Using environments for deployment

You can configure environments with protection rules and secrets. A workflow job that references an environment must follow any protection rules for the environment before running or accessing the environment's secrets.

> Environments, environment secrets, and deployment protection rules are available in public repositories for all current GitHub plans. They are not available on legacy plans, such as Bronze, Silver, or Gold. For access to environments, environment secrets, and deployment branches in private or internal repositories, you must use GitHub Pro, GitHub Team, or GitHub Enterprise.

About 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. For more information about viewing deployments to environments, see "[Viewing deployment history](#)."

You can configure environments with protection rules and secrets. When a workflow job references an environment, the job won't start until all of the environment's protection rules pass. A job also cannot access secrets that are defined in an environment until all the deployment protection rules pass.

Optionally, you can bypass an environment's protection rules and force all pending jobs referencing the environment to proceed. For more information, see "[Reviewing deployments](#)."

Deployment protection rules ⚭

Deployment protection rules require specific conditions to pass before a job referencing the environment can proceed. You can use deployment protection rules to require a manual approval, delay a job, or restrict the environment to certain branches. You can also create and implement custom protection rules powered by GitHub Apps to use third-party systems to control deployments referencing environments configured on your

GitHub Enterprise Server instance.

Third-party systems can be observability systems, change management systems, code quality systems, or other manual configurations that you use to assess readiness before deployments are safely rolled out to environments.

> **Note:** Any number of GitHub Apps-based deployment protection rules can be installed on a repository. However, a maximum of 6 deployment protection rules can be enabled on any environment at the same time.

Required reviewers 🔗

Use required reviewers to require a specific person or team to approve workflow jobs that reference the environment. You can list up to six users or teams as reviewers. The reviewers must have at least read access to the repository. Only one of the required reviewers needs to approve the job for it to proceed.

For more information on reviewing jobs that reference an environment with required reviewers, see "[Reviewing deployments](#)."

Wait timer 🔗

Use a wait timer to delay a job for a specific amount of time after the job is initially triggered. The time (in minutes) must be an integer between 0 and 43,200 (30 days).

Deployment branches 🔗

Use deployment branches to restrict which branches can deploy to the environment. Below are the options for deployment branches for an environment:

- **All branches**: All branches in the repository can deploy to the environment.

- **Protected branches**: Only branches with branch protection rules enabled can deploy to the environment. If no branch protection rules are defined for any branch in the repository, then all branches can deploy. For more information about branch protection rules, see "[About protected branches](#)."

- **Selected branches**: Only branches that match your specified name patterns can deploy to the environment.

  If you specify `releases/*` as a deployment branch rule, only a branch whose name begins with `releases/` can deploy to the environment. (Wildcard characters will not match `/`. To match branches that begin with `release/` and contain an additional single slash, use `release/*/*`.) If you add `main` as a branch rule, a branch named `main` can also deploy to the environment. For more information about syntax options for deployment branches, see the [Ruby `File.fnmatch` documentation](#).

Allow administrators to bypass configured protection rules 🔗

By default, administrators can bypass the protection rules and force deployments to specific environments. For more information, see "[Reviewing deployments](#)."

Alternatively, you can configure environments to disallow bypassing the protection rules for all deployments to the environment.

Custom deployment protection rules 🔗

> **Note:** Custom deployment protection rules are currently in public beta and subject to change.

You can enable your own custom protection rules to gate deployments with third-party services. For example, you can use services such as Datadog, Honeycomb, and ServiceNow to provide automated approvals for deployments to your GitHub Enterprise Server instance. For more information, see "[Creating custom deployment protection rules](#)".

Once custom deployment protection rules have been created and installed on a repository, you can enable the custom deployment protection rule for any environment in the repository. For more information about configuring and enabling custom deployment protection rules, see "[Configuring custom deployment protection rules](#)."

Environment secrets 🔗

Secrets stored in an environment are only available to workflow jobs that reference the environment. If the environment requires approval, a job cannot access environment secrets until one of the required reviewers approves it. For more information about secrets, see "[Using secrets in GitHub Actions](#)."

> **Note:** Workflows that run on self-hosted runners are not run in an isolated container, even if they use environments. Environment secrets should be treated with the same level of security as repository and organization secrets. For more information, see "[Security hardening for GitHub Actions](#)."

Environment variables 🔗

Variables stored in an environment are only available to workflow jobs that reference the environment. These variables are only accessible using the `vars` context. For more information, see "[Variables](#)."

Creating an environment 🔗

To configure an environment in a personal account repository, you must be the repository owner. To configure an environment in an organization repository, you must have `admin` access.

1. On your GitHub Enterprise Server instance, navigate to the main page of the repository.

2. Under your repository name, click ⚙ **Settings**. If you cannot see the "Settings" tab, select the ••• dropdown menu, then click **Settings**.
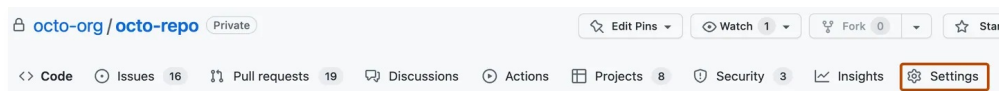
3. In the left sidebar, click **Environments**.

4. Click **New environment**.

5. Enter a name for the environment, then click **Configure environment**. Environment names are not case sensitive. An environment name may not exceed 255 characters and must be unique within the repository.

6. Optionally, specify people or teams that must approve workflow jobs that use this environment. For more information, see "[Required reviewers](#)."

a. Select **Required reviewers**.

b. Enter up to 6 people or teams. Only one of the required reviewers needs to approve the job for it to proceed.

c. Click **Save protection rules**.

7 Optionally, specify the amount of time to wait before allowing workflow jobs that use this environment to proceed. For more information, see "Wait timer."

a. Select **Wait timer**.

b. Enter the number of minutes to wait.

c. Click **Save protection rules**.

8 Optionally, disallow bypassing configured protection rules. For more information, see "Allow administrators to bypass configured protection rules."

a. Deselect **Allow administrators to bypass configured protection rules**.

b. Click **Save protection rules**.

9 Optionally, enable any custom deployment protection rules that have been created with GitHub Apps. For more information, see "Custom deployment protection rules."

a. Select the custom protection rule you want to enable.

b. Click **Save protection rules**.

10 Optionally, specify what branches can deploy to this environment. For more information, see "Deployment branches."

a. Select the desired option in the **Deployment branches** dropdown.

b. If you chose **Selected branches**, to add a new rule, click **Add deployment branch rule**

c. Enter the name pattern for the branch that you want to allow.

d. Click **Add rule**.

11 Optionally, add environment secrets. These secrets are only available to workflow jobs that use the environment. Additionally, workflow jobs that use this environment can only access these secrets after any configured rules (for example, required reviewers) pass. For more information, see "Environment secrets."

a. Under **Environment secrets**, click **Add Secret**.

b. Enter the secret name.

c. Enter the secret value.

d. Click **Add secret**.

12. Optionally, add environment variables. These variables are only available to workflow jobs that use the environment, and are only accessible using the `vars` context. For more information, see "Environment variables."

   a. Under **Environment variables**, click **Add Variable**.

   b. Enter the variable name.

   c. Enter the variable value.

   d. Click **Add variable**.

You can also create and configure environments through the REST API. For more information, see "Deployment environments," "GitHub Actions Secrets," "GitHub Actions Variables," and "Deployment branch policies."

Running a workflow that references an environment that does not exist will create an environment with the referenced name. The newly created environment will not have any protection rules or secrets configured. Anyone that can edit workflows in the repository can create environments via a workflow file, but only repository admins can configure the environment.

Using an environment 🔗

Each job in a workflow can reference a single environment. Any protection rules configured for the environment must pass before a job referencing the environment is sent to a runner. The job can access the environment's secrets only after the job is sent to a runner.

When a workflow references an environment, the environment will appear in the repository's deployments. For more information about viewing current and previous deployments, see "Viewing deployment history."

You can specify an environment for each job in your workflow. To do so, add a `jobs.<job_id>.environment` key followed by the name of the environment.

For example, this workflow will use an environment called `production`.

```
name: Deployment

on:
  push:
    branches:
      - main

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

When the above workflow runs, the `deployment` job will be subject to any rules configured for the `production` environment. For example, if the environment requires reviewers, the job will pause until one of the reviewers approves the job.

You can also specify a URL for the environment. The specified URL will appear on the

deployments page for the repository (accessed by clicking **Environments** on the home page of your repository) and in the visualization graph for the workflow run. If a pull request triggered the workflow, the URL is also displayed as a **View deployment** button in the pull request timeline.

```
name: Deployment

on:
  push:
    branches:
      - main

jobs:
  deployment:
    runs-on: ubuntu-latest
    environment:
      name: production
      url: https://github.com
    steps:
      - name: deploy
        # ...deployment-specific steps
```

Deleting an environment 🔗

To configure an environment in a personal account repository, you must be the repository owner. To configure an environment in an organization repository, you must have `admin` access.

Deleting an environment will delete all secrets and protection rules associated with the environment. Any jobs currently waiting because of protection rules from the deleted environment will automatically fail.

1. On your GitHub Enterprise Server instance, navigate to the main page of the repository.

2. Under your repository name, click ⚙ **Settings**. If you cannot see the "Settings" tab, select the ••• dropdown menu, then click **Settings**.
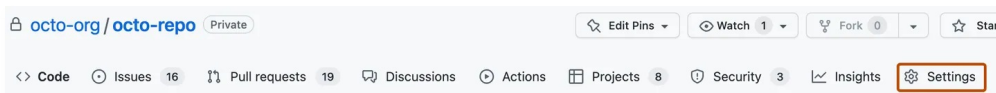
3. In the left sidebar, click **Environments**.

4. Next to the environment that you want to delete, click 🗑.

5. Click **I understand, delete this environment**.

You can also delete environments through the REST API. For more information, see "[Repositories](#)."

How environments relate to 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.

You can access these objects through the REST API or GraphQL API. You can also subscribe to these webhook events. For more information, see "[Repositories](#)" (REST API), "[Objects](#)" (GraphQL API), or "[Webhook events and payloads](#)."

Next steps 🔗

GitHub Actions provides several features for managing your deployments. For more information, see "[Deploying with GitHub Actions](#)."