Environments in GitHub Actions

Environments in GitHub Actions are a powerful feature that allow you to configure and manage deployment targets, such as production, staging, or development. They provide a way to ensure that your workflows adhere to protection rules and have access to specific secrets and environment variables.

About Environments

Environments are used to describe deployment targets, such as production or staging. When a GitHub Actions workflow deploys to an environment, the environment's name is displayed on the repository's main page, making it easy to track deployment history.

Protection Rules

Protection rules help ensure the security and reliability of your deployments. GitHub Actions provides several protection rule options:

- **Required Reviewers**: You can specify individuals or teams that must approve workflow jobs before they can proceed. Only one of the required reviewers needs to approve the job for it to proceed.
- **Wait Timer**: Delay a job for a specific amount of time after it is triggered. The wait time can be between 0 and 43,200 minutes (30 days).
- **Deployment Branches and Tags**: Restrict which branches and tags can deploy to the environment. You can specify patterns for branch and tag names to control deployment access.
- **Custom Deployment Protection Rules**: Use third-party services to automate approvals and assessments for deployments. You can create custom deployment protection rules powered by GitHub Apps.

Environment Secrets

Secrets stored in an environment are only available to workflow jobs that reference the environment. Secrets can only be accessed after any protection rules pass. Treat environment secrets with the same level of security as repository and organization secrets.

Environment Variables

Variables stored in an environment are accessible using the `vars` context and are only available to workflow jobs that reference the environment. Environment variables are available for all public repositories, and for users on GitHub Pro or GitHub Team plans, they are also available for private repositories.

Creating an Environment

To create an environment, you must have specific permissions based on the repository's visibility:

- Personal Account Repository: Repository owner can configure environments.
- Organization Repository: Admins can configure environments.

Using an Environment

Each job in a workflow can reference a single environment. Jobs that reference an environment won't start until all protection rules pass. They can access environment secrets and variables only after the job is sent to a runner.

Deleting an Environment

Deleting an environment will remove all associated secrets and protection rules. Any jobs waiting due to protection rules will fail. Only repository admins can configure environments, but anyone who can edit workflows can create environments.

How Environments Relate to Deployments

When a workflow job references an environment, it creates a deployment object with the environment property set to the name of the environment. Deployment status objects are also created, providing details about the job's progress. These objects can be accessed via the REST API, GraphQL API, or webhook events.

For more detailed information on environments, protection rules, and best practices, please refer to GitHub Actions Documentation.

Note: Availability of certain features for private repositories may vary based on your GitHub plan.

Testing Environments in Workflow

1. Create Workflow Files:

- Create two new YAML files named `19-01-Environments.yml` and `19-02-Environments.yml` in the `.github/workflows` directory of your repository.
- **19-01-Environments.yml**

```yaml
name: 19-01-Environments

on:
  workflow_dispatch:
    inputs:
      target-env:
        type: environment
        default: staging
```

```yaml
jobs:
  echo:
    runs-on: ubuntu-latest
    environment: ${{ inputs.target-dev }}
    env:
      my-env-value: ${{ vars.MY_ENV_VALUE || 'default value' }}
    steps:
      - name: Echo vars
        run: |
          echo "Env variable: ${{ env.my-env-value }}"
```

- **19-02-Environments.yml**

```yaml
name: 19-02-Environments

on:
  workflow_dispatch:

jobs:
  deploy-staging:
    runs-on: ubuntu-latest
    environment: staging
    env:
      my-env-value: ${{ vars.MY_ENV_VALUE || 'default value' }}
    steps:
      - name: Echo vars
        run: |
          echo "Env variable: ${{ env.my-env-value }}"
  e2e-tests:
    runs-on: ubuntu-latest
    needs: deploy-staging
    steps:
      - name: E2E tests
        run: echo "Running E2e"
  deploy-prod-frontend:
    runs-on: ubuntu-latest
    needs: e2e-tests
    environment: prod
    env:
      my-env-value: ${{ vars.MY_ENV_VALUE || 'default value' }}
    steps:
      - name: Echo vars
        run: |
          echo "Deploying prod frontend"
  deploy-prod-backend1:
    runs-on: ubuntu-latest
    needs: e2e-tests
    environment: prod
    env:
      my-env-value: ${{ vars.MY_ENV_VALUE || 'default value' }}
    steps:
```

```
      - name: Echo vars
        run: |
          echo "Deploying prod backend 1"
  deploy-prod-backend2:
    runs-on: ubuntu-latest
    needs: e2e-tests
    environment: prod
    env:
      my-env-value: ${{ vars.MY_ENV_VALUE || 'default value' }}
    steps:
      - name: Echo vars
        run: |
          echo "Deploying prod backend 2"
```

- Copy and paste the provided YAML configurations into these files.

2. Understanding the Workflows:

- **19-01-Environments.yml**
  - This workflow is triggered manually via the GitHub UI (`workflow_dispatch`).
  - It allows the user to specify a target environment as an input parameter (`target-env`).
  - The job `echo` runs on `ubuntu-latest` and is associated with the specified target environment.
  - It sets an environment variable `my-env-value` using a default value if not defined.
- **19-02-Environments.yml**
  - This workflow is triggered manually via the GitHub UI (`workflow_dispatch`).
  - It contains multiple jobs for deploying to different environments: staging and production (`prod`).
  - Each job runs on `ubuntu-latest` and is associated with its respective environment.
  - Each job sets an environment variable `my-env-value` using a default value if not defined.

3. Testing the Workflows:

- Commit and push the workflow files (`19-01-Environments.yml` and `19-02-Environments.yml`) to your repository.
- Navigate to the "Actions" tab in your GitHub repository.
- Manually trigger the workflows by clicking on the "Run workflow" button for each workflow.
- Monitor the workflow runs and verify that the jobs execute successfully, echoing the environment variables as expected.