Essential features of GitHub Actions

In this article

GitHub Actions are designed to help you build robust and dynamic automations. This guide will show you how to craft GitHub Actions workflows that include environment variables, customized scripts, and more.

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.

Overview

GitHub Actions allow you to customize your workflows to meet the unique needs of your application and team. In this guide, we'll discuss some of the essential customization techniques such as using variables, running scripts, and sharing data and artifacts between jobs.

Using variables in your workflows

GitHub Actions include default environment variables for each workflow run. If you need to use custom environment variables, you can set these in your YAML workflow file. This example demonstrates how to create custom variables named `POSTGRES_HOST` and `POSTGRES_PORT`. These variables are then available to the `node client.js` script.

```
jobs:
  example-job:
      steps:
        - name: Connect to PostgreSQL
          run: node client.js
          env:
            POSTGRES_HOST: postgres
            POSTGRES_PORT: 5432
```

For more information, see "Variables."

Adding scripts to your workflow

You can use actions to run scripts and shell commands, which are then executed on the assigned runner. This example demonstrates how an action can use the `run` keyword to execute `npm install -g bats` on the runner.

```yaml
jobs:
  example-job:
    steps:
      - run: npm install -g bats
```

For example, to run a script as an action, you can store the script in your repository and supply the path and shell type.

```yaml
jobs:
  example-job:
    steps:
      - name: Run build script
        run: ./.github/scripts/build.sh
        shell: bash
```

For more information, see "Workflow syntax for GitHub Actions."

Sharing data between jobs

If your job generates files that you want to share with another job in the same workflow, or if you want to save the files for later reference, you can store them in GitHub as *artifacts*. Artifacts are the files created when you build and test your code. For example, artifacts might include binary or package files, test results, screenshots, or log files. Artifacts are associated with the workflow run where they were created and can be used by another job. All actions and workflows called within a run have write access to that run's artifacts.

For example, you can create a file and then upload it as an artifact.

```yaml
jobs:
  example-job:
    name: Save output
    steps:
      - shell: bash
        run: |
          expr 1 + 1 > output.log
      - name: Upload output file
        uses: actions/upload-artifact@v2
        with:
          name: output-log-file
          path: output.log
```

To download an artifact from a separate workflow run, you can use the `actions/download-artifact` action. For example, you can download the artifact named `output-log-file`.

```yaml
jobs:
  example-job:
    steps:
      - name: Download a single artifact
        uses: actions/download-artifact@v2
        with:
          name: output-log-file
```

To download an artifact from the same workflow run, your download job should specify `needs: upload-job-name` so it doesn't start until the upload job finishes.

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

Next steps

To continue learning about GitHub Actions, see "About workflows."