YAML для Playwright у GitHub Actions

Практичний посібник з налаштування CI/CD для E2E-тестування за допомогою Playwright у GitHub Actions — від базового workflow до Docker-контейнерів та публікації HTML-звітів.

DEVOPS E2E TESTING GITHUB ACTIONS

Вступ

Що таке Playwright?

Playwright — сучасний фреймворк для End-to-End тестування від Microsoft, що підтримує кілька браузерів (Chromium, Firefox, WebKit) і мов програмування (JavaScript, TypeScript, Python, .NET, Java). Він забезпечує надійне, швидке та стабільне тестування веб-застосунків у реальних браузерних середовищах.

Playwright активно розвивається і є де-факто стандартом для E2E-тестування у сучасних CI/CD-пайплайнах завдяки вбудованій підтримці паралельного запуску, трейсингу, відео та скриншотів при падінні тестів.

Що охоплює цей посібник

01

Базовий workflow

Налаштування кроків checkout, setup-node, install, run у GitHub Actions

02

Запуск у Docker

Використання офіційного образу Microsoft з браузерами Playwright

03

HTML-звіт

Збереження та публікація інтерактивного звіту як артефакту GitHub

Усі приклади YAML-конфігурацій протестовані на ubuntu-latest runner та сумісні з актуальними версіями GitHub Actions та Playwright.

Базовий workflow: залежності, тести, звіт

Найпростіший і найпоширеніший сценарій — запускати тести при кожному **push** або **pull_request** на основних гілках. Це гарантує, що жоден код не потрапить у main без проходження E2E-тестів.

Конфігурація workflow

```
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@v3
        with:
          node-version: 18
      - 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@v3
        if: always()
        with:
          name: playwright-report
          path: playwright-report/
          retention-days: 30
```

Ключові кроки пояснення

actions/checkout@v4

Клонує репозиторій на runner для доступу до коду та тестів

npm ci

Чиста установка залежностей строго за `package-lock.json` — детермінована та швидка

--with-deps

Встановлює браузери разом з усіма системними залежностями ОС

if: always()

Артефакт завантажується як при успіху, так і при падінні тестів — критично для дебагінгу

Запуск у Docker-контейнері

Використання офіційного Docker-образу від Microsoft **mcr.microsoft.com/playwright** — найелегантніший спосіб гарантувати повністю ізольоване та відтворюване середовище виконання тестів. Образ містить усі необхідні браузери та їхні залежності, що усуває потребу у кроці `playwright install --with-deps`.

Docker workflow конфігурація

```
name: Playwright Tests
on:
  push:
    branches: [main, master]
  pull_request:
    branches: [main, master]
jobs:
  playwright:
    name: Playwright Tests
    runs-on: ubuntu-latest
    container:
      image: mcr.microsoft.com/playwright:v1.39.0-jammy
    steps:
    - uses: actions/checkout@v4
    - uses: actions/setup-node@v3
      with:
        node-version: 18
    - name: Install dependencies
      run: npm ci
    - name: Run your tests
      run: npx playwright test
```

Переваги Docker-підходу

Ізольоване середовище

Однакові умови на будь-якому runner — немає «у мене працює»

Швидший старт

Браузери вже встановлені в образі — менше кроків налаштування

Версійність

Тег образу фіксує точну версію Playwright і браузерів

⚠️ **Важливо:** Версію образу (наприклад, `v1.39.0-jammy`) необхідно узгоджувати з версією Playwright, вказаною у `package.json` проекту. Розбіжність версій може призвести до непередбачуваної поведінки тестів.

HTML-звіт: обмеження та варіанти

Playwright генерує детальний інтерактивний HTML-звіт із результатами тестів, скриншотами, відео та трейсами. Проте є важлива особливість: GitHub Actions **не підтримує відображення інтерактивного HTML-вмісту** безпосередньо в інтерфейсі — звіт доступний лише як архів для завантаження.

🗂️ Варіант 1: Артефакт GitHub

Зберігати HTML-звіт як артефакт workflow. Команда завантажує ZIP-архів локально та відкриває у браузері. Найпростіший варіант без додаткової інфраструктури. Обмеження: термін зберігання артефактів (за замовчуванням 30 днів), незручно для великих команд.

🌐 Варіант 2: Сторонній хостинг

Розгортати звіт на статичному хостингу — GitHub Pages, Netlify, AWS S3 або внутрішній сервер. Звіт доступний за постійним посиланням, яке можна додати до PR-коментаря. Зручно для великих команд і аудиту якості. Потребує налаштування deployment-кроків.

30

Днів зберігання

Стандартний retention period для артефактів у GitHub Actions

3

Браузери у звіті

Chromium, Firefox та WebKit — результати по кожному окремо

100%

Покриття звіту

Усі тести — успішні, провалені, пропущені — присутні у HTML-звіті

Завантаження та збереження артефакту звіту

GitHub Actions надає два вбудовані action для роботи з артефактами: **upload-artifact** для збереження файлів після виконання job та **download-artifact** для отримання їх в іншому job або workflow. Ця пара особливо корисна у мульти-job пайплайнах, де один job запускає тести, а інший — публікує звіт.

📤 Збереження звіту

```
- name: Save report
  uses: actions/upload-artifact@v3
  with:
    name: playwright-html-report
    path: playwright-report/
    # або path: html/report —
    # залежно від конфігурації
    # reporter у проекті
  if: failure() || success()
```

Параметр `path` залежить від налаштування репортера у `playwright.config.ts`. Перевірте значення `outputFolder` у конфігурації проекту.

📥 Завантаження в іншому job

```
- name: Get report
  uses: actions/download-artifact@v3
  with:
    name: playwright-html-report
  if: failure() || success()
```

Цей крок використовується у **окремому job** (після завершення тестового job) для подальшої обробки: публікації на хостинг, надсилання сповіщень або генерації зведеного звіту у PR-коментарі.

if: failure() || success() — виконувати крок незалежно від результату тестів. Еквівалентно до `if: always()`, але більш явна форма запису.

→ **Job 1: test**

Запускає Playwright тести → генерує HTML-звіт → зберігає через `upload-artifact`

→ **Job 2: publish (опційно)**

Отримує артефакт через `download-artifact` → деплоїть на GitHub Pages або внутрішній сервер

→ **Команда переглядає**

Завантажує ZIP з вкладки Actions → розпаковує → відкриває `index.html` локально

Висновок

Налаштування Playwright у GitHub Actions — це прямолінійний процес, який значно підвищує якість коду завдяки автоматизованому E2E-тестуванню на кожен PR та push. Правильна конфігурація гарантує стабільне середовище та зручний доступ до результатів тестів для всієї команди.

1

Базовий сценарій

`checkout` → `setup-node` → `npm ci` → `playwright install --with-deps` → `playwright test` → `upload-artifact` для звіту

2

Docker-підхід

Образ `mcr.microsoft.com/playwright` спрощує середовище. Версію образу **обов'язково** узгоджувати з версією Playwright у `package.json`

3

HTML-звіт

Зберігати як артефакт для локального перегляду або публікувати на GitHub Pages / окремий хостинг для зручного командного доступу

> Автоматизований CI/CD для E2E-тестів — це не розкіш, а необхідність. Playwright у поєднанні з GitHub Actions дає команді впевненість у якості кожного деплою.

ГОТОВО ДО ПРОДАКШНУ ДОКУМЕНТАЦІЯ PLAYWRIGHT