Tutorial completo: CI/CD para .NET con NUnit y GitHub Actions

---

1️⃣ Crear la estructura del proyecto

Vamos a hacer un proyecto con **librería, aplicación principal y pruebas unitarias con NUnit**.

```
MiProyecto/
├── .github/
│   └── workflows/
│       └── ci-cd.yml
├── MiProyecto.sln
├── MiProyectoApp/
│   ├── MiProyectoApp.csproj
│   └── Program.cs
├── MiProyectoLib/
│   ├── MiProyectoLib.csproj
│   └── Calculadora.cs
├── MiProyectoTests/
│   ├── MiProyectoTests.csproj
│   └── CalculadoraTests.cs
└── README.md
```

2️⃣ Crear la solución y los proyectos

2.1 Crear la solución

```
dotnet new sln -n MiProyecto
```

2.2 Crear la librería de clases

```
dotnet new classlib -n MiProyectoLib
dotnet sln add MiProyectoLib/MiProyectoLib.csproj
```

2.3 Crear la aplicación principal

```
dotnet new console -n MiProyectoApp
dotnet sln add MiProyectoApp/MiProyectoApp.csproj
```

dotnet add MiProyectoApp/MiProyectoApp.csproj reference MiProyectoLib/MiProyectoLib.csproj

2.4 Crear proyecto de pruebas con NUnit

dotnet new nunit -n MiProyectoTests

dotnet sln add MiProyectoTests/MiProyectoTests.csproj

dotnet add MiProyectoTests/MiProyectoTests.csproj reference MiProyectoLib/MiProyectoLib.csproj

---

3️⃣ Código del proyecto

3.1 Librería: MiProyectoLib/Calculadora.cs

```
namespace MiProyectoLib
{
  public class Calculadora
  {
    public int Sumar(int a, int b) => a + b;
    public int Restar(int a, int b) => a - b;
    public int Multiplicar(int a, int b) => a * b;
    public double Dividir(int a, int b)
    {
      if (b == 0) throw new DivideByZeroException("No se puede dividir entre cero");
      return (double)a / b;
    }
  }
}
```

3.2 Aplicación principal: MiProyectoApp/Program.cs

```
using System;
using MiProyectoLib;

namespace MiProyectoApp
{
  class Program
```

```
  {
    static void Main(string[] args)
    {
      var calc = new Calculadora();

      Console.WriteLine($"Suma: {calc.Sumar(5,3)}");
      Console.WriteLine($"Resta: {calc.Restar(5,3)}");
      Console.WriteLine($"Multiplicación: {calc.Multiplicar(5,3)}");
      Console.WriteLine($"División: {calc.Dividir(5,3)}");
    }
  }
}
```

3.3 Pruebas NUnit: MiProyectoTests/CalculadoraTests.cs

```
using NUnit.Framework;
using MiProyectoLib;
using System;

namespace MiProyectoTests
{
  public class CalculadoraTests
  {
    private Calculadora calc;

    [SetUp]
    public void Setup()
    {
      calc = new Calculadora();
    }

    [Test]
```

```
    public void Suma_Correcta()
    {
        Assert.AreEqual(8, calc.Sumar(5,3));
    }

    [Test]
    public void Resta_Correcta()
    {
        Assert.AreEqual(2, calc.Restar(5,3));
    }

    [Test]
    public void Multiplicacion_Correcta()
    {
        Assert.AreEqual(15, calc.Multiplicar(5,3));
    }

    [Test]
    public void Division_Correcta()
    {
        Assert.AreEqual(1.6666666666666667, calc.Dividir(5,3), 0.0001);
    }

    [Test]
    public void DivisionPorCero_LanzaExcepcion()
    {
        Assert.Throws<DivideByZeroException>(() => calc.Dividir(5,0));
    }
  }
}
```

4️⃣ Crear el flujo CI/CD en GitHub Actions

Archivo: .github/workflows/ci-cd.yml

```
name: .NET CI/CD Pipeline

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

jobs:
  build-and-test:
    name: Build and Test
    runs-on: ubuntu-latest

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

      - name: Setup .NET
        uses: actions/setup-dotnet@v3
        with:
          dotnet-version: '7.0.x'

      - name: Restore dependencies
        run: dotnet restore

      - name: Build solution
        run: dotnet build --configuration Release --no-restore

      - name: Run tests
```

```
    run: dotnet test MiProyectoTests/MiProyectoTests.csproj --no-build --verbosity normal

publish:
  name: Publish Application
  runs-on: ubuntu-latest
  needs: build-and-test

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

    - name: Setup .NET
      uses: actions/setup-dotnet@v3
      with:
        dotnet-version: '7.0.x'

    - name: Restore dependencies
      run: dotnet restore

    - name: Build project
      run: dotnet build MiProyectoApp/MiProyectoApp.csproj --configuration Release

    - name: Publish project
      run: dotnet publish MiProyectoApp/MiProyectoApp.csproj -c Release -o ./publish

    - name: Upload artifact
      uses: actions/upload-artifact@v3
      with:
        name: MiProyectoApp
        path: ./publish
```

5️⃣ Explicación detallada de cada paso

Job: build-and-test

1. **Checkout repository** → Clona el código del repositorio.
2. **Setup .NET** → Instala .NET 7 en el runner.
3. **Restore dependencies** → Descarga paquetes NuGet necesarios.
4. **Build solution** → Compila todos los proyectos en Release.
5. **Run tests** → Ejecuta todas las pruebas NUnit en MiProyectoTests.

Job: publish

1. **needs: build-and-test** → Solo se ejecuta si las pruebas pasan.
2. **Restore + Build** → Prepara la app para publicar.
3. **Publish** → Genera la carpeta ./publish lista para deploy.
4. **Upload artifact** → Permite descargar la app desde GitHub Actions.

6️⃣ Ejecutar el flujo CI/CD

1. Haz commit y push de todo tu proyecto a GitHub.
2. GitHub Actions detectará cambios en la rama main.
3. Verás dos jobs:
    - **Build and Test** → compila y ejecuta pruebas.
    - **Publish Application** → genera la versión lista para deploy.
4. Puedes descargar la app publicada desde el **artifact** en la interfaz de Actions.