Тема 7

Мониторинг: метрики, дашборды, алерты



Узнаём о проблемах раньше пользователей

План лекции

Сегодня

1. Зачем мониторинг
2. Метрики, логи, трейсы
3. Стек: Prometheus, Grafana, cAdvisor
4. Метрики сервиса и PromQL
5. Дашборды и алерты

После лекции вы сможете

- Найти на графиках свои контроллеры и методы
- Писать базовые запросы PromQL
- Читать перцентили, а не средние
- Настраивать алерты по симптомам

Какие вопросы закрывает мониторинг



Жив ли?

Работает ли сервис прямо сейчас



Насколько плохо?

Сколько ошибок, как медленно



Что изменилось?

После какого деплоя стало хуже



Хватит ли ресурсов?

Память и CPU против лимитов

Три сигнала наблюдаемости

Метрики

Что происходит и насколько плохо. Числа во времени: Prometheus, Grafana

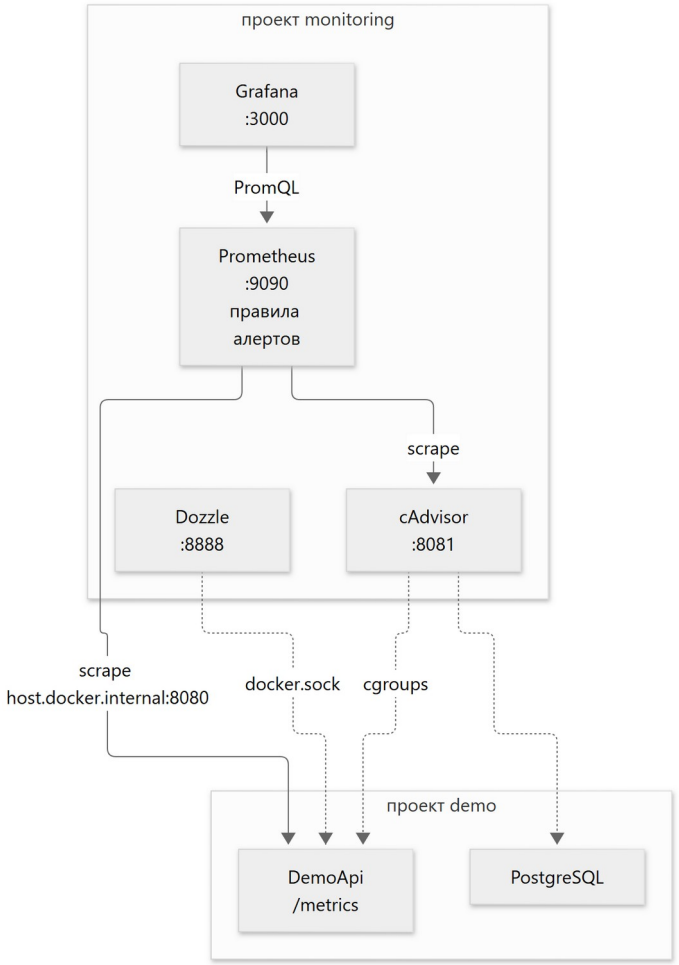
Логи

Почему это произошло. События с деталями: `docker logs`, Dozzle

Трейсы

Где тормозит запрос в цепочке сервисов. OpenTelemetry — за рамками курса

Стек мониторинга



- Сервис отдаёт метрики на `/metrics`
- cAdvisor — метрики каждого контейнера из cgroups
- Prometheus сам забирает метрики раз в 5 с и вычисляет алерты
- Grafana строит дашборды
- Dozzle — логи всех контейнеров в браузере
- Мониторинг — отдельный compose-проект: деплой его не трогает

Метрики нашего сервиса: /metrics


```
http_requests_received_total{code="201",method="POST",
  controller="Tasks",action="Create"} 97
http_requests_received_total{code="409",method="PATCH",
  controller="Tasks",action="ChangeStatus"} 3
http_request_duration_seconds_bucket{...,le="0.064"} 120
process_working_set_bytes 8.1e+07
```

1. **Одна строка кода**

   `app.UseHttpMetrics()`

2. **controller / action**

   Ваш класс и метод: `TasksController.Create`

3. **Группируйте по ним**

   Для 500 от обработчика ошибок `endpoint` пустой

Типы метрик

Counter

Только растёт: число запросов. Читаем через `rate()` и `increase()`

Gauge

Текущее значение: память контейнера. Читаем как есть

Histogram

Распределение по корзинам: время ответа. Читаем `histogram_quantile()`

Что смотреть: RED и USE



RED — сервис

- Rate: запросы в секунду
- Errors: доля 5xx
- Duration: p95 / p99 времени ответа



USE — ресурсы

- Utilization: память / лимит
- Saturation: троттлинг, очереди
- Errors: ошибки ресурса

p95 = 1 с: 95% запросов быстрее секунды. Среднее скрывает медленные хвосты

PromQL: язык запросов


```
# RPS по кодам ответа
sum by (code) (rate(http_requests_received_total{job="demo-api"}[1m]))

# Доля ошибок сервера
sum(rate(http_requests_received_total{code=~"5.."}[1m]))
  / sum(rate(http_requests_received_total[1m]))

# p95 по действиям контроллеров
histogram_quantile(0.95,
  sum by (le, controller, action)
    (rate(http_request_duration_seconds_bucket[1m])))
```

1. **`[1m]`**
   Окно: последняя минута

2. **`rate()`**
   Скорость роста счётчика в секунду

3. **`=~"5.."`**
   Регулярное выражение: все 5xx

4. **`le`**
   Граница корзины — нужна для перцентиля

Дашборды как код

Дашборд сервиса

Жив ли, сколько запросов, сколько ошибок, как быстро, сколько памяти — с первого взгляда

JSON в git

`monitoring/grafana/dashboards/` — переживает пересоздание Grafana и проходит ревью

Алерты

```
- alert: ApiHighErrorRate
  expr: <доля 5xx> > 0.05
  for: 30s                    # держится, а не мигнуло
  labels: { severity: warning }
  annotations:
    summary: "Доля 5xx выше 5%"
```

1. **inactive**

   Условие ложно

2. **pending**

   Условие истинно, ждём `for`

3. **firing**

   Алерт сработал

Принципы хороших алертов

По симптомам

Ошибки, латентность, недоступность — то, что видит пользователь. Не «CPU 70%»

Каждый требует действия

Не нужно реагировать — это шум. Удалите

`for` против всплесков

Кратковременный пик не должен будить людей

Типичные ошибки

- `rate()` по gauge или счётчик без `rate()`
- Высокая кардинальность лейблов
- Алерты без `for`: к шуму привыкают
- Дашборды «накликаны» руками, не в git
- Мониторинг в одном compose-проекте с сервисом

Проверьте себя

1. Почему для счётчика нужен `rate()`?
2. Почему алертим на p95, а не на среднее?
3. Чем pending отличается от firing?
4. Что увидит только cAdvisor, а не `/metrics` сервиса?
5. Рост 409 на `ChangeStatus` — проблема сервиса или клиентов?



Практика

Лаба 7. Мониторинг сервиса

Подключаем Prometheus и Grafana и смотрим на сбои из лабы 6 глазами мониторинга

- Поднимаем стек мониторинга рядом с сервисом
- Находим свои контроллеры и методы в метриках
- Пишем PromQL и добавляем панель на дашборд
- Вызываем сбои и смотрим, как срабатывают алерты