Неделя 14. Лаба 7. Метрики и алерты: мониторинг сервиса задач

Часть 2 из 2: разделы «5. Дашборд как код» — «⭐ Дополнительно». Первая часть лабы — в материалах предыдущей недели; перед продолжением выполните подготовку ниже.

Цель: поднять Prometheus, Grafana, cAdvisor и Dozzle, научиться читать метрики сервиса и контейнера, увидеть срабатывание алертов на реальных сбоях. **Время:** 120 мин. **Теория:** лекция 7. **Инструменты:** Prometheus, Grafana, cAdvisor, Dozzle (контейнеры), Docker Desktop, браузер, PowerShell 7.

Подготовка:

```
cd C:\dev\demo-api
$K = "C:\dev\kit"              # путь к kit из лабы 3
Test-Path $K\monitoring        # True
$api = 'http://localhost:8080'
docker compose ps              # api и db — healthy
```

Стек мониторинга займёт порты 9090, 3000, 8081 и 8888. Проверьте, что они свободны (пустой вывод — свободны):

```
Get-NetTCPConnection -State Listen -LocalPort 9090,3000,8081,8888 -ErrorAction SilentlyContinue |
  Select-Object LocalPort, @{ n = 'Process'; e = { (Get-Process -Id $_.OwningProcess).ProcessName } }
```

5. Дашборд как код

Откройте дашборд *DemoApi*. Добавьте панель **«Tasks created per minute»** с запросом из задания 4.1.

Provisioned-дашборд Grafana не даст сохранить в UI. Экспортируйте JSON: *Export* → *Export as JSON*, формат **Classic**, переключатель экспорта «для другого экземпляра Grafana» **выключен** → *Copy to clipboard*. Замените этим содержимым файл `monitoring/grafana/dashboards/demo-api.json`, поле `uid` должно остаться `demo-api`. Через ~10 секунд Grafana подхватит файл: обновите страницу и проверьте, что панель на месте.

Зачем: дашборд в git переживёт пересоздание Grafana и проходит ревью, как код.

6. Алерты на реальных сбоях

Prometheus → *Alerts*. Для каждого сценария запишите время запуска, время перехода в **pending** и в **firing**. Во время каждого сценария смотрите **Grafana** (какая панель это показывает) и **Dozzle** (что в логах `demo-api-1`).

a. Ошибки → `ApiHighErrorRate`

```
1..60 | ForEach-Object { curl.exe -s -o NUL $api/api/chaos/error; Start-Sleep 1 }
```

b. Тормоза → `ApiSlowRequests`

```
1..40 | ForEach-Object { curl.exe -s -o NUL "$api/api/chaos/slow?ms=1500" }
```

c. Падение → `ContainerRestarted`

```
curl.exe -X POST $api/api/chaos/crash
```

`ContainerRestarted` смотрит на окно в 5 минут и останется в firing ещё 5 минут. **Перед сценарием d дождитесь, пока он вернётся в inactive** — иначе не поймёте, от какого сбоя алерт.

d. Утечка памяти → `ContainerHighMemory`, затем `ContainerRestarted`

```
curl.exe -s -m 10 -X POST "$api/api/chaos/leak?mb=80"
curl.exe -s -m 10 -X POST "$api/api/chaos/leak?mb=100"    # память ~85% лимита
Start-Sleep 30                                             # держится выше 80% — алерт успевает сработать
curl.exe -s -m 10 -X POST "$api/api/chaos/leak?mb=60"     # OOM: контейнер убит и перезапущен
```

Если после двух запросов память ниже 80% лимита (колонка `MEM %` в `docker stats` или дашборд), отправьте ещё один запрос с `mb=20` и подождите.

e. Недоступность → `ApiDown`

```
docker compose stop api          # после перехода алерта в firing:
docker compose start api
```

7. Закоммитьте

```
git add monitoring scripts
git commit -m "feat: add Prometheus, Grafana, cAdvisor and Dozzle monitoring"
git push -u origin feat/monitoring
```

PR → ревью → merge.

Сдача

- [ ] Скриншот *Target health* со всеми UP
- [ ] Четыре своих PromQL-запроса из п. 4
- [ ] Смерженный PR с панелью «Tasks created per minute» в JSON дашборда
- [ ] Таблица по сценариям a–e с временем pending и firing
- [ ] Ответы:
    1. Почему между началом сбоя и `firing` проходит время? Из чего оно складывается?
    2. Почему в сценарии **c** не сработал `ApiDown`?
    3. Какую проблему из лабы 6 (п. 9) теперь видно, а раньше — нет?

☆ Дополнительно

Добавьте бизнес-метрику `demo_tasks_created_total` — счётчик созданных задач. В начало `src/DemoApi/Services/TaskService.cs` добавьте `using Prometheus;`, в класс — поле `private static readonly Counter TasksCreated = Metrics.CreateCounter("demo_tasks_created_total", "Созданные задачи");`, а в `CreateAsync` после `tasks.CreateAsync(...)` вызовите `TasksCreated.Inc();`. Выведите метрику на дашборд и сравните с запросом из задания 4.1. Когда они разойдутся?