

CI/CD and DevOps in 3 Weeks

Week 3

Revision 1.6 – 02/19/23

Tech Skills Transformations LLC / Brent Laster

Before you start: Make sure you did the "homework" from last week to setup your token and secret

You will need the classic Personal Access Token created and stored in a secret named PIPELINE_USE.

HELPFUL HINT: For some of the labs, you will need to copy code from a file in the "extra" subdirectory of the greetings-ci project. To make it easy to copy the code into the editor, you can open up a second tab in your browser to the same greetings-ci repository, and bring the file up in the "extra" subdirectory in that second tab. Then you can just copy and paste from the second tab into where you are editing on the first one.

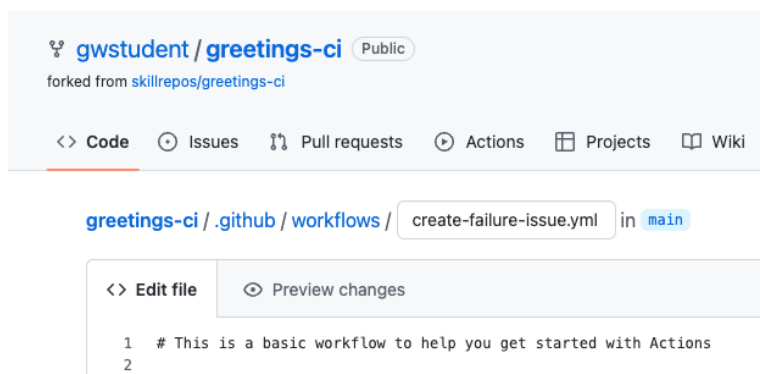
Lab 9: Working with fast feedback and automatically reporting issues

Purpose: Learning how to get fast feedback and automatic failure reporting in our pipeline

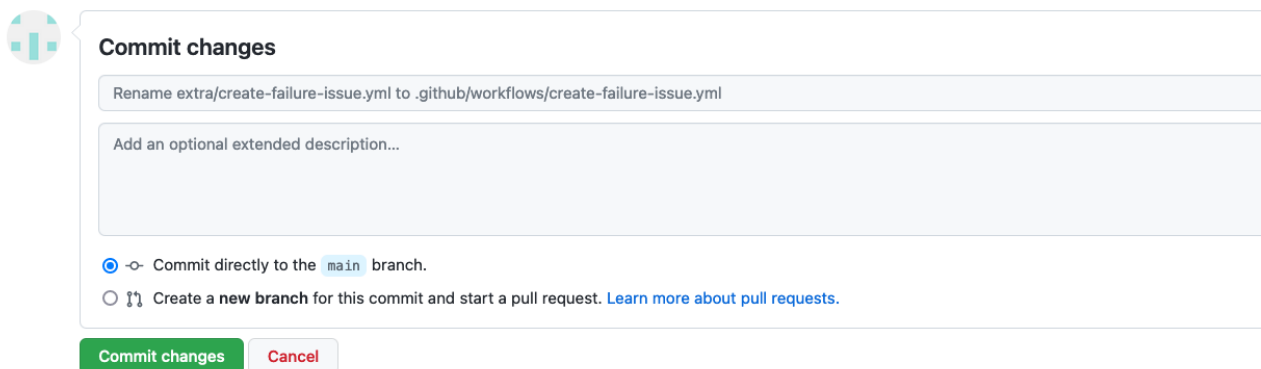
1. Start out in the **"greetings-ci" repository** in GitHub under your **primary GitHub** userid, and in the **main** branch.
2. We're going to create a new workflow that will be able to automatically create a GitHub issue in our repository. And then we will invoke that workflow from our current workflow. The workflow to create the issue using a REST API call is already written to save time. It is in the main project under "extra/create-failure-issue.yml". You need to get this file in the .github/workflows directory. To do that, you can clone and move it. Or you can just do it via GitHub with the following steps.
 - a. In the repository, browse to the "extra" folder and to the "create-failure-issue.yml" file.
 - b. Take a few moments to look over the file and see what it does. Notice that:
 - i. it has a workflow_dispatch section in the "on" area, which means it can be run manually.
 - ii. It has two inputs - a title and body for the issue.
 - iii. The primary part of the body is simply a REST call (using the GITHUB_TOKEN) to create a new issue.
 - c. Click the pencil icon to edit it.



- d. In the filename field, change the name of the file. Use the backspace key to backspace over "extra/" making sure to backspace over the word. Then type in the path to put it in the workflows ".github/workflows/create-failure-issue.yml". (Backspace over *extra/* and type in *.github/workflows/* where it was.)



- e. To complete the change, scroll to the bottom of the page, and click on the green "Commit changes" button.



3. Go back to the Actions tab. You'll see a new workflow execution due to the rename. Also, in the Workflows section on the left, you should now see a new workflow titled "create-failure-issue". Click on that. Since it has a *workflow_dispatch* event trigger available, we can try it out. Click on the "Run workflow" button and enter in some text for the "title" and "body" fields. Then click "Run workflow".

- After a moment, you should see the workflow run start and then complete. If you now click on the Issues tab at the top, you should see your new issue there.

- Now that we know that our new workflow works as expected, we can make the changes to the previous workflow to "call" this if we fail. Edit the *pipeline.yml* file and add the following lines as a new job and set of steps at the end of the workflow.

(For convenience, these lines are also in the file "extra/create-issue-on-failure.txt" if you want to copy and paste from there.)

The "create-issue-on-failure" job name should align with the "test-run" job name. See screenshot further down.

create-issue-on-failure:

```
runs-on: ubuntu-latest
needs: test-run
if: always() && failure()
steps:
  - name: invoke workflow to create issue
    run: >
      curl -X POST
      -H "authorization: Bearer ${ secrets.PIPELINE_USE }"
      -H "Accept: application/vnd.github.v3+json"
```

```

    "https://api.github.com/repos/${{ github.repository }}/actions/workflows/create-
failure-issue.yml/dispatches"
    -d '{"ref":"main",
      "inputs":
        {"title":"Automated workflow failure issue for commit ${{ github.sha }}",
         "body":"This issue was automatically created by the GitHub Action workflow
** ${{ github.workflow }} **"}
      }'

```

```

57     name: greetings-jar
58     path: |
59       build/libs
60       test-script.sh
61
62
63   test-run:
64
65     runs-on: ubuntu-latest
66     needs: build
67
68     steps:
69     - name: Download candidate artifacts
70       uses: actions/download-artifact@v3
71       with:
72         name: greetings-jar
73
74     - name: Execute test
75       shell: bash
76       run: |
77         chmod +x ./test-script.sh
78         ./test-script.sh ${{ needs.build.outputs.artifact-tag }} ${{ github.
79
80   create-issue-on-failure:
81
82     runs-on: ubuntu-latest
83     needs: test-run
84     if: always() && failure()
85     steps:
86     - name: invoke workflow to create issue
87       run: >
88         curl -X POST
89         -H "authorization: Bearer ${{ secrets.PIPELINE_USE }}"
90         -H "Accept: application/vnd.github.v3+json"
91         "https://api.github.com/repos/${{ github.repository }}/actions/workflows/create-issue-on-failure.yml/dispatches"
92         -d '{"ref":"main",
93           "inputs":
94             {"title":"Automated workflow failure issue for commit ${{ git
95             "body":"This issue was automatically created by the GitHub A
96           }'
97

```

- Commit your changes. After this is committed and the workflow runs, you can look at the output for the run and you'll see that the "create-issue-on-failure" job was skipped. That makes sense because we have the checks in the code and there was no failure on previous jobs.

feat: add create issue on failure Java CI with Gradle #38 Re-run all jobs ...

Summary

Triggered via push 7 minutes ago

Jobs	Status	Total duration	Artifacts
gwstudent pushed → 6e0a6b4 main	Success	47s	1

pipeline.yml
on: push

```

graph LR
    build[build 25s] --> test-run[test-run 2s]
    test-run --> create-issue-on-failure[create-issue-on-failure 0s]
  
```

7. To have this executed via the "if" statement, we need to have a failure. Let's try some different input with special characters that may not print out as expected. Go to the Actions menu, and then select our main "Java CI with Gradle" workflow. Click on the "Run workflow" button and enter text like below: (that's two backslashes between the "de" and "f"). As long as you have two backslashes somewhere, this should fail.

abc de\\f ghi

The screenshot shows the GitHub Actions interface for the workflow "Java CI with Gradle". On the left, there's a sidebar with "Workflows" and "All workflows". The main area shows a list of workflow runs. The first run is "Java CI with Gradle #72: Manually run by gwstudent2" with a green checkmark. The second run is "fix: version 3" with a green checkmark. The third run is "fix: version2" with a red 'x' icon, indicating a failure. The fourth run is "Java CI with Gradle" with a green checkmark. A "Run workflow" button is visible in the top right corner of the workflow runs list.

8. After the workflow run completes for this, there should be a failure in our testing. This will in turn, cause our other workflow to create an issue. You can verify the failure in testing by looking at the logs.

The screenshot shows the GitHub Actions interface for the workflow "Java CI with Gradle". The "test-run" job is highlighted in the left sidebar. The main area shows the logs for the "test-run" job. The logs indicate a failure in the "Execute test" step. The error message is "Error: Process completed with exit code 1." The logs also show the output of the "test-script.sh" script, which is "abc".

You can also verify the new issue got created as a result of the failure through the logs of that job and by looking in the Issues menu at the top.

The screenshot shows the GitHub Actions interface for a workflow named 'create-issue-on-failure'. The workflow is listed as 'succeeded 5 minutes ago in 1s'. The job 'test-run' is marked as failed, while 'build' and 'create-issue-on-failure' are successful. The logs for the 'create-issue-on-failure' job are expanded, showing a successful run of a curl command that creates a GitHub issue. The command is: `curl -X POST -H "authorization: ***" -H "Accept: application/vnd.github.v3+json" "https://api.github.com/repos/gwstudent/greetings-ci/actions/workflows/create-failure-issue.yml/dispatches" -d '{"ref":"main", "inputs": {"title": "Automated workflow failure issue for commit 3d1d1ca2a645955953cc2a6978aba577319dd2b7", "body": "This issue was automatically created by the GitHub Action workflow ** Java CI with Gradle **"} }'`. Below the command, a table shows the progress of the curl command, indicating it completed successfully with a 100% success rate.

The screenshot shows a GitHub issue titled 'Automated workflow failure issue for commit 3d1d1ca2a645955953cc2a6978aba577319dd2b7 #7'. The issue is created by the 'github-actions' bot 4 minutes ago. The issue body contains the text: 'This issue was automatically created by the GitHub Action workflow ** Java CI with Gradle **'. The issue is currently open and has no comments. The right sidebar shows the issue's metadata, including assignees, labels, projects, milestones, and development status. The bottom of the page features a comment box and a 'Close issue' button.

END OF LAB

Lab 10 – Securing inputs

Purpose: In this lab, we'll look at how to plug a potential security hole with our inputs.

1. Switch to the pipeline.yml file in the .github/workflows directory and look at the "test-run" job and in particular, this line in the "Execute test" step:

```
./test-script.sh ${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion
}} ${ github.event.inputs.myValues }}
```

```
75
76 - name: Execute test
77   shell: bash
78   run: |
79     chmod +x ./test-script.sh
80     ./test-script.sh ${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }} ${ github.event.inputs.myValues }}
81
82 create-issue-on-failure:
```

2. When we create our pipelines that execute code based on generic inputs, we must be cognizant of potential security vulnerabilities such as injection attacks. This code is subject to such an attack. To demonstrate this, use the workflow_dispatch event for the workflow in the Actions menu, put in a version and pass in the following as the arguments in the arguments field (NOTE: That is two backquotes around ls -la) ``ls -la``

The screenshot shows the GitHub Actions interface for the 'Java CI with Gradle' workflow. The workflow has a 'workflow_dispatch' event trigger. A modal is open for running the workflow, showing the 'Input Version' as '1.0.2' and 'Input Values' as '`ls -la`'. The workflow runs are listed below the modal.

Run	Status	Branch	Actor
Java CI with Gradle #73: Manually run by gwstudent2	Failed		gwstudent2
Java CI with Gradle #72: Manually run by gwstudent2	Success		gwstudent2
fix: version 3	Success	main	gwstudent2
fix: version2	Failed		gwstudent2

3. After the run completes, look at the output of the step. Notice that it ran successfully, but it has actually run the ``ls -la`` command directly on the runner system. The command was innocuous in this case, but this could have been a more destructive command.

Code Issues 6 Pull requests Actions Projects Wiki Security Insights

Java CI with Gradle Java CI with Gradle #46 Re-run all jobs ...

Summary

Jobs

- build
- test-run
- create-issue-on-failure

test-run
succeeded 1 hour ago in 2s

Search logs

- Set up job 1s
- Download candidate artifacts 0s
- Execute test 0s
 - 1 Run `chmod +x ./test-script.sh`
 - 5 total
 - 6 16
 - 7 `drwxr-xr-x`
 - 8 `drwxr-xr-x`
 - 9 `drwxr-xr-x`
 - 10 3
 - 11 3
 - 12 3
 - 13 runner
 - 14 runner
 - 15 runner
 - 16 runner

- Let's fix the command to not be able to execute the code in this way. We can do that by placing the output into an environment variable first and then passing that to the step. Edit the `pipeline.yaml` file and change the code to look like the following (pay attention to how things line up):

```
env:
  ARGS: ${github.event.inputs.myValues}
run: |
  chmod +x ./test-script.sh
  ./test-script.sh ${needs.build.outputs.artifact-tag || github.event.inputs.myVersion} "$ARGS"
```

```

65 test-run:
66
67   runs-on: ubuntu-latest
68   needs: build
69
70   steps:
71   - name: Download candidate artifacts
72     uses: actions/download-artifact@v3
73     with:
74       name: greetings-jar
75
76   - name: Execute test
77     shell: bash
78     env:
79       ARGS: ${github.event.inputs.myValues}
80     run: |
81       chmod +x ./test-script.sh
82       ./test-script.sh ${needs.build.outputs.artifact-tag || github.event.inputs.myVersion} "$ARGS"
83
84   create-issue-on-failure:
85
86     runs-on: ubuntu-latest
87     needs: test-run

```

- Commit back the changes and wait till the action run for the push completes.
- Now, you can execute the code again with the same arguments as before.

Workflows New workflow **Java CI with Gradle** pipeline.yml

All workflows

Java CI with Gradle

create-failure-issue

Filter workflow runs

75 workflow runs

Event Status Branch Actor

This workflow has a workflow_dispatch event trigger. Run workflow

- ✓ **fix: security update**
Java CI with Gradle #75: Commit 0f3f6df pushed by gwstudent2 main
- ✓ **Java CI with Gradle**
Java CI with Gradle #74: Manually run by gwstudent2
- ✗ **Java CI with Gradle**
Java CI with Gradle #73: Manually run by gwstudent2
- ✓ **Java CI with Gradle**

Use workflow from
Branch: main
Input Version
1.0.3
Input Values
`'ls -la'`
Run workflow

2 hours ago

7. Notice that this time, the output did not run the commands, but just echoed them back out as desired.

END OF LAB

Lab 11 – Separating out jobs into a separate action

Purpose: In this lab, we'll look at how to separate our testing job into a separate action.

- We're going to make our test script into a composite action. To do this, let's first create a new branch to work with called "test-action". In the "Code" tab, click on the branch dropdown that says "main". Then in the text area that says, "Find or create a branch...", enter the text "test-action". Then click on the **Create branch: test-action from 'main'** link.

<> Code Issues 18 Pull requests

main 3 branches 17 tags

Switch branches/tags

test-action

Branches Tags

Create branch: test-action from 'main'

View all branches

- You should now be on the "test-action" branch. The "test-script.sh" file will be the basis for our new composite action. So, let's move it to a separate local area for local actions. Select the test-script.sh file, edit it, and then add **".github/actions/test-action"** to the path as shown below.

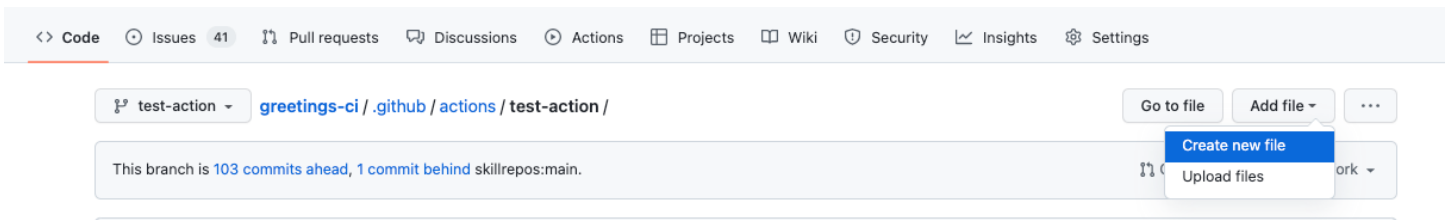
The screenshot shows the GitHub interface for the repository 'greetings-ci'. The file 'test-script.sh' is open in the 'test-action' branch. The file content is as follows:

```

1 # Simple test script for greetings jar
2
3 set -e
4
5 java -jar build/libs/greetings-ci-$1.jar ${@:2} > output.bench
6 IFS=' ' read -ra ARR <<< "${@:2}"
7 for i in "${ARR[@]}; do
8     grep "$i$" output.bench
9 done
10

```

- Click on the green button to commit your changes in the "test-action" branch. Notice that no workflows were kicked off because we don't have events defined in our workflow related to the "test-action" branch.
- Now, let's create the action.yml file for our test action. You will need to create a new file in the path "greetings-ci/.github/actions/test-action" directory by going there, clicking on "Add file" and then clicking on "Create new file"



- Name the new file "action.yml". So the full path would be "greetings-ci/.github/actions/test-action/action.yml". For the file contents, you can either copy and paste from below or from the file at [extra/action.yml](#) Commit the file when done to the "test-action" branch.

```

name: 'Test Action'
description: 'Runs a simple execution to validate compiled built deliverable'
author: 'attendee'
inputs:
  artifact-version: # semantic version of the artifact from build
    description: 'built version of artifact'
    required: true
    default: '1.0.0'
  arguments-to-print: # rest of arguments to echo out
    description: 'arguments to print out'
runs:
  using: "composite"
  steps:
    - name: Download candidate artifacts
      uses: actions/download-artifact@v3
      with:
        name: greetings-jar
    - id: test-run
      env:
        ARGS: ${ inputs.arguments-to-print }
      run: |
        chmod +x ${ github.action_path }/test-script.sh
        ${ github.action_path }/test-script.sh ${ inputs.artifact-version } "$ARGS"
      shell: bash

```

greetings-ci / .github / actions / test-action / action.yml in test-action

<> Edit file Preview changes

```

1 name: 'Test Action'
2 description: 'Runs a simple execution to validate compiled built deliverable'
3 author: 'attendee'
4 inputs:
5   artifact-version: # semantic version of the artifact from build
6     description: 'built version of artifact'
7     required: true
8     default: '1.0.0'
9   arguments-to-print: # rest of arguments to echo out
10     description: 'arguments to print out'
11 runs:
12   using: "composite"
13   steps:
14     - name: Download candidate artifacts
15       uses: actions/download-artifact@v3
16       with:
17         name: greetings-jar
18     - id: test-run
19       env:
20         ARGS: ${ inputs.arguments-to-print }
21       run: |
22         chmod +x ${ github.action_path }/test-script.sh
23         ${ github.action_path }/test-script.sh ${ inputs.artifact-version } "$ARGS"
24       shell: bash
25
```

6. This is all we need for our basic composite action. Notice that we've essentially copied over a couple of steps into our composite action that were in the original workflow file. So, we can go back and modify the original workflow file to use our new action. **Still in the "test-action" branch, edit the file "greetings-ci/.github/workflows/pipeline.yaml".**

Replace the current steps of test-run, with the new set as shown below (checking alignment). Notice that we need to add a checkout action here to have the necessary pieces from our test-action directory present for the action to get to. Then we just call our new action passing in the parameters. **Commit the file when done.**

- uses: actions/checkout@v3
- name: run-test
 uses: ../.github/actions/test-action
 with:
 artifact-version: \${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }
 arguments-to-print: \${ github.event.inputs.myValues }

```

65
66 test-run:
67
68   runs-on: ubuntu-latest
69   needs: build
70
71   steps:
72
73     - uses: actions/checkout@v3
74
75     - name: run-test
76       uses: ../.github/actions/test-action
77       with:
78         artifact-version: ${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }
79         arguments-to-print: ${ github.event.inputs.myValues }
80
81 create-issue-on-failure:

```

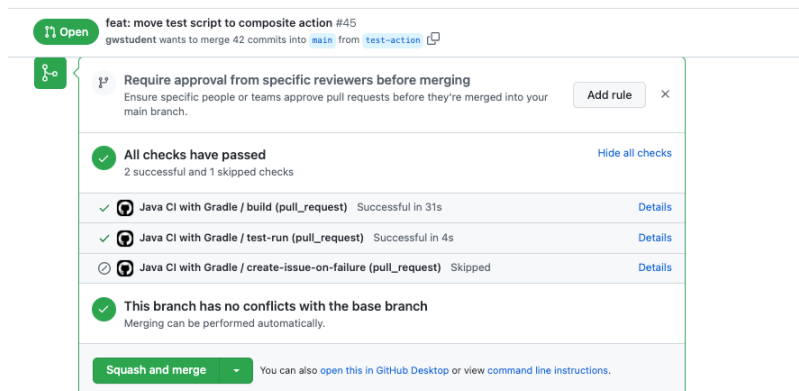
7. Finally, let's merge in the "test-action" branch to the "main" branch. Click on the top-level "Pull requests" menu. You should see a yellow bar with text that indicates the "test-action" branch had recent pushes. Click on the green "Compare & pull request" button.

The screenshot shows the GitHub interface with the 'Pull requests' tab selected. A yellow banner at the top states 'test-action had recent pushes 3 minutes ago' with a green 'Compare & pull request' button. Below the banner, there's a search bar with 'is:pr is:open' and filters for 'Labels: 9' and 'Milestones: 0'. A 'New pull request' button is on the right. The main area shows '0 Open' and '3 Closed' pull requests, with columns for Author, Label, Projects, Milestones, Reviews, Assignee, and Sort.

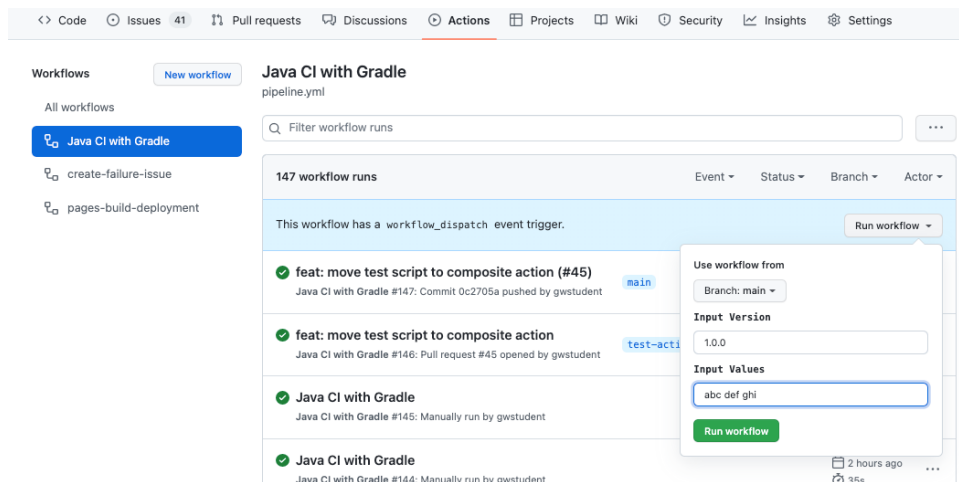
8. As we've done before, change the "base" portion to be the current repo. After this, it should show that you can merge from the "test-action" branch to the "main" branch of your "greetings-ci" repository. Fill in an appropriate comment and then click the green "Create pull request" button.

The screenshot shows the 'Comparing changes' page in GitHub. The 'base repository' is 'gwstudent/greetings-ci' and the 'base' is 'main'. The 'head repository' is 'gwstudent/greetings-ci' and the 'compare' is 'test-action'. A dropdown menu for 'Choose a Base Repository' is open, showing 'gwstudent/greetings-ci' selected. The page indicates 'Able to merge. These branches can be automatically merged.' Below this, there's a text input field with 'feat: move test script to composite action'. The 'Write' tab is active, showing a 'Leave a comment' section. On the right, there are sections for 'Reviewers' (No reviews), 'Assignees' (No one—assign yourself), 'Labels' (None yet), 'Projects' (None yet), 'Milestone' (No milestone), and 'Development'.

- With the Pull Request created, the automated merge checks should run and succeed. After that, you can click on the "Squash and merge" button to complete the merge. (Select "Squash and merge" from the down arrow on the right of the green button if needed.). Confirm when asked. The merge should complete, and the Pull Request should be closed.



- A workflow run will have occurred because of the merge. But if you want to try out the merged code with the action more fully, you can do a manual workflow run as before.



END OF LAB

Lab 12 – Adding Environments and Releases

Purpose: In this lab, we'll look at how to add staging (blue, green) and production environments and releases.

- Let's add some deploy jobs to our pipeline.yaml file. **Edit the .github/workflows/pipeline.yaml file.** For simplicity, we can just do this **in the main branch**.



2. We can illustrate blue/green deployment with new branches such as "blue" and "green". So, let's modify the "on:" section first to run the workflow on a push to any of these. Modify the **on: push:** command to be like the following.

```
on:
  push:
    branches: [ "main", "blue", "green" ]
7
8   name: Java CI with Gradle
9
10  on:
11    push:
12      branches: [ "main", "blue", "green" ]
13    pull_request:
14      branches: [ "main" ]
```

3. You can also remove the "pull_request" portion.

```
9
10  on:
11    push:
12      branches: [ "main", "blue", "green" ]
13    workflow_dispatch:
14      inputs:
15        myVersion:
```

4. Now, let's add the job for deploying a "stage" environment/release. This job can be inserted between the "test-run" job and the "create-issue-on-failure" job. The code for this job is already done for you and can be copied from the file [extra/deploy-stage.txt](#) Just copy and paste.

(Note: You may need to add a space or two at the front of the first line of output once pasted to get it to line up correctly.)

This code essentially does the following:

- Waits for the build and test jobs to complete (line 79)
- Checks to see if the branch being pushed to is "blue" or "green" (line 80)
- Establishes an environment called "staging" (line 83)
- Sets the associated URL for the environment to the releases page (line 85)
- Checkouts the source code (line 87-90)
- Downloads the jar we built (line 92-95)
- Calls a GitHub Action to create a release that: (line 97-105)
 - is based on the tag we got from the build
 - is set as a draft and prerelease
 - includes the jar file we've built

```

66     steps:
67
68     - uses: actions/checkout@v3
69
70     - name: run-test
71       uses: ../github/actions/test-action
72       with:
73         artifact-version: ${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }
74         arguments-to-print: ${ github.event.inputs.myValues }
75
76
77   deploy-stage:
78
79     needs: [build, test-run]
80     if: github.ref == 'refs/heads/blue' || github.ref == 'refs/heads/green'
81
82     runs-on: ubuntu-latest
83     environment:
84       name: staging
85       url: https://github.com/${ github.repository }}/releases/tag/v${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }
86
87     steps:
88
89     - uses: actions/checkout@v3
90       with:
91         fetch-depth: 0
92
93     - name: Download candidate artifacts
94       uses: actions/download-artifact@v3
95       with:
96         name: greetings-jar
97
98     - name: GH Release
99       uses: softprops/action-gh-release@v0.1.14
100       with:
101         tag_name: v${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }
102         prerelease: true
103         draft: true
104
105     - name: ${ github.ref_name }
106       files: |
107         greetings-ci-${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }.jar
108

```

- Now, let's add the job for deploying a "prod" (production) environment/release from a pull-request being merged into "main". This job can be inserted between the "deploy-stage" job and the "create-issue-on-failure" job. The code for this job is already done for you and can be copied from the file [extra/deploy-prod.txt](#). Just copy and paste.

(Note: You may need to add a space or two at the front of the first line of output once pasted to get it to line up correctly.)

This code essentially does the following:

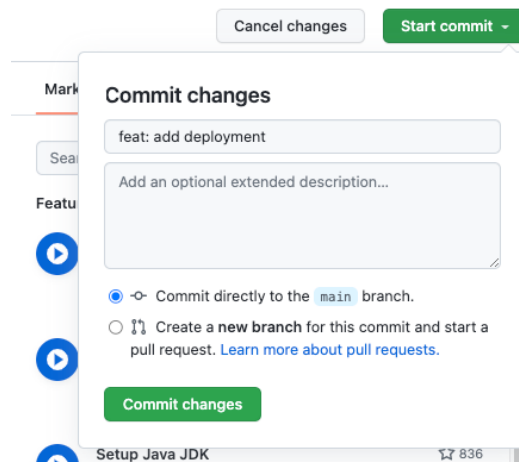
- Waits for the build and test jobs to complete (line 114)
- Checks to see if we got here on the main branch (line 115)
- Establishes an environment called "production" (line 119)
- Sets the associated URL for the environment to the releases page (line 120)
- Checkouts the source code (line 123-125)
- Downloads the jar we built (line 127-130)
- Calls a GitHub Action to create a release that: (line 132-140)
 - is based on the tag we got from the build
 - is named as "Production"
 - includes the jar file we've built and the CHANGELOG

```

105
106     files: |
107         greetings-ci-`${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar
108
109
110
111
112     deploy-prod:
113
114         needs: [build, test-run]
115         if: github.ref == 'refs/heads/main'
116
117         runs-on: ubuntu-latest
118         environment:
119             name: production
120             url: https://github.com/${ github.repository }}/releases/tag/v`${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
121         steps:
122
123             - uses: actions/checkout@v3
124               with:
125                 fetch-depth: 0
126
127             - name: Download candidate artifacts
128               uses: actions/download-artifact@v3
129               with:
130                 name: greetings-jar
131
132             - name: GH Release
133               uses: softprops/action-gh-release@v0.1.14
134               with:
135                 tag_name: v`${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}
136                 generate_release_notes: true
137                 name: Production
138                 files: |
139                     CHANGELOG.md
140                     greetings-ci-`${ needs.build.outputs.artifact-tag || github.event.inputs.myVersion }}.jar
141
142
143     create-issue-on-failure:

```

- Go ahead and commit your changes **to the main branch**. You can include a "feat" conventional commit message.



- This will kick off a new run of the workflow and will create an initial production deployment because of a change in main. After the run completes, you can click on the link in the deploy-prod job in the "Jobs" view to see the release.

✓ feat: add deployment #30

Summary

Jobs

- ✓ build
- ✓ test-run
- ⌚ deploy-stage
- ✓ deploy-prod
- ⌚ create-issue-on-failure

Run details

- ⌚ Usage
- 📄 Workflow file

Triggered via push 2 minutes ago

gwstudent pushed · 123d14c main

Status

Success

Total duration

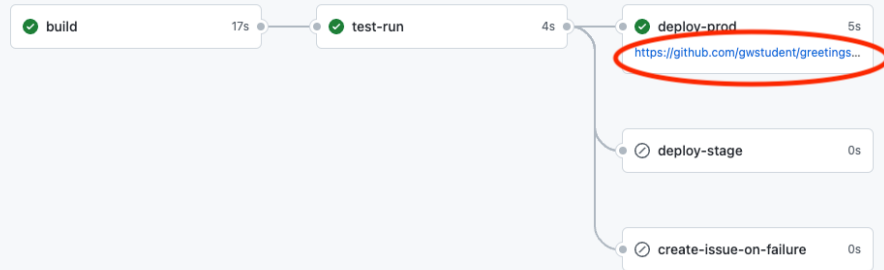
53s

Artifacts

1

pipeline.yml

on: push



gwstudent / greetings-ci Public

forked from skillrepos/greetings-ci

Pin

Watch 0

Fork 52

- <> Code
- ⌚ Issues 2
- 🔗 Pull requests
- ▶ Actions
- 📁 Projects
- 📖 Wiki
- 🛡 Security
- 📊 Insights
- ⚙ Settings

Releases / v0.6.0

Production

Latest

Compare

📄 🗑

github-actions released this 8 minutes ago · v0.6.0 · 7017919

What's Changed

- Test action by @gwstudent in #7

Full Changelog: v0.5.1...v0.6.0

Contributors



gwstudent

Assets 3

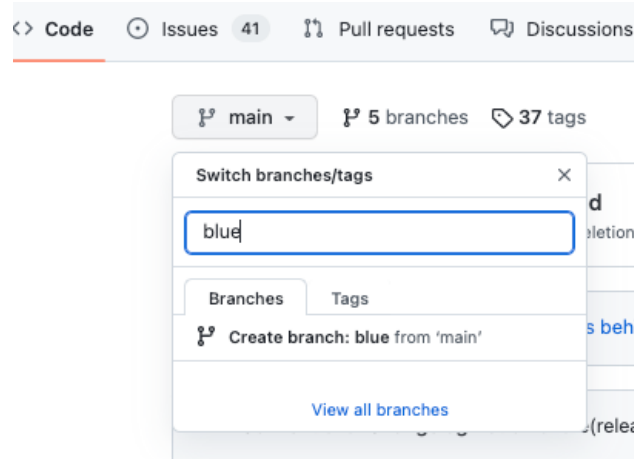
CHANGELOG.md	1.15 KB	8 minutes ago
Source code (zip)		8 minutes ago
Source code (tar.gz)		8 minutes ago

END OF LAB

Lab 13 – Exercising the entire workflow

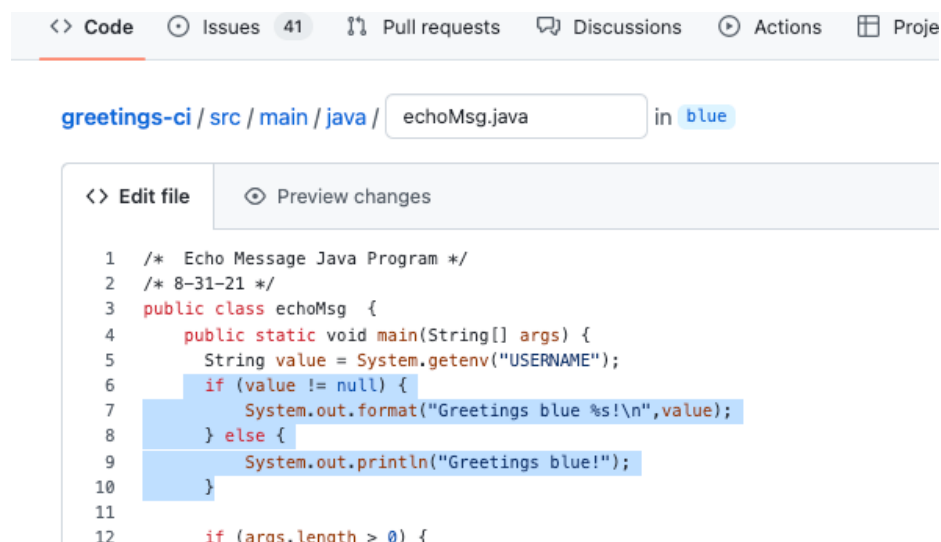
Purpose: In this lab, we'll see how to make a change in source code and have it processed through the pipeline.

1. In the example of using a "blue/green" environment, let's **create a branch called "blue" from the "main" branch** to make some changes on. Do this just as you've done before. (Note that you will need to make sure you're on the main branch before creating the blue one.)

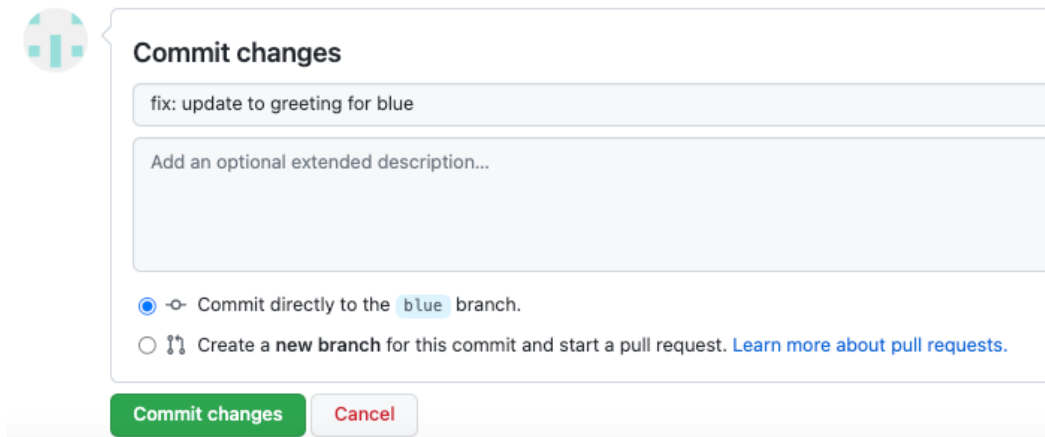


2. In the "blue" branch, edit the file `src/main/java/echoMsg.java`. Make a simple, non-breaking change like adding "blue" to the lines that print out "Greetings". See text and figure below.

```
if (value != null) {  
    System.out.format("Greetings blue %s!\n",value);  
} else {  
    System.out.println("Greetings blue!");  
}
```



3. Commit the changes with an appropriate "fix: " conventional commit message.



Commit changes

fix: update to greeting for blue

Add an optional extended description...

☒ Commit directly to the `blue` branch.

☐ Create a new branch for this commit and start a pull request. [Learn more about pull requests.](#)

Commit changes **Cancel**

4. After the workflow run completes, you can click on the run and look at the job graph. You should be able to see that it executed the build and test pieces and then deployed it to the stage environment.

✓ fix: update to greeting for blue Java CI with Gradle #170 Re-run all jobs ...

Summary

Jobs

- ✓ build
- ✓ test-run
- ✓ deploy-stage
- deploy-prod
- create-issue-on-failure

Triggered via push 2 minutes ago

gwstudent pushed → c9a2ff0 `blue`

Status: **Success** Total duration: **52s** Artifacts: **1**

pipeline.yml
on: push

```

graph LR
    build[build 15s] --> test-run[test-run 4s]
    test-run --> deploy-prod[deploy-prod 0s]
    test-run --> deploy-stage[deploy-stage 4s]
    test-run --> create-issue-on-failure[create-issue-on-failure 0s]
  
```

deploy-stage
<https://github.com/gwstudent/greetings...>

5. Now, click on the link in the "deploy-stage" box. This will take you to the tagged version of the source repo.

<> Code Issues 41 Pull requests Discussions Actions Projects Wiki Security Insights Settings

Releases **Tags**

Create release from tag **Delete**

v0.12.2
a5926b9

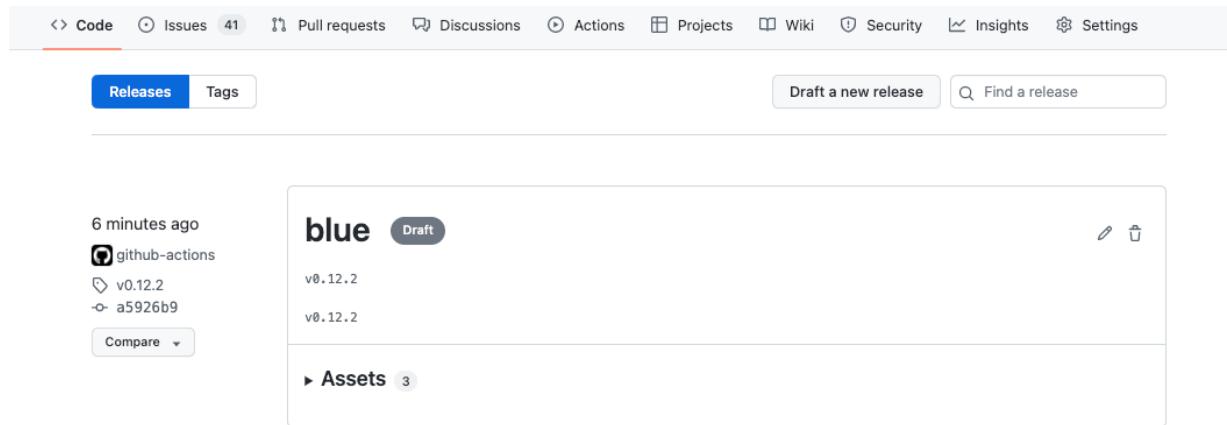
Compare

v0.12.2
tagged this 4 minutes ago
v0.12.2

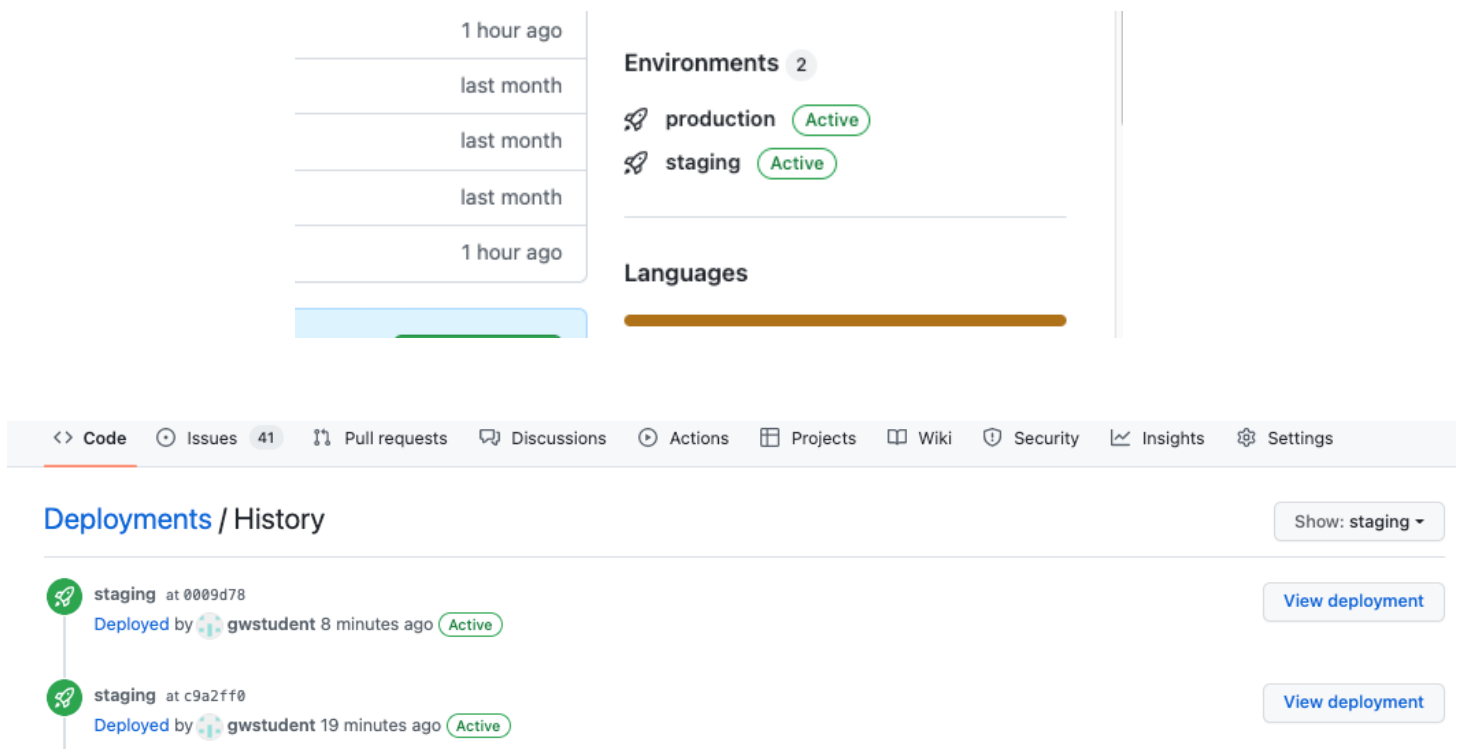
▼ **Assets** 2

Source code (zip)	4 minutes ago
Source code (tar.gz)	4 minutes ago

6. If you click on the "Releases" item next to "Tags", you can see the draft release that was created.

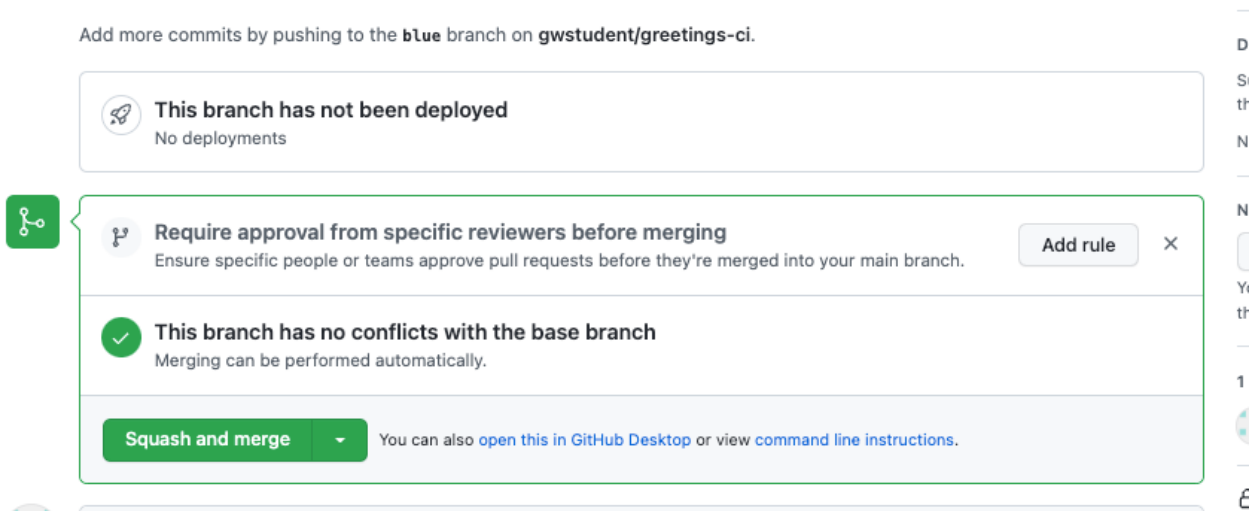


7. And, if you click on the main code page, in the lower right, you'll be able to see a new "Staging" environment. You can click on that to see a list of recent deployments there.

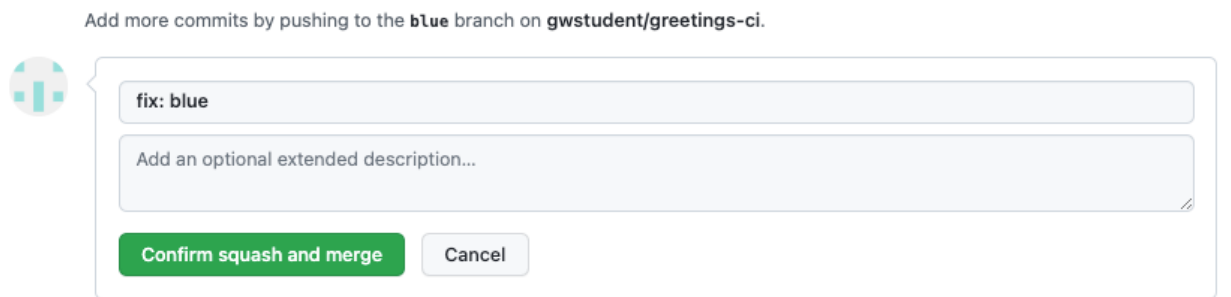


8. Since everything built ok, we can deploy this change to the production environment. To merge the changes, we can just create a pull request to main and merge it. In the "Code" page for your repository, there may be a yellow bar that says "blue had recent pushes..." If so, click on the big green "Compare & pull request" button.

10. At this point, you can go ahead and click the "Squash and merge" button when available and confirm.



11. You can edit the main comment to have something like "fix: blue" in it and do what you want with the other commit messages. Then go ahead and click the "Confirm squash and merge" button.



12. This should kick off another run of the action workflow in main. Because it runs in main, it should kick off the deploy-prod job.

✓ fix: blue Java CI with Gradle #185

Summary

Jobs

- ✓ build
- ✓ test-run
- ⌚ deploy-stage
- ✓ deploy-prod
- ⌚ create-issue-on-failure

Triggered via push 1 minute ago

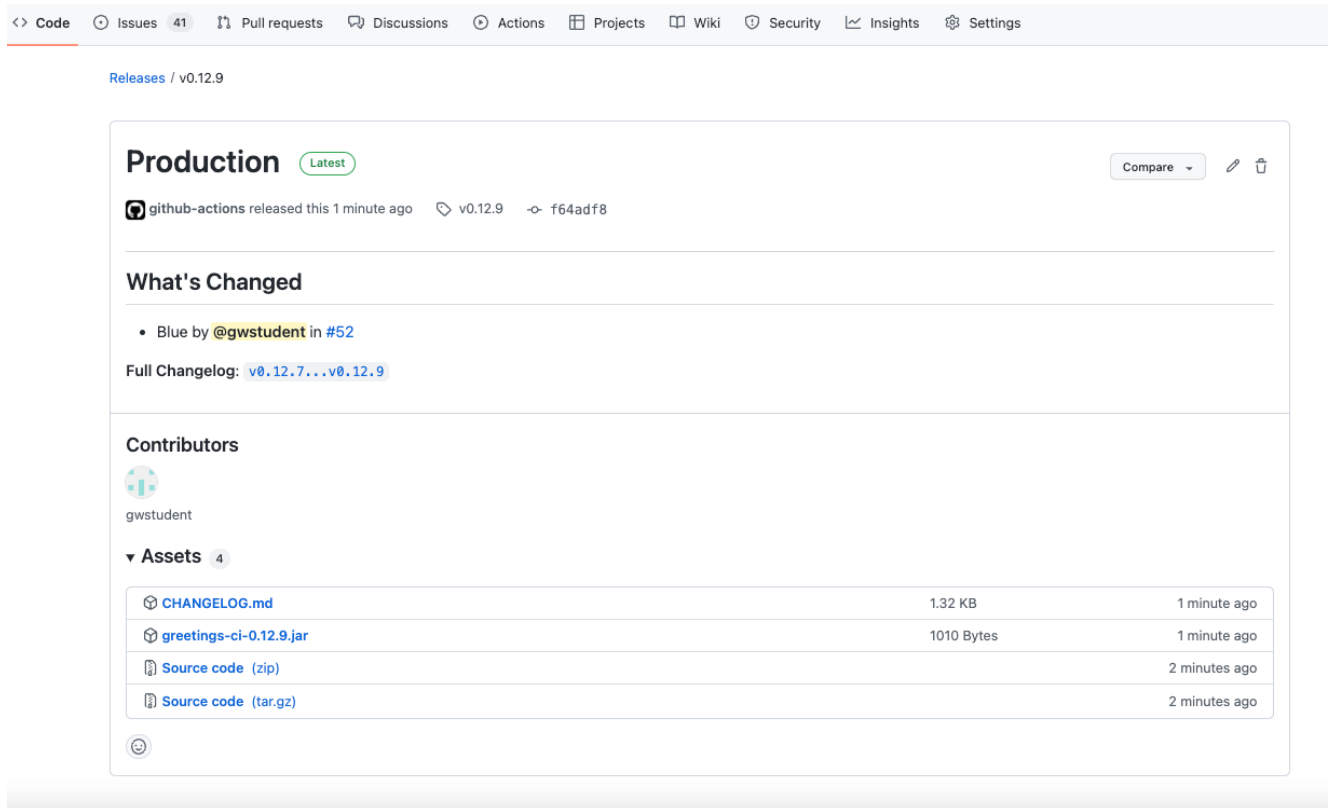
gwstudent pushed 56cb457 main

Status	Total duration	Artifacts
Success	56s	1

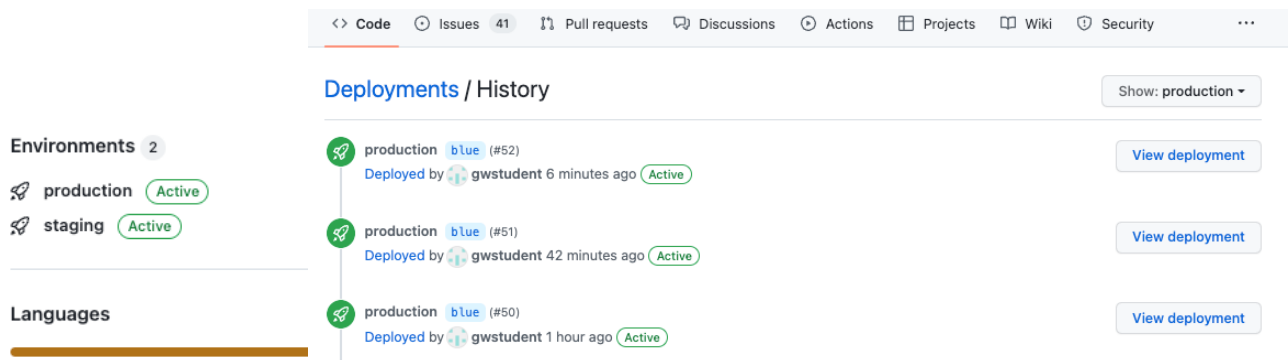
pipeline.yml
on: push

```
graph LR; build[build 14s] --> test-run[test-run 4s]; test-run --> deploy-prod[deploy-prod 10s]; test-run --> deploy-stage[deploy-stage 0s]; test-run --> create-issue-on-failure[create-issue-on-failure 0s];
```

13. After this completes, you can click on the link in the "deploy-prod" box to see the release it created.



14. You can also now see a Production environment available from the main repo page. You can click on it and see the deployments to production. Clicking on "View deployment" will take you to the same kind of page as the previous step did.



15. If you want, you can repeat the same exercise with a "green" branch to see how it works the same.

END OF LAB