzer/
helm repo update
helm install kubecost kubecost/cost-analyzer --namespace
kubecost --create-namespace
```

Création du chart Helm personnalisé

Un chart Helm personnalisé a été créé pour déployer l'application. Les fichiers principaux sont :

Chart.yaml :

```
apiVersion: v2
name: mon-app
description: A Helm chart for mon-app Flask application
type: application
version: 0.1.0
appVersion: "1.0.0"
```

values.yaml :

```
replicaCount: 1

image:
  repository: clementgarcia/mon-app
  tag: latest
  pullPolicy: Always
```

```
service:
  type: ClusterIP
  port: 80
  targetPort: 8080

ingress:
  enabled: true
  className: nginx
  hosts:
    - host: mon-app.local
      paths:
        - path: /
          pathType: Prefix
```

Le chart comprend également des templates pour :

- Le déploiement (deployment.yaml)
- Le service (service.yaml)
- L'ingress (ingress.yaml)

Le déploiement est configuré pour exposer l'application sur le port 8080 et l'ingress est configuré pour accéder à l'application via `http://mon-app.local/`.

Problèmes rencontrés et solutions

1. Compatibilité Python

Problème : L'application est prévue pour Python 3.13.2, mais cette version n'est pas disponible dans les images Docker officielles.

Solution : Utilisation de Python 3.9, qui est compatible avec l'application.

2. Problèmes de déploiement Kubernetes

Problème : Les pods étaient initialement dans un état `ErrImagePull` et `ImagePullBackOff`.

Solution : Correction des références d'image dans le chart Helm et vérification de l'accessibilité du registry Docker.

3. Erreur d'exécution Python

Problème : Erreur `exec /usr/local/bin/python: exec format error` lors de l'exécution du conteneur.

Solution : Utilisation d'une image multi-architecture ou spécification d'une architecture compatible.

4. Configuration de l'accès à l'application

Problème : Comment accéder à l'application depuis l'extérieur du cluster.

Solution : Configuration d'un ingress et ajout d'une entrée dans le fichier hosts local pour mapper `mon-app.local` vers l'adresse IP externe de l'ingress.

Conclusion

Ce projet démontre la mise en œuvre d'un pipeline DevOps complet pour une application simple. Toutes les étapes du cycle de vie de l'application ont été automatisées :

- Développement
- Intégration continue
- Construction et analyse d'images Docker
- Déploiement sur Kubernetes

Le projet utilise des outils modernes comme GitHub Actions, Docker, Kubernetes, Helm et Terraform, démontrant l'application de pratiques DevOps avancées.

Annexes

Fichiers de configuration

deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: {{ include "mon-app.fullname" . }}
  labels:
    {{- include "mon-app.labels" . | nindent 4 }}
spec:
  replicas: {{ .Values.replicaCount }}
  selector:
    matchLabels:
      {{- include "mon-app.selectorLabels" . | nindent 6 }}
  template:
    metadata:
      labels:
        {{- include "mon-app.selectorLabels" . | nindent 8 }}
    spec:
      containers:
        - name: {{ .Chart.Name }}
          image: "{{ .Values.image.repository }}:
{{ .Values.image.tag }}"
          imagePullPolicy: {{ .Values.image.pullPolicy }}
          ports:
            - name: http
              containerPort: 8080
              protocol: TCP
          livenessProbe:
            httpGet:
              path: /
              port: http
          readinessProbe:
            httpGet:
              path: /
              port: http
```

service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: {{ include "mon-app.fullname" . }}
```

```
  labels:
    {{- include "mon-app.labels" . | nindent 4 }}
spec:
  type: {{ .Values.service.type }}
  ports:
    - port: {{ .Values.service.port }}
      targetPort: {{ .Values.service.targetPort }}
      protocol: TCP
      name: http
  selector:
    {{- include "mon-app.selectorLabels" . | nindent 4 }}
```

ingress.yaml

```
{{- if .Values.ingress.enabled -}}
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {{ include "mon-app.fullname" . }}
  labels:
    {{- include "mon-app.labels" . | nindent 4 }}
spec:
  ingressClassName: {{ .Values.ingress.className }}
  rules:
    {{- range .Values.ingress.hosts }}
    - host: {{ .host | quote }}
      http:
        paths:
          {{- range .paths }}
          - path: {{ .path }}
            pathType: {{ .pathType }}
            backend:
              service:
                name: {{ include "mon-app.fullname" $ }}
                port:
                  number: {{ $.Values.service.port }}
          {{- end }}
    {{- end }}
{{- end }}
```

Liens repo :

https://github.com/M1ck3y-ru/dev_auto