Неделя 12. Лаба 6. Жизнь контейнера: ломаем сервис и разбираем сбои

Часть 2 из 2: разделы «6. Нехватка памяти (OOM)» — «⭐ Дополнительно». Первая часть лабы — в материалах предыдущей недели; перед продолжением выполните подготовку ниже.

Цель: воспроизвести типовые сбои и научиться определять причину по логам, событиям, exit-коду и ресурсам. **Время:** 90 мин. **Теория:** лекция 6. **Инструменты:** Docker Desktop (CLI и вкладка Containers), PowerShell 7.

Подготовка: в лабе вы будете временно менять `compose.yaml`, поэтому работаем с локальной сборкой актуального `main`:

```
cd C:\dev\demo-api
git switch main; git pull
docker compose up -d --build --wait      # оба сервиса healthy
$api = 'http://localhost:8080'
```

Откройте **три терминала**:

Терминал	Команда
T1	для команд лабы
T2	`docker events --filter container=demo-api-1 --filter event=start --filter event=kill --filter event=die --filter event=oom --filter event=stop --filter event=health_status`
T3	`docker stats demo-api-1 demo-db-1`

Фильтры в T2 отсекают служебные события `exec_*`: их каждые 10 секунд порождает HEALTHCHECK, и в них тонут нужные `die` и `oom`.

По ходу лабы заполняйте таблицу для сдачи:

Сценарий	События (T2)	Exit code	Рестарт?	Что в логах

6. Нехватка памяти (OOM)

В T3 следите за колонкой `MEM USAGE / LIMIT`.

```
1..4 | ForEach-Object { curl.exe -s -m 10 -X POST "$api/api/chaos/leak?mb=80"; "" }
```

На 3–4 запросе соединение оборвётся. В T2: `oom` → `die` (`exitCode=137`) → `start`.

```
docker inspect demo-api-1 --format 'OOMKilled={{.State.OOMKilled}} Restarts={{.RestartCount}}'
```

`OOMKilled=false`, хотя OOM был. **Почему?** Где тогда искать правду?

7. Зависимость недоступна

```
docker compose stop db
curl.exe -s -w " %{http_code}`n" $api/health/live       # Healthy 200
curl.exe -s -w " %{http_code}`n" $api/health/ready      # Unhealthy 503
curl.exe -s -o NUL -w "%{http_code}`n" $api/api/tasks   # 500
docker compose ps                                       # api всё ещё healthy
docker compose start db
```

`/health/ready` = 503 — это и есть сигнал «не готов». API жив и healthy, но не может обслуживать запросы к БД. Контейнер не перезапускается — и это правильно: рестарт не починит БД.

8. Unhealthy

Временно сломайте проверку: добавьте в `compose.yaml` к сервису `api`:

```
    healthcheck:
      test: ["CMD", "curl", "-fsS", "http://localhost:8080/no-such-endpoint"]
      interval: 5s
      retries: 2
```

```
docker compose up -d --no-build
Start-Sleep 40
docker compose ps                                                # (unhealthy)
docker inspect demo-api-1 --format '{{json .State.Health}}'      # лог проверок
docker inspect demo-api-1 --format '{{.RestartCount}}'           # не вырос
```

Docker не перезапускает unhealthy контейнер. Верните файл и пересоздайте контейнер:

```
git restore compose.yaml
docker compose up -d --no-build --wait
```

9. Медленные ответы

```
curl.exe -s -o NUL -w "%{time_total}s`n" "$api/api/tasks"
curl.exe -s -o NUL -w "%{time_total}s`n" "$api/api/chaos/slow?ms=3000"
```

Контейнер жив, healthy, ошибок нет — а пользователю плохо. **Какой инструмент покажет эту проблему?** Ответ — в лабе 7.

Сдача

- [ ] Заполненная таблица по сценариям 2–8
- [ ] Ответы:
    1. Чем сценарий 2 отличается от сценария 3 для пользователя и для Docker?
    2. Почему после `docker compose stop api` нет рестарта при `unless-stopped`?
    3. Почему `OOMKilled=false` после OOM и где искать правду?
    4. Почему в `HEALTHCHECK` используется live, а не ready?
    5. Какие сбои из лабы **не видны** по `docker compose ps`?

☆ Дополнительно

Напишите `scripts/diag.ps1 <container>`. Скрипт выводит статус и health, `RestartCount`, лимит и текущее потребление памяти (`docker stats --no-stream`), последние 20 строк логов и события `die`/`oom` за 15 минут (`docker events --since 15m --until 0s`). Закоммитьте через PR.