Releasing and maintaining actions

In this article

Introduction
Developing and releasing actions
Working with the community
Further reading

You can leverage automation and open source best practices to release and maintain actions.

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

After you create an action, you'll want to continue releasing new features while working with community contributions. This tutorial describes an example process you can follow to release and maintain actions in open source. The example:

- Leverages GitHub Actions for continuous integration, dependency updates, release management, and task automation.
- Provides confidence through automated tests and build badges.
- Indicates how the action can be used, ideally as part of a broader workflow.
- Signal what type of community contributions you welcome. (For example, issues, pull requests, or vulnerability reports.)

For an applied example of this process, see github-developer/javascript-action.

Developing and releasing actions

In this section, we discuss an example process for developing and releasing actions and show how to use GitHub Actions to automate the process.

About JavaScript actions

JavaScript actions are Node.js repositories with metadata. However, JavaScript actions have additional properties compared to traditional Node.js projects:

- Dependent packages are committed alongside the code, typically in a compiled and minified form. This means that automated builds and secure community contributions are important.

- Many actions make use of GitHub's APIs and third party APIs, so we encourage robust end-to-end testing.

Setting up GitHub Actions workflows

To support the developer process in the next section, add two GitHub Actions workflows to your repository:

1. Add a workflow that triggers when a commit is pushed to a feature branch or to `main` or when a pull request is created. Configure the workflow to run your unit and integration tests. For an example, see this workflow.

2. Add a workflow that triggers when a release is published or edited. Configure the workflow to ensure semantic tags are in place. You can use an action like JasonEtco/build-and-tag-action to compile and bundle the JavaScript and metadata file and force push semantic major, minor, and patch tags. For an example, see this workflow. For more information about semantic tags, see "About semantic versioning."

Example developer process

Here is an example process that you can follow to automatically run tests, create a release, and publish your action.

1. Do feature work in branches per GitHub flow. For more information, see "GitHub flow."

   - Whenever a commit is pushed to the feature branch, your testing workflow will automatically run the tests.

2. Create pull requests to the `main` branch to initiate discussion and review, merging when ready.

   - When a pull request is opened, either from a branch or a fork, your testing workflow will again run the tests, this time with the merge commit.
   - **Note:** for security reasons, workflows triggered by `pull_request` from forks have restricted `GITHUB_TOKEN` permissions and do not have access to secrets. If your tests or other workflows triggered upon pull request require access to secrets, consider using a different event like a manual trigger or a `pull_request_target`. Read more here.

3. Create a semantically tagged release. For more information, see "Managing releases in a repository".

   - When a release is published or edited, your release workflow will automatically take care of compilation and adjusting tags.
   - We recommend creating releases using semantically versioned tags – for example, `v1.1.3` – and keeping major (`v1`) and minor (`v1.1`) tags current to the latest appropriate commit. For more information, see "About custom actions" and "About semantic versioning.

Results

Unlike some other automated release management strategies, this process intentionally does not commit dependencies to the `main` branch, only to the tagged release commits.

By doing so, you encourage users of your action to reference named tags or `sha`s, and you help ensure the security of third party pull requests by doing the build yourself during a release.

Using semantic releases means that the users of your actions can pin their workflows to a version and know that they might continue to receive the latest stable, non-breaking features, depending on their comfort level:

Working with the community

GitHub Enterprise Server provides tools and guides to help you work with the open source community. Here are a few tools we recommend setting up for healthy bidirectional communication. By providing the following signals to the community, you encourage others to use, modify, and contribute to your action:

- Maintain a `README` with plenty of usage examples and guidance. For more information, see "About READMEs."
- Include a workflow status badge in your `README` file. For more information, see "Adding a workflow status badge." Also visit shields.io to learn about other badges that you can add.
- Keep issues current by utilizing actions like actions/stale.

Further reading

Examples where similar patterns are employed include:

- github/super-linter
- octokit/request-action
- github-developer/javascript-action