JUL 29, 2023

# Should you consider migrating CI/CD pipelines from Azure DevOps to GitHub Actions?

As engineering teams look for ways to streamline their processes and enhance their development workflow, the adoption of CI/CD (Continuous Integration/Continuous Delivery) pipelines has become quite the norm. Azure DevOps was one of the first platforms to offer YAML based CI/CD templates and quickly became the favorite tool amongst a lot of developers. After Microsoft's acquisition of Github, they quickly introduced the much-desired enterprise-grade CI/CD tooling in GitHub Actions, and it quickly rose to popularity. While Azure DevOps remains a feature-rich tool, with GitHub's continuous upgrades and versatile features, an increasing number of developers are migrating from Azure DevOps to GitHub Actions.

With the increasing popularity and capabilities of GitHub Actions (GHA), many teams are considering migrating their CI/CD pipelines from Azure DevOps (ADO) to GitHub Actions. This article aims to provide an outline of the advantages of GHA, and a general roadmap for the migration process, highlighting key considerations and steps.

## Why Migrate to GitHub Actions?

GitHub Actions has seen rapid adoption since its introduction due to its tight integration with GitHub repositories and its event-driven model. With this model, workflows can be triggered on almost any GitHub event, enabling highly flexible and customizable automation.

While both ADO and GHA are YAML using relatively similar YAML templates , certain characteristics and features may make GHA a more appealing choice for most teams or projects. Here are a few key advantages of GitHub Actions over Azure DevOps.

### Marketplace And Custom Actions

GitHub Actions marketplace is more extensive and active, with thousands of actions available for various tasks. This contrasts with Azure DevOps, where task extensions are less numerous and community contribution is not as robust.

## Ease of Integration

GitHub Actions marketplace hosts a plethora of actions, ranging from setting up different environments (like actions/setup-node for Node.js, actions/setup-python for Python) to deploying code to different cloud providers (like aws-actions/aws-s3-sync for syncing a directory to an S3 bucket). In comparison, Azure DevOps does offer an extensions marketplace, but it's not as comprehensive or user-friendly as GitHub's.

For instance, consider the action of deploying a Docker image to a Kubernetes cluster. The GitHub marketplace has an action like azure/k8s-deploy, which simplifies the process to a few lines of YAML.

```yaml
- name: Deploy to Kubernetes
  uses: azure/k8s-deploy@v1
  with:
    namespace: mynamespace
    manifests: |
      manifest1.yml
      manifest2.yml
    images: |
      myimage1:$(IMAGE1_NAME)
      myimage2:$(IMAGE2_NAME)
```

kubernetes.yaml hosted with ❤ by GitHub view raw

While Azure DevOps can achieve similar results, it usually requires a more complex series of tasks to accomplish the same goal.

## Community Contributions

GitHub Actions benefits greatly from being built on the most popular platform for open-source software. Many actions in the marketplace are contributed by the community and go beyond mere CI/CD tasks, including actions for automating the management of issues, pull requests, and project boards. For example, the actions/stale action marks issues and pull requests that have not seen activity for a specified amount of time as stale, encouraging more effective project management.

```yaml
- name: Mark stale issues and pull requests
  uses: actions/stale@v4
  with:
    days-before-stale: 60
    days-before-close: 7
```

community-actions-example.yaml hosted with ❤ by GitHub view raw

## Triggers And Event-Driven Workflows

GitHub Actions supports a broader range of event triggers. Any event that happens within a GitHub repository can be used to trigger a workflow. On the other hand, Azure DevOps primarily focuses on push and pull request events for CI/CD workflows.

### Pull Request Triggers

The workflow runs whenever a pull request is opened, synchronized, or closed.

For example, with ADO, you have to do a lot of PowerShell magic to trigger a pipeline for a pull request closure.

```yaml
- task: PowerShell@2
  inputs:
    targetType: 'inline'
    script: |
      # Call REST API and set the result to $pr
      $token = "{PAT}" # Replace it with your PAT
      $token = [System.Convert]::ToBase64String([System.Text.Encoding]::ASCII.GetByte
      $url="https://dev.azure.com/{organization}/{project}/_apis/git/pullrequests/{pu
      $head = @{ Authorization =" Basic $token" }
      $pr = Invoke-RestMethod -Uri $url -Method Get  -Headers $head -ContentType appl
      # Pass the status result to future tasks. You can also pass it to future jobs/s
      $status = $pr.status
      Write-Host "##vso[task.setvariable variable=prstatus;]$status"

- task: ... # The task you use to destroy apps
  condition: eq(variables.prstatus, 'completed')
```

pull-request-triggers-example.yaml hosted with ❤ by GitHub view raw

With GHA, there are pre-build actions, and conditions to do this quite easily.

```yaml
name: Pull Request Closed Workflow

on:
  pull_request:
    types: [closed]
```

gha-pull-request-closed-workflow-example.yaml hosted with ❤ by GitHub view raw

## Issue Comments Triggers

The workflow runs when a comment is added to an issue in the repository.

While ADO also has support for comment-based triggers, this is only supported if you are using GitHub as the source repository. ( in most instances, if you are using ADO Pipelines, you are likely to be using ADO Git repos, so this feature is actually useless).

With GHA, this is straightforward to do.

```yaml
on:
  issue_comment:
    types: [created, edited]
```

gha-comment-triggers.yaml hosted with ❤ by GitHub view raw

## Release Triggers

In Azure DevOps, this is typically done using Release Pipelines, which are designed to handle the deployment of your application. You can create a release manually, or you can configure the release pipeline to automatically create a new release when it detects a new build artifact (which could be created by a build pipeline when you create a GitHub release, for example).

```yaml
trigger: none

```

```yaml
stages:
- stage: Deploy
  jobs:
  - deployment: DeployToAzure
    displayName: Deploy to Azure
    pool:
      vmImage: 'ubuntu-latest'
    environment: 'Production'
    strategy:
      runOnce:
        deploy:
          steps:
          - script: echo Deploying!
```

ado-release-triggers.yaml hosted with ❤ by GitHub view raw

In GitHub Actions, you can trigger a workflow directly from your YAML file when a release is created by using the release event. You can further specify the types of release events that will trigger the workflow, such as created, edited, published, etc.

```yaml
on:
  release:
    types: [published, created, edited]
```

gha-release-triggers.yaml hosted with ❤ by GitHub view raw

## Parallelism And Matrix Builds

Both platforms support running jobs in parallel and matrix builds. However, GitHub Actions offers a more intuitive syntax and configuration for matrix builds.

Azure DevOps has the concept of a "Matrix Strategy" that allows you to run jobs in parallel with different configurations. Here is an example:

```yaml
jobs:
- job: Build
  strategy:
    matrix:
      linux:
        imageName: 'ubuntu-latest'
      windows:
        imageName: 'windows-latest'
  pool:
    vmImage: $(imageName)
  steps:
  - script: echo Hello, world!
```

ado-matrix-build.yaml hosted with ❤ by GitHub view raw

GitHub Actions also supports a similar feature, also known as "Matrix Strategy". However, GitHub Actions provides a more flexible approach, allowing you to combine different sets of variables. Here is an example:

```yaml
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest]
        node: [12, 14]
    steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Use Node.js ${{ matrix.node }}
      uses: actions/setup-node@v2
      with:
        node-version: ${{ matrix.node }}
    - run: npm ci
    - run: npm test
```

gha-matrix-example.yaml hosted with ❤ by GitHub view raw

## Runner Environment

GitHub Actions and Azure DevOps both offer self-hosted runners, allowing you to run workflows on your own hardware and in your own environment. Self-hosted runners can be used when you have specific requirements that aren't met by the hosted runners, like when you need a specific OS or tool that's not available on the hosted runners, or when your code needs to access resources that are only available in your local network.

However, GitHub Actions goes a step further by offering the ability to run actions in Docker containers on their hosted runners. This means that you can create a Dockerfile to specify the exact environment that your workflow needs, and this Dockerfile will be built and run on the GitHub-hosted runner.

Here's an example of how this work:

First, you create a Dockerfile that specifies the exact environment your workflow requires. For instance, if your workflow requires `Node.js` and a specific version of the AWS CLI, your Dockerfile might look like this:

```
FROM node:14

RUN curl "https://d1vvhvl2y92vvt.cloudfront.net/awscli-exe-linux-x86_64-2.1.30.zip" -o
    unzip awscliv2.zip && \
    sudo ./aws/install
```

Dockerfile hosted with ❤️ by GitHub view raw

Next, you define a custom action in the same repository as your workflow. This action uses the Dockerfile you just created. The `action.yml` for this custom action would look like this:

```
name: 'Custom Action'
description: 'Run my scripts in a custom environment'
runs:
  using: 'docker'
  image: 'Dockerfile'
```

custom-action.yaml hosted with ❤️ by GitHub view raw

Finally, you use this action in your workflow:

```
jobs:
  custom-job:
    runs-on: ubuntu-latest
    steps:
    - uses: ./.github/actions/custom-action
```

workflow.yaml hosted with ❤ by GitHub view raw

In this setup, the custom-job job will run on a GitHub-hosted runner, but the action will run in the Docker container that you've specified. This gives you the ability to specify the exact environment that your action needs, without the need to set up and manage your own self-hosted runner.

## GitHub Advanced Security

Although Microsoft now owns GitHub, and there's a lot of feature parity between the two platforms, Microsoft has announced that the Advanced Security service will not be available for ADO. GitHub's advanced security features definitely add significant value to the platform and can serve as compelling reasons to consider migration from Azure DevOps to GitHub Actions. Here are some key features:

**Code Scanning:** This feature scans your code as soon as it's pushed to GitHub, and can surface and help prevent vulnerabilities before they reach production. The tool is built on the open-source project Semmle's CodeQL, allowing you to explore your source code as a database. It's a powerful tool for identifying security vulnerabilities in many popular programming languages.

**Secret Scanning**: Formerly known as token scanning, this feature scans repositories for known secret formats to prevent fraudulent use of credentials committed accidentally. Once it finds potential secrets, it alerts the provider or the repository owner, ensuring quick action can be taken.

**Dependabot Alerts and Security Updates**: Dependabot pulls in information from the GitHub Advisory Database and other sources to identify dependencies with known vulnerabilities. It can alert you to the problem and can even automatically create a pull request to update to the fixed version, subject to your configuration.

**Private Instances**: For businesses that need to meet complex security, compliance and policy constraints, GitHub provides private instances. It is a fully managed offering from GitHub, which provides enhanced security, simplified administration, and scalable performance to serve large or regulated organizations.

These features integrate seamlessly into the developer workflow. Since everything happens within the GitHub ecosystem, there's no need to manage and maintain multiple tools or jump from one platform to another to manage your code, which saves time and reduces complexity.

## Step-by-Step Migration Guide

### Step 1: Assess Your Current Azure Pipelines

Before diving into the migration, it's important to understand your existing Azure Pipelines in detail.

- Which tasks are being performed?
- What dependencies exist between tasks?
- How is the pipeline triggered, and what conditions lead to its execution?

This analysis will provide a blueprint for recreating your pipelines in GitHub Actions.

### Step 2: Understand GitHub Actions Structure

GitHub Actions are composed of workflows that respond to events on your GitHub repository. Each workflow consists of one or more jobs, which are sets of steps that perform individual tasks. Steps can run commands or actions, which are pre-built pieces of code.

Understand the key components:

- **Workflow**: Defined in a YAML file in the **.github/workflows** directory in your repository.
- **Jobs**: Run on separate instances of the GitHub-hosted runner.
- **Steps**: The smallest unit in a job, can run commands or actions.
- **Actions**: Prebuilt pieces of code to perform common tasks.

## Step 3: Create An Equivalent GitHub Actions Workflow

Create a **.github/workflows** directory in your GitHub repository, and add a new YAML file to define your workflow. Use your analysis from Step 1 to guide the definition of jobs and steps.

Consider using prebuilt actions from the GitHub marketplace to replace custom scripts in your Azure Pipeline, but be aware of versioning and potential security concerns.

## Step 4: Handle Secrets And Environment Variables

GitHub Actions provide mechanisms to handle secrets and environment variables. Use the GitHub web interface to configure secrets at the repository or organization level.

Environment variables can be set at the workflow, job, or step level within the YAML workflow file. They can also reference secrets, providing a way to securely handle sensitive data.

## Step 5: Implement Conditional Logic

GitHub Actions supports conditional jobs and steps, using a syntax similar to Azure DevOps. Conditional statements are written in JavaScript expression syntax, providing flexibility for complex logic.

## Step 6: Testing The Migration

Before fully transitioning to GitHub Actions, make sure to thoroughly test your new workflows. This can be done by manually triggering the workflow or pushing changes to your repository that would trigger the workflow.

## Migrating Azure functions deployment from ADO to GHA

This example is a simple comparison of what an Azure Function deployment using Azure DevOps and GitHub Actions.

### Azure DevOps YAML pipeline for Azure Function Deployment

```yaml
trigger:
- main

variables:
  # Azure Resource Manager connection created during pipeline creation
  azureSubscription: 'my-azure-subscription'

  # Function app name
  functionAppName: 'my-function-app'

  # Agent VM image name
  vmImageName: 'ubuntu-latest'

stages:
- stage: Build
  displayName: Build stage

  jobs:
  - job: Build
    displayName: Build
    pool:
      vmImage: $(vmImageName)

    steps:
    - task: DotNetCoreCLI@2
      displayName: Build
      inputs:
        command: build
        projects: '**/*.csproj'
        arguments: --output $(System.DefaultWorkingDirectory)/publish_output

- stage: Deploy
  displayName: Deploy stage

  jobs:
  - deployment: Deploy
    displayName: Deploy
    environment: 'production'
    pool:
      vmImage: $(vmImageName)

    strategy:
      runOnce:
        deploy:
          steps:
          - task: AzureFunctionApp@1
            displayName: 'Azure functions app deploy'
            inputs:
              azureSubscription: '$(azureSubscription)'
              appType: functionApp
              appName: $(functionAppName)
              package: $(System.DefaultWorkingDirectory)/publish_output
```

ado-function-app-deployment.yaml hosted with ❤️ by GitHub view raw

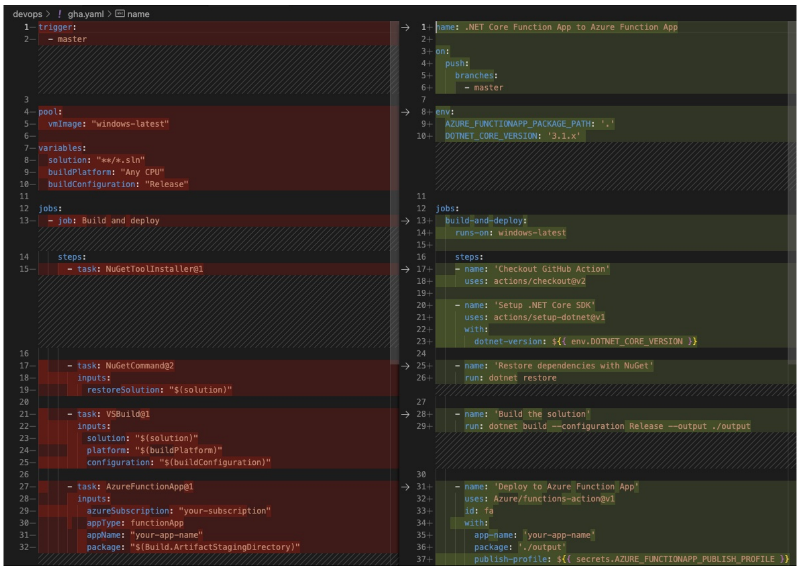
## GitHub Actions Workflow for Azure Function deployment

```yaml
name: Build and Deploy to Azure Functions

on:
  push:
    branches:
      - main

env:
  AZURE_FUNCTIONAPP_NAME: my-function-app # set this to your application's name
  AZURE_FUNCTIONAPP_PACKAGE_PATH: './publish_output' # set this to the path to your a
  DOTNET_VERSION: '3.1.x' # set this to the dotnet version to use
```

```yaml

jobs:
  build-and-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: 'Checkout GitHub Action'
      uses: actions/checkout@main

    - name: Setup DotNet ${{ env.DOTNET_VERSION }} Environment
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: ${{ env.DOTNET_VERSION }}

    - name: Build with dotnet
      run: dotnet build --configuration Release --output ${{ env.AZURE_FUNCTIONAPP_PA

    - name: 'Login via Azure CLI'
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: 'Run Azure Functions Action'
      uses: Azure/functions-action@v1
      id: fa
      with:
        app-name: ${{ env.AZURE_FUNCTIONAPP_NAME }}
        package: ${{ env.AZURE_FUNCTIONAPP_PACKAGE_PATH }}

    - name: 'Logout of Azure CLI'
      run: az logout
```

gha-function-app-deployment.yaml hosted with ❤ by GitHub view raw

In the GitHub Actions workflow, we're also doing two main tasks: Building the app and deploying it. The build is done with a run command that directly uses the dotnet CLI, and the deployment is done with the Azure/functions-action@v1 action.



```yaml
trigger:
  - master

pool:
  vmImage: "windows-latest"

variables:
  solution: "**/*.sln"
  buildPlatform: "Any CPU"
  buildConfiguration: "Release"

jobs:
  - job: Build and deploy

    steps:
      - task: NuGetToolInstaller@1

      - task: NuGetCommand@2
        inputs:
          restoreSolution: "$(solution)"

      - task: VSBuild@1
        inputs:
          solution: "$(solution)"
          platform: "$(buildPlatform)"
          configuration: "$(buildConfiguration)"

      - task: AzureFunctionApp@1
        inputs:
          azureSubscription: "your-subscription"
          appType: functionApp
          appName: "your-app-name"
          package: "$(Build.ArtifactStagingDirectory)"
```

```yaml
name: .NET Core Function App to Azure Function App

on:
  push:
    branches:
      - master

env:
  AZURE_FUNCTIONAPP_PACKAGE_PATH: '.'
  DOTNET_CORE_VERSION: '3.1.x'

jobs:
  build-and-deploy:
    runs-on: windows-latest

    steps:
    - name: 'Checkout GitHub Action'
      uses: actions/checkout@v2

    - name: 'Setup .NET Core SDK'
      uses: actions/setup-dotnet@v1
      with:
        dotnet-version: ${{ env.DOTNET_CORE_VERSION }}

    - name: 'Restore dependencies with NuGet'
      run: dotnet restore

    - name: 'Build the solution'
      run: dotnet build --configuration Release --output ./output

    - name: 'Deploy to Azure Function App'
      uses: Azure/functions-action@v1
      id: fa
      with:
        app-name: 'your-app-name'
        package: './output'
        publish-profile: ${{ secrets.AZURE_FUNCTIONAPP_PUBLISH_PROFILE }}
```

Side-by-side comparison of ADO ( left) and GHA (right) deploying an Azure Functions App.

As you can see, both the Azure DevOps pipeline and the GitHub Actions workflow achieve the same goal, and the syntax and structure are quite similar, so the migration process is relatively simpler.

## Conclusion

GitHub Actions offer significantly better DevOps/DevSecOps tooling. Migrating from Azure DevOps to GitHub Actions involves translating Azure Pipelines YAML to GitHub Actions YAML, while mapping tasks to actions. While the sample provided here is specific to Azure Function App deployment, the process is similar for other applications.

It's worth noting that the convenience and increasing capabilities offered by GitHub Actions make it a compelling option. However, the decision should be based on the specific needs and context of your projects.

Feel free to reach out if you need help with preparing a migration strategy to GitHub Actions.

*Further readings:*

Understanding GitHub Actions

Migration from Azure Pipelines to GitHub Actions

## Interested in hearing more?

## Lets connect.

## Thoughts, stories and ideas.