Migrating from Jenkins to GitHub Actions

In this article

GitHub Actions and Jenkins share multiple similarities, which makes migration to GitHub Actions relatively straightforward.

Introduction

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

- Jenkins creates workflows using *Declarative Pipelines*, which are similar to GitHub Actions workflow files.
- Jenkins uses *stages* to run a collection of steps, while GitHub Actions uses jobs to group one or more steps or individual commands.
- Jenkins and GitHub Actions support container-based builds. For more information, see "Creating a Docker container action."
- Steps or tasks can be reused and shared with the community.

For more information, see "Understanding GitHub Actions."

Key differences

- Jenkins has two types of syntax for creating pipelines: Declarative Pipeline and Scripted Pipeline. GitHub Actions uses YAML to create workflows and configuration files. For more information, see "Workflow syntax for GitHub Actions."
- Jenkins deployments are typically self-hosted, with users maintaining the servers in their own data centers. GitHub Actions offers a hybrid cloud approach by hosting its own runners that you can use to run jobs, while also supporting self-hosted runners. For more information, see About self-hosted runners.

Comparing capabilities

Distributing your builds

Jenkins lets you send builds to a single build agent, or you can distribute them across multiple agents. You can also classify these agents according to various attributes, such as operating system types.

Similarly, GitHub Actions can send jobs to GitHub-hosted or self-hosted runners, and you can use labels to classify runners according to various attributes. For more information, see "Understanding GitHub Actions" and "About self-hosted runners."

Using sections to organize pipelines

Jenkins splits its Declarative Pipelines into multiple sections. Similarly, GitHub Actions organizes its workflows into separate sections. The table below compares Jenkins sections with the GitHub Actions workflow.

Jenkins Directives	GitHub Actions

`agent`	`jobs.<job_id>.runs-on`
	`jobs.<job_id>.container`

`post`	None
`stages`	`jobs`
`steps`	`jobs.<job_id>.steps`

Using directives

Jenkins uses directives to manage *Declarative Pipelines*. These directives define the characteristics of your workflow and how it will execute. The table below demonstrates how these directives map to concepts within GitHub Actions.

Jenkins Directives	GitHub Actions

`environment`	`jobs.<job_id>.env`
	`jobs.<job_id>.steps[*].env`

`options`	`jobs.<job_id>.strategy`
	`jobs.<job_id>.strategy.fail-fast`
	`jobs.<job_id>.timeout-minutes`

`parameters`	`inputs` `outputs`

`triggers`	`on`
	`on.<event_name>.types`
	`on.<push>.<branches|tags>`
	`on.<pull_request>.<branches>`
	`on.<push|pull_request>.paths`

`triggers { upstreamprojects() }`	`jobs.<job_id>.needs`
Jenkins cron syntax	`on.schedule`

`stage`	`jobs.<job_id>`
	`jobs.<job_id>.name`

`tools`	Specifications for GitHub-hosted runners
`input`	`inputs`
`when`	`jobs.<job_id>.if`

Using sequential stages

Parallel job processing

Jenkins can run the `stages` and `steps` in parallel, while GitHub Actions currently only runs jobs in parallel.

Jenkins Parallel	GitHub Actions
`parallel`	`jobs.<job_id>.strategy.max-parallel`

Matrix

Both GitHub Actions and Jenkins let you use a matrix to define various system combinations.

Jenkins	GitHub Actions
`axis`	`strategy/matrix` `context`
`stages`	`steps-context`
`excludes`	None

Using steps to execute tasks

Jenkins groups `steps` together in `stages`. Each of these steps can be a script, function, or command, among others. Similarly, GitHub Actions uses `jobs` to execute specific groups of `steps`.

Jenkins	GitHub Actions
`steps`	`jobs.<job_id>.steps`

Examples of common tasks

Scheduling a pipeline to run with `cron`

Jenkins pipeline with `cron`

```
pipeline {
  agent any
  triggers {
    cron('H/15 * * * 1-5')
  }
}
```

GitHub Actions workflow with `cron`

```
on:
  schedule:
    - cron: '*/15 * * * 1-5'
```

Configuring environment variables in a pipeline

Jenkins pipeline with an environment variable

```
pipeline {
  agent any
  environment {
    MAVEN_PATH = '/usr/local/maven'
  }
}
```

GitHub Actions workflow with an environment variable

```
jobs:
  maven-build:
    env:
      MAVEN_PATH: '/usr/local/maven'
```

Building from upstream projects

Jenkins pipeline that builds from an upstream project

```
pipeline {
  triggers {
    upstream(
      upstreamProjects: 'job1,job2',
      threshold: hudson.model.Result.SUCCESS
    )
  }
}
```

GitHub Actions workflow that builds from an upstream project

```
jobs:
  job1:
  job2:
    needs: job1
  job3:
    needs: [job1, job2]
```

Building with multiple operating systems

Jenkins pipeline that builds with multiple operating systems

```
pipeline {
  agent none
  stages {
    stage('Run Tests') {
      matrix {
        axes {
          axis {
            name: 'PLATFORM'
            values: 'macos', 'linux'
          }
        }
        agent { label "${PLATFORM}" }
        stages {
          stage('test') {
            tools { nodejs "node-16" }
            steps {
              dir("scripts/myapp") {
                sh(script: "npm install -g bats")
                sh(script: "bats tests")
              }
            }
          }
        }
      }
    }
  }
}
```

GitHub Actions workflow that builds with multiple operating systems

```yaml
name: demo-workflow
on:
  push:
jobs:
  test:
    runs-on: ${{ matrix.os }}
    strategy:
      fail-fast: false
      matrix:
        os: [macos-latest, ubuntu-latest]
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v3
        with:
          node-version: 16
      - run: npm install -g bats
      - run: bats tests
        working-directory: ./scripts/myapp
```