Неделя 4. Лаба 2. Docker: упаковываем сервис задач и запускаем на Docker Desktop

Часть 2 из 2: разделы «6. Сервис на Docker Desktop» — «⭐ Дополнительно». Первая часть лабы — в материалах предыдущей недели; перед продолжением выполните подготовку ниже.

Цель: написать multi-stage Dockerfile для сервиса задач, собрать образ, запустить сервис на Docker Desktop и поработать с ним — из терминала и из окна Docker Desktop. **Время:** 90 мин. **Теория:** лекция 2. **Инструменты:** Docker Desktop, Visual Studio 2026, PowerShell 7.

6. Сервис на Docker Desktop

```
docker run -d --name api -p 8080:8080 demo-api:1.1
Start-Sleep 20
docker ps                                   # STATUS: Up ... (healthy)
irm http://localhost:8080/                  # service: DemoApi, version: dev, host: <id контейнера>
irm -Method Post http://localhost:8080/api/projects -ContentType application/json -Body '{"name":"Из контейнера"}'
irm http://localhost:8080/api/projects
```

Сервис работает в контейнере точно так же, как в Visual Studio, но на компьютере не нужен ни .NET SDK, ни Visual Studio — только Docker.

Откройте коллекцию `demo-api` в Postman из лабы 1 и отправьте те же запросы: адрес тот же, `localhost:8080`, только теперь на нём отвечает контейнер. Клиенту не видно, где запущен сервис.

`host` в ответе `GET /` — это id контейнера: у каждого контейнера своё имя хоста.

7. Смотрим в Docker Desktop

Откройте окно Docker Desktop → **Containers** → `api`:

Вкладка	Что найти
Logs	JSON-строку `Создан проект` — это ваш запрос из п. 6. Строку `Хранилище: в памяти`
Inspect	Переменные окружения `APP_VERSION`, `ASPNETCORE_HTTP_PORTS`; проброс порта 8080
Exec	Терминал внутри контейнера: выполните `whoami` (→ `app`) и `ls /app` (DLL сервиса)
Stats	Сколько памяти и CPU потребляет сервис

То же из терминала:

```
docker logs api --tail 5
docker exec api whoami                                  # app — не root
docker inspect api --format '{{json .State.Health}}'    # результаты HEALTHCHECK
docker stats --no-stream api
```

8. Контейнер и данные

```
docker stop api
docker ps -a --filter name=api          # Exited (0)
docker start api
Start-Sleep 10
irm http://localhost:8080/api/projects  # ?
```

Проект «Из контейнера» пропал. **Почему?** Процесс сервиса завершился при `docker stop`, а вместе с ним — данные в памяти. То же будет при `docker rm` и новом `docker run`. Запишите ответ.

Вывод: состояние нельзя хранить в памяти процесса. В теме 3 сервис будет хранить данные в PostgreSQL, а сама БД — на томе, который переживает контейнеры.

9. Один образ — разные окружения

В контейнере окружение `Production`, поэтому страницы `/swagger` нет:

```
curl.exe -sL -o NUL -w "%{http_code}`n" http://localhost:8080/swagger     # 404

docker run -d --name api-dev -p 8082:8080 -e ASPNETCORE_ENVIRONMENT=Development demo-api:1.1
Start-Sleep 10
curl.exe -sL -o NUL -w "%{http_code}`n" http://localhost:8082/swagger     # 200
```

Откройте **http://localhost:8082/swagger** в браузере и создайте проект уже в этом контейнере.

Один образ — разное поведение за счёт переменных окружения. Именно так один и тот же образ идёт в тестовое окружение и в прод.

10. Уборка

```
docker rm -f api api-dev
```

Образы `demo-api` остаются — они понадобятся в теме 3.

Сдача

- [ ] Ваши `Dockerfile` и `.dockerignore`
- [ ] Вывод второго `docker build` с `CACHED` на шаге restore
- [ ] `docker ps` со статусом `(healthy)` и ответ `GET /`
- [ ] Скриншоты вкладок Logs и Exec в Docker Desktop
- [ ] Ответ на вопрос из п. 8
- [ ] Коды 404 и 200 из п. 9 и объяснение разницы

☆ Дополнительно

1. Соберите **single-stage** вариант: всё на `sdk:10.0`, `ENTRYPOINT ["dotnet", "run", "--project", "src/DemoApi"]`. Сравните размер образа и время старта с multi-stage.
2. Попробуйте `aspnet:10.0-noble-chiseled` в качестве рантайма. Что сломается в `HEALTHCHECK` и в `docker exec api sh`, и почему?