This version of GitHub Enterprise was discontinued on 2023-03-15. No patch releases will be made, even for critical security issues. For better performance, improved security, and new features, upgrade to the latest version of GitHub Enterprise. For help with the upgrade, contact GitHub Enterprise support.

Building and testing Go

In this article

You can create a continuous integration (CI) workflow to build and test your Go project.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Introduction

This guide shows you how to build, test, and publish a Go package.

GitHub-hosted runners have a tools cache with preinstalled software, which includes the dependencies for Go. For a full list of up-to-date software and the preinstalled versions of Go, see "About GitHub-hosted runners."

Prerequisites

You should already be familiar with YAML syntax and how it's used with GitHub Actions. For more information, see "Workflow syntax for GitHub Actions."

We recommend that you have a basic understanding of the Go language. For more information, see Getting started with Go.

Using the Go starter workflow

GitHub provides a Go starter workflow that should work for most Go projects. This guide includes examples that you can use to customize the starter workflow. For more information, see the Go starter workflow.

To get started quickly, add the starter workflow to the `.github/workflows` directory of your repository.

YAML

```yaml
name: Go package

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      - name: Set up Go
        uses: actions/setup-go@v2
        with:
          go-version: '1.15'

      - name: Build
        run: go build -v ./...

      - name: Test
        run: go test -v ./...
```

Specifying a Go version

The easiest way to specify a Go version is by using the `setup-go` action provided by GitHub. For more information see, the `setup-go` action.

To use a preinstalled version of Go on a GitHub-hosted runner, pass the relevant version to the `go-version` property of the `setup-go` action. This action finds a specific version of Go from the tools cache on each runner, and adds the necessary binaries to `PATH`. These changes will persist for the remainder of the job.

The `setup-go` action is the recommended way of using Go with GitHub Actions, because it helps ensure consistent behavior across different runners and different versions of Go. If you are using a self-hosted runner, you must install Go and add it to `PATH`.

Using multiple versions of Go

YAML

```yaml
name: Go

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        go-version: [ '1.14', '1.15', '1.16.x' ]

    steps:
      - uses: actions/checkout@v2
      - name: Setup Go ${{ matrix.go-version }}
        uses: actions/setup-go@v2
        with:
          go-version: ${{ matrix.go-version }}
      # You can test your matrix by printing the current Go version
      - name: Display Go version
        run: go version
```

Using a specific Go version

You can configure your job to use a specific version of Go, such as `1.16.2`. Alternatively, you can use semantic version syntax to get the latest minor release. This example uses the latest patch release of Go 1.16:

YAML

```yaml
      - name: Setup Go 1.16.x
        uses: actions/setup-go@v2
        with:
          # Semantic version range syntax or exact version of Go
          go-version: '1.16.x'
```

Installing dependencies

You can use `go get` to install dependencies:

YAML

```yaml
    steps:
      - uses: actions/checkout@v2
      - name: Setup Go
        uses: actions/setup-go@v2
        with:
          go-version: '1.16.x'
      - name: Install dependencies
        run: |
          go get .
          go get example.com/octo-examplemodule
          go get example.com/octo-examplemodule@v1.3.4
```

Building and testing your code

You can use the same commands that you use locally to build and test your code. This example workflow demonstrates how to use `go build` and `go test` in a job:

YAML

```yaml
name: Go
on: [push]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v2
      - name: Setup Go
        uses: actions/setup-go@v2
        with:
          go-version: '1.16.x'
      - name: Install dependencies
        run: go get .
      - name: Build
        run: go build -v ./...
      - name: Test with the Go CLI
        run: go test
```

Packaging workflow data as artifacts

After a workflow completes, you can upload the resulting artifacts for analysis. For example, you may need to save log files, core dumps, test results, or screenshots. The following example demonstrates how you can use the `upload-artifact` action to upload test results.

For more information, see "Storing workflow data as artifacts."

YAML

```yaml
name: Upload Go test results

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        go-version: [ '1.14', '1.15', '1.16.x' ]

    steps:
      - uses: actions/checkout@v2
      - name: Setup Go
        uses: actions/setup-go@v2
        with:
          go-version: ${{ matrix.go-version }}
      - name: Install dependencies
        run: go get .
      - name: Test with Go
        run: go test -json > TestResults-${{ matrix.go-version }}.json
      - name: Upload Go test results
        uses: actions/upload-artifact@v2
        with:
          name: Go-results-${{ matrix.go-version }}
          path: TestResults-${{ matrix.go-version }}.json
```