TP 1 : Automatisation du SDLC avec GitHub Actions, C# et Docker

Objectifs pédagogiques

À la fin de ce TP, l'étudiant sera capable de :

- Créer un dépôt Git pour un projet C#.
- Mettre en place un pipeline CI/CD avec GitHub Actions.
- Intégrer des tests unitaires et un outil d'analyse statique.
- Construire une image Docker et la publier localement.
- Simuler un déploiement sécurisé avec gestion des secrets.

Pré-requis (à installer)

1. **Visual Studio 2022 Community** :
   ➜ https://visualstudio.microsoft.com/fr/vs/community
   Cocher ".NET desktop development" à l'installation.

2. **.NET SDK 8.0** :
   ➜ https://dotnet.microsoft.com/fr-fr/download/dotnet/8.0

3. **Docker Desktop** (compte DockerHub facultatif) :
   ➜ https://www.docker.com/products/docker-desktop/

4. **Compte GitHub** : pour utiliser GitHub Actions

Étape 1 : Création du projet

1. Ouvrir Visual Studio > Nouveau projet > **Application console (.NET Core)**

2. Nommer le projet : `ModernSoftwareApp`

3. Ajouter une classe simple avec une méthode à tester :

```
public class Calculator
{
    public int Add(int a, int b) => a + b;
}
```

4. Créer un projet de test :

    - Clic droit sur la solution > Ajouter > Nouveau projet > **xUnit Test Project**
    - Ajouter la référence vers le projet principal
    - Écrire un test :

```
public class CalculatorTests
{
    [Fact]
    public void Add_ShouldReturnCorrectSum()
    {
        var calc = new Calculator();
        Assert.Equal(5, calc.Add(2, 3));
    }
}
```

Étape 2 : Git + GitHub

1. Initialiser Git dans le dossier du projet :

```
git init
```

2. Créer un dépôt GitHub et le lier :

```
git remote add origin
https://github.com/votre-utilisateur/ModernSoftwareApp.git
git add .
git commit -m "Init projet C#"
git push -u origin main
```

Étape 3 : CI avec GitHub Actions

1. Dans le dépôt GitHub, créer un fichier `.github/workflows/dotnet.yml` :

```
name: Build and Test

on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest

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

    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: '8.0.x'

    - name: Restore dependencies
      run: dotnet restore

    - name: Build
      run: dotnet build --no-restore

    - name: Run Tests
      run: dotnet test --no-build --verbosity normal
```

2. Faire un commit/push → GitHub va exécuter automatiquement les tests.

Étape 4 : Analyse de qualité

1. Ajouter **dotnet-format** :

```
dotnet tool install -g dotnet-format
```

2. Ajouter une étape dans GitHub Actions pour formatter le code :

```
 - name: Check code formatting
    run: dotnet format --verify-no-changes
```

Étape 5 : Dockerisation

1. Créer un `Dockerfile` :

```
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
WORKDIR /app

FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
WORKDIR /src
COPY . .
RUN dotnet publish -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=build /app/publish .
ENTRYPOINT ["dotnet", "ModernSoftwareApp.dll"]
```

2. Construire et exécuter localement :

```
docker build -t modernsoftwareapp .
docker run modernsoftwareapp
```

Étape 6 : Simulation de secrets

1. Créer un secret GitHub dans `Settings > Secrets > Actions` :
    - `MY_SECRET=super_secret_value`
2. L'utiliser dans le pipeline :

```
- name: Read secret
  run: echo "Secret value is ${{ secrets.MY_SECRET }}"
```

Étape 7 : Simuler un déploiement

1. Ajouter un script bash de déploiement simulé (`deploy.sh`) :

```
#!/bin/bash
echo "Déploiement de l'app..."
```

2. L'ajouter dans le pipeline :

```
- name: Simulated deploy
  run: bash ./deploy.sh
```

Étape 8 :

1. Intégrer SonarQube localement

- Installer SonarQube avec Docker :

docker run -d --name sonarqube -p 9000:9000 sonarqube

- Scanner avec SonarScanner for .NET

2. Intégrer un contrôle de qualité avec dotnet format ou StyleCop

- Ajouter dans le pipeline un job `dotnet format`
- Ajouter un outil de linting C# (StyleCop.Analyzers)

3. Déploiement dans GitHub Pages avec un frontend (React, Blazor, etc.)

- Déployer un frontend via GitHub Actions (bonus)

4. Docker + GitHub Container Registry

- Construire une image Docker et la publier sur ghcr.io
- Utiliser les secrets `GHCR_TOKEN` pour push

5. Scénario AWS (si compte éducation AWS Educate disponible)

- Créer un bucket S3 pour héberger l'API statique ou front React
- Déployer automatiquement avec `aws s3 sync`

À rendre (livrable)

- Lien GitHub du dépôt
- Capture d'écran de l'exécution du pipeline
- Capture d'écran de l'image Docker exécutée
- Explication courte : "Quel est le rôle du CI/CD et des tests dans un projet réel ?"