TP 1 - Déploiement Automatisé avec Docker et GitHub Actions

Pipeline CI/CD avec Conteneurisation

Prérequis : Compte GitHub, Docker installé, Git installé

Objectifs du TP

À l'issue de ce TP, vous serez capable de :

- Conteneuriser une application web avec Docker
- Créer un pipeline CI/CD avec GitHub Actions
- Construire et publier des images Docker automatiquement
- Déployer des containers sur GitHub Container Registry (GHCR)
- Automatiser le versioning des images

Prérequis Techniques

Avant de commencer, assurez-vous d'avoir :

- Un compte GitHub
- Docker Desktop installé et fonctionnel
- Git installé sur votre machine
- Un éditeur de code (VS Code recommandé)
- Connaissances de base en Docker et HTML/CSS/JavaScript

Vérifier l'installation Docker

```
docker --version
docker run hello-world
```

Architecture du Projet Final

```
devops-tp-docker-[votre-nom]/
├── .github/
│   └── workflows/
│       └── docker-deploy.yml     # Workflow CI/CD
├── src/
│   ├── index.html                # Page principale
│   ├── style.css                 # Styles
│   └── app.js                    # JavaScript
├── nginx/
│   └── nginx.conf                # Configuration Nginx
├── Dockerfile                    # Instructions Docker
├── .dockerignore                 # Fichiers à exclure
└── README.md                     # Documentation
```

Étape 1 : Création du Projet

1.1 Créer un nouveau repository GitHub

1. Aller sur GitHub.com et se connecter
2. Cliquer sur "New repository"
3. Configurer :
   - Nom : devops-tp-docker-[votre-nom]
   - Visibilité : Public
   - Ajouter un README
4. Cliquer sur "Create repository"

1.2 Cloner localement

```
git clone https://github.com/[username]/devops-tp-docker-[votre-nom]
cd devops-tp-docker-[votre-nom]
```

1.3 Créer la structure

```
mkdir -p src nginx
```

Étape 2 : Créer l'Application Web

2.1 Fichier src/index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>TP DevOps Docker</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <header>
            <h1>TP DevOps avec Docker</h1>
            <p class="subtitle">Pipeline CI/CD et Conteneurisation</p>
            <div class="version">Version: <span id="version">1.0.0</span></div>
        </header>

        <main>
            <section class="info-box">
                <h2>Informations du Container</h2>
                <p><strong>Status :</strong> <span id="status">En cours d'exécution</s
                <p><strong>Image :</strong> <span id="image">ghcr.io/[username]/devops
                <p><strong>Dernière MAJ :</strong> <span id="timestamp"></span></p>
            </section>

            <section class="action-box">
                <h2>Test de Fonctionnalité</h2>
                <button onclick="testContainer()" class="btn-primary">Tester le Contai
                <div id="result" class="result-box"></div>
            </section>

            <section class="docker-info">
                <h2>Informations Docker</h2>
                <div class="tech-stack">
                    <div class="tech-item">
                        <strong>Base Image</strong>
                        <p>nginx:alpine</p>
                    </div>
                    <div class="tech-item">
                        <strong>Registry</strong>
                        <p>GitHub Container Registry</p>
                    </div>
```

```
                    <div class="tech-item">
                        <strong>CI/CD</strong>
                        <p>GitHub Actions</p>
                    </div>
                </div>
            </section>
        </main>

        <footer>
            <p>Container ID: <span id="container-id">N/A</span></p>
        </footer>
    </div>

    <script src="app.js"></script>
</body>
</html>
```

2.2 Fichier src/style.css

```css
* {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
}

body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
    min-height: 100vh;
    padding: 20px;
}

.container {
    max-width: 900px;
    margin: 0 auto;
    background: white;
    border-radius: 10px;
    box-shadow: 0 10px 40px rgba(0,0,0,0.2);
    overflow: hidden;
}

header {
    background: #1E3A8A;
    color: white;
    padding: 40px;
    text-align: center;
    position: relative;
}

header h1 {
    font-size: 2.5em;
    margin-bottom: 10px;
}

.subtitle {
    font-size: 1.2em;
    opacity: 0.9;
}

.version {
    position: absolute;
    top: 20px;
    right: 20px;
```

```
    background: rgba(255,255,255,0.2);
    padding: 8px 15px;
    border-radius: 20px;
    font-size: 0.9em;
}

main {
    padding: 40px;
}

.info-box, .action-box, .docker-info {
    background: #f8f9fa;
    padding: 25px;
    border-radius: 8px;
    margin-bottom: 20px;
    border-left: 4px solid #1E3A8A;
}

.info-box h2, .action-box h2, .docker-info h2 {
    color: #1E3A8A;
    margin-bottom: 15px;
}

.info-box p, .action-box p {
    margin: 10px 0;
    font-size: 1.1em;
}

#status {
    color: #16A34A;
    font-weight: bold;
}

#image {
    font-family: 'Courier New', monospace;
    background: #e9ecef;
    padding: 2px 8px;
    border-radius: 4px;
}

.btn-primary {
    background: #1E3A8A;
    color: white;
    border: none;
    padding: 12px 30px;
    font-size: 1.1em;
```

```
    border-radius: 5px;
    cursor: pointer;
    transition: all 0.3s ease;
}

.btn-primary:hover {
    background: #2563EB;
    transform: translateY(-2px);
    box-shadow: 0 5px 15px rgba(0,0,0,0.2);
}

.result-box {
    margin-top: 20px;
    padding: 15px;
    border-radius: 5px;
    display: none;
    font-family: 'Courier New', monospace;
}

.result-box.success {
    display: block;
    background: #DCFCE7;
    border: 1px solid #16A34A;
    color: #166534;
}

.tech-stack {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
    gap: 20px;
    margin-top: 15px;
}

.tech-item {
    background: white;
    padding: 15px;
    border-radius: 5px;
    box-shadow: 0 2px 5px rgba(0,0,0,0.1);
}

.tech-item strong {
    display: block;
    color: #1E3A8A;
    margin-bottom: 5px;
}
```

```css
.tech-item p {
    color: #666;
    font-size: 0.95em;
}

footer {
    background: #f8f9fa;
    padding: 20px;
    text-align: center;
    color: #666;
    border-top: 1px solid #dee2e6;
}

#container-id {
    font-family: 'Courier New', monospace;
    color: #1E3A8A;
}
```

2.3 Fichier src/app.js

```
function updateTimestamp() {
    const now = new Date();
    const timestamp = now.toLocaleString('fr-FR', {
        year: 'numeric',
        month: 'long',
        day: 'numeric',
        hour: '2-digit',
        minute: '2-digit',
        second: '2-digit'
    });
    document.getElementById('timestamp').textContent = timestamp;
}

function generateContainerId() {
    const chars = '0123456789abcdef';
    let id = '';
    for (let i = 0; i < 12; i++) {
        id += chars[Math.floor(Math.random() * chars.length)];
    }
    return id;
}

function testContainer() {
    const resultBox = document.getElementById('result');
    const statusElement = document.getElementById('status');

    statusElement.textContent = 'Test en cours...';
    statusElement.style.color = '#EA580C';

    setTimeout(() => {
        statusElement.textContent = 'Container opérationnel';
        statusElement.style.color = '#16A34A';

        resultBox.innerHTML = `
            <strong>Test du Container Réussi</strong><br><br>
            docker ps<br>
            CONTAINER ID   IMAGE                                          STATUS<br>
            ${document.getElementById('container-id').textContent}      ghcr.io/user/d
            - Serveur Nginx : OK<br>
            - Application Web : OK<br>
            - Port 80 : LISTENING<br>
            - Health Check : PASSED
        `;
        resultBox.className = 'result-box success';
```

```
    }, 1500);
}

document.addEventListener('DOMContentLoaded', function() {
    updateTimestamp();
    setInterval(updateTimestamp, 1000);

    const containerId = generateContainerId();
    document.getElementById('container-id').textContent = containerId;

    document.getElementById('status').textContent = 'Container opérationnel';
});
```

Étape 3 : Configuration Nginx

3.1 Fichier nginx/nginx.conf

```
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # Configuration pour SPA
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Headers de sécurité
    add_header X-Frame-Options "SAMEORIGIN" always;
    add_header X-Content-Type-Options "nosniff" always;
    add_header X-XSS-Protection "1; mode=block" always;

    # Compression
    gzip on;
    gzip_types text/plain text/css application/json application/javascript text/xml ap

    # Cache pour les assets statiques
    location ~* \.(jpg|jpeg|png|gif|ico|css|js)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }
}
```

Étape 4 : Créer le Dockerfile

4.1 Fichier Dockerfile

```
FROM nginx:alpine

# Métadonnées
LABEL maintainer="TP DevOps"
LABEL description="Application DevOps containerisée avec Nginx"

# Copier la configuration Nginx
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf

# Copier les fichiers de l'application
COPY src/ /usr/share/nginx/html/

# Exposer le port 80
EXPOSE 80

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3     CMD wget --q

# Commande de démarrage
CMD ["nginx", "-g", "daemon off;"]
```

4.2 Fichier .dockerignore

```
.git
.github
README.md
.gitignore
*.md
.DS_Store
```

Étape 5 : Tester Localement avec Docker

5.1 Construire l'image

```
docker build -t devops-tp-docker:latest .
```

5.2 Exécuter le container

```
docker run -d -p 8080:80 --name devops-tp devops-tp-docker:latest
```

5.3 Tester l'application

Ouvrir le navigateur : `http://localhost:8080`

5.4 Vérifier les logs

```
docker logs devops-tp
```

5.5 Inspecter le container

```
docker ps
docker inspect devops-tp
```

5.6 Arrêter et supprimer

```
docker stop devops-tp
docker rm devops-tp
```

Étape 6 : Configuration GitHub Actions

6.1 Créer le workflow

```
mkdir -p .github/workflows
```

6.2 Fichier .github/workflows/docker-deploy.yml

```yaml
name: Build and Push Docker Image

on:
  push:
    branches: [ main ]
    tags:
      - 'v*'
  pull_request:
    branches: [ main ]

env:
  REGISTRY: ghcr.io
  IMAGE_NAME: ${{ github.repository }}

jobs:
  build-and-push:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      packages: write

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Set up Docker Buildx
        uses: docker/setup-buildx-action@v2

      - name: Log in to Container Registry
        uses: docker/login-action@v2
        with:
          registry: ${{ env.REGISTRY }}
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Extract metadata
        id: meta
        uses: docker/metadata-action@v4
        with:
          images: ${{ env.REGISTRY }}/${{ env.IMAGE_NAME }}
          tags: |
            type=ref,event=branch
            type=ref,event=pr
            type=semver,pattern={{version}}
```

```
            type=semver,pattern={{major}}.{{minor}}
            type=sha,prefix={{branch}}-

      - name: Build and push Docker image
        uses: docker/build-push-action@v4
        with:
          context: .
          push: true
          tags: ${{ steps.meta.outputs.tags }}
          labels: ${{ steps.meta.outputs.labels }}
          cache-from: type=gha
          cache-to: type=gha,mode=max

      - name: Image digest
        run: echo ${{ steps.meta.outputs.digest }}
```

Étape 7 : Premier Déploiement

7.1 Commit et push

```
git add .
git commit -m "Initial commit: Dockerized application with CI/CD"
git push origin main
```

7.2 Vérifier le workflow

1. Aller dans l'onglet **Actions**
2. Voir le workflow "Build and Push Docker Image"
3. Attendre la fin de l'exécution

7.3 Vérifier l'image publiée

1. Aller sur votre profil GitHub
2. Cliquer sur **Packages**
3. Voir votre image Docker publiée sur GHCR

Étape 8 : Utiliser l'Image Publiée

8.1 Rendre le package public

1. Aller dans votre Package sur GitHub
2. Cliquer sur **Package settings**
3. Changer la visibilité en **Public**

8.2 Pull et run depuis GHCR

```
docker pull ghcr.io/[username]/devops-tp-docker-[nom]:main
docker run -d -p 8080:80 ghcr.io/[username]/devops-tp-docker-[nom]:main
```

8.3 Tester

Ouvrir `http://localhost:8080`

Étape 9 : Versioning avec Tags

9.1 Créer un tag Git

```
git tag -a v1.0.0 -m "Version 1.0.0"
git push origin v1.0.0
```

9.2 Observer le workflow

Le workflow se déclenche automatiquement et crée :

- Image avec tag `v1.0.0`
- Image avec tag `1.0`
- Image avec tag `latest`

9.3 Pull une version spécifique

```
docker pull ghcr.io/[username]/devops-tp-docker-[nom]:v1.0.0
```

Étape 10 : Optimisation de l'Image

10.1 Vérifier la taille de l'image

```
docker images | grep devops-tp-docker
```

10.2 Analyser les layers

```
docker history devops-tp-docker:latest
```

10.3 Dockerfile optimisé (optionnel)

```
FROM nginx:alpine

# Métadonnées
LABEL maintainer="TP DevOps"
LABEL description="Application DevOps optimisée"

# Copier en une seule couche
COPY nginx/nginx.conf /etc/nginx/conf.d/default.conf
COPY src/ /usr/share/nginx/html/

# Supprimer les fichiers inutiles
RUN rm -rf /usr/share/nginx/html/*.md

EXPOSE 80

HEALTHCHECK --interval=30s --timeout=3s     CMD wget -q --spider http://localhost/ || 

CMD ["nginx", "-g", "daemon off;"]
```

Résultats Attendus

À la fin de ce TP, vous devez avoir :

Infrastructure Docker :

- Application web conteneurisée avec Nginx
- Dockerfile optimisé et fonctionnel

- Image Docker publiée sur GHCR
- Versioning automatique des images

Pipeline CI/CD :

- Build automatique à chaque push
- Push automatique vers GHCR
- Tags Git créent des versions d'images
- Cache optimisé pour build rapides

Compétences acquises :

- Conteneurisation d'applications web
- Configuration Nginx dans Docker
- GitHub Container Registry
- GitHub Actions pour Docker
- Bonnes pratiques Docker

Commandes Docker Utiles

```
# Lister les images
docker images

# Lister les containers
docker ps -a

# Logs d'un container
docker logs [container-id]

# Exécuter une commande dans un container
docker exec -it [container-id] sh

# Arrêter tous les containers
docker stop $(docker ps -q)

# Supprimer tous les containers
docker rm $(docker ps -aq)

# Nettoyer les images non utilisées
docker image prune -a

# Voir l'utilisation disque
docker system df
```
