



YAML для Cypress у GitHub Actions

Практичний посібник для розробників та DevOps-інженерів, які налаштовують End-to-End тестування у GitHub Actions. Від базового запуску до паралелізації та Cypress Cloud — все, що потрібно знати для ефективного CI.

⌚ E2E ТЕСТУВАННЯ

⌚ GITHUB ACTIONS

✖️ CYPRESS

Вступ

Cypress — один з найпопулярніших фреймворків для End-to-End тестування у браузері. Він дозволяє писати тести, що імітують реальну поведінку користувача, перевіряючи весь стек застосунку від UI до бекенду. Завдяки зручному API та чудовому інструменту для відлагодження, Cypress став стандартом де-факто для багатьох команд.

Офіційна action [cypress-io/github-action](#) значно спрощує інтеграцію Cypress з GitHub Actions. Вона бере на себе інсталяцію залежностей, кешування та запуск тестів, дозволяючи зосередитися на конфігурації, а не на інфраструктурних деталях. Це рішення підходить як для невеликих проектів, так і для масштабних систем із паралельними запусками.



Базовий запуск

Мінімальна конфігурація для першого запуску тестів у CI без зайвих налаштувань.



Браузери та Docker

Запуск у Chrome, Firefox, Edge та ізольованих Docker-контейнерах.



Cypress Cloud

Запис результатів, артефакти, аналітика та паралелізація через хмарний сервіс.



Паралелізація

Розподіл тестів між кількома контейнерами для скорочення часу прогону.

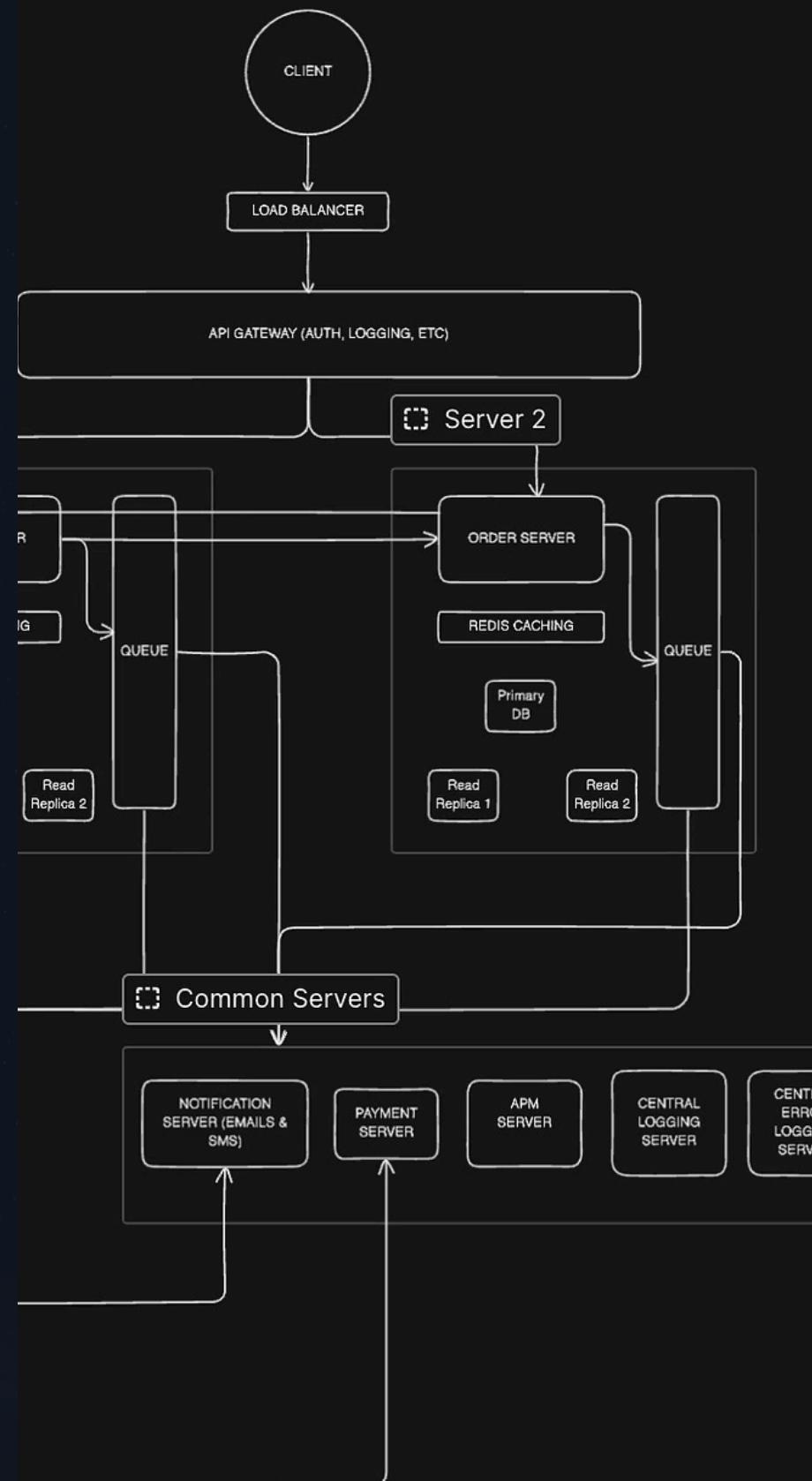
Простий workflow: усі тести при push

Мінімальний варіант workflow — це найкраща відправна точка для першого запуску та перевірки базових налаштувань. Він містить лише два кроки: checkout коду та запуск Cypress. Без жодних додаткових параметрів дія автоматично встановлює залежності та запускає всі знайдені тести.

- Без параметра **browser** тести виконуються в **Electron** — вбудованому браузері Cypress, що не потребує окремої інсталяції.

```
name: End-to-end tests
on: push
jobs:
  cypress-run:
    runs-on: ubuntu-22.04
    steps:
      - name: Checkout
        uses: actions/checkout@v4
      - name: Cypress run
        uses: cypress-io/github-action@v6
```

Цей workflow автоматично запускатиметься при кожному **push** у будь-яку гілку. Для production-проектів рекомендується обмежити тригер конкретними гілками або подіями, такими як `pull_request`, щоб не витрачати ресурси CI на кожну дрібну зміну.



Запуск у різних браузерах: Chrome

Workflow для Chrome

```
name: Chrome
on: push
jobs:
  chrome:
    runs-on: ubuntu-22.04
    name: E2E on Chrome
    steps:
      - uses: actions/checkout@v4
      - uses: cypress-io/github-action@v6
        with:
          browser: chrome
```

Чому Chrome?

Chrome є найпоширенішим браузером серед користувачів, тому тестування саме в ньому забезпечує найбільше покриття реальних сценаріїв. GitHub Actions надає Chrome попередньо встановленим на образах ubuntu-22.04.

При використанні параметра `browser: chrome` [cypress-io/github-action](#) автоматично знаходить встановлену версію Chrome та передає її Cypress. Не потрібно вказувати повний шлях або версію вручну.

- Встановлений на ubuntu-22.04 за замовчуванням
- Підтримує headless та headed режими
- Найкраще покриття реальних сценаріїв

Запуск у різних браузерах: Firefox та Edge

Кросбраузерне тестування дозволяє виявити проблеми, специфічні для конкретного рушія. GitHub Actions підтримує Firefox та Edge поряд з Chrome, що робить налаштування мультибраузерного CI відносно простим завданням.



```
- uses: cypress-io/github-action@v6
  with:
    browser: firefox
```

Використовує рушій Gecko.
Важливо для виявлення
відмінностей у рендерингу та
поведінці CSS.



```
- uses: cypress-io/github-action@v6
  with:
    browser: edge
```

Базується на Chromium, але має
власні особливості. Актуально
для корпоративних середовищ.



```
- uses: cypress-io/github-action@v6
# browser не вказано
```

Найшвидший варіант.
Вбудований браузер Cypress, не
потребує інсталяції.

- ☐ Якщо параметр **browser** не вказано — використовується **Electron**. Це найшвидший режим, але він не відображає поведінку реальних браузерів користувача.

Запуск у режимі headed (з вікном браузера)

Headless (за замовчуванням)

У CI Cypress за замовчуванням працює в режимі **headless** — без графічного інтерфейсу. Це швидше, споживає менше ресурсів і ідеально підходить для автоматизованого тестування в продуктивному середовищі.

- Швидший запуск
- Менше споживання RAM та CPU
- Стандарт для CI/CD
- Не потребує дисплея або Xvfb

Headed (для відлагодження)

Режим **headed** відкриває реальне вікно браузера, що корисно при відлагодженні складних тестів або візуальних проблем.

```
- uses: cypress-io/github-action@v6
  with:
    browser: chrome
    headed: true
```

Зверніть увагу: для headed режиму в Linux-середовищі може знадобитися віртуальний дисплей (Xvfb), проте офіційна action обробляє це автоматично.

Запуск у Docker-контейнері

Використання офіційного Docker-образу Cypress cypress/browsers:latest гарантує повністю відтворюване середовище зі встановленими браузерами. Це усуває проблему «у мене працює, у CI — ні», оскільки і розробник, і CI використовують ідентичне оточення.

```
name: Test in Docker
on: push
jobs:
  cypress-run:
    runs-on: ubuntu-22.04
    container:
      image: cypress/browsers:latest
      options: --user 1001
    steps:
      - uses: actions/checkout@v4
      - uses: cypress-io/github-action@v6
        with:
          browser: chrome
```

Ізоляція середовища

Контейнер не залежить від ПЗ, встановленого на runners GitHub.

Відтворюваність

Однакова версія браузера та Node.js кожного разу.

--user 1001

Уникнення проблем з правами доступу у контейнері GitHub Actions.

Передача змінних середовища (env)

Параметр `env` дії `cypress-io/github-action` дозволяє передавати змінні середовища безпосередньо до Cypress. Це зручно для конфігурування тестового оточення: вказівки хостів API, портів, feature flags та інших налаштувань, які можуть відрізнятися між середовищами (dev, staging, production).

Передача через параметр action

```
- name: Cypress run with env  
uses: cypress-io/github-action@v6  
with:  
  env: host=api.dev.local,port=4222
```

Список параметрів передається **в один рядок через кому**. Не додавайте пробіли між парами ключ=значення — це може привести до некоректного парсингу.

Доступ у тестах

У коді тестів змінні доступні через:

```
Cypress.env('host')  
Cypress.env('port')
```

Або в `cypress.config.js` через секцію `env`.

- **Важливо:** Параметр `env` у `action` — це NOT те саме, що секція `env` на рівні `workflow` або `job`. Перший передає значення у Cypress, другий — в shell-середовище `runner`.

Глобальні env у workflow

Окрім передачі змінних безпосередньо до Cypress через параметр action, можна визначати змінні середовища на рівні всього workflow або окремого job. Це зручно для значень, які використовуються у кількох кроках, або для конфігурування поведінки runner.

Приклад глобального env

```
env:  
  DAY_OF_WEEK: Monday  
  ENV_VAR: url  
  ENV_VAR_2: "3000"
```

Змінні, оголошені на рівні job, доступні у всіх кроках цього job.
Оголошенні на рівні workflow — у всіх jobs.

⚠️ Безпека секретів

Ніколи не зберігайте паролі, API-ключи або токени у відкритому вигляді у файлах workflow.
Використовуйте **GitHub Secrets**:

```
env:  
  API_KEY: ${{ secrets.API_KEY }}
```

- Secrets маскуються в логах
- Недоступні з форків (за замовчуванням)
- Шифруються при зберіганні

Запуск конкретних spec-файлів (spec)

Параметр **spec** дозволяє гнучко вибирати, які тести запускати в конкретному workflow. Це особливо корисно для розбиття тестів на групи: smoke-тести, тести окремих фіч або регресійний набір. Можна вказати конкретний файл, кілька файлів або glob-патерн.

1

Один файл

```
with:  
spec: cypress/e2e/spec1.cy.js
```

2

Кілька файлів

```
with:  
spec: |  
cypress/e2e/spec-a.cy.js  
cypress/e2e/spec-b.cy.js
```

3

Glob-патерн

```
with:  
spec: |  
cypress/e2e/spec-a.cy.js  
cypress/**/*-b.cy.js
```

Використання glob-патернів дозволяє динамічно вибирати файли, не перелічуючи кожен вручну. Наприклад, `cypress/e2e/smoke/**/*.cy.js` запустить усі тести у папці `smoke`. При використанні багаторядкового синтаксису YAML (символ `|`) кожен файл або патерн вказується з нового рядка.

Кастомна команда (command)

Конфігурація через command

```
- name: Custom tests
  uses: cypress-io/github-action@v6
  with:
    command: npm run e2e:ci
```

Де у package.json визначено:

```
"scripts": {
  "e2e:ci": "cypress run
    --browser chrome
    --spec cypress/e2e/smoke/**"
}
```

Коли використовувати?

Параметр **command** замінює стандартний виклик Cypress власним скриптом. Це корисно, коли:

- Команда запуску вже визначена в package.json
- Потрібно передати специфічні пропори Cypress CLI
- Використовується кастомний wrapper навколо Cypress
- Команда включає pre/post хуки

Увага: При використанні command параметри action browser, spec, env, record та інші — **ігноруються**. Всі налаштування передаються через саму команду.

Завантаження результатів у Cypress Cloud

Cypress Cloud (раніше Cypress Dashboard) — хмарний сервіс для збереження результатів тестів, відео, скріншотів та аналітики. Він надає зручний інтерфейс для перегляду падань, порівняння результатів між запусками та виявлення flaky-тестів.

```
- name: Cypress run
  uses: cypress-io/github-action@v6
  with:
    record: true
  env:
    CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
    GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

record: true

Вмикає запис результатів тестів до Cypress Cloud. Без цього параметра результати зберігаються лише локально на runner.

CYPRESS_RECORD_KEY

Унікальний ключ вашого проєкту у Cypress Cloud. Ідентифікує, до якого проєкту завантажувати результати.

GITHUB_TOKEN

Токен для коректної ідентифікації збірки та інтеграції з GitHub (статуси PR, коментарі).

Отримання CYPRESS_RECORD_KEY та GITHUB_TOKEN

Перед налаштуванням запису результатів необхідно отримати відповідні ключі доступу. Cypress Record Key прив'язаний до конкретного проєкту в Cypress Cloud, а GITHUB_TOKEN використовується для інтеграції зі статусами PR та коментарями.

1 Реєстрація у Cypress Cloud

Зайдіть на cloud.cypress.io та створіть акаунт або увійдіть через GitHub. Безкоштовний план дозволяє записувати обмежену кількість тестів на місяць.

2 Створення проєкту та генерація Record Key

У Cypress Cloud створіть новий проєкт. Після створення перейдіть до **Project Settings** → **Record Key**. Скопіюйте згенерований ключ та збережіть у безпечному місці.

3 GITHUB_TOKEN або Personal Access Token

GitHub автоматично надає secrets.GITHUB_TOKEN для кожного workflow. Якщо потрібен розширеній доступ — створіть [Personal Access Token](#) у налаштуваннях GitHub акаунта з необхідними дозволами.

Додавання секретів у репозиторій

Після отримання ключів їх необхідно безпечно зберегти як GitHub Secrets. Тільки після цього вони будуть доступні у workflow через синтаксис ``${{ secrets.NAME }}``. Процес займає кілька хвилин і виконується через веб-інтерфейс GitHub.



Після додавання секрети стають доступними у всіх workflow репозиторію. Значення секретів **ніколи не відображаються** у відкритому вигляді — ні в логах, ні в інтерфейсі. GitHub автоматично маскує їх у виводі. Для організацій можна налаштовувати секрети на рівні організації або середовища (environment), що спрощує управління при роботі з кількома репозиторіями.

Артефакти: скріншоти та відео на GitHub

Cypress автоматично робить скріншоти при падінні тестів та записує відео всіх прогонів. Без збереження як артефакти ці файли втрачаються після завершення runner. Крок actions/upload-artifact дозволяє зберегти їх і завантажити для аналізу прямо з інтерфейсу GitHub Actions.

```
- uses: cypress-io/github-action@v6

- uses: actions/upload-artifact@v3
  with:
    name: cypress-screenshots
    path: cypress/screenshots
    if-no-files-found: ignore

- uses: actions/upload-artifact@v3
  with:
    name: cypress-videos
    path: cypress/videos
    if-no-files-found: ignore
```

Скріншоти

Автоматично створюються при падінні тесту. Зберігаються у `cypress/screenshots`. Для завантаження лише при помилці — додайте `if: failure()`.

Відео

Записуються для кожного spec-файлу. Зберігаються у `cypress/videos`. Дозволяють переглянути точну послідовність дій перед падінням.

if-no-files-found

Параметр `ignore` запобігає помилці кроку, якщо файлів немає (наприклад, всі тести пройшли і скріншоти не створювались).

- Для економії місця та прискорення CI рекомендується завантажувати скріншоти лише при падінні: `if: failure()` на кроці `upload`.

Паралельний запуск тестів (matrix + Cypress Cloud)

Паралелізація — один з найпотужніших інструментів для скорочення часу тестування. Замість послідовного виконання всіх тестів на одному runner, вони розподіляються між кількома контейнерами, які працюють одночасно. Cypress Cloud автоматично балансує навантаження, розподіляючи spec-файли між доступними контейнерами.

```
strategy:  
fail-fast: false  
matrix:  
  containers: [1, 2, 3]  
  
steps:  
  - uses: actions/checkout@v4  
  - uses: cypress-io/github-action@v6  
  with:  
    record: true  
    parallel: true  
    group: 'Actions example'  
env:  
  CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}  
  GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

3x

Паралельних потокі

Кількість елементів у containers визначає кількість паралельних runners.

~66%

Скорочення часу

Теоретичне прискорення при 3 паралельних контейнерах (залежить від розподілу тестів).

Параметр **fail-fast: false** гарантує, що при падінні одного контейнера інші не зупиняються. Це важливо для отримання повної картини стану тестів. Cypress Cloud інтелектуально розподіляє тести на основі попередньої статистики виконання, забезпечуючи рівномірне завантаження контейнерів.

Унікальний ci-build-id при re-run

При повторному запуску (re-run) того ж workflow GitHub Actions Cypress Cloud може відхилити запуск з повідомленням «Збірка вже завершена», оскільки отримує той самий ідентифікатор збірки. Рішення — генерувати унікальний **ci-build-id** для кожного запуску за допомогою окремого підготовчого job.

```
jobs:  
  prepare:  
    runs-on: ubuntu-22.04  
    outputs:  
      uuid: ${{ steps.uuid.outputs.value }}  
  steps:  
    - name: Generate unique ID  
      id: uuid  
      run: echo "value=sha-$GITHUB_SHA-time-$(date +%s)" >> $GITHUB_OUTPUT  
  
  smoke-tests:  
    needs: [prepare]  
    strategy:  
      fail-fast: false  
    matrix:  
      containers: [1, 2, 3]  
    steps:  
      - uses: actions/checkout@v4  
      - uses: cypress-io/github-action@v6  
        with:  
          record: true  
          parallel: true  
          ci-build-id: ${{ needs.prepare.outputs.uuid }}  
    env:  
      CYPRESS_RECORD_KEY: ${{ secrets.CYPRESS_RECORD_KEY }}
```

Як формується uuid?

Комбінація GITHUB_SHA (хеш коміту) та Unix timestamp \$(date +%s) гарантує унікальність навіть при кількох re-run одного коміту.

Передача між jobs

Значення передається через механізм outputs: job **prepare** генерує uuid та публікує його, job **smoke-tests** отримує через needs.prepare.outputs.uuid.

needs: [prepare]

Залежність гарантує, що uuid буде згенеровано до початку паралельних тестових jobs. Усі контейнери матриці отримують одинаковий ci-build-id.

Висновок

Cypress у GitHub Actions — потужна та гнучка комбінація для налаштування надійного CI/CD для E2E-тестів. Починаючи з мінімального двокрокового workflow та поступово додаючи складність, можна побудувати повноцінну систему тестування, яка закріє всі потреби від невеликого проєкту до enterprise-рішення.

01

Базовий запуск

Checkout + cypress-io/github-action з потрібним **browser** і **spec**. Мінімум налаштувань для першого результату.

02

Гнучкість конфігурації

Docker для ізоляції, **env** для параметризації, **spec** для вибіркових запусків, **command** для кастомних сценаріїв.

03

Артефакти для діагностики

Скріншоти та відео через `upload-artifact` — незамінний інструмент для розбору падань без доступу до runner.

04

Cypress Cloud + паралелізація

record: true, matrix-стратегія та унікальний **ci-build-id** при `re-run` — для швидкого та надійного CI у масштабі.

- ❑  **Рекомендований шлях:** Почніть з базового workflow → додайте потрібний браузер → налаштуйте артефакти → підключіть Cypress Cloud для аналітики → увімкніть паралелізацію при зростанні тестової бази.