**GitHub Actions Workflow for .NET Framework.**

This workflow file provides automated deployment for your .NET Framework application. Here's how it works:

1. It triggers on pushes and pull requests to main, test, and dev branches

2. Sets up the necessary tools (MSBuild and NuGet)

3. Builds your solution in Release configuration

4. Deploys to different folders based on the branch:

- main → C:\Deployments\Production

- test → C:\Deployments\Test

- dev → C:\Deployments\Development

To use this workflow:

1. Save it as `.github/workflows/deploy.yml` in your repository

2. Replace `YourSolution.sln` with your actual solution file name

3. Update the source path in the deployment step to match your project's output directory

4. Ensure the Windows runner has necessary permissions to write to these locations

You may also want to consider:

- Adding environment variables for the deployment paths

- Including additional steps like running tests

- Adding approval workflows for production deployments

- Including cleanup steps for old deployments

Would you like me to modify any part of this workflow?

**Key changes made for .NET 8.0:**

See deploy.yml to see how the test is implemented in the workflow.

1. Replaced MSBuild/NuGet setup with .NET SDK setup

2. Updated to use `dotnet` CLI commands instead of MSBuild

3. Added a specific publish step before deployment

4. Simplified the build process using built-in .NET CLI features

5. Added environment variable for .NET version

The main differences from the .NET Framework version are:

- Uses `actions/setup-dotnet` instead of `setup-msbuild` and `setup-nuget`

- Uses `dotnet restore/build/publish` commands instead of `nuget restore` and `msbuild`

- Includes a dedicated publish step that creates ready-to-deploy artifacts

- Deployment copies from the publish directory instead of the bin directory

The workflow will now:

1. Install .NET 8.0 SDK

2. Restore NuGet packages

3. Build the solution

4. Publish the application

5. Deploy to the appropriate directory based on the branch

**Key changes made to add testing:**

See deploy-test.txt to see how the test is implemented in the workflow.

1. Added a test step using `dotnet test` that:

- Runs all tests in the solution

- Collects code coverage data

- Outputs results to a coverage directory

2. Added test reporting features:

- Uploads test results as artifacts using `actions/upload-artifact`

- Generates a test report using `dorny/test-reporter`

- Preserves test results even if tests fail (`if: always()`)

3. Added conditional deployments:

- Publication and deployment only occur if tests pass (`if: success()`)

- Test results are uploaded regardless of test outcome

You can view test results and code coverage in the GitHub Actions UI after each run.

Would you like me to add any specific test configurations or additional test reporting features? For example, we could:

1. Add specific test filters

2. Include code coverage thresholds

3. Add parallel test execution

4. Include additional test reporting formats

**Show .NET 8.0 version**

See deploy-NETversion.txt to see how to determine the version being used.

I'll show you a couple of ways to determine the .NET 8.0 version in your GitHub Actions workflow.

I've added a new "Check .NET version" step that will show you:

1. All installed .NET SDKs

2. All installed .NET Runtimes

3. The current active .NET version

Additionally, you can:

1. Specify an exact version instead of using '8.0.x':

```yaml

DOTNET\_VERSION: '8.0.100' # Or whatever specific version you need

```

2. Check the version in your application code:

```csharp

Console.WriteLine(System.Runtime.InteropServices.RuntimeInformation.FrameworkDescription);

```

3. Add version information to your published artifacts by including it in your project file:

```xml

<PropertyGroup>

<VersionPrefix>8.0.0</VersionPrefix>

<VersionSuffix>$(VersionSuffix)</VersionSuffix>

</PropertyGroup>

```

**Can I use ‘8.0.x’?**

Yes, using '8.0.x' in your workflow is a good practice!

When you specify '8.0.x' in the DOTNET\_VERSION environment variable, GitHub Actions will automatically use the latest patch version of .NET 8.0. This means:

1. You'll automatically get the latest security updates and bug fixes within the 8.0 series

2. Your workflow stays compatible with the 8.0 release you're targeting

3. You don't have to update your workflow file every time a new patch version is released

This is what we have in the current workflow:

```yaml

env:

DOTNET\_VERSION: '8.0.x'

```

If you ever need to lock to a specific version (though this is usually not necessary), you could use an exact version like '8.0.100', but '8.0.x' is generally the recommended approach for most scenarios as it provides a good balance between stability and security.