About custom actions

In this article

Actions are individual tasks that you can combine to create jobs and customize your workflow. You can create your own actions, or use and customize actions shared by the GitHub community.

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.

About custom actions

You can create actions by writing custom code that interacts with your repository in any way you'd like, including integrating with GitHub's APIs and any publicly available third-party API. For example, an action can publish npm modules, send SMS alerts when urgent issues are created, or deploy production-ready code.

Actions can run directly on a machine or in a Docker container. You can define an action's inputs, outputs, and environment variables.

Types of actions

You can build Docker container, JavaScript, and composite actions. Actions require a metadata file to define the inputs, outputs and main entrypoint for your action. The metadata filename must be either `action.yml` or `action.yaml`. For more information, see "Metadata syntax for GitHub Actions."

Type	Linux	macOS	Windows
Docker container	✓	×	×
JavaScript	✓	✓	✓

Composite Actions	✓	✓	✓

Docker container actions

Docker containers package the environment with the GitHub Actions code. This creates a more consistent and reliable unit of work because the consumer of the action does not need to worry about the tools or dependencies.

A Docker container allows you to use specific versions of an operating system, dependencies, tools, and code. For actions that must run in a specific environment configuration, Docker is an ideal option because you can customize the operating system and tools. Because of the latency to build and retrieve the container, Docker container actions are slower than JavaScript actions.

Docker container actions can only execute on runners with a Linux operating system. Self-hosted runners must use a Linux operating system and have Docker installed to run Docker container actions. For more information about the requirements of self-hosted runners, see "About self-hosted runners."

JavaScript actions

JavaScript actions can run directly on a runner machine, and separate the action code from the environment used to run the code. Using a JavaScript action simplifies the action code and executes faster than a Docker container action.

To ensure your JavaScript actions are compatible with all GitHub-hosted runners (Ubuntu, Windows, and macOS), the packaged JavaScript code you write should be pure JavaScript and not rely on other binaries. JavaScript actions run directly on the runner and use binaries that already exist in the runner image.

If you're developing a Node.js project, the GitHub Actions Toolkit provides packages that you can use in your project to speed up development. For more information, see the actions/toolkit repository.

Composite Actions

A *composite* action allows you to combine multiple workflow steps within one action. For example, you can use this feature to bundle together multiple run commands into an action, and then have a workflow that executes the bundled commands as a single step using that action. To see an example, check out "Creating a composite action".

Choosing a location for your action

If you're developing an action for other people to use, we recommend keeping the action in its own repository instead of bundling it with other application code. This allows you to version, track, and release the action just like any other software.

You can store the action's files in any location in your repository. If you plan to combine action, workflow, and application code in a single repository, we recommend storing actions in the `.github` directory. For example, `.github/actions/action-a` and `.github/actions/action-b`.

Compatibility with GitHub Enterprise Server

To ensure that your action is compatible with GitHub Enterprise Server, you should make sure that you do not use any hard-coded references to GitHub Enterprise Server API URLs. You should instead use environment variables to refer to the GitHub Enterprise Server API:

- For the REST API, use the `GITHUB_API_URL` environment variable.
- For GraphQL, use the `GITHUB_GRAPHQL_URL` environment variable.

For more information, see "Variables."

Using release management for actions

This section explains how you can use release management to distribute updates to your actions in a predictable way.

Good practices for release management

If you're developing an action for other people to use, we recommend using release management to control how you distribute updates. Users can expect an action's patch version to include necessary critical fixes and security patches, while still remaining compatible with their existing workflows. You should consider releasing a new major version whenever your changes affect compatibility.

Under this release management approach, users should not be referencing an action's default branch, as it's likely to contain the latest code and consequently might be unstable. Instead, you can recommend that your users specify a major version when using your action, and only direct them to a more specific version if they encounter issues.

To use a specific action version, users can configure their GitHub Actions workflow to target a tag, a commit's SHA, or a branch named for a release.

Using tags for release management

We recommend using tags for actions release management. Using this approach, your users can easily distinguish between major and minor versions:

- Create and validate a release on a release branch (such as `release/v1`) before creating the release tag (for example, `v1.0.2`).
- Create a release using semantic versioning. For more information, see "Managing releases in a repository."
- Move the major version tag (such as `v1`, `v2`) to point to the Git ref of the current release. For more information, see "Git basics - tagging."
- Introduce a new major version tag (`v2`) for changes that will break existing workflows. For example, changing an action's inputs would be a breaking change.
- Major versions can be initially released with a `beta` tag to indicate their status, for example, `v2-beta`. The `-beta` tag can then be removed when ready.

This example demonstrates how a user can reference a major release tag:

```
steps:
    - uses: actions/javascript-action@v1
```

This example demonstrates how a user can reference a specific patch release tag:

```
steps:
    - uses: actions/javascript-action@v1.0.1
```

Using branches for release management

If you prefer to use branch names for release management, this example demonstrates how to reference a named branch:

```
steps:
    - uses: actions/javascript-action@v1-beta
```

Using a commit's SHA for release management

Each Git commit receives a calculated SHA value, which is unique and immutable. Your action's users might prefer to rely on a commit's SHA value, as this approach can be more reliable than specifying a tag, which could be deleted or moved. However, this means that users will not receive further updates made to the action. You must use a commit's full SHA value, and not an abbreviated value.

```
steps:
    - uses: actions/javascript-action@a824008085750b8e136effc585c3cd6082bd575f
```

Creating a README file for your action

We recommend creating a README file to help people learn how to use your action. You can include this information in your `README.md`:

- A detailed description of what the action does
- Required input and output arguments
- Optional input and output arguments
- Secrets the action uses
- Environment variables the action uses
- An example of how to use your action in a workflow

Comparing GitHub Actions to GitHub Apps

GitHub Marketplace offers tools to improve your workflow. Understanding the differences and the benefits of each tool will allow you to select the best tool for your job. For more information about building apps, see "About creating GitHub Apps."

Strengths of GitHub Actions and GitHub Apps

While both GitHub Actions and GitHub Apps provide ways to build automation and workflow tools, they each have strengths that make them useful in different ways.

GitHub Apps:

- Run persistently and can react to events quickly.
- Work great when persistent data is needed.
- Work best with API requests that aren't time consuming.
- Run on a server or compute infrastructure that you provide.

GitHub Actions:

- Provide automation that can perform continuous integration and continuous deployment.
- Can run directly on runner machines or in Docker containers.
- Can include access to a clone of your repository, enabling deployment and publishing tools, code formatters, and command line tools to access your code.
- Don't require you to deploy code or serve an app.
- Have a simple interface to create and use secrets, which enables actions to interact with third-party services without needing to store the credentials of the person using

the action.

Further reading

- "Workflow commands for GitHub Actions"