DevOps и CI/CD

Автоматизация

Скорость доставки новых функций определяет успех продукта. Компании, которые выпускают обновления несколько раз в день, опережают конкурентов с ежемесячными релизами. Как этого достичь?

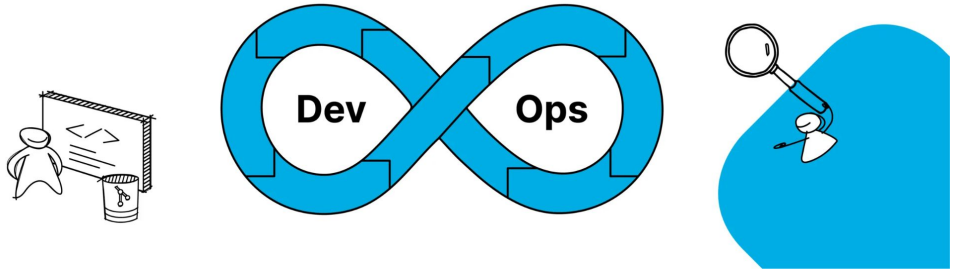
=> требуется **автоматизация**

- сократить количество человеческих ошибок
- сделать процессы эффективными и воспроизводимыми
- иметь полный контроль над инфраструктурой приложения

Проблема

DevOps

DevOps («Development» и «Operations») — это **культура** и **набор практик**, направленных на устранение барьеров между командами разработки и эксплуатации. Это **философия**, которая меняет подход к созданию и эксплуатации ПО.

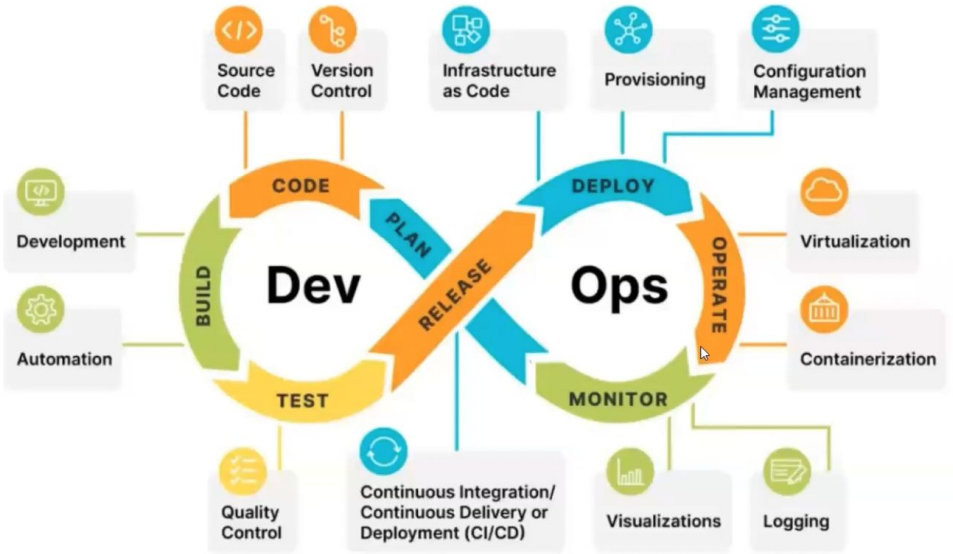


DevOps охватывает весь жизненный цикл продукта: от планирования и разработки до мониторинга и поддержки.

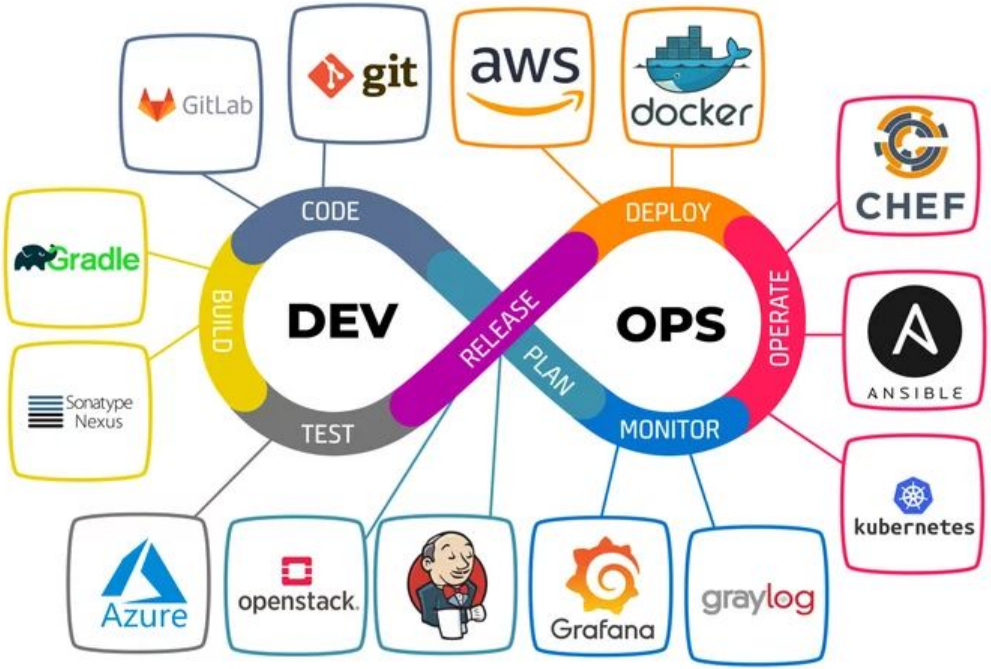
Цикл DevOps

Культура DevOps носит непрерывный (continuous) характер. Его можно представить в виде бесконечного цикла, состоящего из фаз:

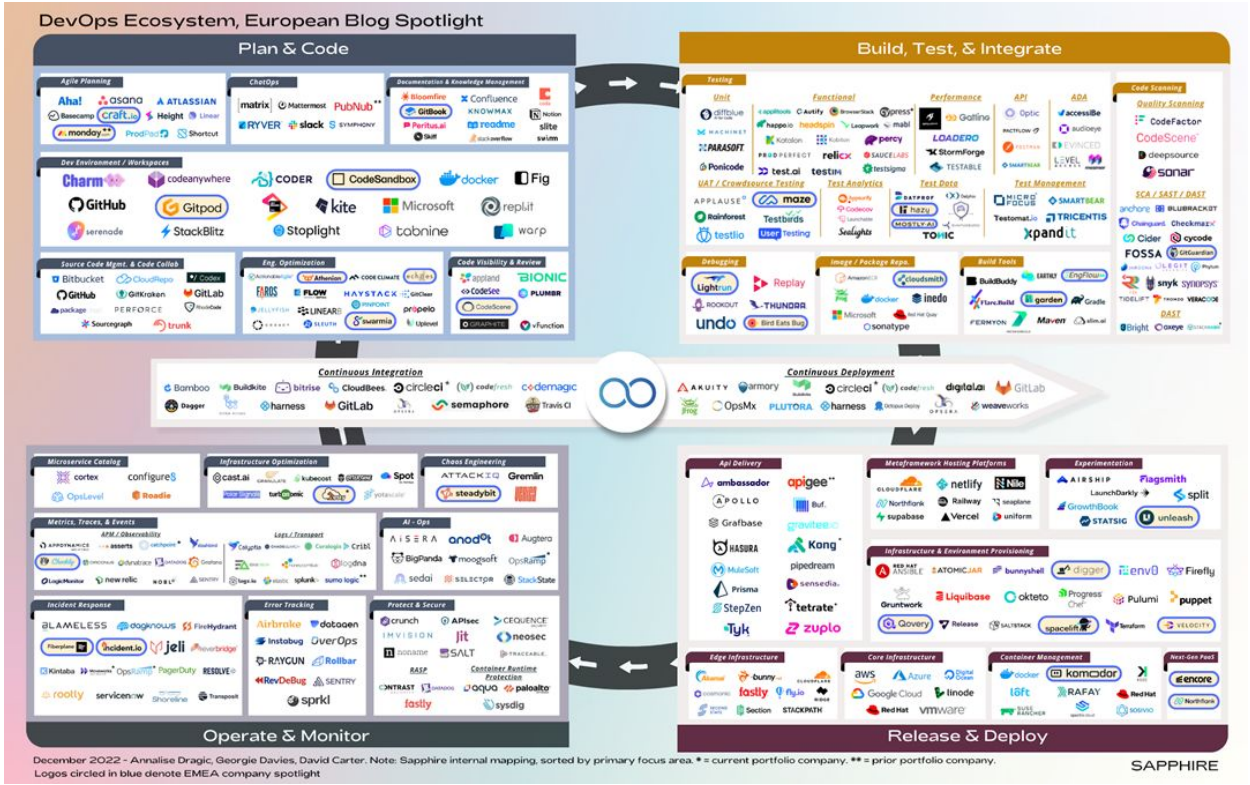
- **Plan**: планирование разработки
- **Code**: совместная разработка
- **Build**: автоматическая сборка кода
- **Test**: проверка качества
- **Release:** подготовка кода к развертыванию
- **Deploy**: подготовка инфраструктуры, предоставление ресурсов, конфигурация
- **Operate**: управление приложением на инфраструктуре
- **Monitor**: непрерывный мониторинг за состоянием приложения



Технологии DevOps



Технологии DevOps



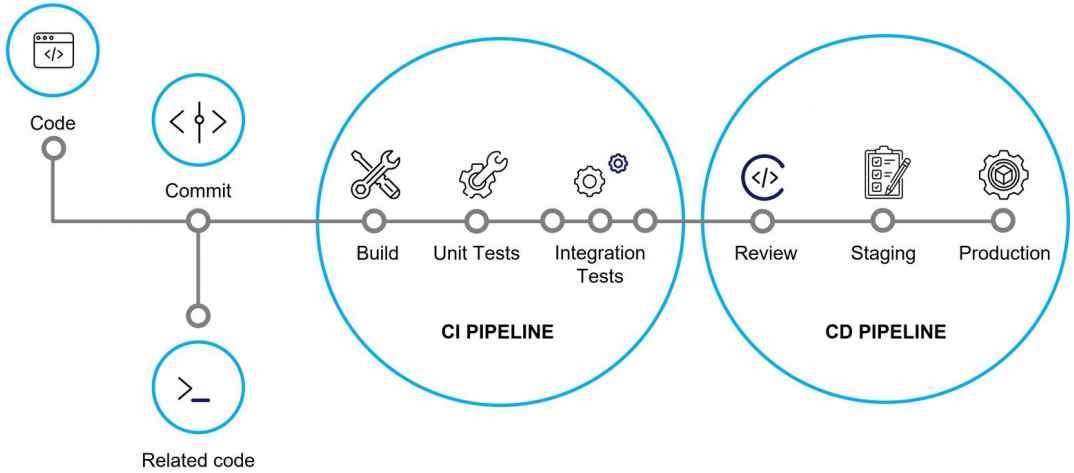
7 C's of DevOps

Как мы уже упоминали, в DevOps всё **непрерывно** — от планирования до мониторинга.
Разобьём весь жизненный цикл на семь фаз (**7 C`s**). Любая фаза жизненного цикла может повторяться в рамках проекта несколько раз, пока не завершится.

=> CI/CD Pipeline

CI/CD Pipeline

CI/CD (или **CI/CD pipeline**) — одна из ключевых практик DevOps. Фокусируется на автоматизации процессов интеграции, тестирования и доставки кода.



CI — Continuous Integration — это практика, при которой разработчики часто интегрируют код в общий репозиторий. Каждая интеграция проверяется автоматической сборкой и тестами.

CD может означать две вещи:
> **Continuous Delivery** — непрерывная доставка, когда код автоматически готовится к релизу, но развёртывание происходит вручную.
> **Continuous Deployment** — непрерывное развёртывание, когда каждое изменение автоматически попадает в продакшн после прохождения всех проверок.

Можно сказать, что CI/CD — это сердце DevOps. Без автоматизированных пайплайнов невозможно достичь той скорости и надёжности, которые требует DevOps-подход. При этом DevOps включает и другие практики: IaC, мониторинг и логирование, AQA, управление инцидентами и т. д.

Инструмент CI/CD

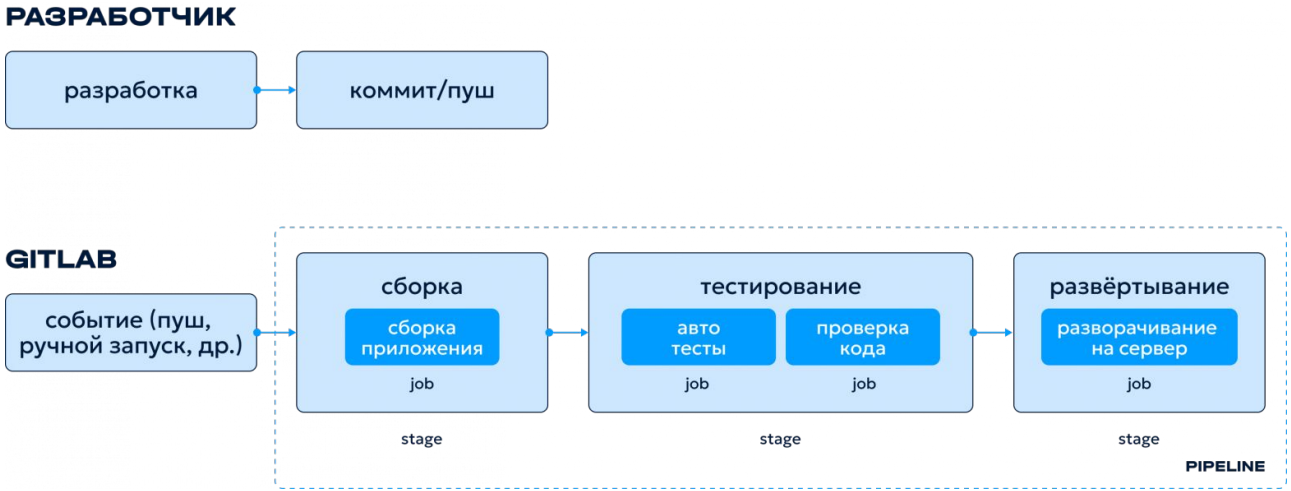
GitLab CI/CD

Gitlab CI/CD — полностью интегрированная в GitLab система для автоматизации сборки, тестирования и развертывания программного кода.

Этот процесс является частью более крупного рабочего процесса:

Общая схема

Пайплайн (pipeline) представляет собой целиковый процесс из этапов или **стадий (stage)**, которые состоят из **задач (job)**. Каждая задача выполняется в изолированном процессе (используется **GitLab Runner**).



Создание pipeline-а

GitLab ищет в корне репозитория конфиг **.gitlab-ci.yml** и, когда находит, запускает пайплайн согласно описанной в конфиге логике.

- **Этапы (stages)**: сборка, тестирование, деплой.
- **Задачи (jobs)**: конкретные действия на каждом этапе.
- **Условия запуска (rules)**: когда и при каких событиях выполнять ту или иную задачу (по коммиту, MR или расписанию).
- **Переменные окружения (env)**: безопасная работа с ключами, логинами и паролями.

YAML

```yaml
stages:
    - build
    - test

build_job:
  stage: build
  script:
    - npm install
    - npm run build

test_job:
  stage: test
  script:
    - npm run test
```

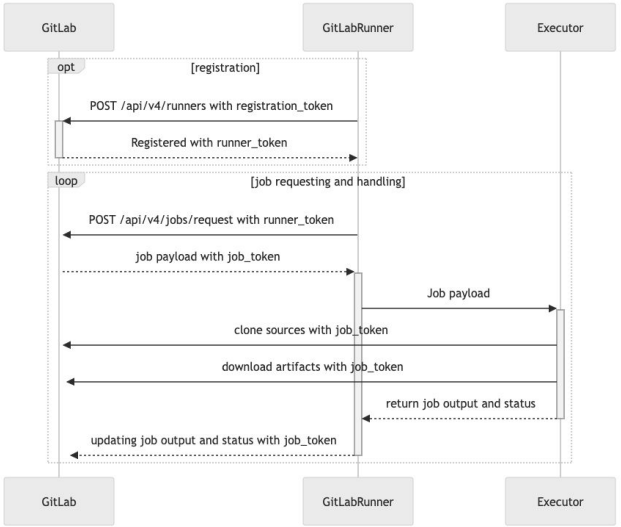
Gitlab Runners

Раннер (gitlab runner) — это отдельное приложение (агент), которое подключается к вашему репозиторию, а затем исполняет команды, описанные в .gitlab-ci.yml.

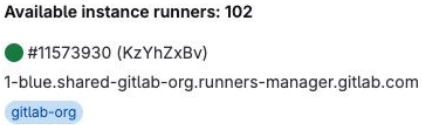
Оно может работать на любом сервере: локальной машине, VPS или в облаке.

Вы также можете указать образ контейнера, который хотите использовать при запуске задания.

Settings -> CI/CD -> Runners



ex:
Shell executor
Docker executor



CI/CD переменные

Переменные GitLab CI/CD — это пары «ключ-значение», которые используются для хранения и передачи параметров конфигурации и конфиденциальной информации, например паролей или ключей API, заданиям в конвейере.

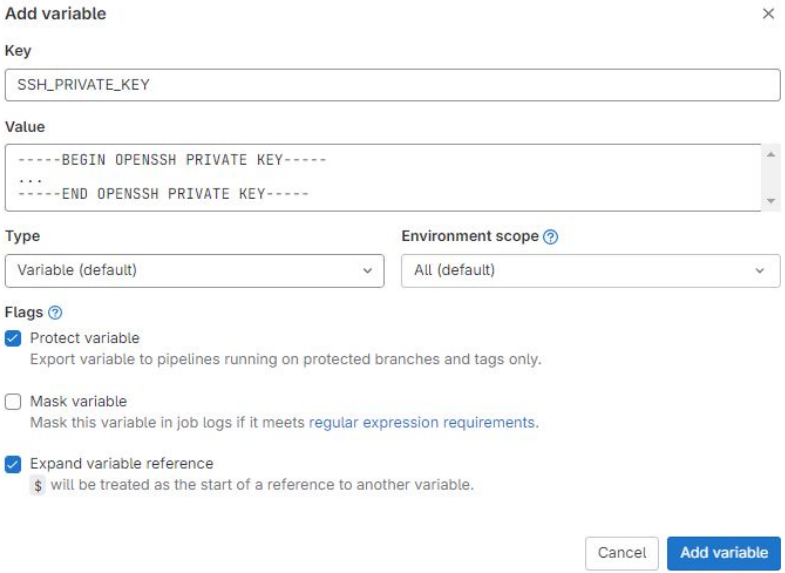
Типы:

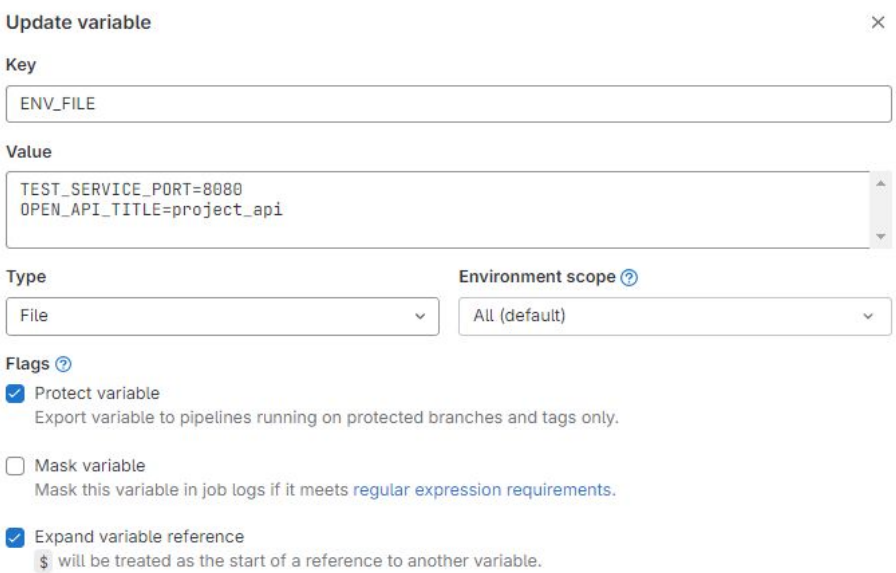
- **Пользовательские:** Создавайте и управляйте ими в пользовательском интерфейсе GitLab [Settings -> CI/CD -> Variables]. Также задаются с помощью ключевого слова variables.
- **Системные:** Автоматически устанавливаются GitLab и предоставляют информацию о текущем задании, конвейере и среде.

В целях безопасности могут быть помечены:

- **Protected:** доступны только для заданий, работающих на защищенных ветках или тегах.
- **Masked:** скрывать значение в логах заданий, чтобы предотвратить раскрытие конфиденциальной информации.

CI/CD переменные



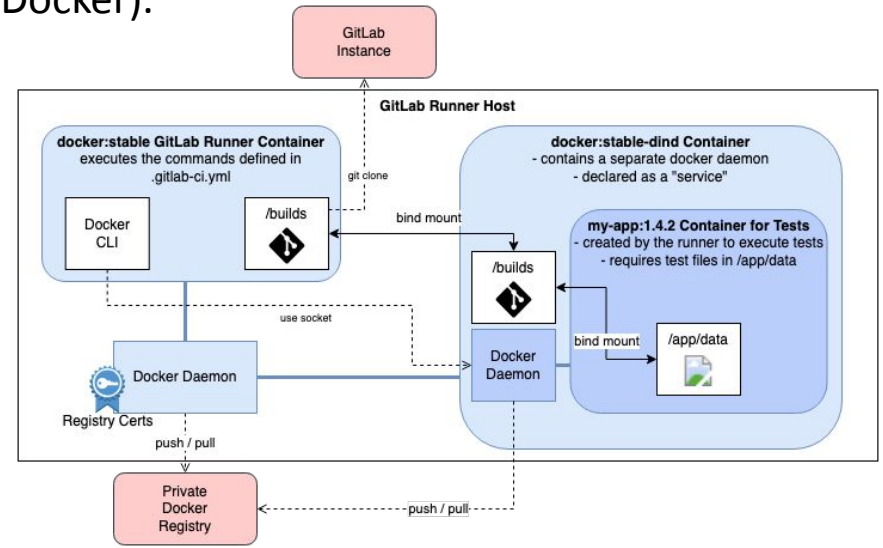


Демонстрация

…

Сборка и публикация Docker образа

Для создания образов и публикации Docker образов используйте **DIND** (Docker in Docker).




```yaml
docker:build:
  stage: build
  image: docker:latest # as docker client
  services:
  - docker:dind # as docker server
  rules:
    - if: $CI_COMMIT_BRANCH == 'gitlab-ci'
  before_script:
    - docker login -u ${CI_REGISTRY_USER} -p ${CI_JOB_TOKEN} ${CI_REGISTRY}
  script:
    - docker build --pull -t ${CI_DOCKER_IMAGE} .
    - docker push ${CI_DOCKER_IMAGE}
```

** Клиент Docker и демон могут работать на одной системе, или вы можете подключить клиент Docker к удалённому демону Docker.*

Распространённые ошибки и антипаттерны

- **«У меня работает»**: код исправно функционирует у разработчика, но неожиданно падает в CI. Причина кроется в различиях окружений: версии библиотек, переменные среды, операционные системы. Используйте Docker!
- **Медленная обратная связь**: когда пайплайн выполняется часами. Правильно декомпозируйте и применяйте оптимизации (например, КЭШ)
- **Нестабильные (Flaky) тесты**: тесты, которые периодически падают без видимых причин, — подрывают доверие к CI/CD.
- **Ручные шаги в автоматизированном процессе**: инструкции вроде «не забудь обновить конфигурацию на сервере» в документации к релизу — верный признак проблем в будущем. Человеческий фактор рано или поздно сработает.
- **Отсутствие стратегии отката**: когда релиз идёт не по плану, а чётких правил отката нет.
- **Несоответствие тестовых и боевых окружений**: становятся источником неприятных сюрпризов при релизе.
- **Попытка автоматизировать всё сразу**: команды часто стремятся построить идеальный пайплайн с первого раза

Задание: GitLab CI/CD

Оценка: 2

Description: Выбрать любой проект и

- настроить CI pipeline (статический анализ и тесты) (1 балл)
- настроить CD pipeline (сборка docker и публикация docker образа) (1 балл)

DoD: GitLab/GitHub репозиторий с .gitlab-ci.yml файлом, ссылка на результат работы пайплайнов и Docker Registry (например, Docker Hub)