Deploying with GitHub Actions

In this article

Learn how to control deployments with features like environments and concurrency.

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

GitHub Actions offers features that let you control deployments. You can:

- Trigger workflows with a variety of events.
- Configure environments to set rules before a job can proceed and to limit access to secrets.
- Use concurrency to control the number of deployments running at a time.

For more information about continuous deployment, see "About continuous deployment."

Prerequisites

You should be familiar with the syntax for GitHub Actions. For more information, see "Learn GitHub Actions."

Triggering your deployment

You can use a variety of events to trigger your deployment workflow. Some of the most common are: `pull_request`, `push`, and `workflow_dispatch`.

For example, a workflow with the following triggers runs whenever:

- There is a push to the `main` branch.
- A pull request targeting the `main` branch is opened, synchronized, or reopened.

- Someone manually triggers it.

```yaml
on:
  push:
    branches:
      - main
  pull_request:
    branches:
      - main
  workflow_dispatch:
```

For more information, see "Events that trigger workflows."

Using environments

Environments are used to describe a general deployment target like `production`, `staging`, or `development`. When a GitHub Actions workflow deploys to an environment, the environment is displayed on the main page of the repository. You can use environments to require approval for a job to proceed, restrict which branches can trigger a workflow, gate deployments with custom deployment protection rules, or limit access to secrets. For more information about creating environments, see "Using environments for deployment."

Using concurrency

Concurrency ensures that only a single job or workflow using the same concurrency group will run at a time. You can use concurrency so that an environment has a maximum of one deployment in progress and one deployment pending at a time.

> **Note:** `concurrency` and `environment` are not connected. The concurrency value can be any string; it does not need to be an environment name. Additionally, if another workflow uses the same environment but does not specify concurrency, that workflow will not be subject to any concurrency rules.

For example, when the following workflow runs, it will be paused with the status `pending` if any job or workflow that uses the `production` concurrency group is in progress. It will also cancel any job or workflow that uses the `production` concurrency group and has the status `pending`. This means that there will be a maximum of one running and one pending job or workflow in that uses the `production` concurrency group.

```yaml
name: Deployment

concurrency: production

on:
  push:
    branches:
      - main

jobs:
  deployment:
    runs-on: ubuntu-latest
    environment: production
    steps:
      - name: deploy
        # ...deployment-specific steps
```

You can also specify concurrency at the job level. This will allow other jobs in the workflow to proceed even if the concurrent job is `pending`.

```yaml
name: Deployment

on:
  push:
    branches:
      - main

jobs:
  deployment:
    runs-on: ubuntu-latest
    environment: production
    concurrency: production
    steps:
      - name: deploy
        # ...deployment-specific steps
```

You can also use `cancel-in-progress` to cancel any currently running job or workflow in the same concurrency group.

```yaml
name: Deployment

concurrency:
  group: production
  cancel-in-progress: true

on:
  push:
    branches:
      - main

jobs:
  deployment:
    runs-on: ubuntu-latest
    environment: production
    steps:
      - name: deploy
        # ...deployment-specific steps
```

For guidance on writing deployment-specific steps, see "Finding deployment examples."

Viewing deployment history

When a GitHub Actions workflow deploys to an environment, the environment is displayed on the main page of the repository. For more information about viewing deployments to environments, see "Viewing deployment history."

Monitoring workflow runs

Every workflow run generates a real-time graph that illustrates the run progress. You can use this graph to monitor and debug deployments. For more information see, "Using the visualization graph."

You can also view the logs of each workflow run and the history of workflow runs. For more information, see "Viewing workflow run history."

Tracking deployments through apps

You can also build an app that uses deployment and deployment status webhooks to track deployments. When a workflow job that references an environment runs, it creates a deployment object with the `environment` property set to the name of your

environment. As the workflow progresses, it also creates deployment status objects with the `environment` property set to the name of your environment, the `environment_url` property set to the URL for environment (if specified in the workflow), and the `state` property set to the status of the job. For more information, see "GitHub Apps documentation" and "Webhook events and payloads."

Choosing a runner

You can run your deployment workflow on GitHub-hosted runners or on self-hosted runners. Traffic from GitHub-hosted runners can come from a wide range of network addresses. If you are deploying to an internal environment and your company restricts external traffic into private networks, GitHub Actions workflows running on GitHub-hosted runners may not be able to communicate with your internal services or resources. To overcome this, you can host your own runners. For more information, see "About self-hosted runners" and "Using GitHub-hosted runners."

Displaying a status badge

You can use a status badge to display the status of your deployment workflow. A status badge shows whether a workflow is currently failing or passing. A common place to add a status badge is in the `README.md` file of your repository, but you can add it to any web page you'd like. By default, badges display the status of your default branch. You can also display the status of a workflow run for a specific branch or event using the `branch` and `event` query parameters in the URL.

For more information, see "Adding a workflow status badge."

Finding deployment examples

This article demonstrated features of GitHub Actions that you can add to your deployment workflows.

GitHub Enterprise Server offers deployment starter workflows for several popular services, such as Azure Web App. To learn how to get started using a starter workflow, see "Using starter workflows" or browse the full list of deployment starter workflows. You can also check out our more detailed guides for specific deployment workflows, such as "Deploying Node.js to Azure App Service."

Many service providers also offer actions on GitHub Marketplace for deploying to their service. For the full list, see GitHub Marketplace.