Skip to main content

PlaywrightDocsAPI
Node.js

- Node.js
- Python
- Java
- .NET

Community
Search⌘K

- Getting Started
  - Installation
  - Writing tests
  - Generating tests
  - Running and debugging tests
  - Trace viewer
  - Setting up CI
- Getting started - VS Code
- Release notes
- Canary releases
- Playwright Test
  - Test configuration
  - Test use options
  - Annotations
  - Command line

Setting up CI

Introduction

Playwright tests can be run on any CI provider. This guide covers one way of running tests on GitHub using GitHub actions. If you would like to learn more, or how to configure other CI providers, check out our detailed doc on Continuous Integration.

You will learn

- How to set up GitHub Actions
- How to view test logs
- How to view the HTML report
- How to view the trace
- How to publish report on the web

Setting up GitHub Actions

When installing Playwright using the VS Code extension or with `npm init playwright@latest` you are given the option to add a GitHub Actions workflow. This creates a `playwright.yml` file inside a `.github/workflows` folder containing everything you need so that your tests run on each push and pull request into the main/master branch. Here's how that file looks:

.github/workflows/playwright.yml
```yaml
name: Playwright Tests
on:
  push:
    branches: [ main, master ]
  pull_request:
    branches: [ main, master ]
jobs:
  test:
    timeout-minutes: 60
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-node@v4
      with:
        node-version: lts/*
    - name: Install dependencies
      run: npm ci
    - name: Install Playwright Browsers
      run: npx playwright install --with-deps
    - name: Run Playwright tests
      run: npx playwright test
    - uses: actions/upload-artifact@v4
      if: ${{ !cancelled() }}
      with:
        name: playwright-report
        path: playwright-report/
        retention-days: 30
```

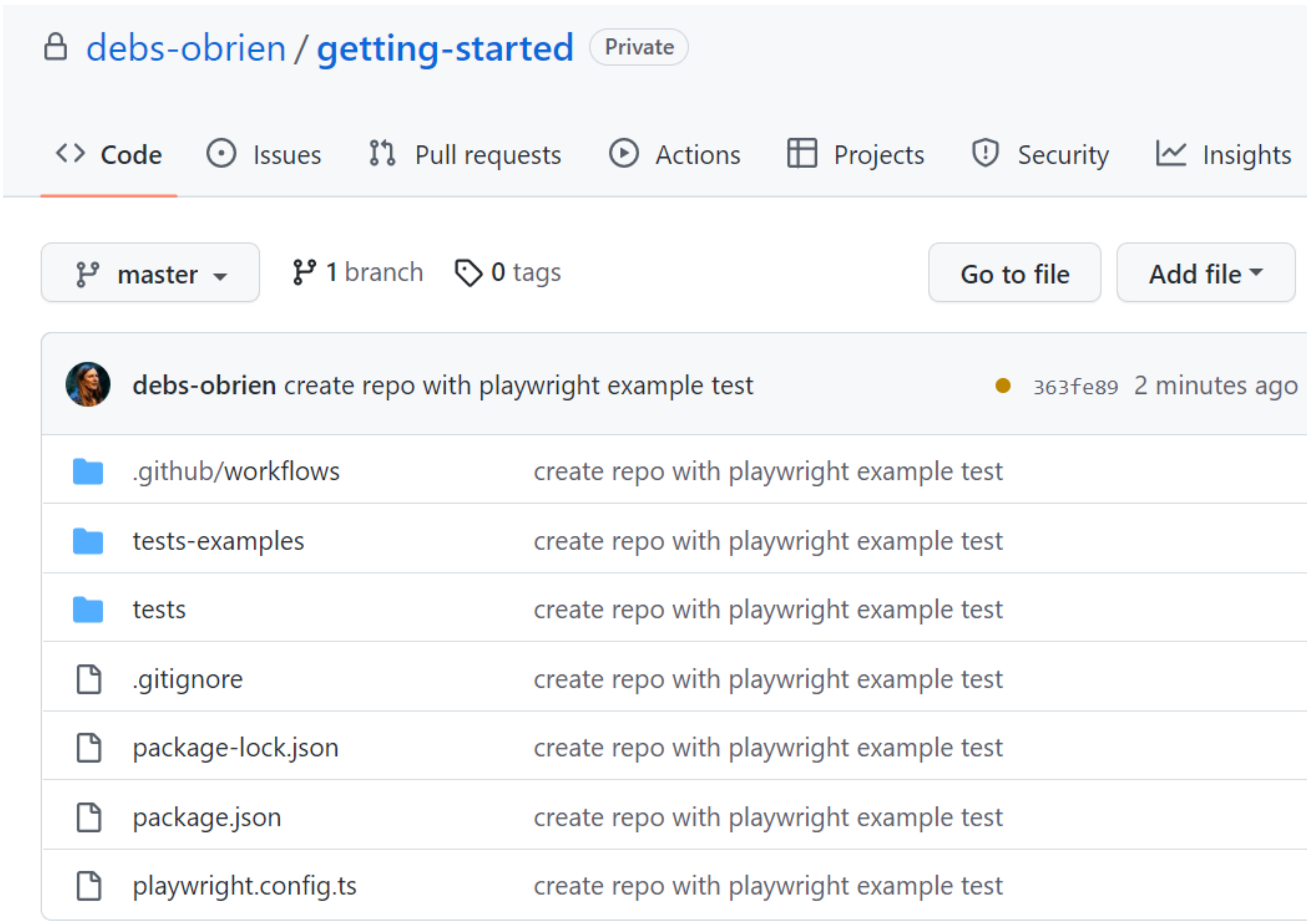
The workflow performs these steps:

1. Clone your repository 2. Install Node.js 3. Install NPM Dependencies 4. Install Playwright Browsers 5. Run Playwright tests 6. Upload HTML report to the GitHub UI

To learn more about this, see "Understanding GitHub Actions".

Create a Repo and Push to GitHub

Once you have your GitHub actions workflow setup then all you need to do is Create a repo on GitHub or push your code to an existing repository. Follow the instructions on GitHub and don't forget to initialize a git repository using the `git init` command so you can add, commit and push your code.

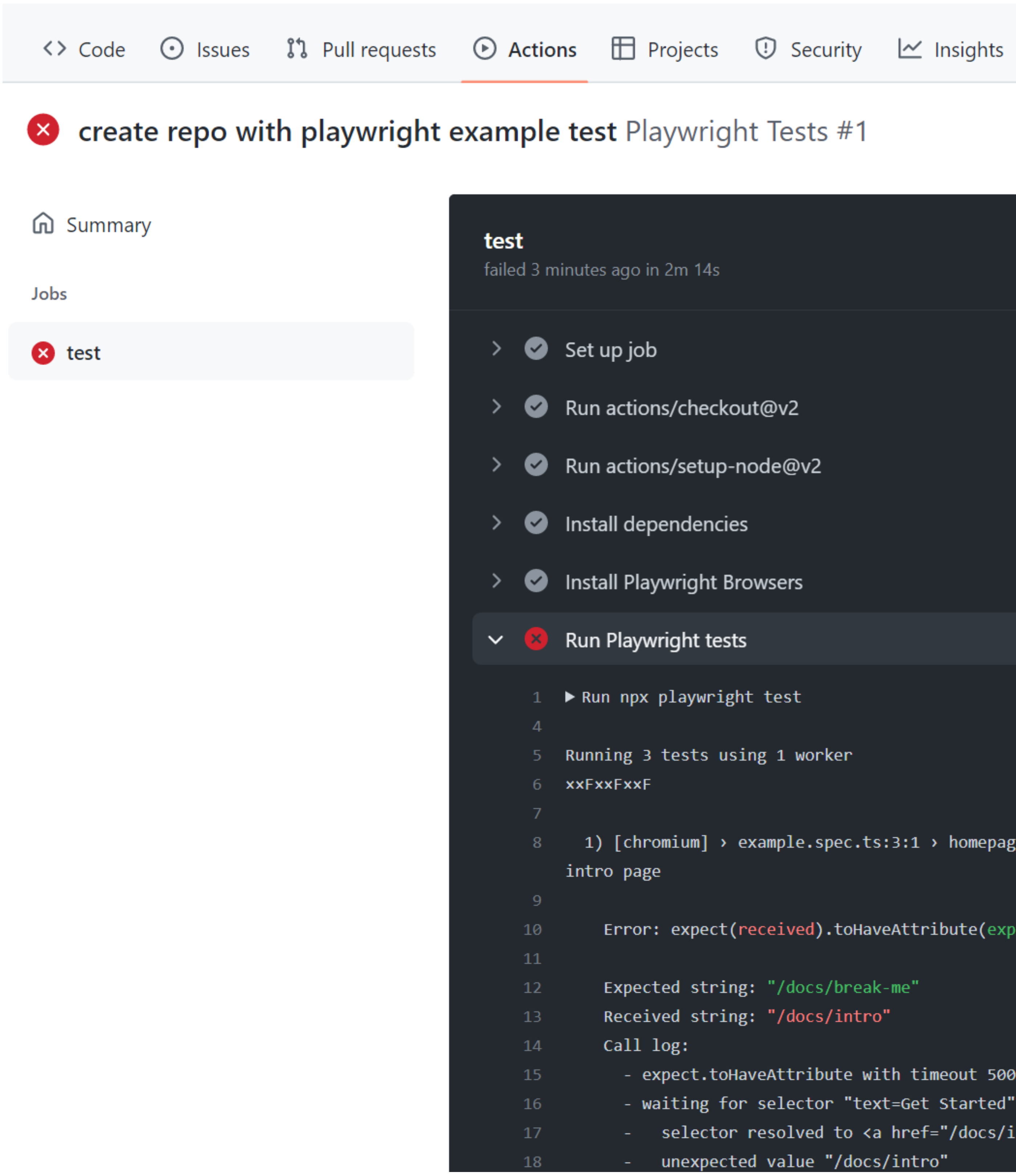

debs-obrien / getting-started Private

debs-obrien create repo with playwright example test ● 363fe89 2 minutes ago

.github/workflows	create repo with playwright example test
tests-examples	create repo with playwright example test
tests	create repo with playwright example test
.gitignore	create repo with playwright example test
package-lock.json	create repo with playwright example test
package.json	create repo with playwright example test
playwright.config.ts	create repo with playwright example test

Opening the Workflows

Click on the **Actions** tab to see the workflows. Here you will see if your tests have passed or failed.



create repo with playwright example test Playwright Tests #1

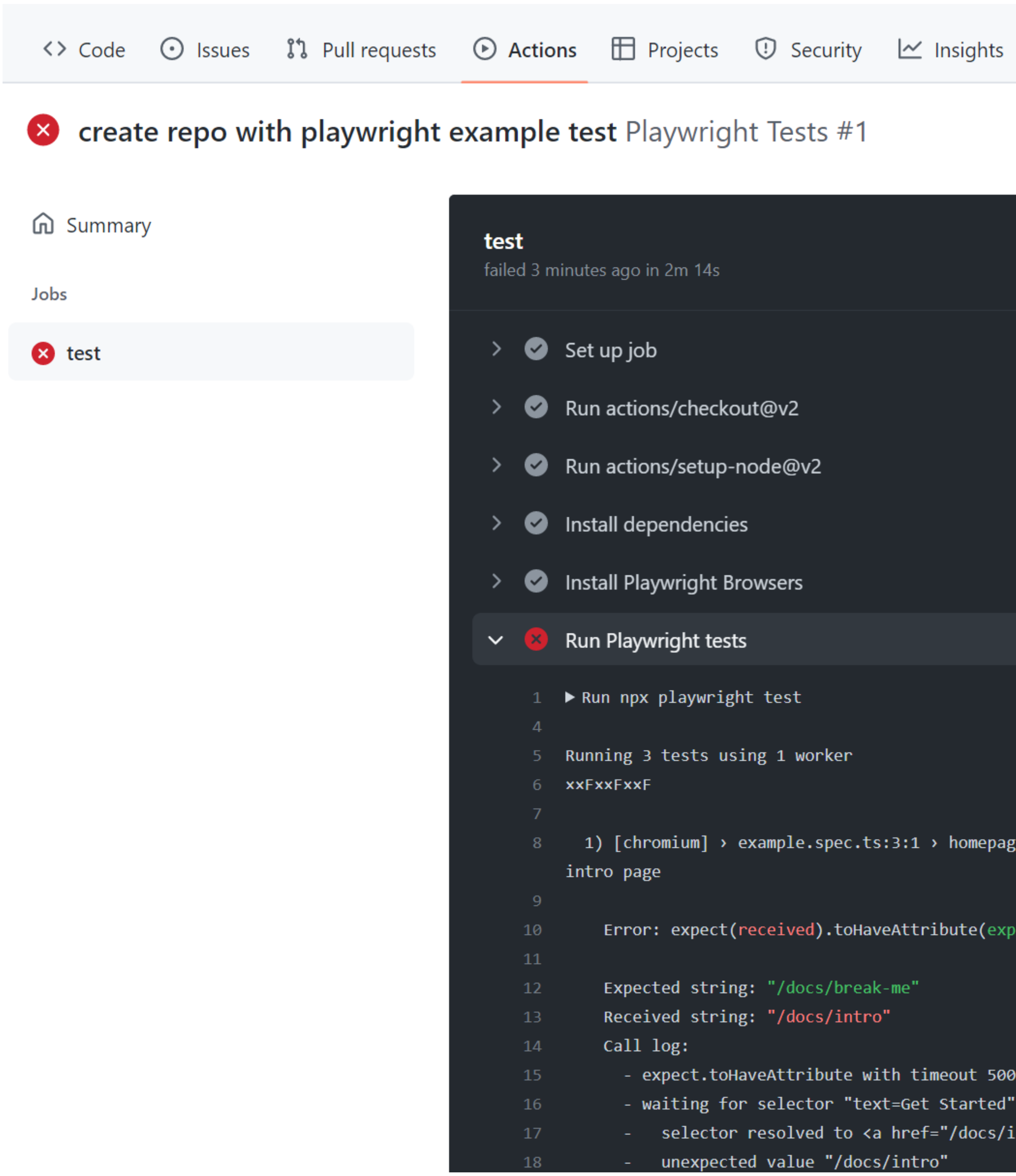
test
failed 3 minutes ago in 2m 14s

Set up job
Run actions/checkout@v2
Run actions/setup-node@v2
Install dependencies
Install Playwright Browsers
Run Playwright tests

```
▶ Run npx playwright test

Running 3 tests using 1 worker
xxFxxFxxF

  1) [chromium] › example.spec.ts:3:1 › homepag
intro page

    Error: expect(received).toHaveAttribute(exp

    Expected string: "/docs/break-me"
    Received string: "/docs/intro"
    Call log:
      - expect.toHaveAttribute with timeout 500
      - waiting for selector "text=Get Started"
      -   selector resolved to <a href="/docs/i
      -   unexpected value "/docs/intro"
```

Viewing Test Logs

Clicking on the workflow run will show you the all the actions that GitHub performed and clicking on **Run Playwright tests** will show the error messages, what was expected and what was received as well as the call log.



create repo with playwright example test Playwright Tests #1

test
failed 3 minutes ago in 2m 14s

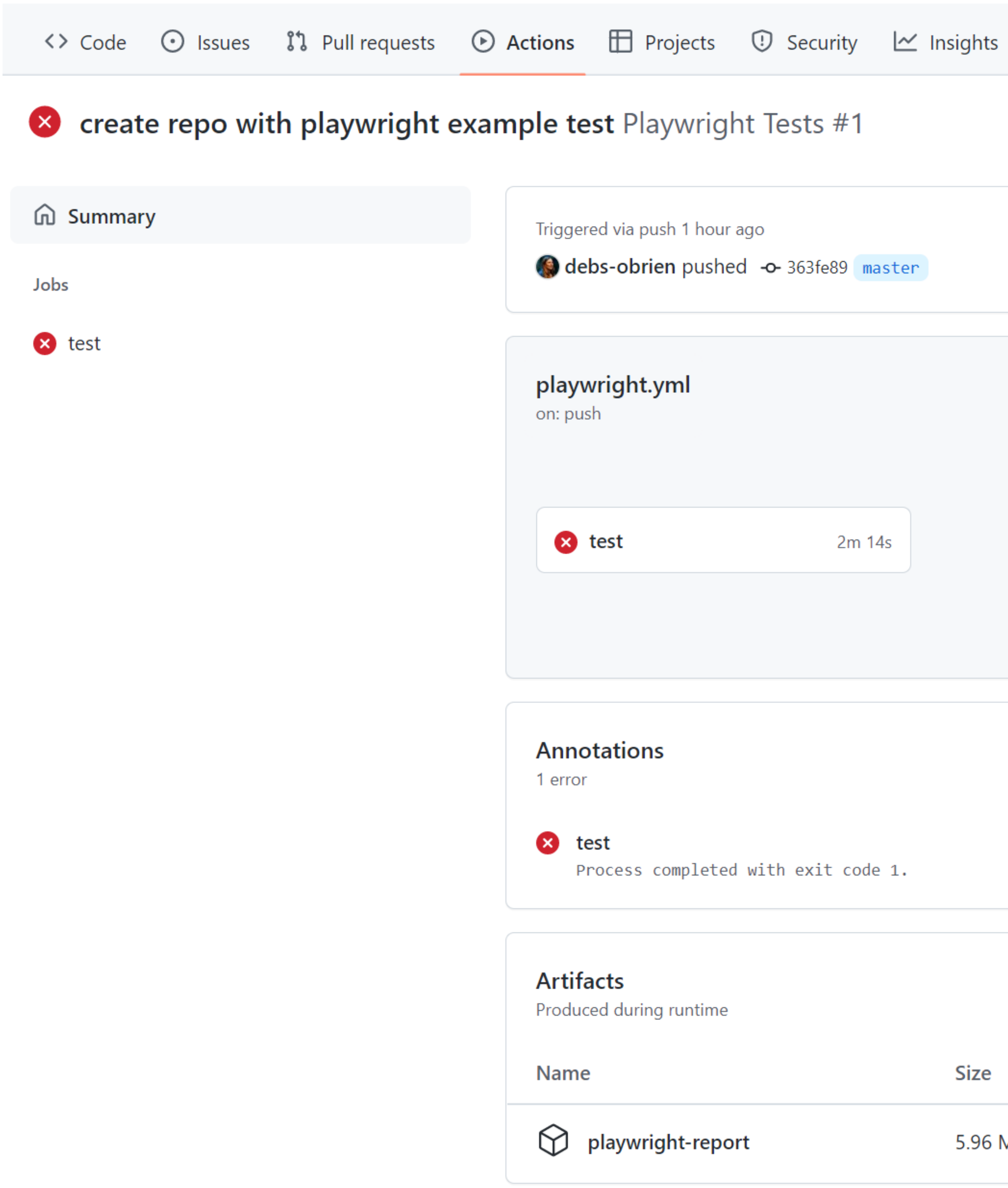
```
▶ Run npx playwright test

Running 3 tests using 1 worker
xxFxxFxxF

  1) [chromium] › example.spec.ts:3:1 › homepag
intro page

    Error: expect(received).toHaveAttribute(exp

    Expected string: "/docs/break-me"
    Received string: "/docs/intro"
    Call log:
      - expect.toHaveAttribute with timeout 500
      - waiting for selector "text=Get Started"
      -   selector resolved to <a href="/docs/i
      -   unexpected value "/docs/intro"
```

HTML Report

The HTML Report shows you a full report of your tests. You can filter the report by browsers, passed tests, failed tests, skipped tests and flaky tests.

Downloading the HTML Report

In the Artifacts section click on the **playwright-report** to download your report in the format of a zip file.



create repo with playwright example test Playwright Tests #1

Triggered via push 1 hour ago

debs-obrien pushed 363fe89 master

playwright.yml
on: push

test 2m 14s

Annotations
1 error

test
Process completed with exit code 1.

Artifacts
Produced during runtime

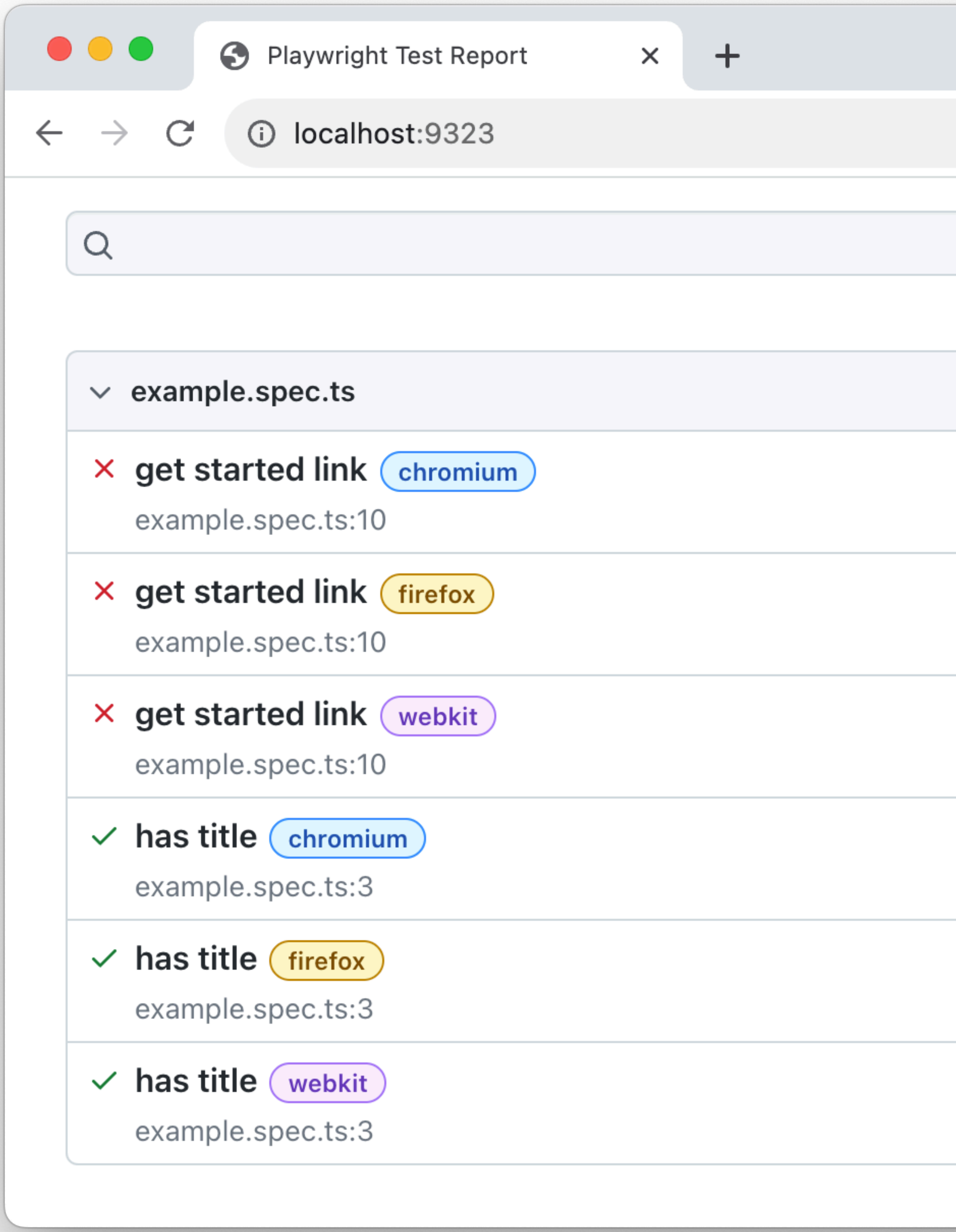
Name	Size
playwright-report	5.96 M

Viewing the HTML Report

Locally opening the report will not work as expected as you need a web server in order for everything to work correctly. First, extract the zip, preferably in a folder that already has

Playwright installed. Using the command line change into the directory where the report is and use `npx playwright show-report` followed by the name of the extracted folder. This will serve up the report and enable you to view it in your browser.

```
npx playwright show-report name-of-my-extracted-playwright-report
```



Playwright Test Report

localhost:9323

example.spec.ts

✗ get started link chromium
example.spec.ts:10

✗ get started link firefox
example.spec.ts:10

✗ get started link webkit
example.spec.ts:10

✓ has title chromium
example.spec.ts:3

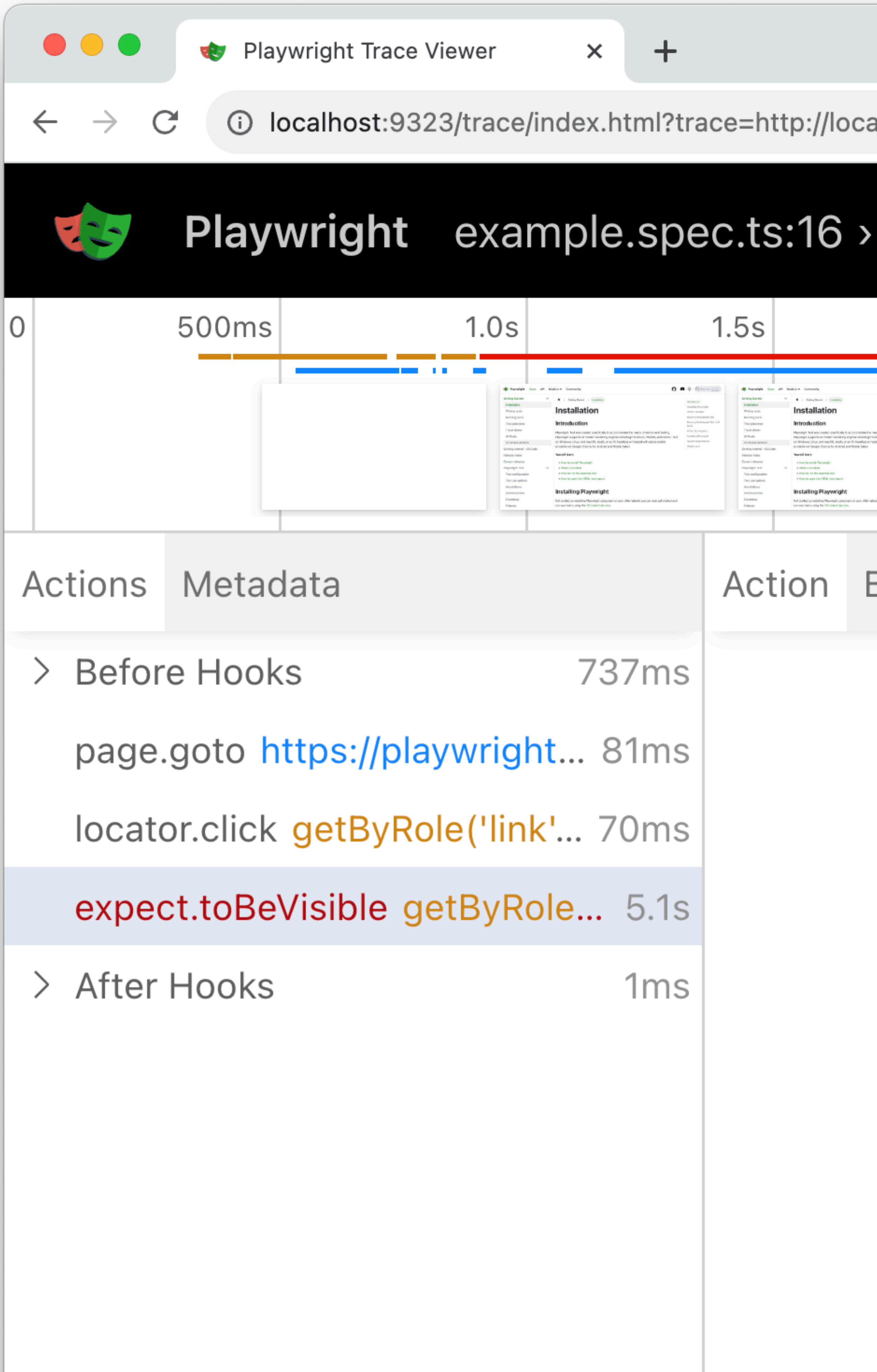
✓ has title firefox
example.spec.ts:3

✓ has title webkit
example.spec.ts:3

To learn more about reports check out our detailed guide on HTML Reporter

Viewing the Trace

Once you have served the report using `npx playwright show-report`, click on the trace icon next to the test's file name as seen in the image above. You can then view the trace of your tests and inspect each action to try to find out why the tests are failing.


Playwright Trace Viewer

localhost:9323/trace/index.html?trace=http://loca

Playwright example.spec.ts:16 ›

Before Hooks 737ms

page.goto https://playwright... 81ms

locator.click getByRole('link'... 70ms

expect.toBeVisible getByRole... 5.1s

After Hooks 1ms

Publishing report on the web

Downloading the HTML report as a zip file is not very convenient. However, we can utilize Azure Storage's static websites hosting capabilities to easily and efficiently serve HTML reports on the Internet, requiring minimal configuration.

1. Create an Azure Storage account.
2. Enable Static website hosting for the storage account.
3. Create a Service Principal in Azure and grant it access to Azure Blob storage. Upon successful execution, the command will display the credentials which will be used in the next step.

   ```
   az ad sp create-for-rbac --name "github-actions" --role "Storage Blob
   Data Contributor" --scopes
   /subscriptions/<SUBSCRIPTION_ID>/resourceGroups/<RESOURCE_GROUP_NAME>
   /providers/Microsoft.Storage/storageAccounts/<STORAGE_ACCOUNT_NAME>
   ```

4. Use the credentials from the previous step to set up encrypted secrets in your GitHub repository. Go to your repository's settings, under GitHub Actions secrets, and add the following secrets:
   - `AZCOPY_SPA_APPLICATION_ID`
   - `AZCOPY_SPA_CLIENT_SECRET`
   - `AZCOPY_TENANT_ID`

   For a detailed guide on how to authorize a service principal using a client secret, refer to this Microsoft documentation.

5. Add a step that uploads the HTML report to Azure Storage.

   .github/workflows/playwright.yml

   ```
   ...
       - name: Upload HTML report to Azure
         shell: bash
         run: |
           REPORT_DIR='run-${{ github.run_id }}-${{ github.run_attempt
   }}'
           azcopy cp --recursive "./playwright-report/*"
   "https://<STORAGE_ACCOUNT_NAME>.blob.core.windows.net/\$web/$REPORT_D
   IR"
           echo "::notice title=HTML report
   url::https://<STORAGE_ACCOUNT_NAME>.z1.web.core.windows.net/$REPORT_D
   IR/index.html"
         env:
           AZCOPY_AUTO_LOGIN_TYPE: SPN
           AZCOPY_SPA_APPLICATION_ID: '${{
   secrets.AZCOPY_SPA_APPLICATION_ID }}'
           AZCOPY_SPA_CLIENT_SECRET: '${{
   secrets.AZCOPY_SPA_CLIENT_SECRET }}'
           AZCOPY_TENANT_ID: '${{ secrets.AZCOPY_TENANT_ID }}'
   ```

The contents of the `$web` storage container can be accessed from a browser by using the public URL of the website.

NOTE

This step will not work for pull requests created from a forked repository because such workflow doesn't have access to the secrets.

What's Next

- Learn how to use Locators
- Learn how to perform Actions
- Learn how to write Assertions
- Learn more about the Trace Viewer
- Learn more ways of running tests on GitHub Actions
- Learn more about running tests on other CI providers // TODO: is this link correct?

Previous
Trace viewer

Next
Getting started - VS Code

- GitHub
- YouTube
- Blog
- Ambassadors