Migrating from Azure Pipelines to GitHub Actions

In this article

GitHub Actions and Azure Pipelines share several configuration similarities, which makes migrating to GitHub Actions relatively straightforward.

Introduction

Azure Pipelines and GitHub Actions both allow you to create workflows that automatically build, test, publish, release, and deploy code. Azure Pipelines and GitHub Actions share some similarities in workflow configuration:

- Workflow configuration files are written in YAML and are stored in the code's repository.
- Workflows include one or more jobs.
- Jobs include one or more steps or individual commands.
- Steps or tasks can be reused and shared with the community.

For more information, see "Understanding GitHub Actions."

Key differences

When migrating from Azure Pipelines, consider the following differences:

- Azure Pipelines supports a legacy *classic editor*, which lets you define your CI configuration in a GUI editor instead of creating the pipeline definition in a YAML file. GitHub Actions uses YAML files to define workflows and does not support a graphical editor.
- Azure Pipelines allows you to omit some structure in job definitions. For example, if you only have a single job, you don't need to define the job and only need to define its steps. GitHub Actions requires explicit configuration, and YAML structure cannot be omitted.
- Azure Pipelines supports *stages* defined in the YAML file, which can be used to create deployment workflows. GitHub Actions requires you to separate stages into separate

YAML workflow files.

- On-premises Azure Pipelines build agents can be selected with capabilities. GitHub Actions self-hosted runners can be selected with labels.

Migrating jobs and steps

Jobs and steps in Azure Pipelines are very similar to jobs and steps in GitHub Actions. In both systems, jobs have the following characteristics:

- Jobs contain a series of steps that run sequentially.
- Jobs run on separate virtual machines or in separate containers.
- Jobs run in parallel by default, but can be configured to run sequentially.

Migrating script steps

You can run a script or a shell command as a step in a workflow. In Azure Pipelines, script steps can be specified using the `script` key, or with the `bash`, `powershell`, or `pwsh` keys. Scripts can also be specified as an input to the Bash task or the PowerShell task.

In GitHub Actions, all scripts are specified using the `run` key. To select a particular shell, you can specify the `shell` key when providing the script. For more information, see "Workflow syntax for GitHub Actions."

Below is an example of the syntax for each system.

Azure Pipelines syntax for script steps

```yaml
jobs:
  - job: scripts
    pool:
      vmImage: 'windows-latest'
    steps:
      - script: echo "This step runs in the default shell"
      - bash: echo "This step runs in bash"
      - pwsh: Write-Host "This step runs in PowerShell Core"
      - task: PowerShell@2
        inputs:
          script: Write-Host "This step runs in PowerShell"
```

GitHub Actions syntax for script steps

```yaml
jobs:
  scripts:
    runs-on: windows-latest
    steps:
      - run: echo "This step runs in the default shell"
      - run: echo "This step runs in bash"
        shell: bash
      - run: Write-Host "This step runs in PowerShell Core"
        shell: pwsh
      - run: Write-Host "This step runs in PowerShell"
        shell: powershell
```

Differences in script error handling

In Azure Pipelines, scripts can be configured to error if any output is sent to `stderr`. GitHub Actions does not support this configuration.

GitHub Actions configures shells to "fail fast" whenever possible, which stops the script immediately if one of the commands in a script exits with an error code. In contrast, Azure Pipelines requires explicit configuration to exit immediately on an error. For more information, see "Workflow syntax for GitHub Actions."

Differences in the default shell on Windows

In Azure Pipelines, the default shell for scripts on Windows platforms is the Command shell (*cmd.exe*). In GitHub Actions, the default shell for scripts on Windows platforms is PowerShell. PowerShell has several differences in built-in commands, variable expansion, and flow control.

If you're running a simple command, you might be able to run a Command shell script in PowerShell without any changes. But in most cases, you will either need to update your script with PowerShell syntax or instruct GitHub Actions to run the script with the Command shell instead of PowerShell. You can do this by specifying `shell` as `cmd`.

Below is an example of the syntax for each system.

Azure Pipelines syntax using CMD by default

```
jobs:
  - job: run_command
    pool:
      vmImage: 'windows-latest'
    steps:
      - script: echo "This step runs in CMD on Windows by default"
```

GitHub Actions syntax for specifying CMD

```
jobs:
  run_command:
    runs-on: windows-latest
    steps:
      - run: echo "This step runs in PowerShell on Windows by default"
      - run: echo "This step runs in CMD on Windows explicitly"
        shell: cmd
```

For more information, see "Workflow syntax for GitHub Actions."

Migrating conditionals and expression syntax

Azure Pipelines and GitHub Actions can both run steps conditionally. In Azure Pipelines, conditional expressions are specified using the `condition` key. In GitHub Actions, conditional expressions are specified using the `if` key.

Azure Pipelines uses functions within expressions to execute steps conditionally. In contrast, GitHub Actions uses an infix notation. For example, you must replace the `eq` function in Azure Pipelines with the `==` operator in GitHub Actions.

Below is an example of the syntax for each system.

Azure Pipelines syntax for conditional expressions

```
jobs:
  - job: conditional
    pool:
```

```
      vmImage: 'ubuntu-latest'
    steps:
      - script: echo "This step runs with str equals 'ABC' and num equals 123"
        condition: and(eq(variables.str, 'ABC'), eq(variables.num, 123))
```

GitHub Actions syntax for conditional expressions

```
jobs:
  conditional:
    runs-on: ubuntu-latest
    steps:
      - run: echo "This step runs with str equals 'ABC' and num equals 123"
        if: ${{ env.str == 'ABC' && env.num == 123 }}
```

For more information, see "Expressions."

Dependencies between jobs

Both Azure Pipelines and GitHub Actions allow you to set dependencies for a job. In both systems, jobs run in parallel by default, but job dependencies can be specified explicitly. In Azure Pipelines, this is done with the `dependsOn` key. In GitHub Actions, this is done with the `needs` key.

Below is an example of the syntax for each system. The workflows start a first job named `initial`, and when that job completes, two jobs named `fanout1` and `fanout2` will run. Finally, when those jobs complete, the job `fanin` will run.

Azure Pipelines syntax for dependencies between jobs

```
jobs:
  - job: initial
    pool:
      vmImage: 'ubuntu-latest'
    steps:
      - script: echo "This job will be run first."
  - job: fanout1
    pool:
      vmImage: 'ubuntu-latest'
    dependsOn: initial
    steps:
      - script: echo "This job will run after the initial job, in parallel with
fanout2."
  - job: fanout2
    pool:
      vmImage: 'ubuntu-latest'
    dependsOn: initial
    steps:
      - script: echo "This job will run after the initial job, in parallel with
fanout1."
  - job: fanin:
    pool:
      vmImage: 'ubuntu-latest'
    dependsOn: [fanout1, fanout2]
    steps:
      - script: echo "This job will run after fanout1 and fanout2 have finished."
```

GitHub Actions syntax for dependencies between jobs

```
jobs:
  initial:
```

```
    runs-on: ubuntu-latest
    steps:
      - run: echo "This job will be run first."
  fanout1:
    runs-on: ubuntu-latest
    needs: initial
    steps:
      - run: echo "This job will run after the initial job, in parallel with
fanout2."
  fanout2:
    runs-on: ubuntu-latest
    needs: initial
    steps:
      - run: echo "This job will run after the initial job, in parallel with
fanout1."
  fanin:
    runs-on: ubuntu-latest
    needs: [fanout1, fanout2]
    steps:
      - run: echo "This job will run after fanout1 and fanout2 have finished."
```

For more information, see "Workflow syntax for GitHub Actions."

Migrating tasks to actions

Azure Pipelines uses *tasks*, which are application components that can be re-used in multiple workflows. GitHub Actions uses *actions*, which can be used to perform tasks and customize your workflow. In both systems, you can specify the name of the task or action to run, along with any required inputs as key/value pairs.

Below is an example of the syntax for each system.

Azure Pipelines syntax for tasks

```
jobs:
  - job: run_python
    pool:
      vmImage: 'ubuntu-latest'
    steps:
      - task: UsePythonVersion@0
        inputs:
          versionSpec: '3.7'
          architecture: 'x64'
      - script: python script.py
```

GitHub Actions syntax for actions

```
jobs:
  run_python:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/setup-python@v4
        with:
          python-version: '3.7'
          architecture: 'x64'
      - run: python script.py
```

You can find actions that you can use in your workflow in GitHub Marketplace, or you can create your own actions. For more information, see "Creating actions."