UnCodemy

Playwright in CI with GitHub Actions PDF

To run Playwright in GitHub Actions, create a workflow that checks out code, sets up Node, installs dependencies, installs browsers with dependencies, runs npx playwright test and uploads the HTML report as an artifact. Use secrets for credentials, cache what you can and set retries and workers suited to the runner.

Key takeaways

- A basic workflow needs checkout, Node setup, install, browser install, test and artifact upload.
- Use playwright install --with-deps to add system libraries on Linux runners.
- Upload the report with if: always() so failures keep their evidence.
- Store credentials as repository secrets, never in the workflow file.
- Use sharding to split long suites across jobs.

What is Playwright in GitHub Actions?

Playwright in GitHub Actions means running the Playwright test suite automatically inside a GitHub-hosted workflow on events such as pushes and pull requests, and saving reports and traces as downloadable artifacts.

Why run tests in CI?

Tests that only run on one laptop are easy to forget. A pipeline runs them on every pull request, in a clean environment, and blocks merges when something breaks. It also removes "works on my machine" arguments, because everyone sees the same result and the same report.

What does a basic workflow look like?

```
name: Playwright Tests
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
jobs:
  test:
    timeout-minutes: 30
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-node@v4
        with:
          node-version: lts/*
      - run: npm ci
      - run: npx playwright install --with-deps
      - run: npx playwright test
      - uses: actions/upload-artifact@v4
        if: ${{ !cancelled() }}
        with:
          name: playwright-report
          path: playwright-report/
          retention-days: 14
```

Check the current action versions in the official documentation before copying.

What does each step do?

Step	Purpose
checkout	Gets the repository code
setup-node	Installs a Node version
npm ci	Installs exact locked dependencies
playwright install --with-deps	Downloads browsers and Linux libraries
playwright test	Runs the suite
upload-artifact	Saves the report and traces

How do I handle secrets and environment values?

Add secrets in the repository settings and read them with `${{ secrets.NAME }}` in the `env` block of the step. Pass a base URL for the environment under test the same way. Do not print secrets in logs. For pull requests from forks, secrets are not available, so design those runs to use a public or mock environment.

How do I speed up the run?

Cache the npm folder with the Node setup option, split tests across jobs with `--shard=1/4`, run only changed areas for pull requests if your tests are organised by feature, and keep workers suited to the runner size. Avoid caching browser binaries blindly, because version mismatches cause confusing failures; follow the official guidance.

How do I keep reports useful?

Upload the HTML report and, for failures, the trace files. Team members can download the artifact, unzip it and run `npx playwright show-report`. For readable summaries, add the GitHub reporter or a JUnit file and publish results in the pull request. Set a retention period to control storage.

How do I run against a deployed environment?

Set `BASE_URL` to the staging address, wait for the deployment job to finish with `needs`, and use a smaller smoke set on every deploy with the full suite on a schedule. Add a `schedule` trigger with cron for nightly runs.

What causes CI-only failures?

Different screen size, missing fonts, slower machines, timezone, stricter network, missing environment variables and shared test data. Match the viewport in config, log environment details, run with the same container image locally if possible and read the trace before assuming the test is wrong.

What should I avoid?

Avoid infinite retries, running a huge suite on every commit with no sharding, hard-coded URLs, unpinned tool versions that change under you and silent failures. Keep the workflow file short and readable.

How do I run only the tests that matter on a pull request?

Tag tests with labels such as @smoke and @regression in their titles, then run `npx playwright test --grep @smoke` on pull requests and the full suite after merge or on a nightly schedule. This keeps feedback fast for developers while still protecting the main branch. Review the tag list regularly so that the smoke group stays small, stable and meaningful, and so that critical user journeys such as login, search and checkout are always covered.

How do I notify the team about failures?

Use the pull request status check as the first signal, since it blocks merging. For nightly runs, send a short message to the team chat tool through an action or a webhook stored in a secret, including the workflow link and the failed test names. Avoid noisy alerts: notify only on a failure after retries,

and assign an owner to look at each red run the same day so that failures do not become background noise that everyone ignores.

How do I practise this pipeline safely?

Create a small practice repository with two or three tests, add the workflow, open a pull request and watch the run. Break a test on purpose to see the red status and the artifact, then fix it. Doing this once on a safe project makes the real pipeline far less intimidating.

Frequently asked questions

How do I run Playwright tests in GitHub Actions?

Create a workflow that checks out code, installs Node dependencies and browsers, runs npx playwright test and uploads the report as an artifact.

Why use playwright install --with-deps?

It downloads the browsers and installs the system libraries they need on Linux runners, which avoids launch errors in CI.

How do I keep the report after a failure?

Upload the playwright-report folder with a condition that runs even when tests fail, so the evidence is always saved.

How can I make a long suite faster?

Split it with the shard option across several jobs, cache dependencies and keep workers matched to the size of the runner.

Where do I keep passwords for tests?

Store them as repository or environment secrets and read them through the secrets context, never inside the workflow or code.

Sources and further reading

- Playwright documentation
- Playwright continuous integration
- GitHub Actions documentation

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550