Тема 5

CI/CD: GitHub Actions и автоматический деплой



Каждый merge в `main` сам тестируется и через пару минут работает

План лекции

Сегодня

1. Зачем CI и CD
2. Как устроен GitHub Actions
3. Пайплайн сервиса задач
4. Self-hosted runner и деплой на Docker Desktop
5. Откат, секреты, права

После лекции вы сможете

- Объяснить каждый шаг пайплайна
- Запретить merge при красных тестах
- Деплоить образ по неизменяемому тегу
- Откатиться за минуту

Ручной деплой против пайплайна



Руками

- Забыли запустить тесты
- Собрали образ со своими незакоммиченными правками
- Остановили не тот контейнер
- Неизвестно, какая версия работает



Пайплайн

- Тесты на каждый PR — всегда
- Образ собирается из коммита на чистой машине
- Деплой одинаковый каждый раз
- Версия = sha коммита, видна в `GET /`

Ошибка видна через минуты после push, а не у пользователей

CI, Delivery, Deployment



Continuous Integration

Каждое изменение автоматически собирается и тестируется



Continuous Delivery

Готов артефакт (образ), выкладка по кнопке



Continuous Deployment

Выкладка автоматически после merge. Так сделаем мы

Устройство workflow


```yaml
name: ci-cd
on:                                    # события
  pull_request: { branches: [main] }
  push: { branches: [main], tags: ["v*.*.*"] }

permissions:
  contents: read                       # минимальные права

jobs:
  test:                                # job — одна машина
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v7      # готовый action
      - run: dotnet test               # команда
  image:
    needs: test                        # после test
    if: github.event_name == 'push'
```

1. **Event**

   Что запускает: push, PR, кнопка, расписание

2. **Job**

   Набор шагов на отдельной машине (runner)

3. **Step**

   `uses:` — готовый action, `run:` — команда

4. **needs / if**

   Порядок и условия запуска

Пайплайн сервиса задач



На PR — только test. На тег `v1.2.0` — test и image с тегом `1.2.0`

Job test: тесты с настоящей БД


```
test:
  runs-on: ubuntu-latest
  services:
    postgres:
      image: postgres:18-alpine
      env: { POSTGRES_DB: demo, POSTGRES_USER: demo,
             POSTGRES_PASSWORD: demo }
      ports: ["5432:5432"]
  env:
    Storage: Postgres
    ConnectionStrings__Db: Host=localhost;...;Password=demo
  steps:
    - uses: actions/checkout@v7
    - uses: actions/setup-dotnet@v6
    - run: dotnet test
```

1. **services**

   PostgreSQL в контейнере рядом с job

2. **env**

   Переключают тесты на PostgreSQL

3. **Те же тесты**

   Что вы запускали в Visual Studio

Job image: теги образа в GHCR

Реестр `ghcr.io/<owner>/<repo>` — только нижний регистр. Push через `GITHUB_TOKEN` с правом `packages: write`

Тег	Для чего
`sha-abc1234`	Деплой. Неизменяемый: точно известно, какой коммит запущен
`1.2.0`	Релиз по тегу git
`latest`	Удобство. Не для деплоя

Версия попадает в образ как `APP_VERSION` и видна в `GET /`

Self-hosted runner: деплой на ваш ПК

GitHub

Job deploy ждёт в очереди

Runner на вашем ПК

Сам опрашивает GitHub и забирает задание

Docker Desktop

`compose pull` → `up --wait` → smoke

Только исходящие соединения: белый IP и открытые порты не нужны

Деплой, которому можно доверять

```
- run: docker compose pull api
- run: docker compose up -d --no-build --wait --wait-timeout 90
- run: docker compose exec -T api curl -fsS
http://localhost:8080/health/ready
```

1. **pull**
   Скачать образ `sha-…` из GHCR
2. **--wait**
   Не стал healthy — job красный
3. **smoke test**
   Сервис работает и видит БД



Откат = деплой предыдущего `sha-` тега

Пересобирать ничего не нужно: образ уже в реестре и в кэше Docker. Сначала восстановить сервис — потом разбираться

Секреты, права, окружения

Secrets

`DB_PASSWORD` в Settings → Secrets. В логах GitHub их маскирует

permissions

`contents: read` везде, `packages: write` только в image

Environments

Ручное одобрение деплоя и секреты окружения

Типичные ошибки

- `permissions` не заданы или `write-all`
- Секреты в YAML или в выводе `echo`
- Деплой `latest`: откат невозможен
- Деплой без `--wait` и smoke test
- Интеграционные тесты без БД в CI
- Заглавные буквы в имени образа GHCR

Проверьте себя

1. Почему на PR не собирается образ?
2. Откуда в CI берётся PostgreSQL и как тесты о нём узнают?
3. Зачем деплоить по `sha-` тегу, если есть `latest`?
4. Как GitHub достучится до runner за NAT?
5. Как откатиться за минуту?



Практика

Лаба 5. Тесты, образ и автодеплой

Подключаем пайплайн к репозиторию сервиса

- Тесты на каждый PR и запрет merge при красных
- Self-hosted runner на вашем ПК
- Первый автодеплой: `version` = sha коммита
- Изменение через пайплайн и откат