# Неделя 9. Лаба 5. CI/CD: тесты, образ и автодеплой сервиса

*Часть 1 из 2: разделы «1. Тесты локально» — «5. Секрет».*

**Цель:** пайплайн «тесты → образ в GHCR → автодеплой на свой Docker Desktop → smoke test», блокировка merge при красных тестах, откат. **Время:** 120 мин. **Теория:** лекция 5. **Инструменты:** GitHub Actions, GitHub Container Registry (GHCR), self-hosted runner, Docker Desktop, Visual Studio 2026, PowerShell 7.

**Подготовка:**

```
cd C:\dev\demo-api
$K = "C:\dev\kit"              # путь к kit из лабы 3
Test-Path $K\.github           # True
```

## 1. Тесты локально

В репозитории уже есть юнит- и интеграционные тесты (`tests/DemoApi.Tests`).

```
git switch main; git pull
git switch -c ci/pipeline
docker compose up -d --wait db
dotnet test
```

Тесты подключаются к БД с паролем `demo` из `appsettings.Development.json` — тем же, что в `.env` (лаба 3). Если в `.env` задан `DB_PORT=15432`, перед `dotnet test` выполните: `$env:ConnectionStrings__Db = "Host=localhost;Port=15432;Database=demo;Username=demo;Password=demo"`

Ожидаемо: все тесты зелёные.

**Зачем сначала локально:** если тесты красные у вас, CI не поможет. Пайплайн выполняет ту же команду `dotnet test`, только автоматически.

## 2. Workflow

```
New-Item -ItemType Directory -Force .github\workflows | Out-Null
Copy-Item $K\.github\workflows\ci-cd.yml .github\workflows\
```

Прочитайте файл целиком и **ответьте в описании будущего PR:**

1. Какие jobs запустятся на PR? На push в `main`? На тег `v1.0.0`?
2. Откуда в тестах берётся PostgreSQL?
3. Какие права у `GITHUB_TOKEN` в job `image` и зачем?
4. Где выполняется job `deploy` и почему именно там?
5. Что произойдёт, если новый контейнер не станет `healthy`?

```
git add -A
git commit -m "ci: add build, test, image and deploy pipeline"
git push -u origin ci/pipeline
```

Откройте PR. Во вкладке *Checks* выполнится только **test**; `image` и `deploy` будут помечены как *skipped* — так и должно быть: на PR образ не собирается.

## 3. Тесты как обязательное условие merge

*Settings* → *Rules* → *Rulesets* → *main* → ✓ **Require status checks to pass** → *Add checks* → введите `test` и выберите вариант с иконкой GitHub Actions → Save.

Попросите партнёра сделать *Approve*. Затем сломайте тест в той же ветке: в `tests/DemoApi.Tests/Integration/ProjectsApiTests.cs` в тесте `Create_DuplicateName_ReturnsConflict` замените `HttpStatusCode.Conflict` на `HttpStatusCode.OK`. Закоммитьте и запушьте.

Аппрув есть, но merge **заблокирован**, пока `test` красный. Откройте лог упавшего шага: найдите имя теста и ожидаемый/фактический код. Сделайте скриншот, верните тест, запушьте.

**Зачем:** сломанный код физически не может попасть в `main`, а значит, и на деплой.

**Пока не мержите** — сначала нужен runner.

## 4. Self-hosted runner

*Settings* → *Actions* → *Runners* → *New self-hosted runner* → **Windows**, x64.

Откройте **новый** терминал и перейдите в корень диска: `cd C:\`. Выполните команды со страницы по порядку — они сами создадут папку `C:\actions-runner`. **Не выполняйте их внутри репозитория.** При настройке:

- на все вопросы — Enter (значения по умолчанию);
- **Run as service? → N.**

Запустите runner и **не закрывайте окно** до конца лабы:

```
cd C:\actions-runner
./run.cmd              # Listening for Jobs
```

На странице Runners статус **Idle**.

**Зачем свой runner:** машина GitHub не видит ваш Docker Desktop. Runner на вашем ПК сам забирает задания у GitHub и выполняет деплой локально.

## 5. Секрет

*Settings* → *Secrets and variables* → *Actions* → *New repository secret*: `DB_PASSWORD` = `demo` — тот же, что в `.env`.

> Почему тот же: деплой использует тот же compose-проект `demo` и тот же том `demo_pgdata`. PostgreSQL применяет пароль только при первой инициализации тома (лекция 3).

> У вас `DB_PORT=15432`? Runner не видит ваш `.env` (он в `.gitignore`), поэтому возьмёт порт 5432 и упадёт. Добавьте в workflow в блок `env:` job `deploy` строку `DB_PORT: 15432`.