

# From Code to Production: A Secure, AI-Assisted CI/CD Workflow with GitHub Actions and OpenAI

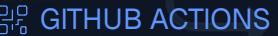
## GitHub Actions: вступ



Повний посібник для розробників та DevOps-інженерів, які хочуть автоматизувати CI/CD безпосередньо у своєму GitHub-репозиторії — без зовнішніх інструментів і зовнішніх сервісів.



CI/CD



GITHUB ACTIONS



АВТОМАТИЗАЦІЯ



Secret Scanning



ESLint & Tests



Dependabot



Vercel



Commit Validation

Changelog Generation



## OpenAI

Changelog Generation



# Вступ

GitHub Actions — це потужна платформа CI/CD, інтегрована безпосередньо у ваш GitHub-репозиторій. Вона дозволяє автоматизувати всі ключові процеси розробки — від тестування до розгортання — без необхідності підключати зовнішні сервіси чи налаштовувати окрему інфраструктуру.

Замість того, щоб платити за Jenkins, CircleCI або інші рішення та підтримувати їх окремо, ви описуєте логіку автоматизації прямо в репозиторії у вигляді YAML-файлів. Це особливо зручно для команд, які вже використовують GitHub як основну платформу для розробки.



## Без зовнішніх сервісів

CI/CD прямо в репозиторії GitHub — все в одному місці.



## E2E-тестування

Cypress, Playwright та інші фреймворки запускаються через YAML workflow.



## Простий старт

Достатньо створити один файл workflow у репозиторії, щоб почати автоматизацію.

Для запуску E2E-тестів, таких як Cypress або Playwright, потрібно лише описати відповідний workflow у файлі YAML та розмістити його у директорії `.github/workflows/`. GitHub подбає про все інше — виділить віртуальну машину, встановить залежності та виконає ваші тести.

# Що таке GitHub Actions

GitHub Actions — це повноцінна система Continuous Integration та Continuous Deployment (CI/CD), вбудована у платформу GitHub. Вона дозволяє автоматично виконувати будь-які сценарії — тести, збірку, деплой, нотифікації — у відповідь на різні події у вашому репозиторії.



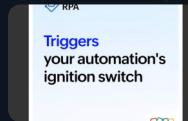
## Віртуальні машини

GitHub надає готові runner-машини на базі **Linux, Windows і macOS**. Ви не витрачаєте час на налаштування середовища — воно вже готове до роботи.



## Self-hosted Runners

Якщо стандартних можливостей недостатньо, можна розгорнути **власні runners** у вашому ЦОД або хмарі для повного контролю над середовищем.



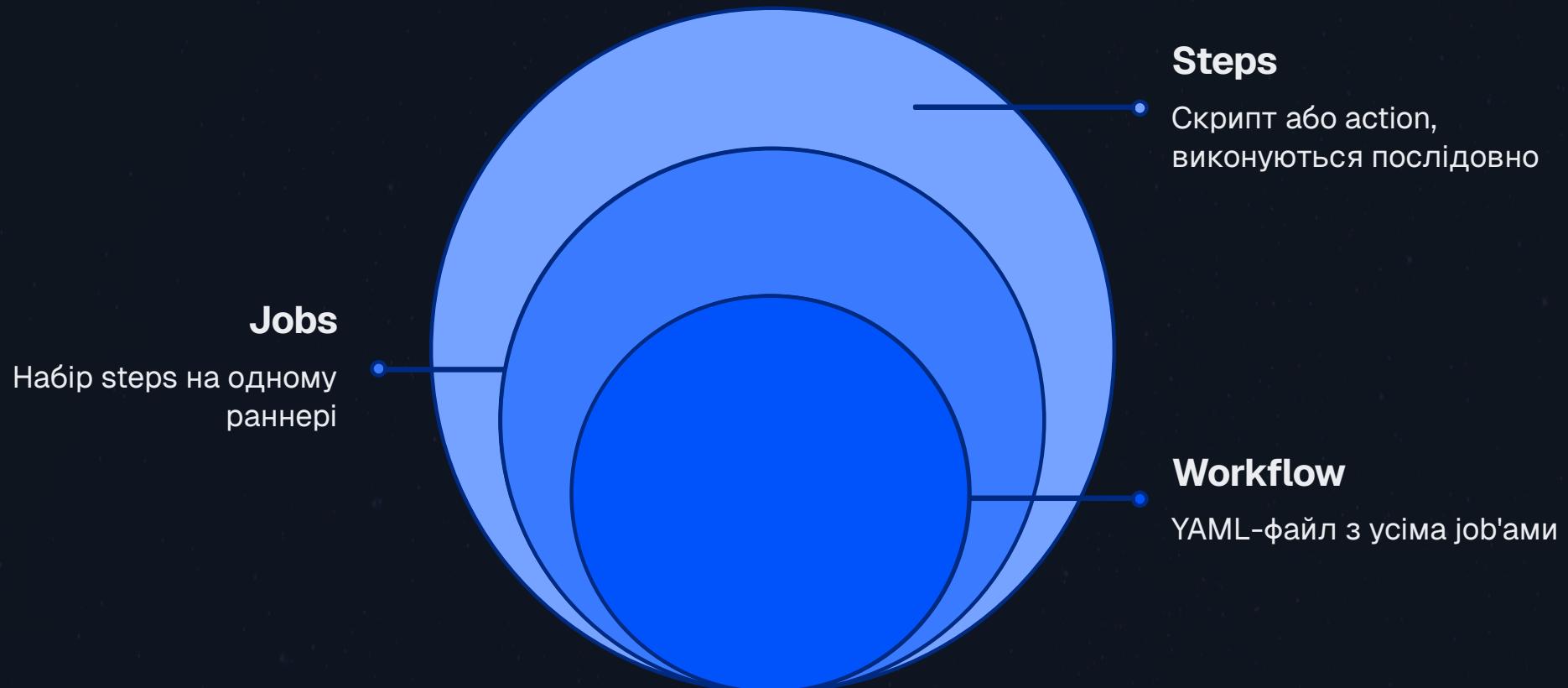
## Гнучкі тригери

Workflow запускається при подіях: **push, pull request, issue, release** — або вручну через UI, за розкладом (cron) чи через REST API.

Ключова перевага GitHub Actions над альтернативами — глибока інтеграція з екосистемою GitHub. Ви можете реагувати на будь-яку подію в репозиторії, мати доступ до контексту PR, issues, releases та інших сутностей прямо у своїх workflow.

# Як влаштований workflow

Архітектура GitHub Actions базується на чіткій ієрархії сущностей. Розуміння цієї структури — ключ до написання ефективних та масштабованих workflow. Кожен рівень ієрархії має свою роль і відповідальність.



Важливо розуміти, що **jobs** за замовчуванням виконуються паралельно, що дозволяє значно прискорити CI-пайплайн. Наприклад, можна паралельно запустити юніт-тести, лінтер і збірку Docker-образу. Якщо ж необхідно задати послідовність, використовується ключове слово `needs`.

## Jobs та паралельність

Jobs виконуються паралельно за замовчуванням. Використовуйте `needs` для задання залежностей між jobs.

- Паралельна збірка для різних архітектур
- Послідовний деплої після успішних тестів
- Матрична стратегія для кількох версій Node.js

## Steps та артефакти

Кожен step має доступ до результатів попередніх кроків. Дані між steps та jobs передаються через:

- Артефакти (файли, звіти, збірки)
- Output-змінні між steps
- Кешування залежностей
- Environment variables

# Тарифікація та ліміти

Одна з перших речей, яку варто з'ясувати перед тим як активно використовувати GitHub Actions у production-проєктах — це модель тарифікації. GitHub пропонує щедрі безкоштовні ліміти, але для активно розроблюваних приватних репозиторіїв варто розуміти обмеження вашого плану.

## Публічні репозиторії — безкоштовно

Якщо ваш репозиторій є публічним, ви отримуєте **необмежений доступ** до стандартних GitHub-hosted runners без жодних витрат. Це ідеально для open-source проєктів.

## Приватні репозиторії — обмежені ліміти

Для приватних репо кількість безкоштовних хвилин залежить від вашого плану (Free, Pro, Team, Enterprise). Після вичерпання ліміту кожна додаткова хвилина тарифікується окремо.

## Контроль витрат

У налаштуваннях організації або акаунту можна встановити **ліміт витрат (spending limit)**, щоб уникнути несподіваних рахунків. За замовчуванням він дорівнює \$0 — тобто понад безкоштовний ліміт нічого не нараховується.

2000

хвилин/місяць

Безкоштовно для GitHub Free  
(приватні репо)

3000

хвилин/місяць

Для плану GitHub Pro / Team

50000

хвилин/місяць

Для GitHub Enterprise Cloud



Деталі тарифів і типів runners: [Офіційна документація GitHub Billing](#)

# Основні поняття: Workflows

Workflow — це центральна сутність GitHub Actions. Він описує весь автоматизований процес: які кроки виконувати, за яких умов запускатися і як організувати роботу між різними jobs. Кожен workflow зберігається як окремий YAML-файл у директорії `.github/workflows/` вашого репозиторію.

## Де зберігається

Файли workflow розміщаються у директорії:

```
.github/  
  workflows/  
    test.yml  
    deploy.yml  
    release.yml
```

Один репозиторій може містити **довільну кількість** workflow-файлів.

## Способи запуску

- **Подія в репо** — push, PR, issue, release
- **workflow\_dispatch** — ручний запуск через UI або API
- **schedule** — за cron-розкладом
- **repository\_dispatch** — зовнішній виклик через REST API

## Приклади workflow у репо

- Запуск тестів при кожному PR
- Деплой при push у main-гілку
- Публікація release при створенні тегу

Важлива перевага підходу «workflow як код» (Workflow as Code) — версіонування. Зміни у логіці CI/CD можна переглядати через git history, робити code review через PR та відкочуватися до попередніх версій, якщо щось пішло не так. Це суттєво підвищує прозорість і надійність процесу автоматизації.

# Основні поняття: Events

Events — це серце системи тригерів GitHub Actions. Саме вони визначають, коли і за яких умов буде запущено ваш workflow. Гнучка система фільтрації дозволяє запускати автоматизацію лише тоді, коли це дійсно необхідно, уникуючи зайвих витрат ресурсів.



## **push / pull\_request**

Найпоширеніші тригери. Запускають workflow при кожному коміті або відкритті/новленні PR. Можна обмежити конкретними гілками через branches.



## **release / issues**

Запуск при створенні release або відкритті issue. Ідеально для автоматичного тегування, публікації changelog або нотифікацій команди.



## **schedule (cron)**

Регулярний запуск за розкладом у форматі cron. Корисно для нічних регресійних тестів або щотижневих звітів про стан репозиторію.



## **workflow\_dispatch**

Ручний запуск через кнопку в UI або REST API. Підтримує вхідні параметри — зручно для деплою з вибором середовища.

Для тонкого налаштування тригерів використовуйте фільтри: branches та branches-ignore для обмеження за гілками, paths та paths-ignore для фільтрації за шляхами файлів. Наприклад, workflow для тестування frontend може запускатися лише тоді, коли змінюються файли у директорії src/, ігноруючи зміни у документації або конфігурації.

# Основні поняття: Jobs

Job — це логічна одиниця виконання у GitHub Actions. Кожен job запускається на окремому runner і складається з послідовних кроків (steps). Розуміння того, як організувати jobs, — ключ до побудови ефективних і швидких пайплайнів.



## Паралельне виконання

За замовчуванням усі jobs у workflow виконуються **одночасно**, що суттєво скорочує загальний час пайплайну.

- Лінтер + тести + збірка — паралельно
- Матрична стратегія для кількох ОС
- Незалежні перевірки безпеки

## Залежності через needs

Ключове слово `needs` дозволяє задати послідовність виконання jobs, коли одна job залежить від результату іншої.

- Деплой тільки після успішних тестів
- Пакування після паралельної збірки
- Нотифікація після завершення деплою

Між steps одного job можна передавати дані через output-змінні та артефакти. Це дозволяє, наприклад, зберегти зібраний артефакт у одному step і використати його у наступному — для завантаження в хмару, публікації в реєстрі або архівування звітів про тестування.

# Основні поняття: Actions

Action — це перевикористовуваний компонент для GitHub Actions. По суті, це готовий скрипт або набір кроків, який можна підключити до свого workflow одним рядком. Actions вирішують проблему дублювання коду і дозволяють командам ділитися корисними автоматизаціями.

Уявіть, що кожного разу вам потрібно писати однакову послідовність команд для checkout репозиторію, встановлення Node.js та кешування залежностей. Actions вирішують цю проблему — замість десятків рядків скриptів ви пишете один рядок `uses: actions/setup-node@v4`.

1

## GitHub Marketplace

Тисячі готових actions від GitHub та спільноти: checkout, setup-node, docker/build-push, AWS credentials, Slack нотифікації та багато іншого.

2

## Власні Actions

Пишіть власні actions для специфічної логіки вашої команди. Підтримуються JavaScript/TypeScript actions та Docker container actions.

3

## Типові сценарії

Підготовка середовища (checkout, setup-\*), автентифікація у хмарних провайдерах, запуск E2E-тестів (Cypress, Playwright), публікація артефактів.



💡 Завжди вказуйте конкретну версію action через тег або SHA-хеш — наприклад, `uses: actions/checkout@v4` — щоб уникнути несподіваних змін у поведінці при оновленні action.

# Основні поняття: Runners

Runner — це сервер, на якому фізично виконується ваш workflow. Розуміння типів runners допомагає вибрати оптимальне рішення для ваших задач — з точки зору продуктивності, безпеки та вартості.

## GitHub-hosted Runners

Готові VM від GitHub: Ubuntu, Windows, macOS. Кожен запуск отримує **чисту нову VM**. Ідеально для більшості проектів — нічого не потрібно налаштовувати.



## Larger Runners

Розширені конфігурації з більшим CPU та RAM для ресурсоємних задач — великі збірки, важкі тести. Доступні за **додаткову плату** для Team та Enterprise.



## Self-hosted Runners

Ваші власні машини або контейнери. Повний контроль над середовищем, спеціальне обладнання (GPU, ARM), доступ до приватної мережі та **економія на хвилинах**.

Важлива деталь: один runner виконує **лише одну job одночасно**. Якщо ваш workflow має кілька паралельних jobs, GitHub автоматично виділяє окремий runner для кожної з них. Для GitHub-hosted runners це відбувається прозоро — ви просто платите за хвилини. Для self-hosted runners потрібно забезпечити достатню кількість runner-агентів.

## Коли обирати GitHub-hosted

- Стандартні проєкти без специфічних вимог
- Невелика команда або open-source
- Немає часу на адміністрування
- Потрібна ізоляція між запусками

## Коли обирати Self-hosted

- Доступ до приватної мережі або VPN
- Специфічне обладнання (GPU, ARM)
- Великий обсяг CI-хвилин (економія)
- Особливі вимоги до безпеки та compliance

# Висновок

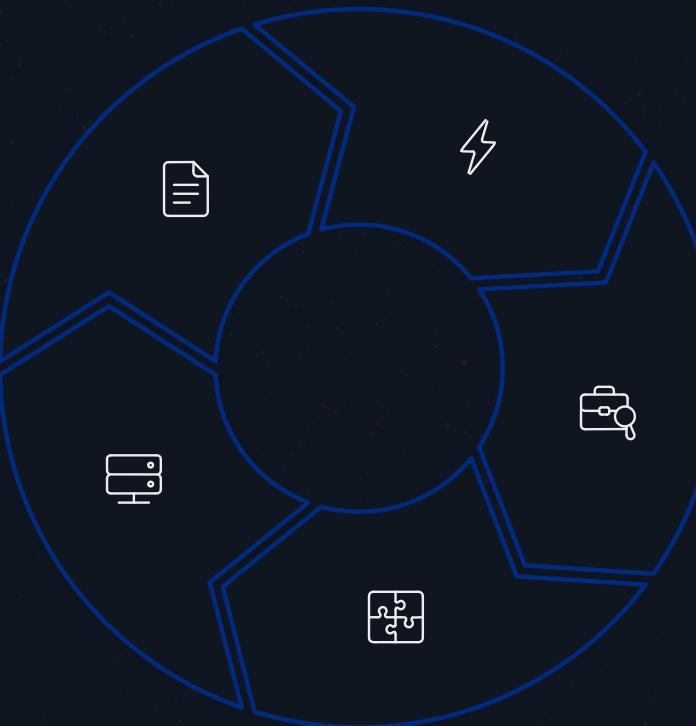
GitHub Actions — це сучасна, потужна і добре інтегрована платформа CI/CD, яка дозволяє командам автоматизувати весь цикл розробки прямо у своєму репозиторії. Відсутність необхідності у зовнішніх інструментах, багата екосистема готових actions та гнучка система тригерів роблять її оптимальним вибором для більшості проектів.

## Workflow

YAML-файл у репозиторії, що описує весь автоматизований процес

## Runners

VM або власні машини, де фізично виконується кожна job



## Events

Тригери: push, PR, schedule, manual — запускають workflow

## Jobs

Набори steps на runner; паралельні або послідовні через needs

## Actions

Перевикористовувані скрипти — з Marketplace або власні

Ключові сутності: **workflow → jobs → steps**; тригери — **events**; виконання на **runners**; логіку виносимо в **actions**.

Наступний крок — заглибитися у YAML-синтаксис GitHub Actions та розглянути практичні приклади для запуску E2E-тестів Cypress та Playwright. Маючи тверде розуміння базових концепцій, описаних у цьому документі, ви зможете швидко освоїти написання реальних workflow для ваших проектів.

👉 НАСТУПНИЙ КРОК: YAML-СИНТАКСИС

🔗 CYPRESS TA PLAYWRIGHT