Creating a JavaScript action

Create an action
3 of 8 in learning path

Next: Creating a composite action

In this article

In this guide, you'll learn how to build a JavaScript action using the actions toolkit.

Introduction

In this guide, you'll learn about the basic components needed to create and use a packaged JavaScript action. To focus this guide on the components needed to package the action, the functionality of the action's code is minimal. The action prints "Hello World" in the logs or "Hello [who-to-greet]" if you provide a custom name.

This guide uses the GitHub Actions Toolkit Node.js module to speed up development. For more information, see the actions/toolkit repository.

Once you complete this project, you should understand how to build your own JavaScript action and test it in a workflow.

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.

Warning: When creating workflows and actions, you should always consider whether your code might execute untrusted input from possible attackers. Certain contexts should be treated as untrusted input, as an attacker could insert their own malicious content. For more information, see "Security hardening for GitHub Actions."

Prerequisites

Before you begin, you'll need to download Node.js and create a public GitHub repository.

1. Download and install Node.js 20.x, which includes npm.

   https://nodejs.org/en/download/

2. Create a new public repository on GitHub.com and call it "hello-world-javascript-action". For more information, see "Creating a new repository."

3. Clone your repository to your computer. For more information, see "Cloning a repository."

4. From your terminal, change directories into your new repository.

   ```shell
   cd hello-world-javascript-action
   ```

5. From your terminal, initialize the directory with npm to generate a `package.json` file.

   ```shell
   npm init -y
   ```

Creating an action metadata file

Create a new file named `action.yml` in the `hello-world-javascript-action` directory with the following example code. For more information, see "Metadata syntax for GitHub Actions."

```yaml
name: 'Hello World'
description: 'Greet someone and record the time'
inputs:
  who-to-greet:  # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  time: # id of output
    description: 'The time we greeted you'
runs:
  using: 'node20'
  main: 'index.js'
```

This file defines the `who-to-greet` input and `time` output. It also tells the action runner how to start running this JavaScript action.

Adding actions toolkit packages

The actions toolkit is a collection of Node.js packages that allow you to quickly build JavaScript actions with more consistency.

The toolkit `@actions/core` package provides an interface to the workflow commands, input and output variables, exit statuses, and debug messages.

The toolkit also offers a `@actions/github` package that returns an authenticated Octokit REST client and access to GitHub Actions contexts.

The toolkit offers more than the `core` and `github` packages. For more information, see the actions/toolkit repository.

At your terminal, install the actions toolkit `core` and `github` packages.

Shell

```
npm install @actions/core
npm install @actions/github
```

Now you should see a `node_modules` directory with the modules you just installed and a `package-lock.json` file with the installed module dependencies and the versions of each installed module.

Writing the action code

This action uses the toolkit to get the `who-to-greet` input variable required in the action's metadata file and prints "Hello [who-to-greet]" in a debug message in the log. Next, the script gets the current time and sets it as an output variable that actions running later in a job can use.

GitHub Actions provide context information about the webhook event, Git refs, workflow, action, and the person who triggered the workflow. To access the context information, you can use the `github` package. The action you'll write will print the webhook event payload to the log.

Add a new file called `index.js`, with the following code.

JavaScript

```
const core = require('@actions/core');
const github = require('@actions/github');

try {
  // `who-to-greet` input defined in action metadata file
  const nameToGreet = core.getInput('who-to-greet');
  console.log(`Hello ${nameToGreet}!`);
  const time = (new Date()).toTimeString();
  core.setOutput("time", time);
  // Get the JSON webhook payload for the event that triggered the workflow
  const payload = JSON.stringify(github.context.payload, undefined, 2)
  console.log(`The event payload: ${payload}`);
} catch (error) {
  core.setFailed(error.message);
}
```

If an error is thrown in the above `index.js` example, `core.setFailed(error.message);` uses the actions toolkit `@actions/core` package to log a message and set a failing exit code. For more information, see "Setting exit codes for actions."

Creating a README

To let people know how to use your action, you can create a README file. A README is most helpful when you plan to share your action publicly, but is also a great way to

remind you or your team how to use the action.

In your `hello-world-javascript-action` directory, create a `README.md` file that specifies the following information:

- 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.

Markdown

```
# Hello world javascript action

This action prints "Hello World" or "Hello" + the name of a person to greet to
the log.

## Inputs

### `who-to-greet`

**Required** The name of the person to greet. Default `"World"`.

## Outputs

### `time`

The time we greeted you.

## Example usage

```yaml
uses: actions/hello-world-javascript-
action@e76147da8e5c81eaf017dede5645551d4b94427b
with:
  who-to-greet: 'Mona the Octocat'
```
```

Commit, tag, and push your action to GitHub

GitHub Enterprise Cloud downloads each action run in a workflow during runtime and executes it as a complete package of code before you can use workflow commands like `run` to interact with the runner machine. This means you must include any package dependencies required to run the JavaScript code. You'll need to check in the toolkit `core` and `github` packages to your action's repository.

From your terminal, commit your `action.yml`, `index.js`, `node_modules`, `package.json`, `package-lock.json`, and `README.md` files. If you added a `.gitignore` file that lists `node_modules`, you'll need to remove that line to commit the `node_modules` directory.

It's best practice to also add a version tag for releases of your action. For more information on versioning your action, see "About custom actions."

Shell

```
git add action.yml index.js node_modules/* package.json package-lock.json
README.md
git commit -m "My first action is ready"
git tag -a -m "My first action release" v1.1
git push --follow-tags
```

Checking in your `node_modules` directory can cause problems. As an alternative, you can use a tool called `@vercel/ncc` to compile your code and modules into one file used for distribution.

1. Install `vercel/ncc` by running this command in your terminal.

   ```
   npm i -g @vercel/ncc
   ```

2. Compile your `index.js` file.

   ```
   ncc build index.js --license licenses.txt
   ```

   You'll see a new `dist/index.js` file with your code and the compiled modules. You will also see an accompanying `dist/licenses.txt` file containing all the licenses of the `node_modules` you are using.

3. Change the `main` keyword in your `action.yml` file to use the new `dist/index.js` file.

   ```
   main: 'dist/index.js'
   ```

4. If you already checked in your `node_modules` directory, remove it.

   ```
   rm -rf node_modules/*
   ```

5. From your terminal, commit the updates to your `action.yml`, `dist/index.js`, and `node_modules` files.

   Shell

   ```
   git add action.yml dist/index.js node_modules/*
   git commit -m "Use vercel/ncc"
   git tag -a -m "My first action release" v1.1
   git push --follow-tags
   ```

Testing out your action in a workflow

Now you're ready to test your action out in a workflow.

Public actions can be used by workflows in any repository. When an action is in a private or internal repository, the repository settings dictate whether the action is available only within the same repository or also to other repositories owned by the same organization or enterprise. For more information, see "Managing GitHub Actions settings for a repository."

Example using a public action

This example demonstrates how your new public action can be run from within an external repository.

Copy the following YAML into a new file at `.github/workflows/main.yml`, and update the `uses: octocat/hello-world-javascript-action@v1.1` line with your username and the name of the public repository you created above. You can also replace the `who-to-greet` input with your name.

YAML

```yaml
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    name: A job to say hello
    steps:
      - name: Hello world action step
        id: hello
        uses: octocat/hello-world-javascript-action@v1.1
        with:
          who-to-greet: 'Mona the Octocat'
      # Use the output from the `hello` step
      - name: Get the output time
        run: echo "The time was ${{ steps.hello.outputs.time }}"
```

When this workflow is triggered, the runner will download the `hello-world-javascript-action` action from your public repository and then execute it.

Example using a private action

Copy the workflow code into a `.github/workflows/main.yml` file in your action's repository. You can also replace the `who-to-greet` input with your name.

.github/workflows/main.yml

YAML

```yaml
on: [push]

jobs:
  hello_world_job:
    runs-on: ubuntu-latest
    name: A job to say hello
    steps:
      # To use this repository's private action,
      # you must check out the repository
      - name: Checkout
        uses: actions/checkout@v4
      - name: Hello world action step
        uses: ./ # Uses an action in the root directory
        id: hello
        with:
          who-to-greet: 'Mona the Octocat'
      # Use the output from the `hello` step
      - name: Get the output time
        run: echo "The time was ${{ steps.hello.outputs.time }}"
```

From your repository, click the **Actions** tab, and select the latest workflow run. Under **Jobs** or in the visualization graph, click **A job to say hello**.

Click **Hello world action step**, and you should see "Hello Mona the Octocat" or the name you used for the `who-to-greet` input printed in the log. To see the timestamp, click **Get the output time**.

Template repositories for creating JavaScript actions

GitHub provides template repositories for creating JavaScript and TypeScript actions. You can use these templates to quickly get started with creating a new action that includes tests, linting, and other recommended practices.

- `javascript-action` template repository
- `typescript-action` template repository

Example JavaScript actions on GitHub.com

You can find many examples of JavaScript actions on GitHub.com.

- DevExpress/testcafe-action
- duckduckgo/privacy-configuration

Previous
Creating a Docker container action

Next
Creating a composite action