

Using GitHub Actions with Substrate

Downloaded from Epic Games Confluence

Date: 2025-07-12 04:07:01

Original URL: <https://confluence-epicgames.atlassian.net/wiki/spaces/CDE/pages/81068367>

Document Level Classification

[300](#)

- [Introduction](#)
- [How can I access GitHub Actions?](#)
- [How would I leverage GitHub actions to deploy my applications and services to Substrate?](#)
- [How do I configure Github Actions to authenticate with my substrate account\(s\)?](#)
- [Can I use Github Actions to deploy infrastructure using Terraform? What is an example workflow?](#)
- [How can I use GitHub Actions to do canary deployments to Substrate?](#)
- [Can I use Github Actions to deploy Services to the Substrate Cluster? What's an example of a full, end to end workflow?](#)
- [Preventing broken deployments when stopping Helm](#)
- [How can I use GitHub Actions to deploy my applications to different environments?](#)
- [How environments relate to deployments](#)

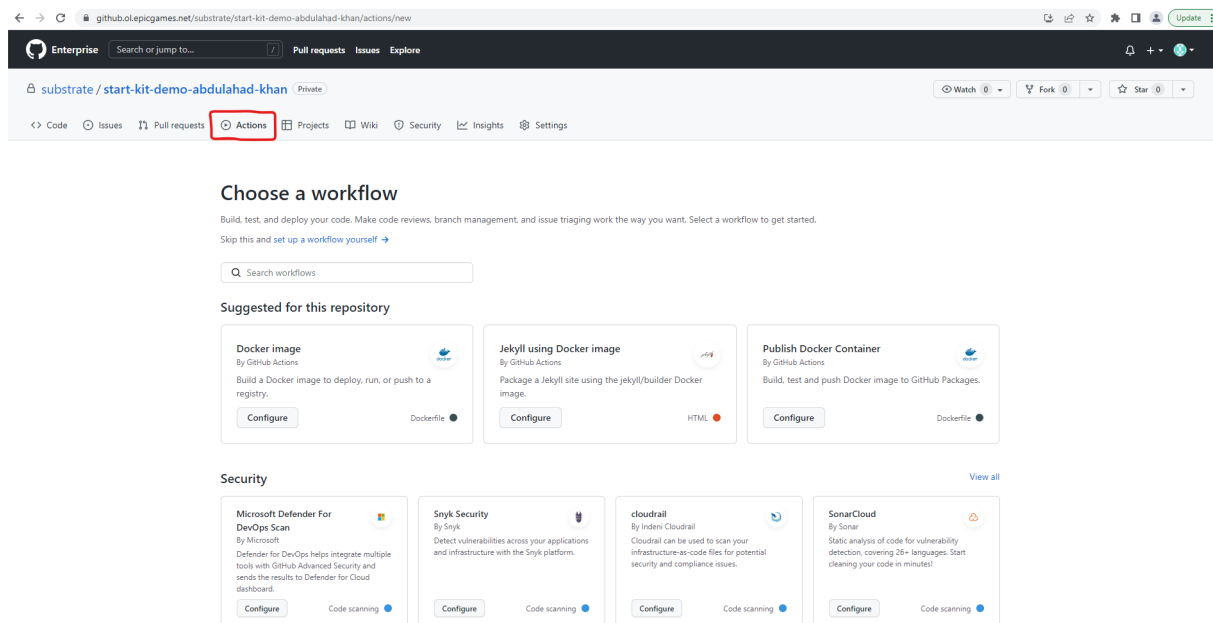
- [Troubleshooting / Error Handling](#)
- [Where to get GitHub Help?](#)
- [Additional Resources](#)
 - [GitHub Actions Documentation:](#)
 - [Epic Games GitHub Actions User Guide](#)

Introduction

When using Substrate EKS Clusters, we need a CI/CD solution to deploy changes to said clusters. We can use CodeFresh for our pipelines but GitHub Actions also offers similar core functionality. In this article we will look at specific code blocks within a GitHub workflow file to authenticate to and deploy our infrastructure, with different deployment options.

How can I access GitHub Actions?

Ensure you are on the VPN, then navigate to <https://github.ol.epicgames.net/>. On the left, you should see a column that mentions Top Repositories. Navigate to any of the repositories that you have access to. Then select the 'Actions' tab as shown below and you can choose a pre-created workflow, or you can just create one yourself.



GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository, or deploy merged pull requests to production.

How would I leverage GitHub actions to deploy my applications and services to Substrate?

If GitHub is already used for source code, setting up GitHub Actions becomes simplified.

Substrate accounts use Github as a source repository, and GitHub Actions only supports Github as a source repository. Simply run your terraform commands from within Github Action Workflows and your AWS or other resources will be deployed to Substrate.

How do I configure Github Actions to authenticate with my substrate account(s)?

You can connect to a Substrate EKS Cluster via GitHub Actions.

Utilize GitHub secrets to store your vault secrets and reference them in your step like so:

```
steps:
- name: configure kubernetes access
  uses: online-web/configure-kubernetes-action@v1
  with:
    vaultRoleId: ${{ secrets.VAULT_ROLE_ID }}
    vaultSecretId: ${{ secrets.VAULT_SECRET_ID }}
    clusterName: bebe-dev-eos-dev-portal
    namespace: team-dev-portal
```

Then, you can create another step and check that your access to the Substrate clusters is valid:

```
- name: verify
  run: |
    curl -LO "https://dl.k8s.io/release/v1.25.0/bin/linux/amd64/kubectl"
    chmod +x kubectl
    ./kubectl get pods
```

Reference: [GitHub Actions User Guide](#)

Can I use Github Actions to deploy infrastructure using Terraform? What is an example workflow?

Yes! That is precisely what GitHub Actions should be doing. Create a Workflow within GitHub Actions and place your usual Terraform commands within this Workflow. See the following example:

```
defaults:
  run:
    working-directory: ${ env.tf_actions_working_dir }
    permissions:
      pull-requests: write
    steps:
      - uses: actions/checkout@v3
      - uses: hashicorp/setup-terraform@v2

      - name: Terraform fmt
        id: fmt
        run: terraform fmt -check

      - name: Terraform Init
        id: init
        run: terraform init

      - name: Terraform Validate
        id: validate
        run: terraform validate -no-color

      - name: Terraform Plan
        id: plan
        run: terraform plan -no-color
```

For example you can define a steps clause and name a few different steps with which commands to run. Run commands such as terraform format, terraform validate, terraform plan, and finally apply.

How can I use GitHub Actions to do canary deployments to Substrate?

Currently there doesn't seem to be a built-in method of doing a canary deployment using GitHub Actions unless Azure's AKS is in use.

Reference: <https://github.com/Azure-Samples/aks-bluegreen-canary>

How can I use GitHub Actions to do blue/green deployments to Substrate?

GitHub Actions can be used to do Blue Green deployments, although not natively (as of this writing). One method to do this is to store an environment variable that the current environment points to.

Run your standard Terraform commands to deploy your infrastructure and flip the value of the color. The value of the color can be stored in AWS SSM Parameter Store.

Essentially, once the terraform apply command is successful, simply switch the color that is stored, via a CLI command in the workflow:

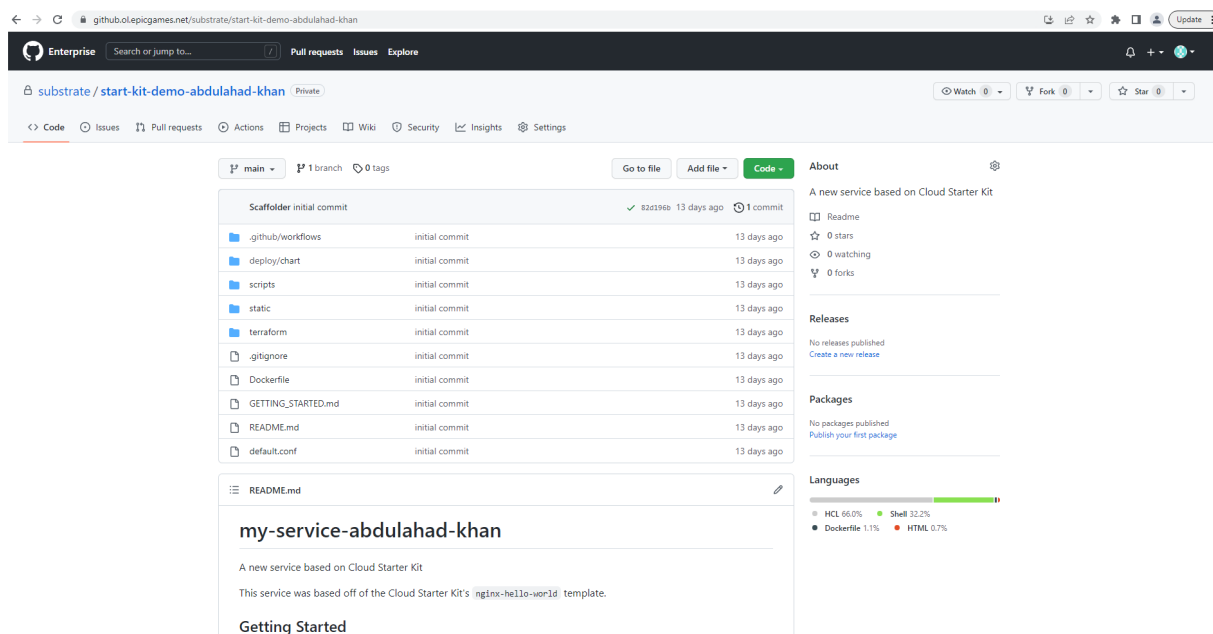
```
aws ssm put-parameter --name "BlueGreenStatus" --type "String" --value
```

If it is blue it should become green and vice versa. You can run an automated test from within GitHub Actions, and if successful, delete the old infra and switch the colors back.

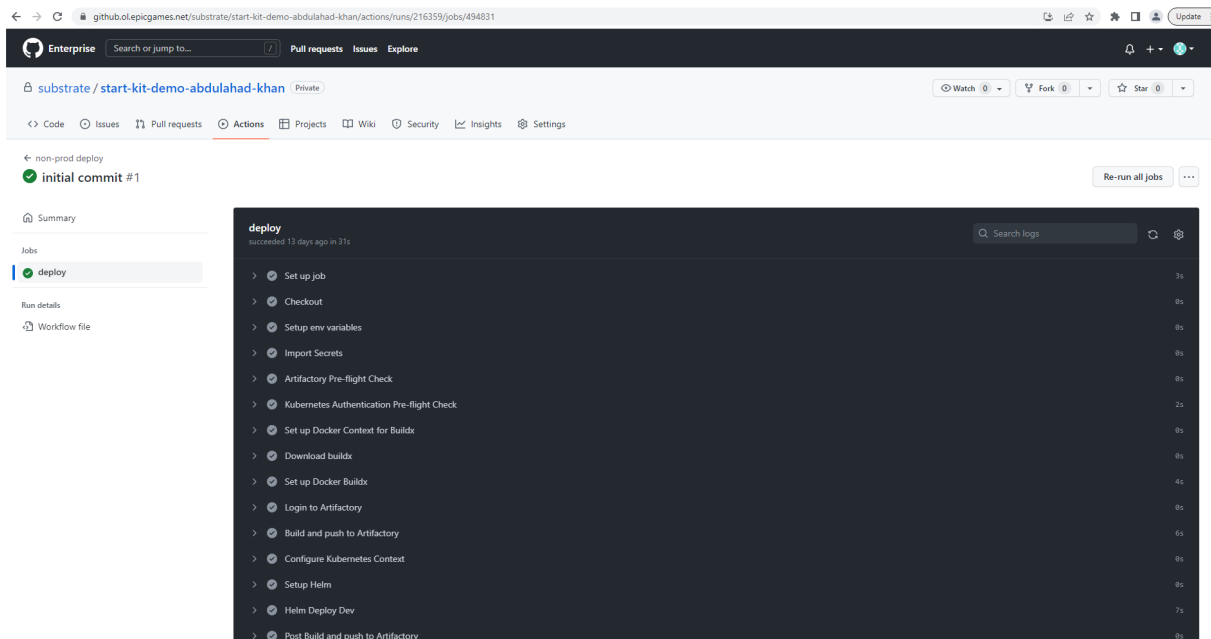
Can I use Github Actions to deploy Services to the Substrate Cluster? What's an example of a full, end to end workflow?

Yes, you can use the [Cloud Starter Kit](#) to deploy a new service to the Substrate cluster. When choosing your service, select GitHub Actions as the CI/CD process.

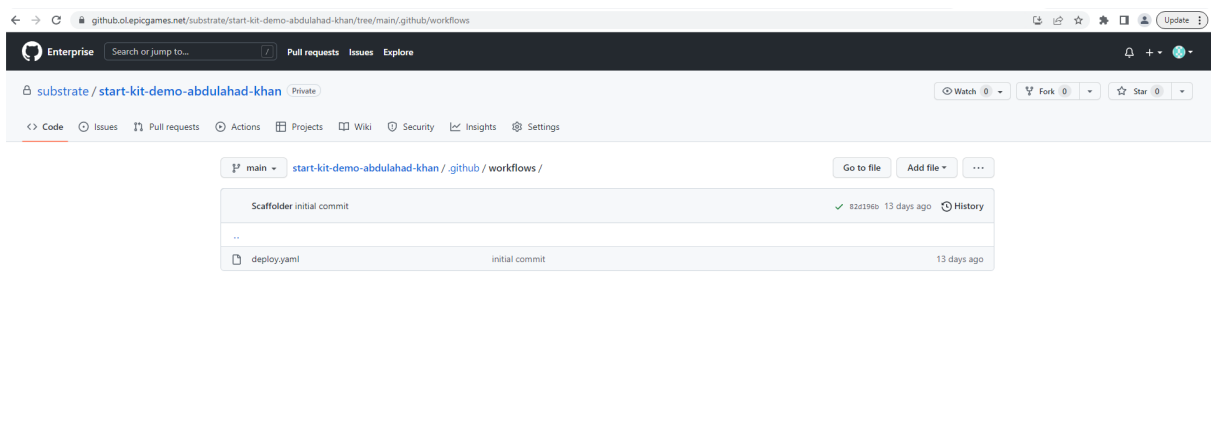
Once you create your application to deploy on Substrate, you'll see that a Repo has been created for your Cloud Starter Kit.



Click on the 'Actions' tab and you'll see a workflow that was triggered as part of the initial commit. In this case, the green checkmark to the side shows that it was successful. The sub-stages shown below (Set up Job, Checkout, Setup env variables, etc...) were also individually successful. If even one of them had failed, the whole workflow would've failed, at least in this case.



The workflow gets automatically run upon any code being pushed to the repo. This is due to the presence of a workflow file within the workflows folder (the name of the file is immaterial):



The workflow file itself may look like this, at least for the Cloud Starter Kit.


```

name: non-prod deploy
on:
  push:
    branches: [ main ]
  workflow_dispatch:
jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Setup env variables
        run: |-
          scripts/set-env.sh

      - name: Import Secrets
        id: secrets
        uses: actions/hashicorp_vault-action@v2.4.1
        with:
          url: https://vault.substrate.on.epicgames.com
          method: approle
          exportEnv: false
          exportToken: false
          roleId: ${ secrets.VAULT_ROLE_ID }
          secretId: ${ secrets.VAULT_SECRET_ID }
          secrets: |
            secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${ secrets.VAULT_SECRET_ID }
            secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${ secrets.VAULT_SECRET_ID }

            # This pre-flight check is designed to help debug access issues w
            # after you have your service deployed successfully.
            # Checks that we have write access to docker repositories from th
      - name: Artifactory Pre-flight Check
        working-directory: .
        env:
          JF_ACCESS_TOKEN: ${ steps.secrets.outputs.JF_ACCESS_TOKEN }"

```

```

run: |
  scripts/preflight-artifactory.sh

  # This pre-flight check is designed to help debug access issues
  # after you have your service deployed successfully.
  # Checks that we can read the deployer credentials for the user's a
- name: Kubernetes Authentication Pre-flight Check
  working-directory: .
  env:
    VAULT_ROLE_ID: ${ secrets.VAULT_ROLE_ID }
    VAULT_SECRET_ID: ${ secrets.VAULT_SECRET_ID }
  run: |
    scripts/preflight-vault.sh

- name: Set up Docker Context for Buildx
  id: buildx-context
  run: |
    docker context create builders
    docker context use builders

- name: Download buildx
  run: |
    mkdir -p ~/.docker/cli-plugins
    wget https://github.com/docker/buildx/releases/download/v0.10.
    chmod +x ~/.docker/cli-plugins/docker-buildx

- name: Set up Docker Buildx
  id: builder1
  uses: actions/docker_setup-buildx-action@v2
  with:
    endpoint: builders

- name: Login to Artifactory
  uses: actions/docker_login-action@v2
  with:
    registry: https://artifacts.01.epicgames.net
    username: ${ steps.secrets.outputs.JF_USERNAME }

```

```

        password: ${{ steps.secrets.outputs.JF_ACCESS_TOKEN }}

- name: Build and push to Artifactory
  uses: actions/docker_build-push-action@v3
  with:
    file: ./Dockerfile
    push: true
    tags: ${{ env.DEV_DOCKER_TAG }}

- name: Configure Kubernetes Context
  uses: online-web/configure-kubernetes-action@v1
  with:
    vaultRoleId: ${{ secrets.VAULT_ROLE_ID }}
    vaultSecretId: ${{ secrets.VAULT_SECRET_ID }}
    clusterName: ${{ env.NPE_CLUSTER_NAME }}
    namespace: ${{ env.NPE_NAMESPACE }}

- name: Setup Helm
  uses: actions/Azure_setup-helm@v3
  with:
    version: 'v3.7.1'
    id: install

- name: Helm Deploy Dev
  run: |
    helm repo add substr-helm https://artifacts.ol.epicgames.net
    helm dependency update ./deploy/chart
    helm upgrade ${{ env.SERVICE_NAME }}-dev ./deploy/chart \
      --install \
      --namespace ${{ env.NPE_NAMESPACE }} \
      --values "./deploy/chart/values.gamedev.yaml" \
      --set "epic-app.imageTag=${{ env.SHORT_SHA }}" \
      --atomic \
      --wait \
      --timeout 5m \

```

There is a 'steps' level under which each individual 'sub-step' gets run in the order shown. You will notice that the names of each steps matches with the sub-steps shown in the workflow run (second screenshot in this section).

Preventing broken deployments when stopping Helm

If you press Cancel on a running workflow while the Helm upgrade command is running, the running workflow will be killed and the Helm release will be stuck in a bad state. To prevent this, the use of the `always()` function in the workflow can prevent the Cancel button from killing the running workflow.

```
name: non-prod deploy
on:
  push:
    branches: [ main ]
  workflow_dispatch:
jobs:
  deploy:
    # Prevent the Cancel Button from stopping Helm installs
    if: always()
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3
    ....
```

In most cases, it is better to separate a large pipeline into separate jobs, butting the Helm steps in it's own job that uses the `always()` function. This allows you to Cancel workflow earlier in the run if an undesired step outcome has occurred, allow you to prevent the Helm deployment from taking place. When the deploy job starts, the Cancel button will have no affect, therefore preventing the Helm release from getting into a bad state.

```

name: non-prod deploy
on:
  push:
    branches: [ main ]
  workflow_dispatch:
jobs:
  prepare:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Setup env variables
        run: |-
          scripts/set-env.sh

      - name: Import Secrets
        id: secrets
        uses: actions/hashicorp_vault-action@v2.4.1
        with:
          url: https://vault.substrate.on.epicgames.com
          method: approle
          exportEnv: false
          exportToken: false
          roleId: ${{ secrets.VAULT_ROLE_ID }}
          secretId: ${{ secrets.VAULT_SECRET_ID }}
          secrets: |
            secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${{ g
            secret/data/ops/ghe-orgs/global/live/runtime/artifactory-${{ g

      # This pre-flight check is designed to help debug access issues w
      # after you have your service deployed successfully.
      # Checks that we have write access to docker repositories from th
      - name: Artifactory Pre-flight Check
        working-directory: .
        env:
          JF_ACCESS_TOKEN: ${{ steps.secrets.outputs.JF_ACCESS_TOKEN }}"

```

```

run: |
    scripts/preflight-artifactory.sh

    # This pre-flight check is designed to help debug access issues
    # after you have your service deployed successfully.
    # Checks that we can read the deployer credentials for the user's artifact
    - name: Kubernetes Authentication Pre-flight Check
      working-directory: .
      env:
        VAULT_ROLE_ID: ${ secrets.VAULT_ROLE_ID }
        VAULT_SECRET_ID: ${ secrets.VAULT_SECRET_ID }
      run: |
        scripts/preflight-vault.sh

- name: Set up Docker Context for Buildx
  id: buildx-context
  run: |
    docker context create builders
    docker context use builders

- name: Download buildx
  run: |
    mkdir -p ~/.docker/cli-plugins
    wget https://github.com/docker/buildx/releases/download/v0.10.0/docker-buildx-v0.10.0-linux-amd64
    chmod +x ~/.docker/cli-plugins/docker-buildx

- name: Set up Docker Buildx
  id: builder1
  uses: actions/docker_setup-buildx-action@v2
  with:
    endpoint: builders

- name: Login to Artifactory
  uses: actions/docker_login-action@v2
  with:
    registry: https://artifacts.01.epicgames.net
    username: ${ steps.secrets.outputs.JF_USERNAME }

```

```

        password: ${ steps.secrets.outputs.JF_ACCESS_TOKEN }}

- name: Build and push to Artifactory
  uses: actions/docker_build-push-action@v3
  with:
    file: ./Dockerfile
    push: true
    tags: ${ env.DEV_DOCKER_TAG }}

deploy:
  name: deploy
  runs-on: ubuntu-latest
  needs: prepare
  # To make sure helm upgrade isn't killed mid-flight, bricking the d
  # we use `always()` and only check if the prepare phase was cancell
  # of the whole job.
  if: always() && !failure() && needs.prepare.result != 'cancelled'
  steps:
    - name: Configure Kubernetes Context
      uses: online-web/configure-kubernetes-action@v1
      with:
        vaultRoleId: ${ secrets.VAULT_ROLE_ID}}
        vaultSecretId: ${ secrets.VAULT_SECRET_ID }}
        clusterName: ${ env.NPE_CLUSTER_NAME }}
        namespace: ${ env.NPE_NAMESPACE }}

    - name: Setup Helm
      uses: actions/Azure_setup-helm@v3
      with:
        version: 'v3.7.1'
        id: install

    - name: Helm Deploy Dev
      run: |
        helm repo add substr-helm https://artifacts.ol.epicgames.net/
        helm dependency update ./deploy/chart
        helm upgrade ${ env.SERVICE_NAME }}-dev ./deploy/chart \

```

```
--install \  
--namespace ${ env.NPE_NAMESPACE } \  
--values "../deploy/chart/values.gamedev.yaml" \  
--set "epic-app.imageTag=${ env.SHORT_SHA }" \  
--atomic \  
--wait \  
--timeout 5m \
```

How can I use GitHub Actions to deploy my applications to different environments?


You can create an environment in GitHub from the Settings menu.

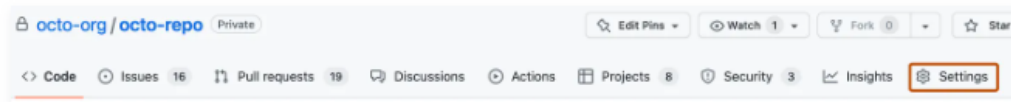
Creating an environment [↗](#)

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.

Notes:

- Creation of an environment in a private repository is available to organizations with GitHub Team and users with GitHub Pro.
- Some features for environments have no or limited availability for private repositories. If you are unable to access a feature described in the instructions below, please see the documentation linked in the related step for availability information.

- 1 On GitHub.com, 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.

Reference: <https://docs.github.com/en/actions/deployment/targeting-different-environments/using-environments-for-deployment#creating-an-environment>

You can also use an environment as an input variable, and just reference it to avoid hard-coding it.

Pass in an input for the environment value:

```
jobs:
  deployment:
    runs-on: ubuntu-latest
```

```
environment: $ {{inputs.environment }}  
steps:  
  - name: deploy  
  # ...deployment-specific steps
```

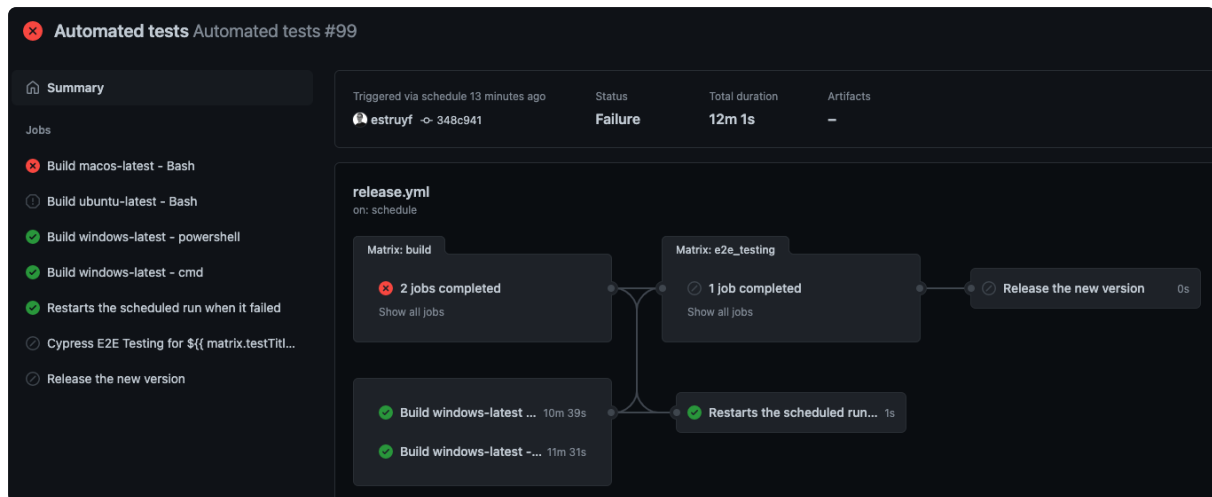
How environments relate to deployments

When a workflow job that references an environment runs, it creates a deployment object with the `environment` property set to the name of the environment. As the workflow progresses, it also creates deployment status objects with the `environment` property set to the name of the 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.

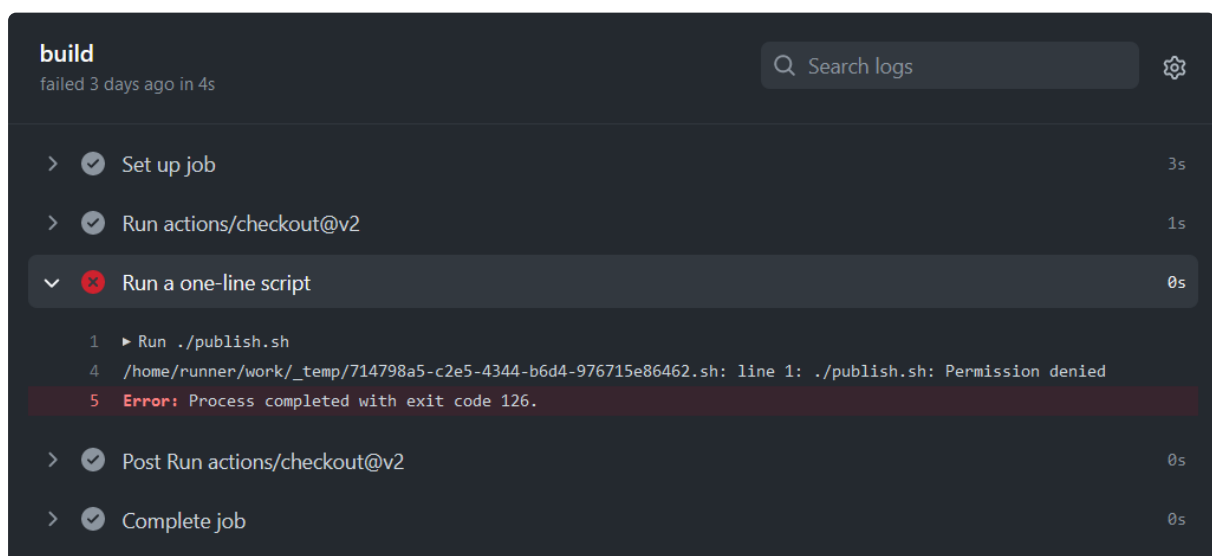
Reference: <https://docs.github.com/en/actions/deployment/targeting-different-environments/using-environments-for-deployment>

Troubleshooting / Error Handling

Each workflow that runs should show a green or red status circle after running. Red indicates failure and green indicates success. Shown below is an example of a matrix workflow that has a mixture of succeeded and failed jobs. In this case, because at least 1 job failed, the remainder of the jobs do not go forward.



You can also click into the failed jobs to see exactly which line they failed in and what exactly the error message is (screenshot taken from a different example):



The same commands that one might run manually in a terminal are the ones that get executed in GitHub Actions via the pipeline. Thus the potential errors you see should be no different from the ones you'd see while running locally.

For more troubleshooting information related to GitHub Actions, see <https://docs.github.com/en/enterprise-server@3.9/admin/github-actions/>

[advanced-configuration-and-troubleshooting/troubleshooting-github-actions-for-your-enterprise](#)

Where to get GitHub Help?

Reach out to the Slack channel [#sig-github-ext](#). If you don't have access to this channel, reach out to your manager to add you.

Additional Resources

GitHub Actions Documentation:

<https://docs.github.com/en/actions>.

Epic Games GitHub Actions User Guide

<https://confluence-epicgames.atlassian.net/wiki/display/TeamOnline/GitHub+Actions+User+Guide>

Page Information:

Page ID: 81068367

Space: Cloud Developer Platform

Downloaded: 2025-07-12 04:07:01