Building and testing Ruby

In this article

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

Introduction

This guide shows you how to create a continuous integration (CI) workflow that builds and tests a Ruby application. If your CI tests pass, you may want to deploy your code or publish a gem.

Prerequisites

We recommend that you have a basic understanding of Ruby, YAML, workflow configuration options, and how to create a workflow file. For more information, see:

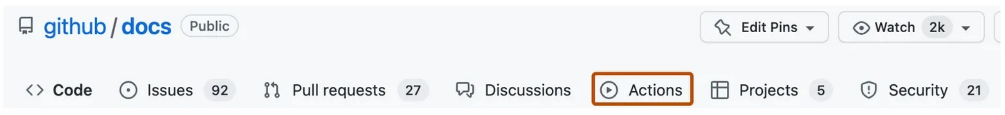
- Learn GitHub Actions
- Ruby in 20 minutes

Using a Ruby starter workflow

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

GitHub provides a starter workflow for Ruby that should work for most Ruby projects. The subsequent sections of this guide give examples of how you can customize this starter workflow.

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.



3. If you already have a workflow in your repository, click **New workflow**.

4. The "Choose a workflow" page shows a selection of recommended starter workflows. Search for "ruby".

5. Filter the selection of workflows by clicking **Continuous integration**.

6. On the "Ruby" workflow, click **Configure**.

7. Edit the workflow as required. For example, change the Ruby versions you want to use.

   Notes:

   - This starter workflow contains an action that is not certified by GitHub. Actions provided by third parties are governed by separate terms of service, privacy policy, and support documentation.
   - If you use actions from third parties you should use a version specified by a commit SHA. If the action is revised and you want to use the newer version, you will need to update the SHA. You can specify a version by referencing a tag or a branch, however the action may change without warning. For more information, see "Security hardening for GitHub Actions."

8. Click **Commit changes**.

   The `ruby.yml` workflow file is added to the `.github/workflows` directory of your repository.

Specifying the Ruby version

The easiest way to specify a Ruby version is by using the `ruby/setup-ruby` action provided by the Ruby organization on GitHub. The action adds any supported Ruby version to `PATH` for each job run in a workflow. For more information and available Ruby versions, see `ruby/setup-ruby`.

Using Ruby's `ruby/setup-ruby` action is the recommended way of using Ruby with GitHub Actions because it ensures consistent behavior across different runners and different versions of Ruby.

The `setup-ruby` action takes a Ruby version as an input and configures that version on the runner.

```
steps:
- uses: actions/checkout@v4
- uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
  with:
    ruby-version: '3.1' # Not needed with a .ruby-version file
- run: bundle install
- run: bundle exec rake
```

Alternatively, you can check a `.ruby-version` file into the root of your repository and `setup-ruby` will use the version defined in that file.

Testing with multiple versions of Ruby

You can add a matrix strategy to run your workflow with more than one version of Ruby. For example, you can test your code against the latest patch releases of versions 3.1, 3.0, and 2.7.

```yaml
strategy:
  matrix:
    ruby-version: ['3.1', '3.0', '2.7']
```

Each version of Ruby specified in the `ruby-version` array creates a job that runs the same steps. The `${{ matrix.ruby-version }}` context is used to access the current job's version. For more information about matrix strategies and contexts, see "Workflow syntax for GitHub Actions" and "Contexts."

The full updated workflow with a matrix strategy could look like this:

```yaml
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Ruby CI

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        ruby-version: ['3.1', '3.0', '2.7']

    steps:
      - uses: actions/checkout@v4
      - name: Set up Ruby ${{ matrix.ruby-version }}
        uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
        with:
          ruby-version: ${{ matrix.ruby-version }}
      - name: Install dependencies
        run: bundle install
      - name: Run tests
        run: bundle exec rake
```

Installing dependencies with Bundler

The `setup-ruby` action will automatically install bundler for you. The version is determined by your `gemfile.lock` file. If no version is present in your lockfile, then the latest compatible version will be installed.

```yaml
steps:
- uses: actions/checkout@v4
- uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
  with:
    ruby-version: '3.1'
- run: bundle install
```

Caching dependencies

The `setup-ruby` actions provides a method to automatically handle the caching of your gems between runs.

To enable caching, set the following.

```
steps:
- uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
    with:
      bundler-cache: true
```

This will configure bundler to install your gems to `vendor/cache`. For each successful run of your workflow, this folder will be cached by GitHub Actions and re-downloaded for subsequent workflow runs. A hash of your gemfile.lock and the Ruby version are used as the cache key. If you install any new gems, or change a version, the cache will be invalidated and bundler will do a fresh install.

Caching without setup-ruby

For greater control over caching, you can use the `actions/cache` action directly. For more information, see "Caching dependencies to speed up workflows."

```
steps:
- uses: actions/cache@v3
  with:
    path: vendor/bundle
    key: ${{ runner.os }}-gems-${{ hashFiles('**/Gemfile.lock') }}
    restore-keys: |
      ${{ runner.os }}-gems-
- name: Bundle install
  run: |
    bundle config path vendor/bundle
    bundle install --jobs 4 --retry 3
```

If you're using a matrix build, you will want to include the matrix variables in your cache key. For example, if you have a matrix strategy for different ruby versions (`matrix.ruby-version`) and different operating systems (`matrix.os`), your workflow steps might look like this:

```
steps:
- uses: actions/cache@v3
  with:
    path: vendor/bundle
    key: bundle-use-ruby-${{ matrix.os }}-${{ matrix.ruby-version }}-${{
hashFiles('**/Gemfile.lock') }}
    restore-keys: |
      bundle-use-ruby-${{ matrix.os }}-${{ matrix.ruby-version }}-
- name: Bundle install
  run: |
    bundle config path vendor/bundle
    bundle install --jobs 4 --retry 3
```

Matrix testing your code

The following example matrix tests all stable releases and head versions of MRI, JRuby and TruffleRuby on Ubuntu and macOS.

```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Matrix Testing

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  test:
    runs-on: ${{ matrix.os }}-latest
    strategy:
      fail-fast: false
      matrix:
        os: [ubuntu, macos]
        ruby: [2.5, 2.6, 2.7, head, debug, jruby, jruby-head, truffleruby,
truffleruby-head]
    continue-on-error: ${{ endsWith(matrix.ruby, 'head') || matrix.ruby ==
'debug' }}
    steps:
      - uses: actions/checkout@v4
      - uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
        with:
          ruby-version: ${{ matrix.ruby }}
      - run: bundle install
      - run: bundle exec rake
```

Linting your code

The following example installs `rubocop` and uses it to lint all files. For more information, see RuboCop. You can configure Rubocop to decide on the specific linting rules.

```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Linting

on: [push]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
        with:
          ruby-version: '2.6'
      - run: bundle install
      - name: Rubocop
        run: rubocop
```

Publishing Gems

You can configure your workflow to publish your Ruby package to any package registry you'd like when your CI tests pass.

You can store any access tokens or credentials needed to publish your package using repository secrets. The following example creates and publishes a package to `GitHub Package Registry` and `RubyGems`.

```
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# GitHub recommends pinning actions to a commit SHA.
# To get a newer version, you will need to update the SHA.
# You can also reference a tag or branch, but the action may change without
warning.

name: Ruby Gem

on:
  # Manually publish
  workflow_dispatch:
  # Alternatively, publish whenever changes are merged to the `main` branch.
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    name: Build + Publish
    runs-on: ubuntu-latest
    permissions:
      packages: write
      contents: read

    steps:
      - uses: actions/checkout@v4
      - name: Set up Ruby 2.6
        uses: ruby/setup-ruby@ec02537da5712d66d4d50a0f33b7eb52773b5ed1
        with:
          ruby-version: '2.6'
      - run: bundle install

      - name: Publish to GPR
        run: |
          mkdir -p $HOME/.gem
          touch $HOME/.gem/credentials
          chmod 0600 $HOME/.gem/credentials
          printf -- "---\n:github: ${GEM_HOST_API_KEY}\n" >
$HOME/.gem/credentials
          gem build *.gemspec
          gem push --KEY github --host https://rubygems.pkg.github.com/${OWNER}
*.gem
        env:
          GEM_HOST_API_KEY: "Bearer ${{secrets.GITHUB_TOKEN}}"
          OWNER: ${{ github.repository_owner }}

      - name: Publish to RubyGems
        run: |
          mkdir -p $HOME/.gem
          touch $HOME/.gem/credentials
          chmod 0600 $HOME/.gem/credentials
          printf -- "---\n:rubygems_api_key: ${GEM_HOST_API_KEY}\n" >
$HOME/.gem/credentials
          gem build *.gemspec
          gem push *.gem
        env:
          GEM_HOST_API_KEY: "${{secrets.RUBYGEMS_AUTH_TOKEN}}"
```