Lab 9: GitHub Environments and Deployments

Lab overview

In this lab activity, you will use GitHub Actions to move code through development, staging, and production environments. This will help you appreciate the work that a DevOps Engineer ought to carry out to ensure a clear separation of environments and enable development teams. In this lab, you will also set up GitHub Actions workflows to rollback code in case a build starts to fail.

In this lab, you will:

- Set up your lab environment
- Create environments using GitHub
- Create a GitHub Actions pipeline that utilizes environments (development, staging, and production)
- Use GitHub Environments to gate sensitive environments
- Roll back to a previous deployment by re-running workflows

Estimated completion time

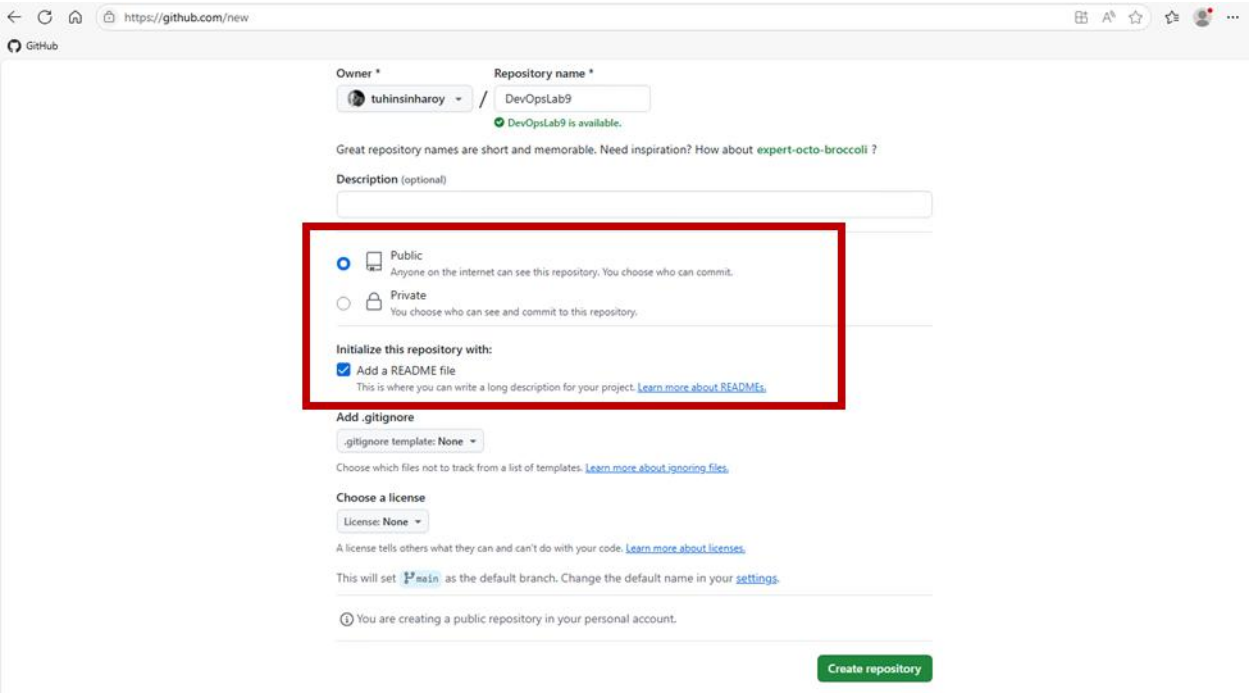
30 minutes

Task 1: Setting up the repository

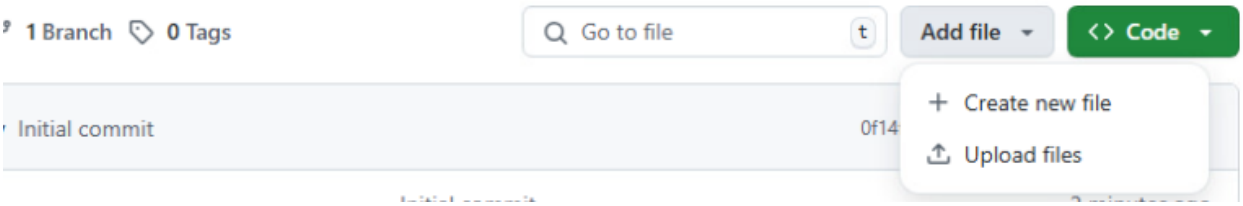
In this task, you will set up your repository.

1. Go to **https://github.com** and create a new **public** repository in your account (e.g., DevOpsLab9). Add a README file.

   Many Environment and Deployment features are only available on paid plans, or for public repositories, therefore go with a public repository for learning purposes.



2. To create a new file, select **Add file** and then choose **Create new file**.



3. Create a new file called **index.html** with the following content to represent our code:

```
<!DOCTYPE html>

<html>

<head>

    <title>DevOps Lab 09</title>
```

```
</head>
<body>
    <h1>Hello, DevOps Lab 09!</h1>
</body>
</html>
```

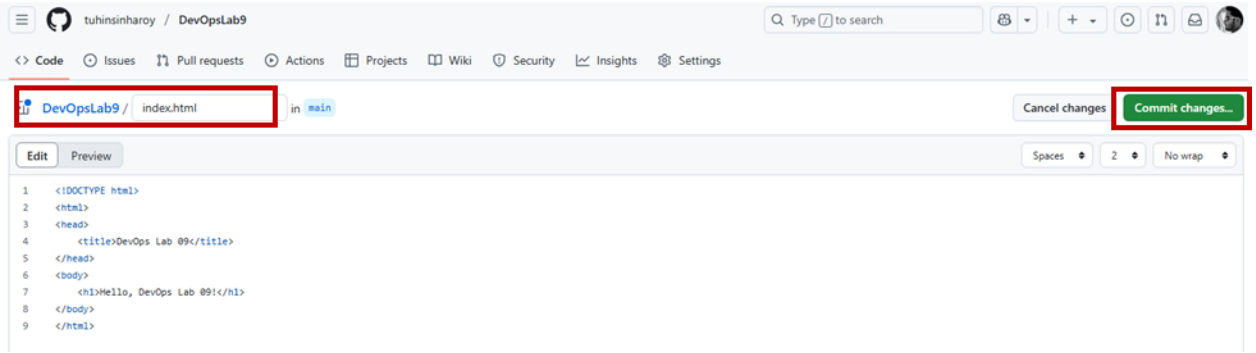
Note

The complete code (**GK840253-Lab9-index.html.txt**) is also available in the DevOpsLab9 folder on the lab desktop.

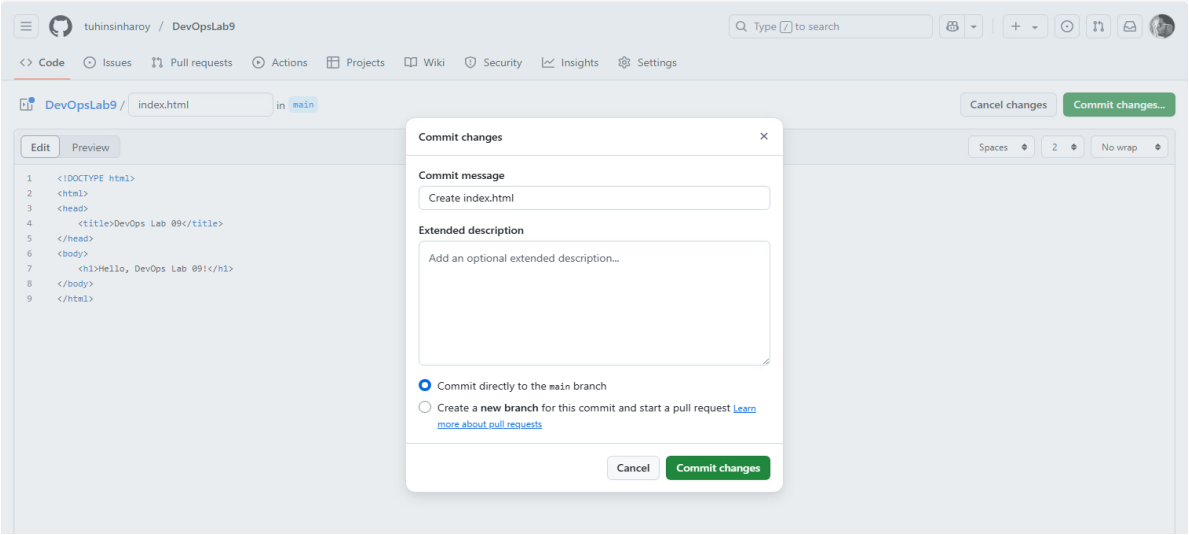
4. Commit the changes to the default branch.

Note

Please ensure that the file is named **index.html**.



5. Approve the commit.



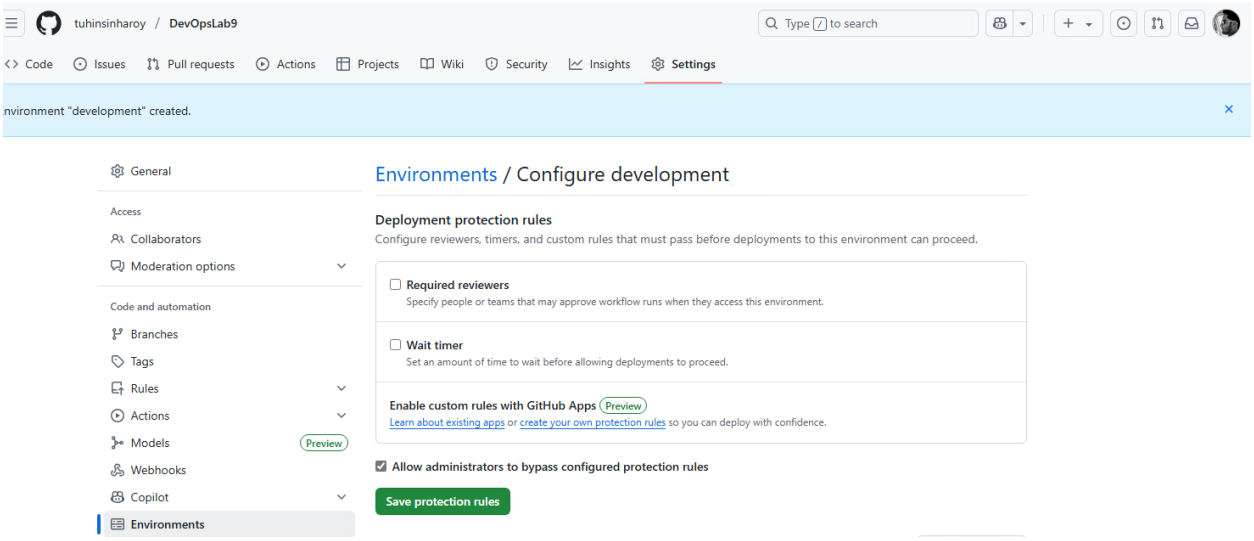
Task 2: Creating environments in GitHub

1. Go to your GitHub Repo > **Settings** > **Environments** and create the following environments.

Figure 1: Environments to create

Environment	Reviewers	Wait timer	Description
development	No	No	Auto-deployment to dev for testing
staging	At least 1	No	Requires approval before deployment
production	At least 1	1 min	Requires approval and wait time before deployment

Figure 2: Configuring development environment



In both staging and production environments:

- Enable required reviewers under Deployment protection rules
- Add your own GitHub username for now though ideally should be an engineering group

Figure 3: Configure staging environment

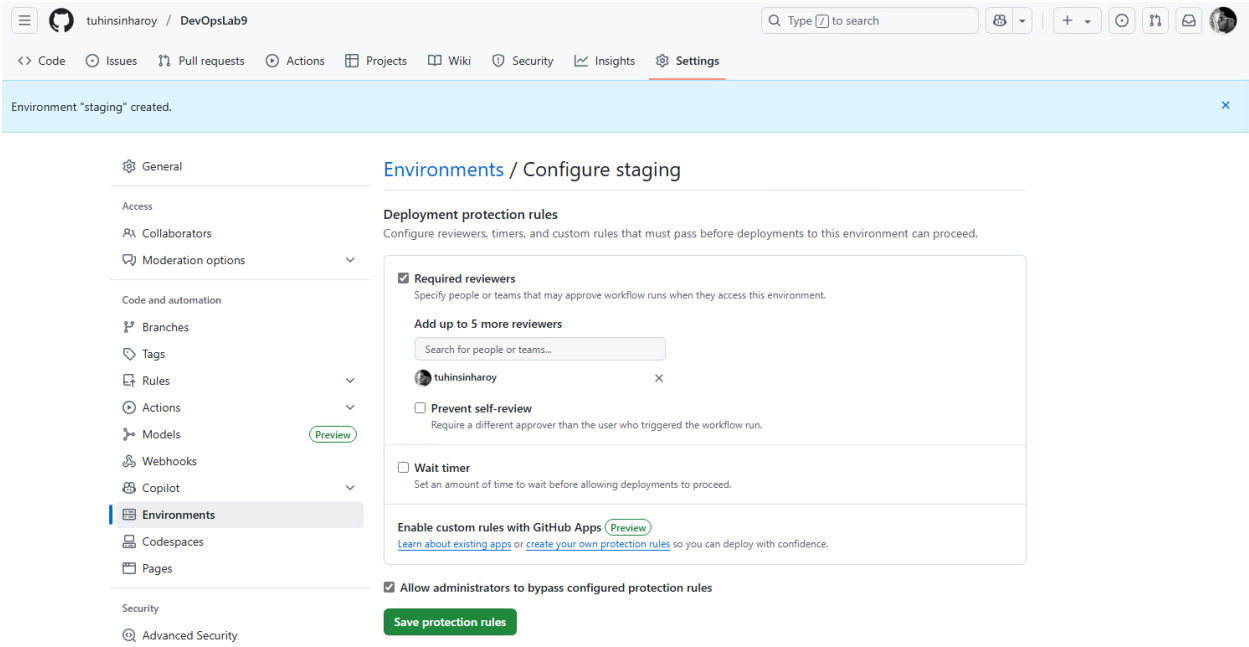
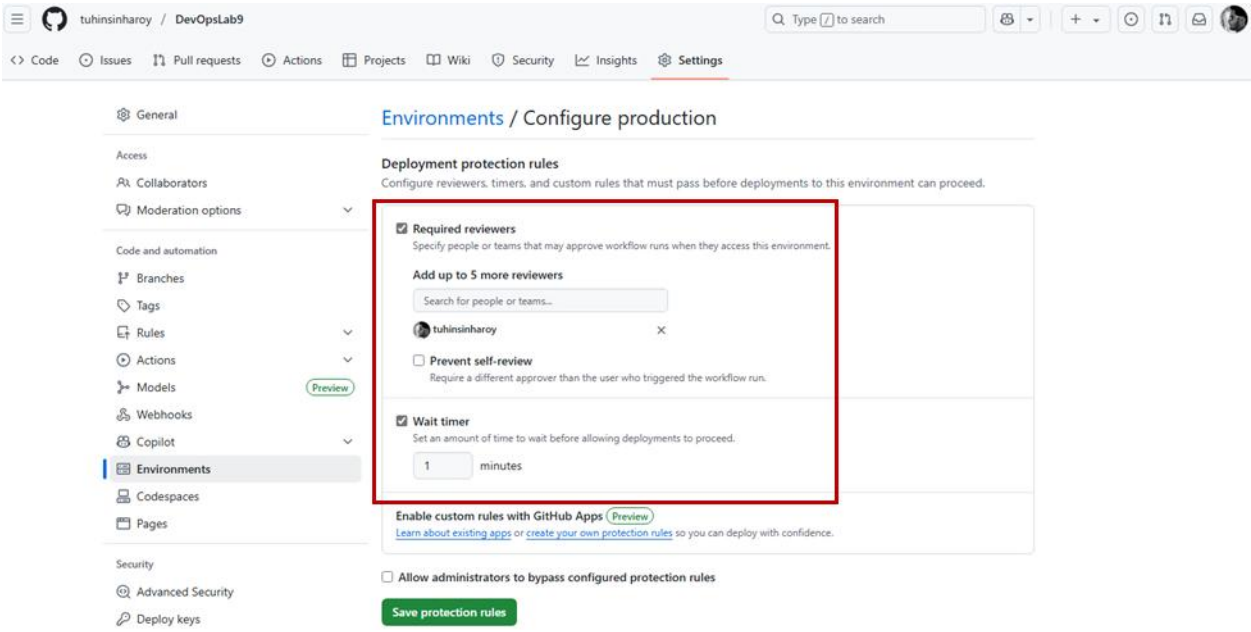
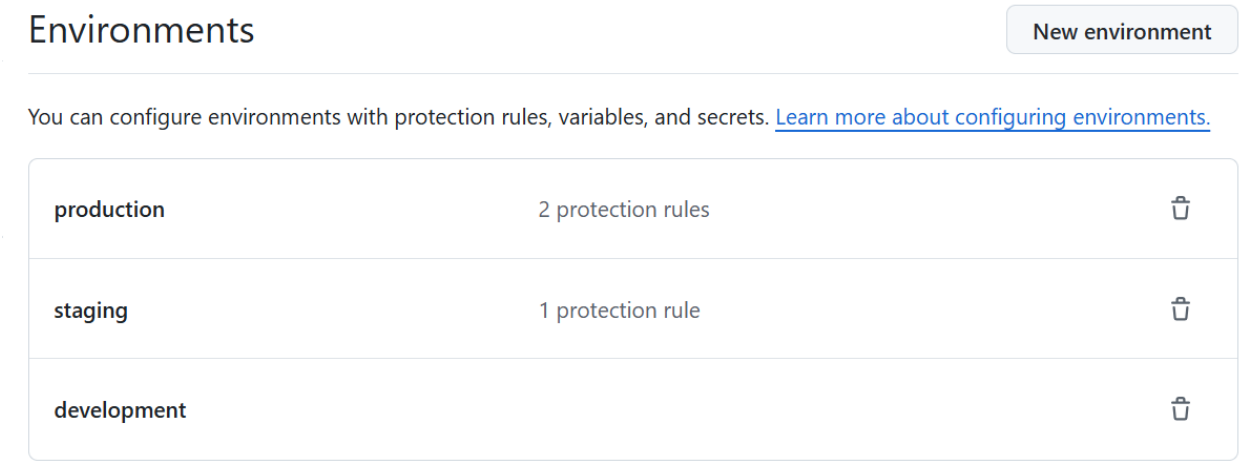


Figure 4: Configure production environment

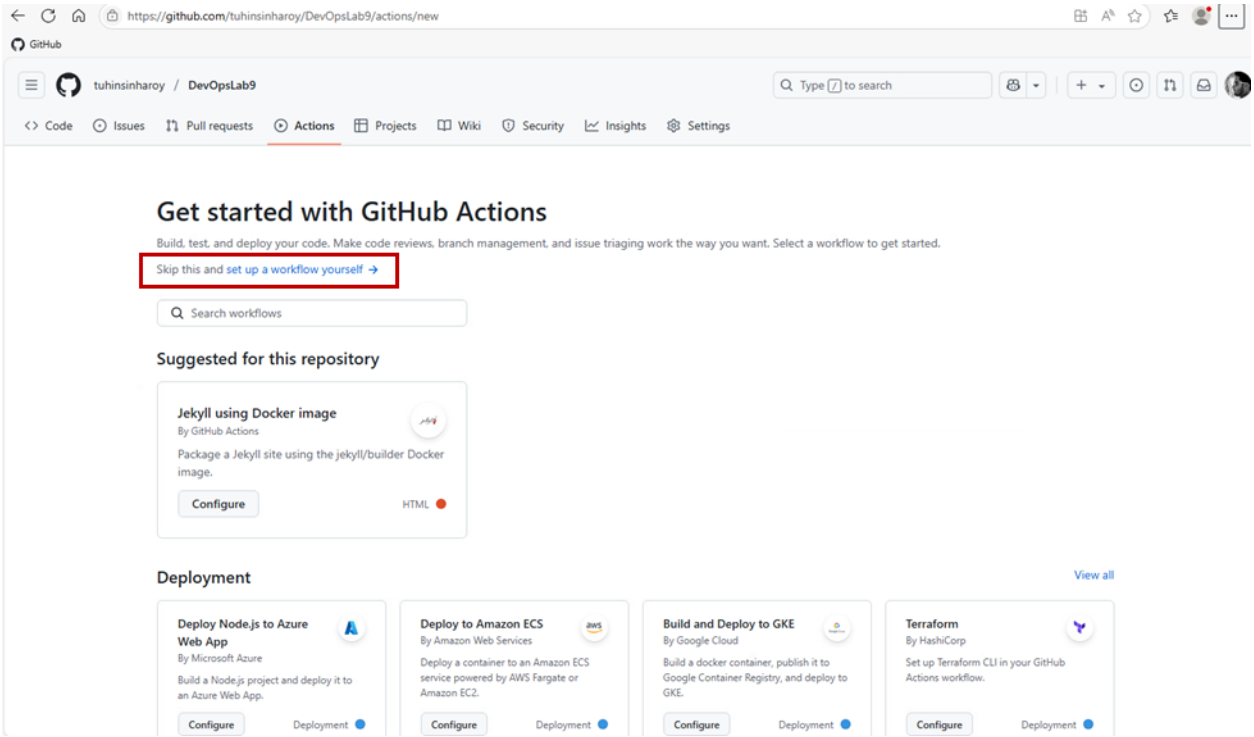


2. This is what the summary of the environments will look like when done.



Task 3: Creating a GitHub Actions pipeline

1. Go to your GitHub Repo > **Actions** > **Set up a workflow yourself** and create the following workflow.



2. Name the file **build-and-deploy.yml** and enter the following code into it.

```
name: Build and Deploy DevOps Lab 9


on:
  push:
    branches:
      - main
  workflow_dispatch:


env:
  APP_NAME: devopslab9


jobs:
  build:
    name: Build and Upload Artifact
    runs-on: ubuntu-latest


    steps:
      - name: Checkout code
        uses: actions/checkout@v3


      - name: Generate artifact
        run: |
          echo "<!-- Built: $(date) -->" >> index.html


      - name: Upload artifact
```

```
        uses: actions/upload-artifact@v4
        with:
          name: ${{ env.APP_NAME }}
          path: ./

  deploy_dev:
    name: Deploy to Development
    runs-on: ubuntu-latest
    needs: build
    environment:
      name: development
      url: https://dev.${{ env.APP_NAME }}.com

    steps:
      - name: Download artifact
        uses: actions/download-artifact@v4
        with:
          name: ${{ env.APP_NAME }}

      - name: Simulate development Deployment
        run: |
          echo "Simulating Deploying to DEVELOPMENT via SSH:
https://dev.${{ env.APP_NAME }}.com"
          cat index.html

  deploy_staging:
    name: Promote to Staging
```

```
    runs-on: ubuntu-latest
    needs: deploy_dev
    environment:
      name: staging
      url: https://staging.${{ env.APP_NAME }}.com

    steps:
      - name: Download artifact
        uses: actions/download-artifact@v4
        with:
          name: ${{ env.APP_NAME }}

      - name: Simulate staging Deployment
        run: |
          echo "Simulating Deploying to STAGING via SSH:
https://staging.${{ env.APP_NAME }}.com"
          cat index.html

  deploy_prod:
    name: Promote to Production
    runs-on: ubuntu-latest
    needs: deploy_staging
    environment:
      name: production
      url: https://${{ env.APP_NAME }}.com

    steps:
```

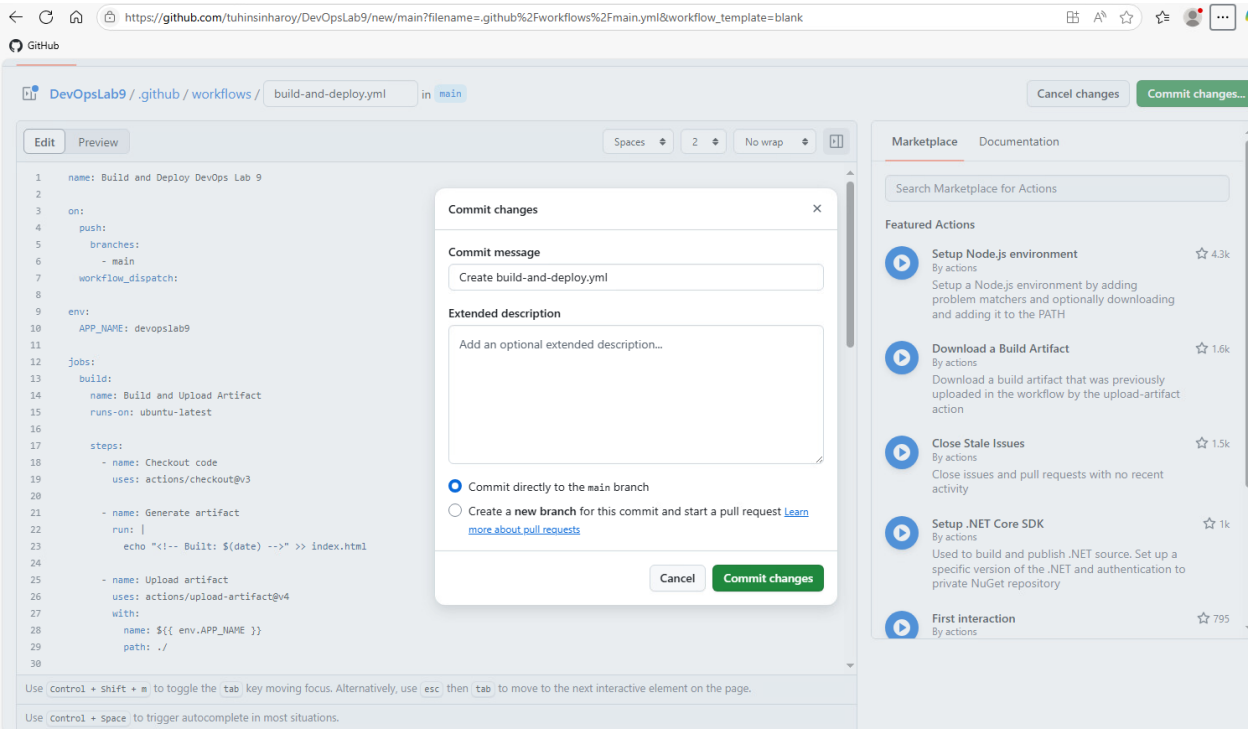
```
      - name: Download artifact
        uses: actions/download-artifact@v4
        with:
          name: ${{ env.APP_NAME }}


      - name: Simulate production Deployment
        run: |
          echo "Simulating Deploying to PRODUCTION via SSH:
https://${{ env.APP_NAME }}.com"
          cat index.html
```

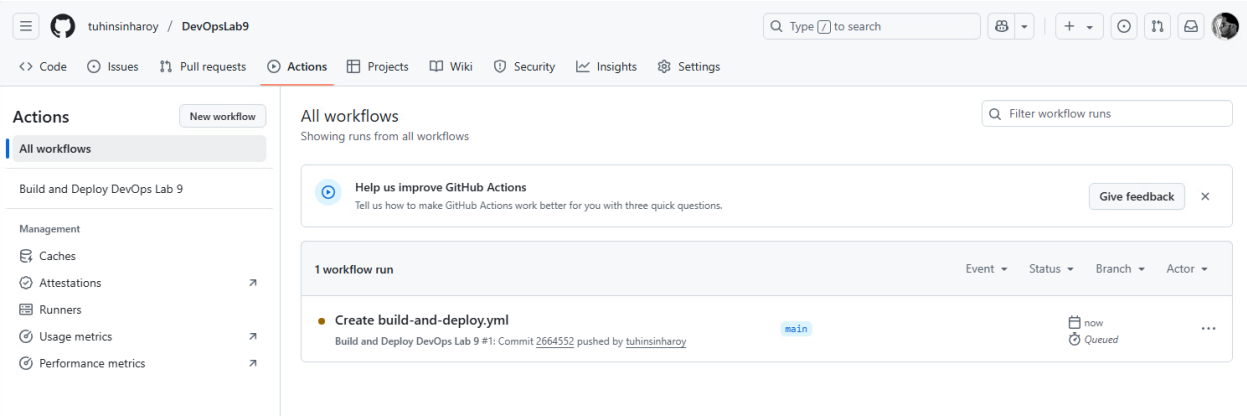
Note

The complete code (**GK840253-Lab9-build-and-deploy.yml.txt**) is also available in the DevOpsLab9 folder on the lab desktop.

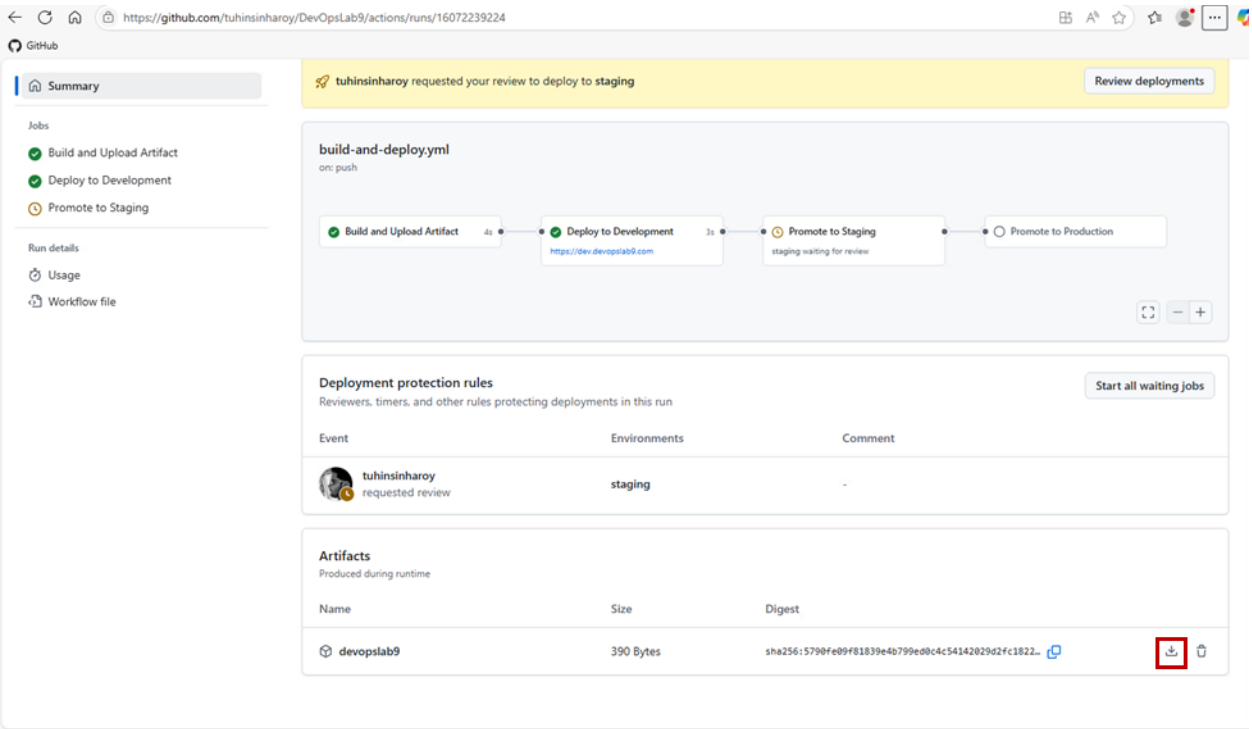
3. Commit the changes.



4. Immediately go to your GitHub Repo > **Actions** and click on the latest job. You will see that the build job has kicked off since a commit to the main branch was made.



5. After a few moments, a deployment to the development environment kicks off because no branch protections are in place. This is typical for **development** because developers like to see their changes as quickly as possible.
6. Click on the **download** button to download the artifact.



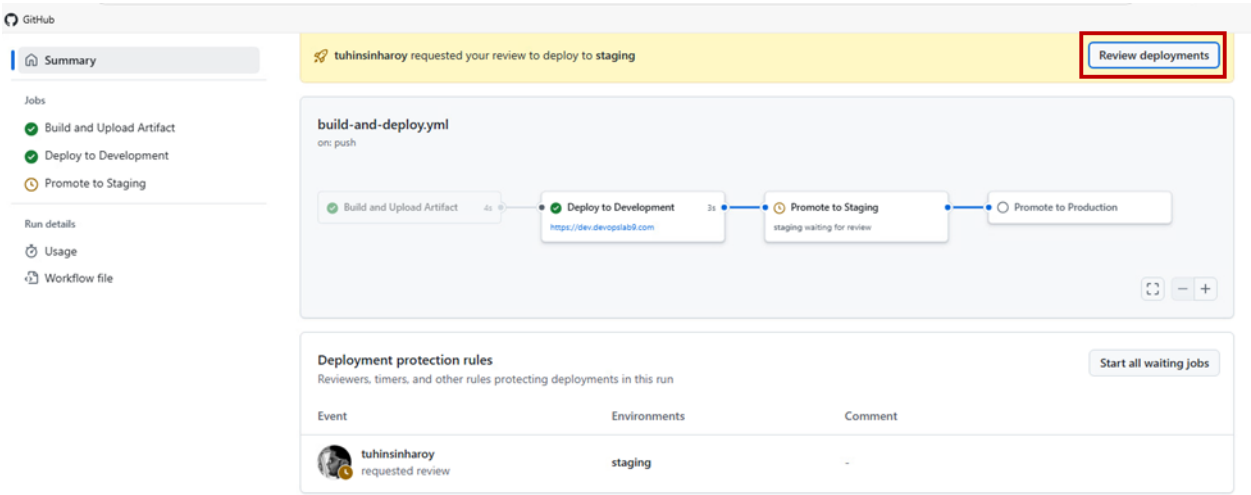
7. Unzip the code, right click on **index.html** and select **Open with code**. You will notice a unique timestamp identifier at the bottom of the file. This is to signify that in an ideal DevOps scenario, the same build that was tested in development environment gets promoted to the staging environment and ultimately to production. Builds are not run again for each environment.

index.html ×

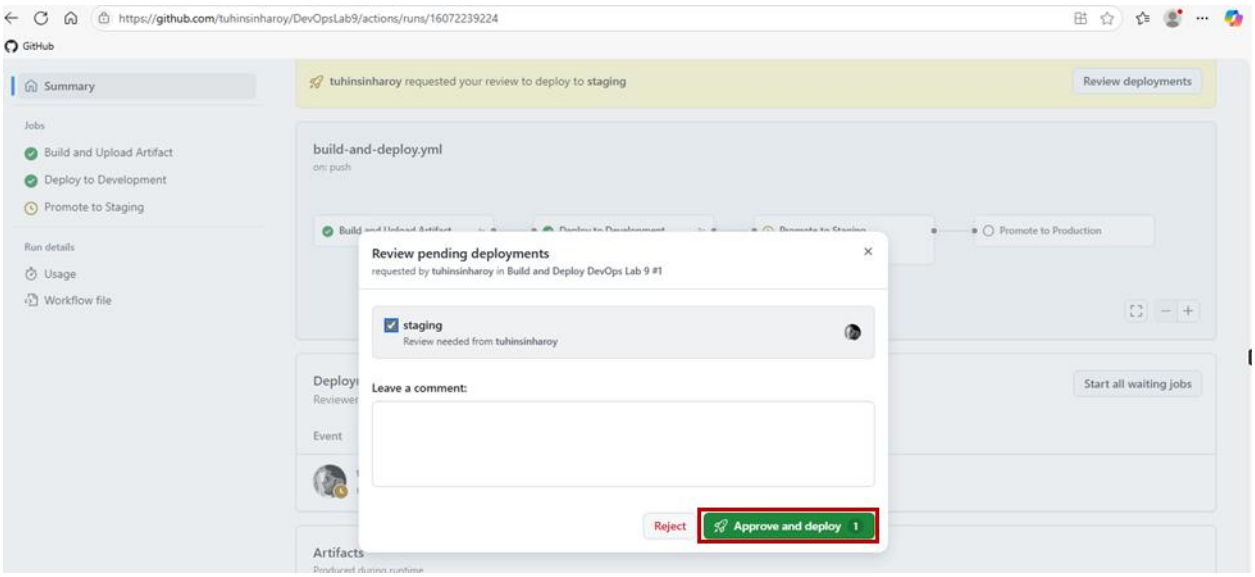
C: > Users > student > Downloads > devopslab9 > index.html > ...

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>DevOps Lab 09</title>
5  </head>
6  <body>
7      <h1>Hello, DevOps Lab 09!</h1>
8  </body>
9  </html>
10 <!-- Built: Fri Jul  4 10:59:51 UTC 2025 -->
11
```

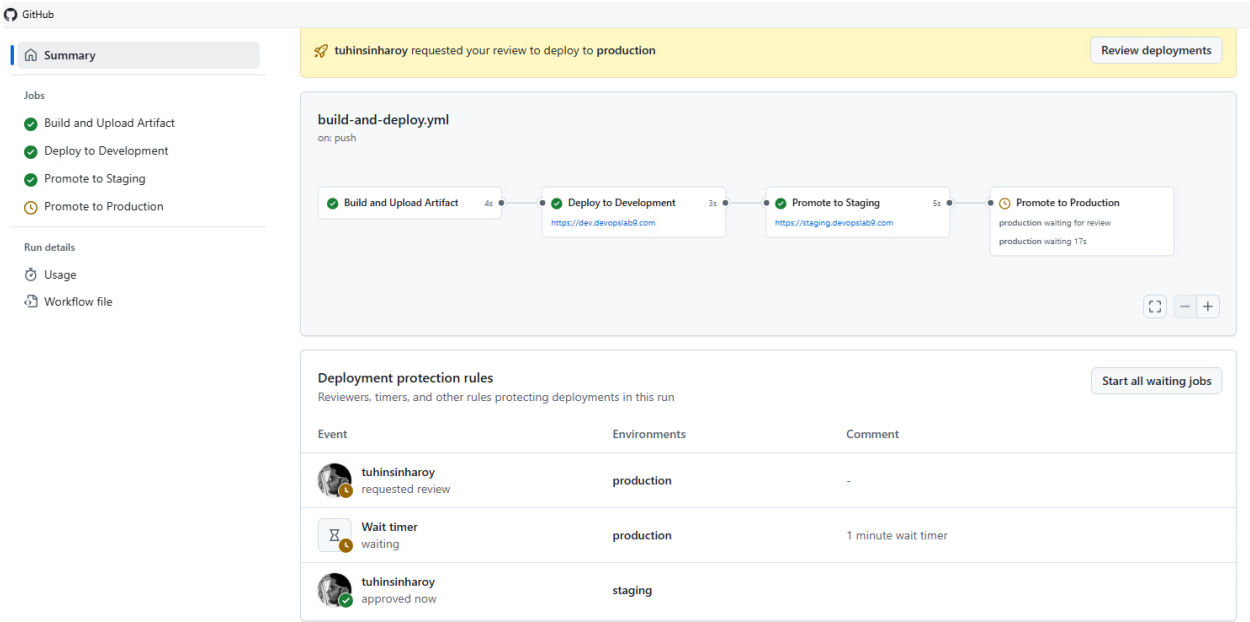
8. After the deployment to the **development** environment is done, you will be invited to review the deployment to **staging**. Click **Review Deployments**.



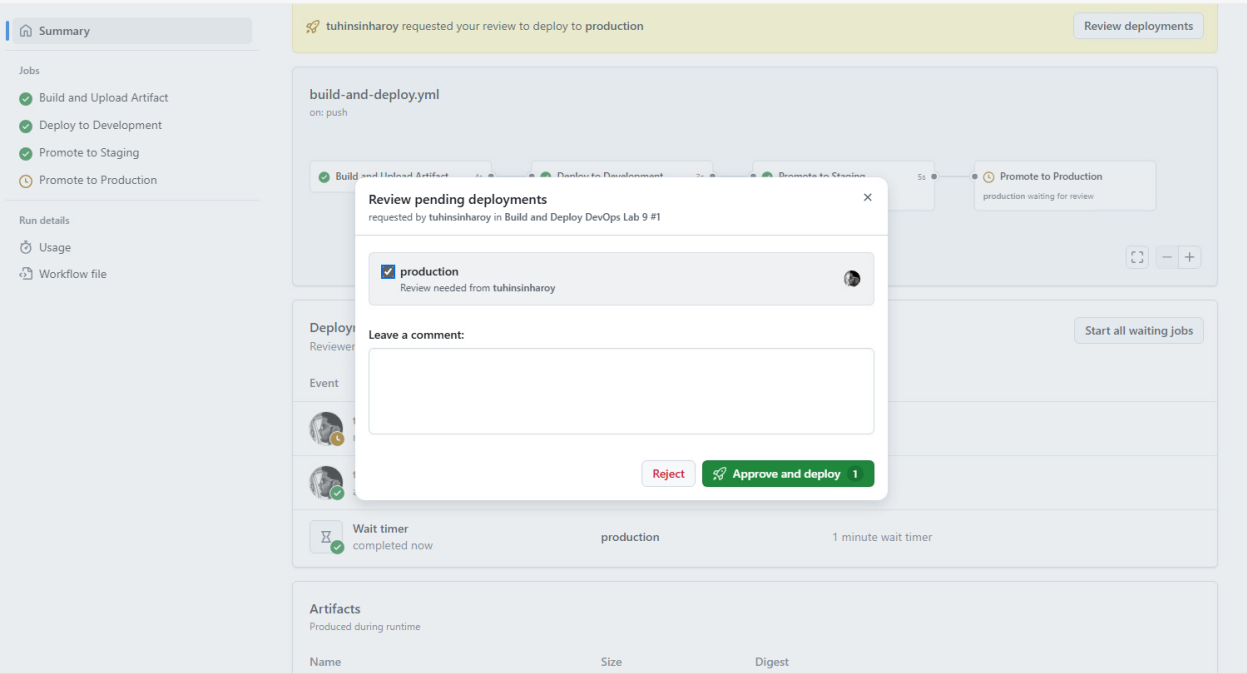
9. Leave an optional review and then proceed with the **staging** deployment.



10. Wait for the **staging** deployment to complete. After it's done, the production wait timer kicks off. Once it's done, review and approve the deployment.



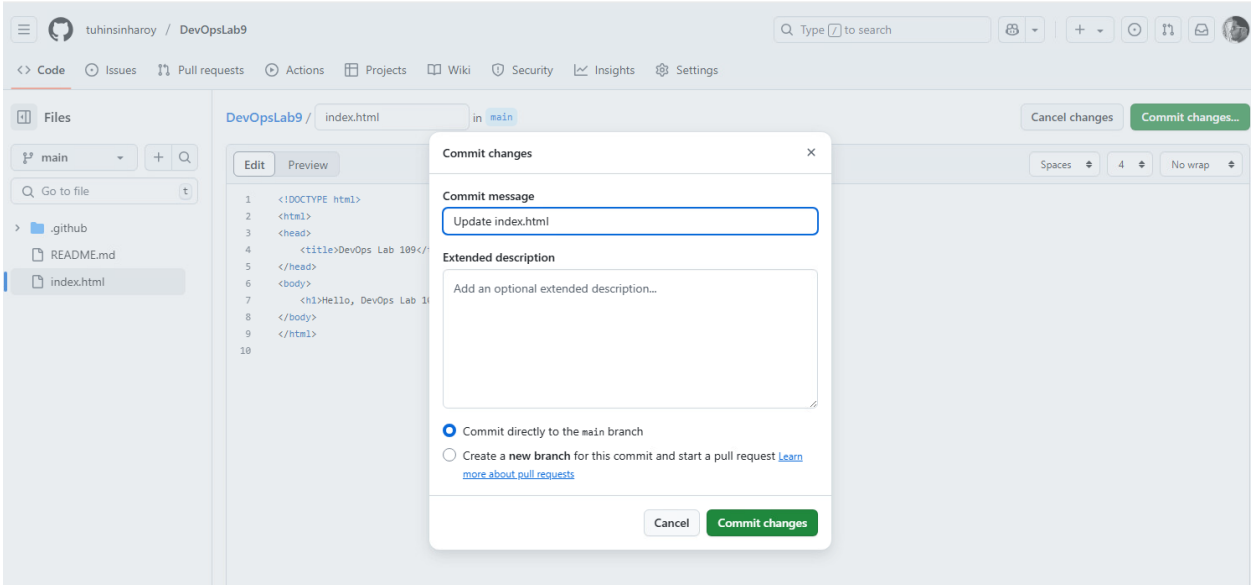
11. When the production wait timer is done, approve the production deployment.



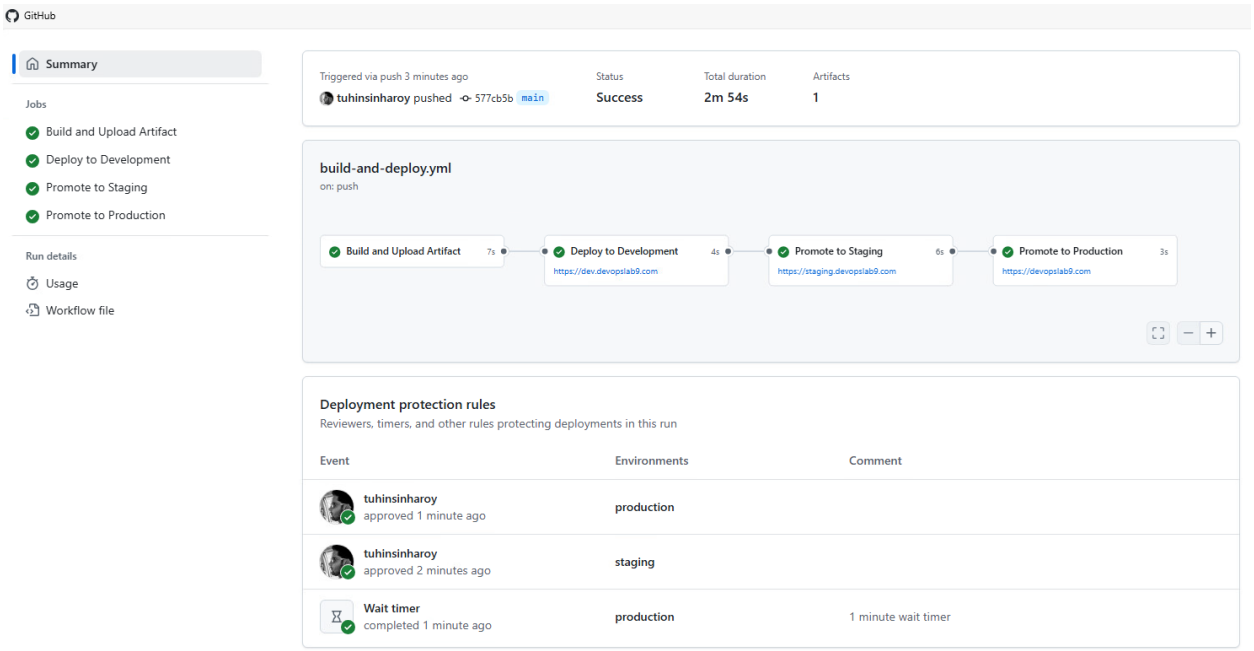
Task 4: Re-deploying an old version

In this task, you will revert to a previous working version when using Deployments.

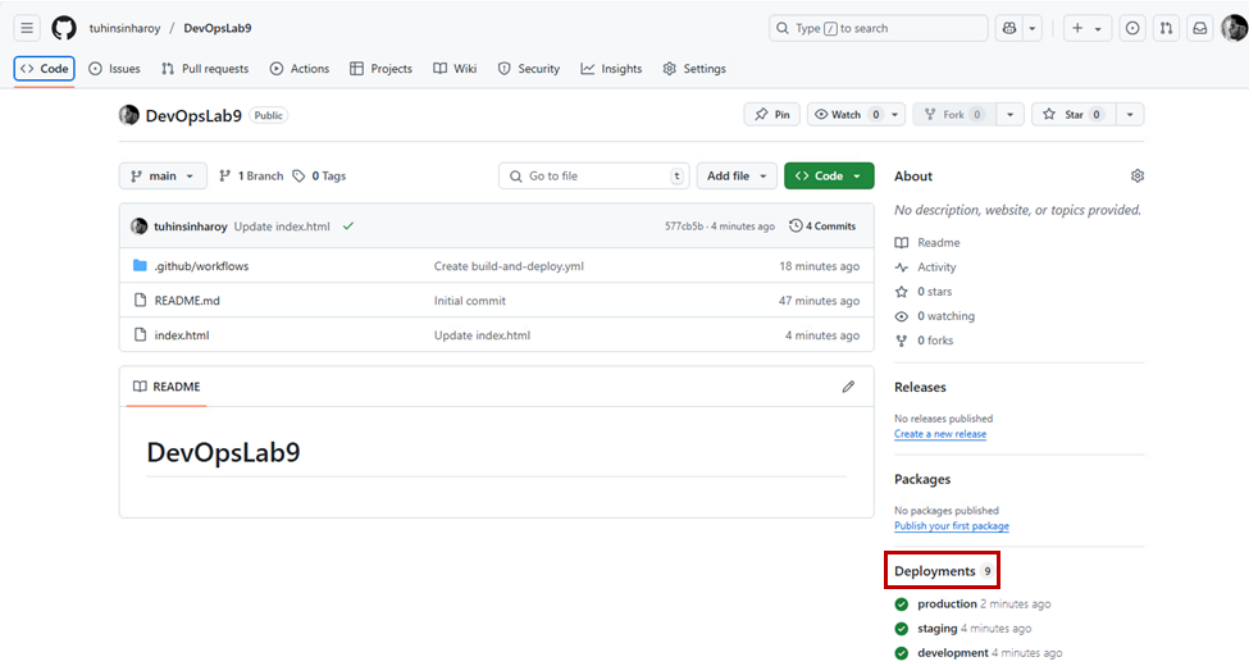
1. Make another code change, this time simulating an erroneous build. Modify **index.html** and commit the changes.



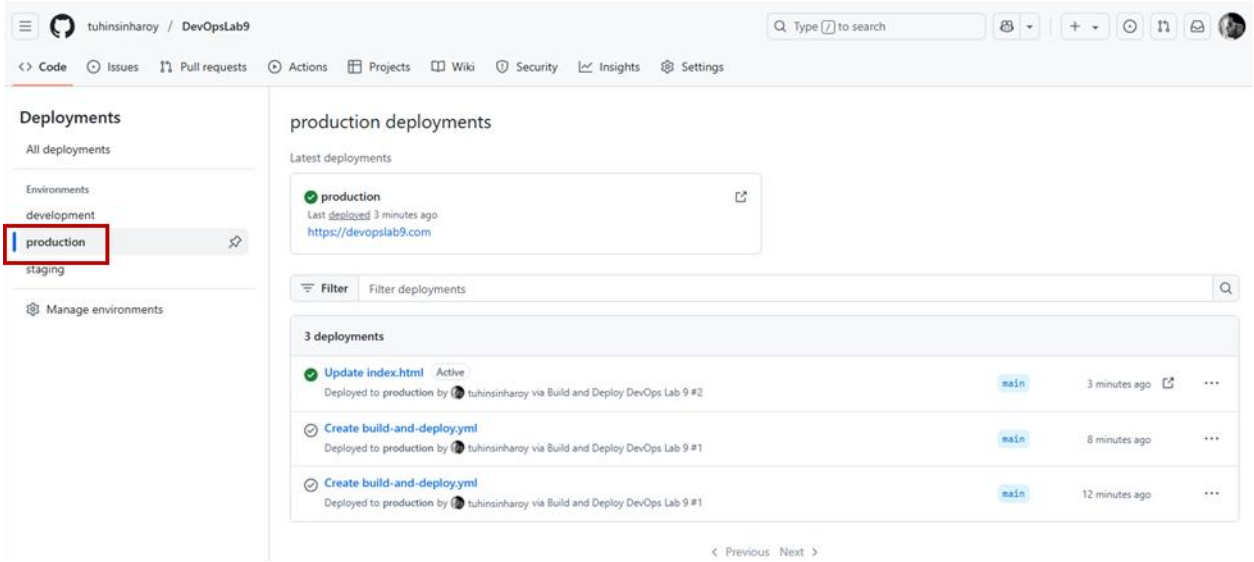
2. Pass the build through **development**, **staging**, and **production**.



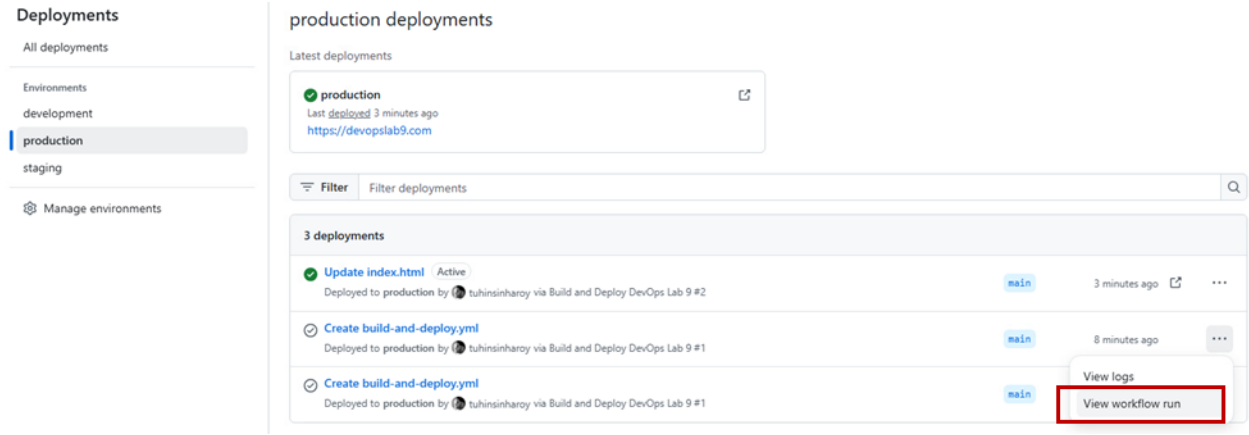
3. Later, you receive complaints from customers that the application is showing DevOps Lab 109 instead of DevOps Lab 09. To fix this issue, you can either write code to fix it, or re-deploy a previous version. You will do the latter by going to the GitHub repository then click **Deployments**.



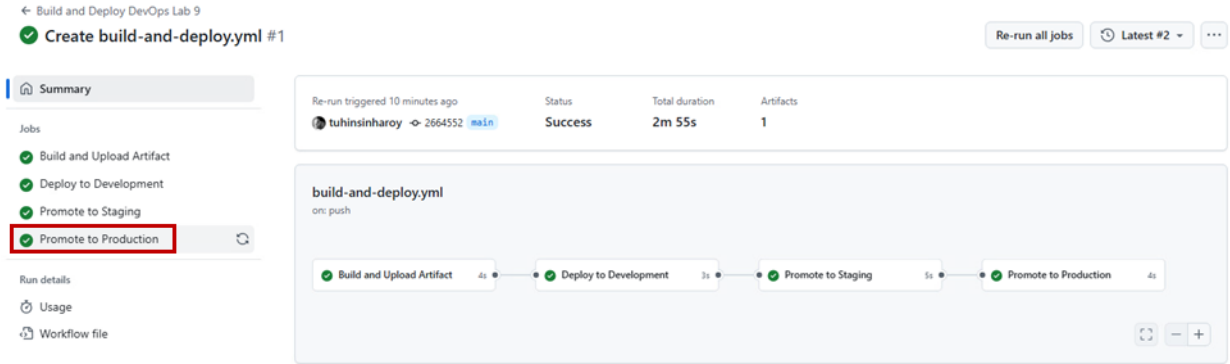
4. Filter for the production deployments.



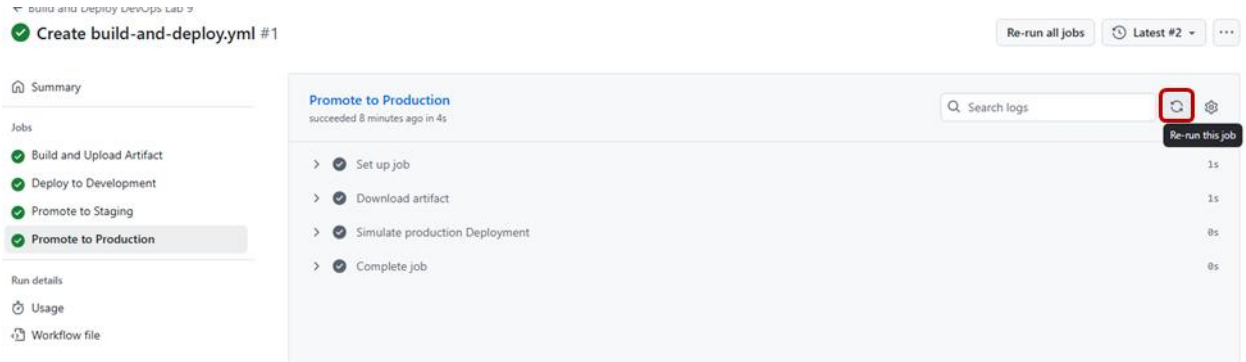
5. View workflow run for the previous deployment.



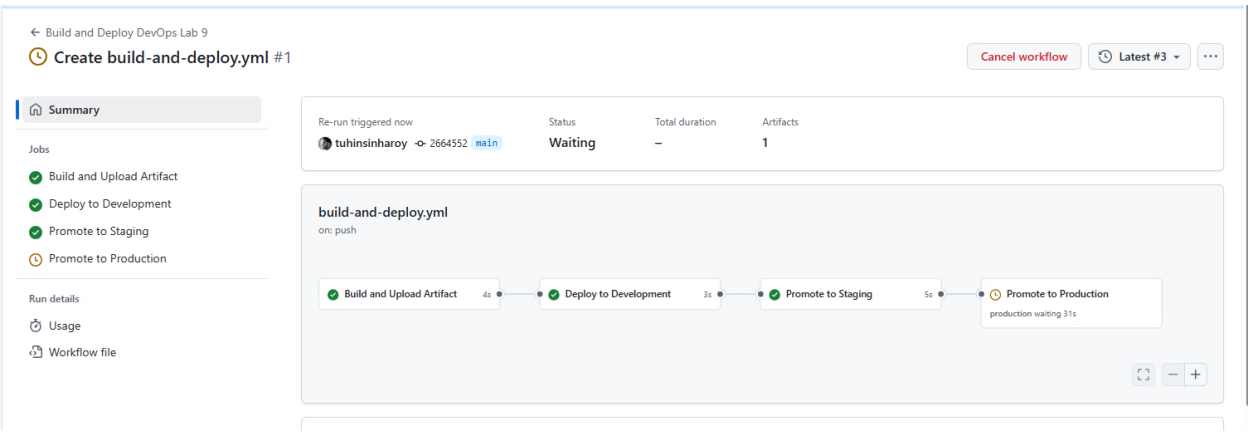
6. Click on the **Promote to Production** job.



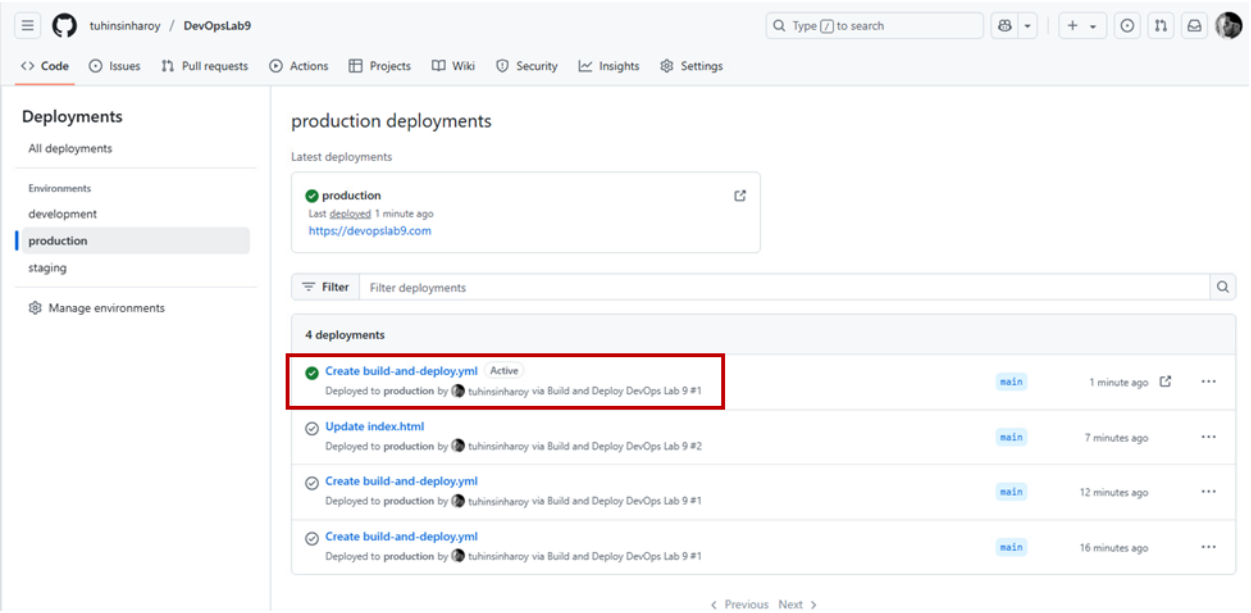
7. Click on **Re-run this job**.



8. Confirm the re-run job. The job is set up for review again and the timer resets. Approve the re-deployment.



9. When the deployment is done, return to the **Deployments** page. You will now see that the active deployment is an older one which was working without issue. This has enabled us to quickly revert a faulty production build without affecting ongoing development of new features.



Lab review

1. What is the primary purpose of using GitHub Environments in a deployment workflow?
   A. To build and compile code efficiently
   B. To test different branches simultaneously
   C. To gate sensitive environments and manage deployment approvals
   D. To speed up the deployment process by skipping testing
2. How does the lab suggest rolling back a faulty production deployment using GitHub Actions?
   A. Manually edit the code and re-commit
   B. Revert the main branch to a previous commit
   C. Clone the previous deployment locally and redeploy
   D. Re-run the previous successful production job from the Deployments page

STOP

You have successfully completed this lab.