Часть 1: Основы CI/CD – что это и зачем нужно; обзор GitHub Actions и GitLab CI

> Когда я только начинал свой путь в сторону DevOps и автоматизации, мне не хватало материалов, которые не просто показывали бы, как настроить CI/CD, но объясняли бы — *зачем* это нужно, *как* это работает, *почему* это стало стандартом. Слишком часто всё сводилось к "вот скопируй `.yml` и всё будет". Этот цикл статей — попытка дать вам то, чего тогда не хватило мне самому.

> Данная статья только начало пути становления, вот небольшая карта:
>
> **Часть 1: Основы CI/CD – что это и зачем нужно; GitHub Actions и GitLab CI**
>
> **^ Вы сейчас здесь ^**
>
> **Часть 2: Настройка GitHub Actions и GitLab CI – первый workflow и деплой**
>
> **Часть3: CI/CD – ветки, условия, секреты и окружения**

Введение

В последние годы вы, вероятно, слышали слова "DevOps", "CI", "CD", возможно, даже "GitHub Actions" или "GitLab CI". Но что это всё значит на практике? Нужно ли быть сеньором DevOps-инженером, чтобы с этим разобраться? К счастью — нет. Эта серия статей написана простым языком для тех, кто только начинает знакомство с автоматизацией разработки.

Сегодня мы разберём базовые понятия: что такое CI/CD, как оно работает, какие инструменты для этого есть — и чем отличаются GitHub Actions и GitLab CI. Мы не будем

углубляться в абстракции или перегружать терминологией. Вместо этого — объясню на примерах, аналогиях и легкостью.

CI/CD — это не какая-то магия или модный корпоративный тренд. Это ваш личный **робот-помощник**, который берёт на себя скучные задачи: запускает тесты, собирает проект, выкладывает его на сервер. Представьте себе автоматизированный **конвейер**, куда вы кладёте свежий код — а на выходе получаете готовое к работе приложение. Звучит круто? Тогда поехали.

Что такое CI и CD

Аббревиатуры CI и CD — одни из самых часто упоминаемых в мире DevOps. Давайте разберёмся с каждой по порядку:

CI — Continuous Integration (непрерывная интеграция)

Идея простая: когда несколько разработчиков работают над проектом, они постоянно вносят изменения в код. Раньше это приводило к хаосу — слияние изменений, баги, несостыковки. С CI при **каждом коммите** изменения автоматически собираются, тестируются и проверяются. Это позволяет быстрее находить ошибки и быть уверенным, что "ничего не сломалось".

Непрерывная интеграция — это практика:

- частого слияния изменений в основной репозиторий (хотя бы раз в день),
- автоматического запуска тестов и проверок при каждом изменении,
- своевременного выявления ошибок до того, как они попадут на продакшн.

Цель — **не допустить "интеграционного ада"**, когда всё работает по отдельности, но ничего не работает вместе.

Пример: вы изменили один файл и закоммитили. Система тут же запускает сборку и юнит-тесты. Если что-то пошло не так — вы узнаете об этом сразу, а не через неделю, когда баг всплывёт у пользователя.

CD — Continuous Delivery / Deployment (непрерывная доставка / развёртывание)

CD идёт дальше. Если CI гарантирует, что ваш код работает корректно после изменений, то CD гарантирует, что **его можно развернуть**. Это может быть:

- **Delivery** — автоматическая подготовка к выпуску, но с ручным одобрением;
- **Deployment** — полностью автоматическое развертывание на сервер после прохождения всех тестов.

Проще говоря: если CI проверяет, что ваш код **не сломан**, то CD делает так, чтобы этот исправный код оказался **на сервере или в продакшене**.

Pipeline (пайплайн)

Пайплайн — это цепочка шагов, которая запускается при изменении кода. Обычно она состоит из:

1. Сборки проекта.
2. Запуска тестов.
3. Развёртывания (если всё прошло успешно).

Аналогия: вы кладёте ингредиенты на вход, конвейер готовит, проверяет и подаёт готовое блюдо. И всё это без участия человека. CI/CD — это не гаджет, а **умный робот**, который каждый день помогает команде экономить часы времени.

Зачем нужна CI/CD

Если коротко — чтобы **разработка шла быстрее, качественнее и спокойнее**.

Преимущества:

- **Раннее обнаружение ошибок.** Лучше поймать баг сразу, чем искать его в продакшене.
- **Быстрая обратная связь.** Коммит → проверка → результат.
- **Автоматизация.** Не нужно запускать тесты вручную — всё происходит само.
- **Готовность к релизу.** Вы всегда уверены, что проект можно выкатить.

Без CI/CD:

- Каждый разработчик вручную запускает сборку и тесты.
- Часто забывают что-то проверить.
- Ошибки всплывают поздно, когда их уже сложно исправить.

С CI/CD:

- Ошибка ловится за минуты.
- Никто не забывает протестировать.
- Релизы стабильнее и выходят быстрее.

CI/CD — это не просто удобство, это **стратегическое преимущество** команды. И главное — настроить это можно с минимальными усилиями, даже если вы работаете один.

GitHub Actions — обзор

Если вы уже пользуетесь GitHub, то хорошая новость — система CI/CD уже встроена прямо туда. Называется она **GitHub Actions**.

Основные термины:

- **Workflow** — сценарий, написанный в YAML-файле. Он описывает, что и когда должно происходить.
- **Job** — задача внутри workflow. Например, "проверить тесты".
- **Step** — шаг внутри job. Например, "установить зависимости", "запустить npm test".
- **Action** — готовая команда или скрипт. Например, `actions/checkout` — чтобы получить код репозитория.

Где находятся конфигурации?

В папке `.github/workflows/` вашего репозитория. Вы можете создать столько файлов, сколько нужно. Например, один файл для тестов, другой — для деплоя.

Пример:

```
name: Simple CI

on:
  push:
    branches: [main]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: echo "Hello, GitHub Actions!"
```

> В данной статье не будем углубляться в код и рассматривать его подробно, об этом поговорим чуть позже, в следующей части.

При каждом пуше в main ветку GitHub запустит пайплайн, и вы увидите результат во вкладке **Actions**.

GitHub также предоставляет Marketplace с тысячами готовых actions: для деплоя, линтеров, публикации на Docker Hub и многого другого.

GitLab CI — обзор

Если вы используете GitLab, то CI/CD тоже встроено — под названием **GitLab CI/CD**. Работает оно немного иначе, но идея та же.

Файл конфигурации:

Один файл `.gitlab-ci.yml` в корне репозитория.

Основные термины:

- **Pipeline** — весь процесс CI/CD.
- **Stage (этап)** — логическая часть пайплайна: build, test, deploy.
- **Job** — конкретная задача в рамках этапа.
- **Script** — список команд, которые выполняются в job-е.

Пример:

```
stages:
  - test

hello_job:
  stage: test
  image: node:16
  script:
    - echo "Hello, GitLab CI!"
```

После коммита GitLab запускает пайплайн. Всё можно отслеживать через интерфейс GitLab — вкладка **CI/CD → Pipelines**.

GitLab CI хорошо интегрирован со своими фичами: issues, merge requests, environments. Также можно использовать как **общие раннеры GitLab**, так и настраивать свои.

Сравнение GitHub Actions и GitLab CI

Параметр	**GitHub Actions**	**GitLab CI**
Файл конфигурации	`.github/workflows/*.yml`	`.gitlab-ci.yml`
Формат YAML	`jobs`, `runs-on`, `steps`	`stages`, `script`, `image`
Триггеры	`on: push/pull_request/...`	`only`, `except`, `rules`
Среда выполнения	Linux, Windows, Mac runners	Чаще Docker-образы (например, `node:14`)

Интерфейс	Вкладка **Actions** в репозитории	Вкладка **CI/CD > Pipelines**
Тарифы	Бесплатно для публичных и приватных репозиториев	Бесплатно до лимита минут

Оба инструмента отлично подходят для автоматизации. Выбор между ними зависит скорее от платформы, на которой вы работаете, и ваших предпочтений.

Примеры использования

CI/CD можно использовать для самых разных проектов:

- **Статический сайт (HTML/CSS/JS):** при пуше автоматически деплоится на GitHub Pages или GitLab Pages.
- **Node.js приложение:** ставим зависимости, запускаем тесты, выкладываем.
- **Python-скрипт:** устанавливаем requirements, запускаем `pytest`, деплоим на сервер.
- **Фронтенд (React, Vue) и бэкенд (Express, Flask):** автоматизируем и сборку, и тесты, и релиз.

Неважно, какой у вас стек — если проект использует git, его можно обернуть в CI/CD.

Терминология

- **CI (Continuous Integration)** — автоматическая сборка и тестирование при каждом коммите.
- **CD (Continuous Delivery/Deployment)** — автоматическая подготовка или публикация релиза.
- **Pipeline** — цепочка шагов: сборка, тесты, деплой.
- **Workflow** — сценарий в GitHub Actions.
- **Job** — отдельная задача: сборка, тесты и т.д.
- **Step** — шаг внутри job-а.
- **Runner** — сервер/контейнер, исполняющий задачи.
- **Stage** — этап в GitLab CI.
- **YAML** — текстовый формат, в котором описывается pipeline.

Итоги

Мы познакомились с основами CI/CD — что это, зачем нужно, и как это устроено. Узнали про два популярных инструмента:

- **GitHub Actions** — CI/CD, встроенное прямо в GitHub.
- **GitLab CI** — мощная система автоматизации в GitLab.

Они позволяют автоматизировать почти любую задачу: от проверки синтаксиса до деплоя приложения.

В следующей статье мы пойдём дальше: **создадим первый workflow и pipeline своими руками**, увидим их в действии и задеплоим сайт на GitHub Pages и GitLab Pages.

Revision #3
Created 2026-03-05 08:12:52 UTC by Эд Кукса
Updated 2026-03-05 09:00:01 UTC by Эд Кукса