

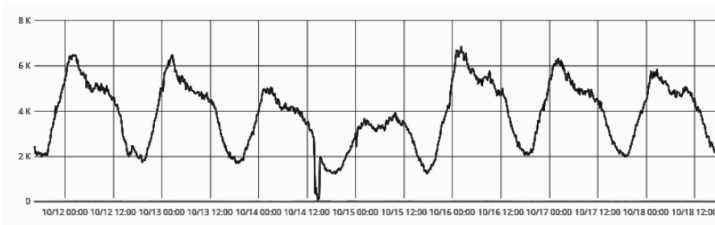
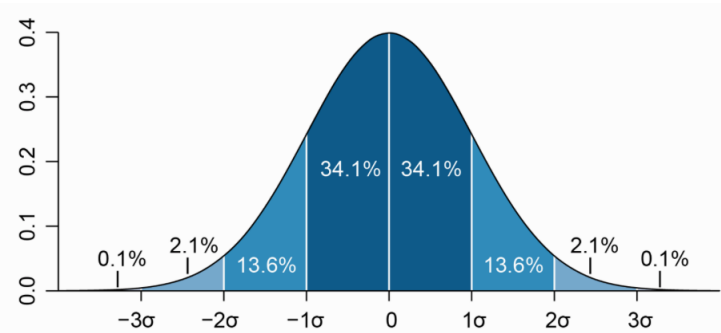
Мониторинг приложения и инфраструктуры. Визуализация и анализ результатов мониторинга. Алертинг

Не забудь включить запись!

План

- Мониторинг инфраструктуры, приложения, бизнес-логики
- Как выбрать что собирать и анализировать?
- Агрегация и визуализация метрик
- Язык запросов PromQL
- Grafana
- Алертинг, on-call, инциденты

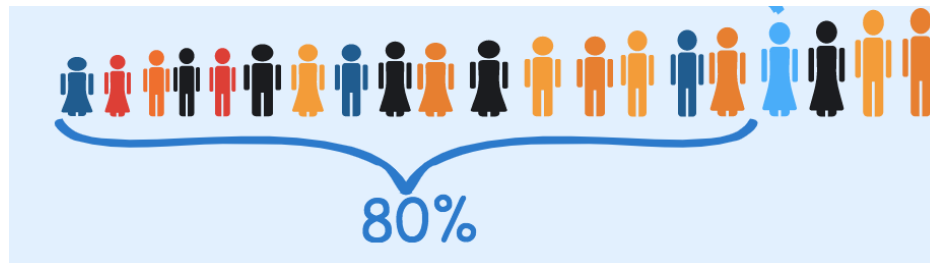
Но сначала немного статистики



- Среднее (Mean or Average)
- Median (медиана)
- Seasonality (повторяемость)
- Standard Deviation (стандартное отклонение)
- Процентили, квантили (Q1 - 25%, Q2 - 50%, Q3 - 75%)

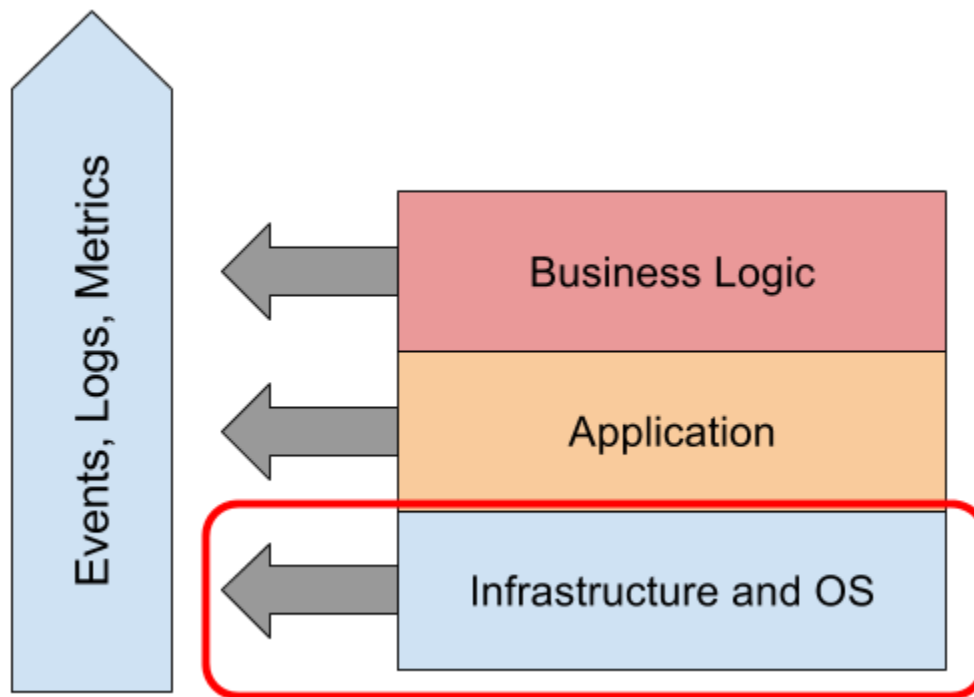
Процентиль, пример

- Число в наборе значений
- Все числа в наборе меньше процентиля, попадают в границы заданного процента значений от всего числа значений в наборе



В классе 20 учеников. Валя занимает 4-е место по росту в классе. Тогда рост Вали (180 см) является 80-м перцентилем. Это означает, что 80 % учеников имеют рост менее 180 см.

Мониторинг инфраструктуры



Метрики хоста

- CPU
- Memory
- Processes
- Disk
- Network
- и т.д.

Метрики хоста: Чем собирать?

Агенты мониторинга системы мониторинга:

- Zabbix-Agent
- Prometheus


Более универсальные инструменты с плагинами:

- `collectd`
- `Telegraf`
- `NetData`

Сервисы платформы:

- Stackdriver (GCP)
- CloudWatch (AWS)

Метрики Docker-контейнеров

- CPU
- Memory
- Network
- Block I/O
-  Docker Daemon

Метрики Docker-контейнеров: чем собирать?

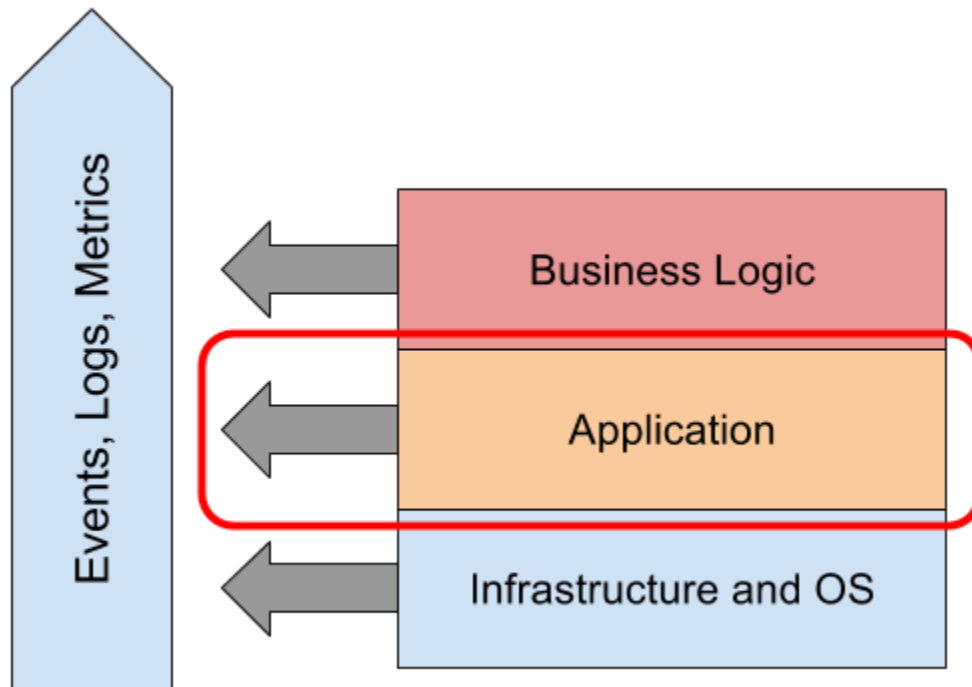
- команда `docker stats`
- cAdvisor
- NetData
- ...



Метрики сервисов

- БД, очереди
- Load balancer
- Сервер приложения
- Сторонние сервисы
- **Все, от чего зависит стабильность работы вашего продукта**

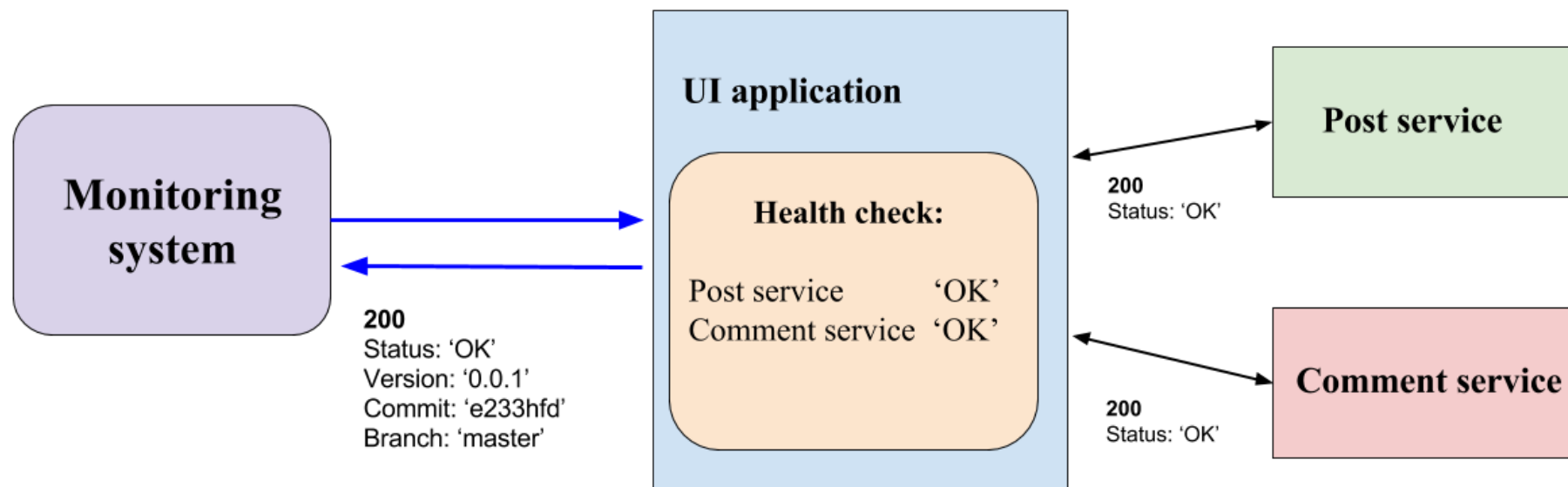
Мониторинг приложения



Health check

- Проверка с целью убедиться, что наше приложение доступно и полноценно работает
- Возврат кода 200 на странице приложения не означает, что оно работает, как ожидается
- Возврат дополнительной информации о работе приложения

Пример реализации



Пример Reddit app

Создание метрик Prometheus:

```
## Create and register metrics
prometheus = Prometheus::Client.registry

comment_health_gauge = Prometheus::Client::Gauge.new(:comment_health, 'Health status
of Comment service')
comment_health_db_gauge =
Prometheus::Client::Gauge.new(:comment_health_mongo_availability, 'Check if MongoDB
is available to Comment')
comment_count = Prometheus::Client::Counter.new(:comment_count, 'A counter of new
comments')
prometheus.register(comment_health_gauge)
prometheus.register(comment_health_db_gauge)
prometheus.register(comment_count)
```

Пример Reddit app

```
## Schedule healthcheck function
if File.exist?('build_info.txt')
  build_info=File.readlines('build_info.txt')

  scheduler = Rufus::Scheduler.new

  scheduler.every '3s' do
    check = JSON.parse(healthcheck(mongo_host, mongo_port))
    comment_health_gauge.set({ version: check['version'].strip, commit_hash:
build_info[0].strip, branch: build_info[1].strip }, check['status'])
    comment_health_db_gauge.set({ version: check['version'].strip, commit_hash:
build_info[0].strip, branch: build_info[1].strip }, check['dependent_services']
['commentdb'])
  end
end

## Define healthcheck endpoint
get '/healthcheck' do
  healthcheck(mongo_host, mongo_port)
end
```


Пример reddit app

```
# Define Healthcheck function
def healthcheck(mongo_host, mongo_port)
  begin
    commentdb_test = Mongo::Client.new( ["#{mongo_host} :#{mongo_port}"],
server_selection_timeout: 2)
    commentdb_test.database_names
    commentdb_test.close
  rescue
    commentdb_status = 0
  else
    commentdb_status = 1
  end

  status = commentdb_status
  version = File.read('VERSION')
  healthcheck = { status: status,
    dependent_services: { commentdb: commentdb_status },
    version: version }
  healthcheck.to_json
end
```

Метрики приложения

APM (Application performance management) - хороший инструмент, но как правило вы лучше знаете, чем занимается ваше приложение и можете реализовать более релевантные метрики

- Позволяют узнать о состоянии и производительности (performance) кода
- Описываются в самом коде приложения
- Отражают опыт использования приложения конечным пользователем

Примеры метрик:

- время ответа на запросы
- количество неудачных логинов пользователей

Как создавать метрики приложения?

- Нужен простой стандарт для создания метрик - общая библиотека
- Многие системы мониторинга имеют готовые клиентские библиотеки (statsd, Prometheus)

Добавляем время поиска пользователя в метод `show`:

```
def show
  STATSD.time("user.find") do
    @user = User.find(params[:id])
  end
  unless current_user.admin?
    unless @user == current_user
      redirect_to :back, :alert => "Access denied."
    end
  end
end
```

Пример reddit app

На GitHub есть [Ruby-библиотека](#) для создания и экспорта метрик Prometheus.

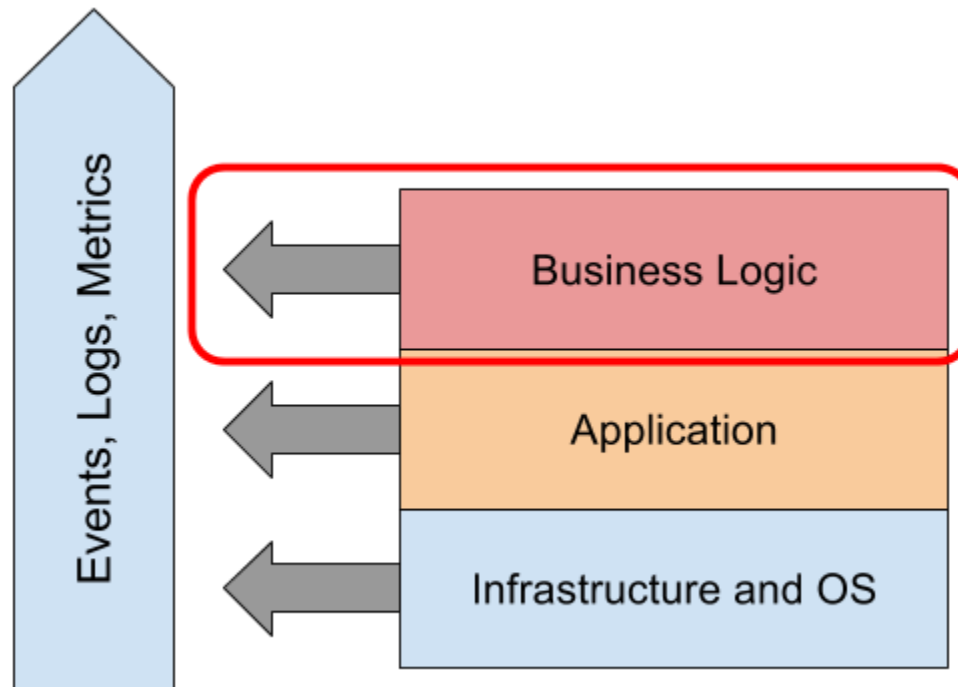
Реализация простого счетчика:

```
require 'prometheus/client'

prometheus = Prometheus::Client.registry
# create a new counter metric
http_requests = Prometheus::Client::Counter.new(:http_requests, 'A counter of HTTP
requests made')
prometheus.register(http_requests)

# start using the counter
http_requests.increment
```

Мониторинг бизнес-логики



Бизнес-метрики

- Выступают средством проверки бизнес-идей
- Индикатор успеха приложения среди пользователей
- Связывают бизнес-KPI и технические метрики

Примеры метрик:

- количество регистраций за последний месяц
- количество продаж
- значение средней покупки

Пример

```
import prometheus_client

POST_COUNT = prometheus_client.Counter('post_count', 'A counter of new posts')

@app.route("/add_post", methods=['POST'])

def add_post():
    title = request.values.get("title")
    link = request.values.get("link")
    created_at = request.values.get("created_at")
    mongo_db.insert({"title": title, "link": link, "created_at": created_at,
"votes": 0})
    POST_COUNT.inc()
    return 'OK'
```

Flashback предыдущей лекции

Метрики `node_exporter`:

- `node_load1` - тип gauge (шкала)
- `node_cpu` - тип counter (счетчик)

Соответственно, когда растет нагрузка, `node_load1` отображает ее увеличение, а для `node_cpu` необходимо использовать функции, чтобы показать **как изменяется метрика**, например:

```
100 - (avg by (host) (irate(node_cpu{mode="idle"}[5m])) * 100)
```

Health check рассмотрим на примере сервиса UI:

- Сам обработчик
- Реализация
- Переменные окружения

Как выбрать что собирать и анализировать?

- Сбор полного набора метрик для сервиса
- Alerting—для настройки оповещений о проблемах
- Troubleshooting—диагностика проблем
- Tuning & Capacity Planning—оптимизация и планирование мощностей

USE-Method

USE-метод от Brendan Gregg:

Больше подходит для выбора инфраструктурных метрик:

- Utilization (использование), например загрузка диска
- Saturation (насыщение), например очередь диска
- Errors (ошибки), например ошибки I/O диска

RED-метод

RED

Больше подходит для выбора метрик приложений и сервисов:

- Rate - запросы в секунду
- Errors - ошибок в секунду
- Duration - время на каждый запрос

Four Golden Signals от Google

Four golden signals - принцип выбора метрик, описанный в книге Site Reliability Engineering от Google:

- Latency - время ответа
- Traffic - частота запросов
- Errors (ошибки) - частота ошибок
- Saturation (насыщение) - насколько утилизирован ресурс

Агрегация и визуализация метрик

Агрегация - это процесс группировки значений различных временных рядов в один.

Агрегация, как правило, использует какую-то функцию для получения результирующих значений. (Напр. `sum`, `min`, `max`, `avg`, ...)

Используется для downsampling (прореживания) данных и построения сводного графика метрик.

Зачем нужна визуализация?

- Наблюдение изменений становится проще
- Позволяет отследить тенденции работы системы
- Анализ

Пример визуализации



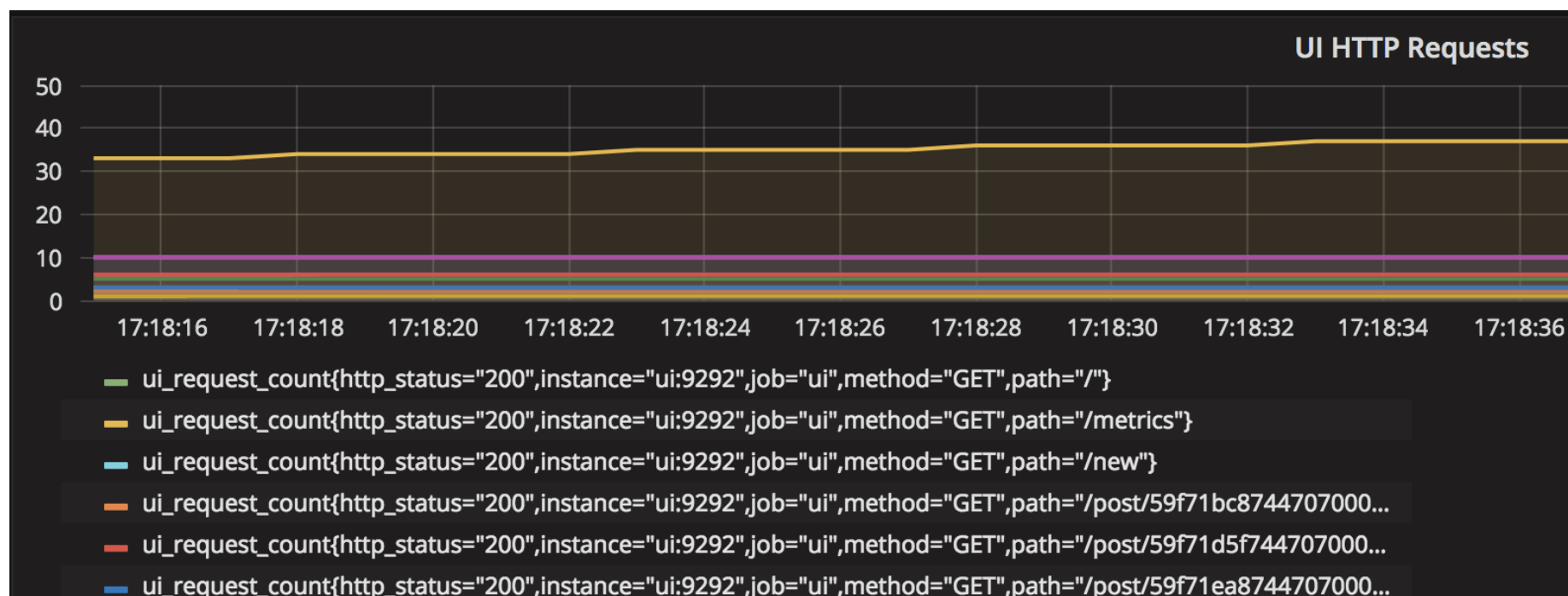
Язык запросов PromQL

PromQL - язык запроса данных, реализованный в Prometheus. Запросы данных в PromQL состоят из:

- **Literals** (литералы) - числа, строки
- **Time series Selectors** - выборка вектора значений метрики из временного ряда за определенный момент или за промежуток времени. Например:
`node_cpu`, `node_cpu{mode='idle'}`, `node_cpu{mode='idle'}`
`offset 5m`, `node_cpu{mode='idle'}[1m]`
- **Operators** (операторы) - различные операторы: арифметические, сравнения, работы над векторами и агрегации
- **Functions** (функции) - большой список функций, которые можно применять к вектору временного ряда

Time series selector

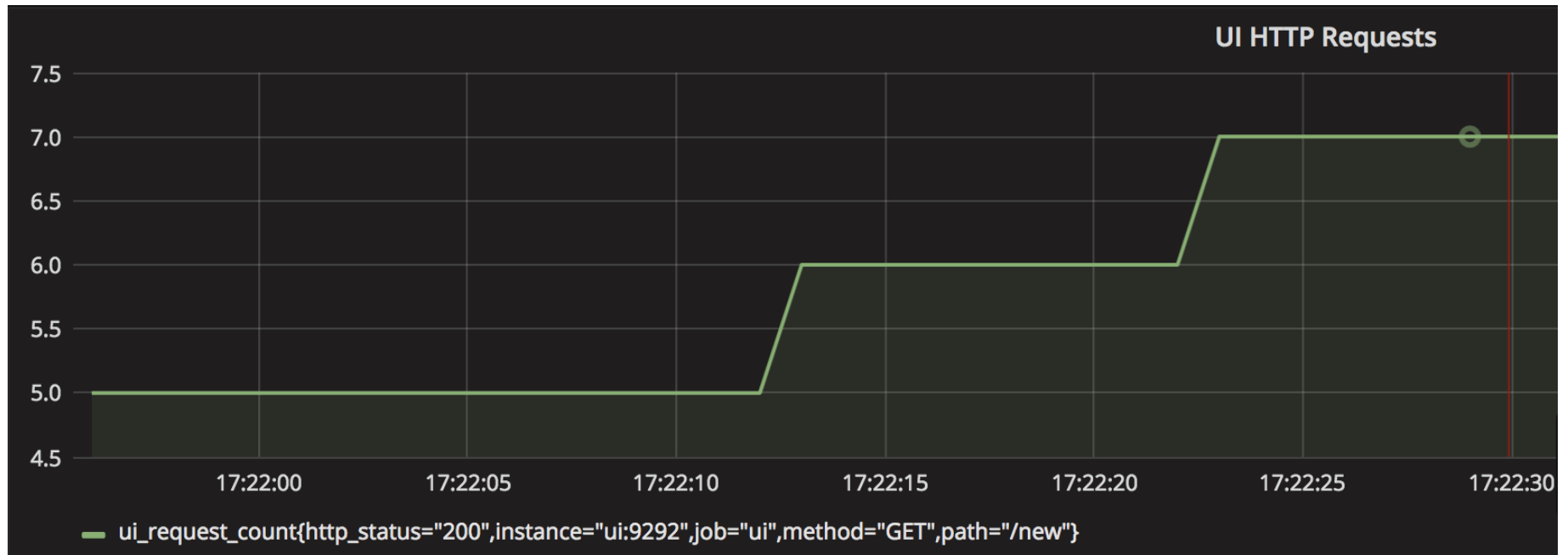
Отообразим метрику `ui_request_count` :



Time series selector

Уточняем метрику, используя лейблы:

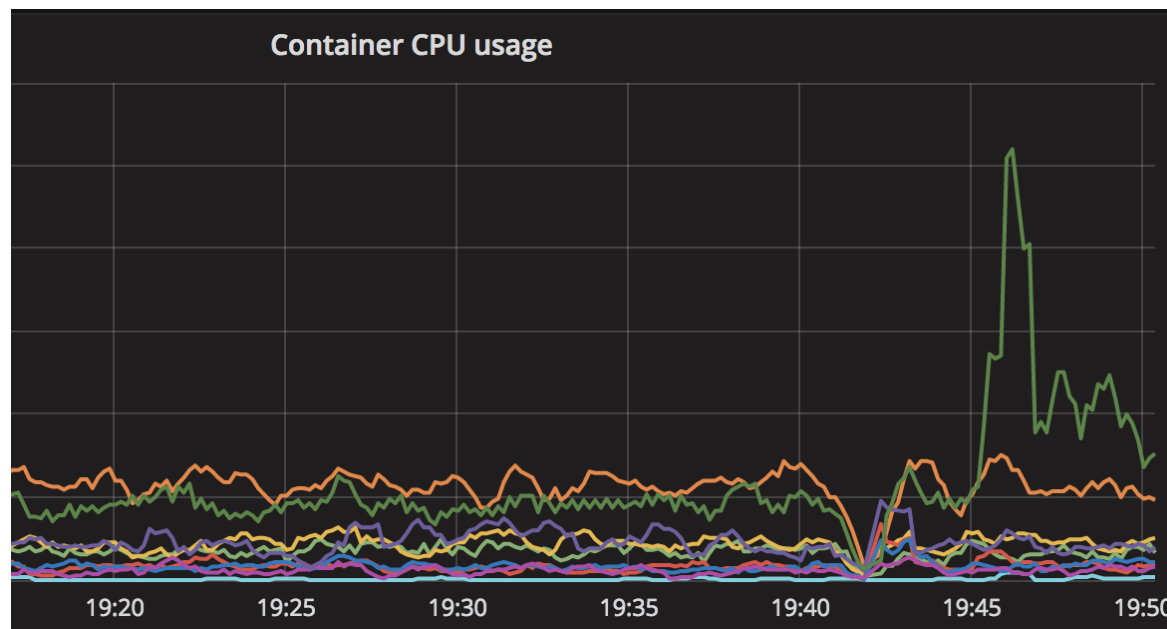
```
ui_request_count{method="GET", path="/new"}
```



Range vector selector

Результат запроса:

```
rate(container_cpu_user_seconds_total{image!=""}[1m])
```



Grafana

- Open source инструмент для построения дашбордов систем мониторинга
- Поддерживает получение данных из Graphite, Elasticsearch, OpenTSDB, Prometheus и InfluxDB и баз SQL
- Поддерживает внешние плагины для интеграции с другими системами мониторинга
- Рендеринг графиков происходит на стороне клиента, поэтому Grafana практически не создает нагрузку на сервер

Grafana и Prometheus

Grafana штатно поддерживает получение данных из Prometheus.

Реализованы следующие сущности:

- Data source (для получения данных)
- Query editor (редактор запросов) поддерживает синтаксис PromQL

Конфигурация

Grafana предоставляет **API**. Таким образом, мы можем добавить в описание инфраструктуры:

- Добавление/удаление источников данных
- Управление дашбордами в Grafana
- Управление пользователями и организациями
- Начиная с версии 5.0 появилась возможность задавать **datasources** и **дашборды в виде кода**

Дашборды

- Дашборды в Grafana реализованы в виде `*.json`-файлов.
- С помощью библиотеки `grafanalib` их можно описать в виде кода на Python
- Есть возможность их импортировать/экспортировать
- Начиная с версии 4.0 поддерживается версионирование дашбордов при изменении с возможностью отката
- При желании, свои дашборды можно опубликовать на `сервисе дашбордов`

Сервис дашбордов

На сайте Grafana доступны **сотни готовых дашбордов**. Доступен поиск и фильтрация по различным критериям.

Dashboards

Official & community built dashboards

Filter by:

Data Source

Prometheus

Panel Type

All

Category

Docker

Collector

nodeExporter

Search within this list

docker

Sort By

Average Rating



Docker Swarm & Container Overview by Basilio Vera

Overview of the most important Docker swarm and container metrics....

ELASTICSEARCH, PROMETHEUS, NODEEXPORTER

Downloads: 7372

Reviews: 1



Docker and system monitoring by Thibaut Mottet

A simple overview of the most important Docker host and container metrics. (cAdvisor/Prometheus)

PROMETHEUS, NODEEXPORTER

Downloads: 18989

Reviews: 1



Docker Host & Container Overview by uschtwill

A simple overview of the most important Docker host and container metrics. (cAdvisor/Prometheus)

PROMETHEUS, NODEEXPORTER

Downloads: 11156

Reviews: 1



Docker monitoring by philicious

Docker monitoring with Prometheus and cAdvisor

PROMETHEUS, NODEEXPORTER

Downloads: 8397

Reviews: 2



Алертинг, Alert

- От англ. "тревога, предупреждение"
- Оповещение о событии или состоянии наблюдаемых систем, ставящих под угрозу надежность их работы
 - "Мы упали и лежим"
 - "Мы скоро упадем!"

Уровни важности событий (severity)

- 🔥 Critical
- ⚠️ Warning
- ⓘ Info
- 🙌 могут быть и другие (или не быть и этих)

Пороговые значения (thresholds)

- Границы нормальных значений метрик
- При выходе за границы осуществляется отправка уведомлений
- В качестве порогов можно использовать SLA (service-level agreement) / SLO (service-level objectives) + SLI (service-level-indicator)

Пример (Prometheus < 2.0)

```
ALERT InstanceDown
  IF up == 0
  FOR 5m
  LABELS { severity = "page" }
  ANNOTATIONS {
    summary = "Instance {{ $labels.instance }} down",
    description = "{{ $labels.instance }} of job {{ $labels.job }} has been down for more than 5 minutes.",
  }
```

Пример (Prometheus >= 2.0)

```
- alert: InstanceDown
  expr: up == 0
  for: 5m
  labels:
    severity: page
  annotations:
    description: '{{ $labels.instance }} of job {{ $labels.job }} has been down
for more than 5 minutes'
    summary: 'Instance {{ $labels.instance }} down'
```

Принципы алертинга

- Реагировать нужно на изменения во временном промежутке, а не на единичные значения
- Если есть возможность решить проблему без вмешательства человека, это нужно сделать (self-healing, auto-remediation)
- Должен быть организован механизм реагирования на алерты (on-call дежурства, написаны runbooks, есть четкий механизм эскалации)

Куда отправлять уведомления?

- командный чат, если уведомление не критичное
- SaaS сервисы уведомлений (VictorOps, OpsGenie, PagerDuty, Amixr), т.н. Pagers для критичных уведомлений
- При невозможности использовать SaaS приходится самостоятельно реализовывать оповещение звонками и смс
- E-Mail (плохая идея в 2020 году)

Пример PagerDuty

Принципы реагирования на инциденты

- On-call инженеры
- Эскалация инцидентов
- Postmortems:
 - Процедура разбора и анализа инцидентов
 - Должны документироваться вместе с контекстом
 - Принцип blameless

Руководства от PagerDuty **про on-call** и **инцидент менеджмент**

Дежурства (on-call)

Дежурства это всегда тяжело, вот простые способы облегчить жизнь дежурных:

- Искоренять "ложные" алерты
- Уменьшить число пожаров:
 - Частью работы дежурных должны быть задачи по повышению отказоустойчивости инфраструктуры (очевидно, когда не заняты устранением проблем)
 - В планы команды на неделю (или спринты) *необходимо* явно включать задачи по улучшению стабильности инфраструктуры

Дежурства (on-call)

- Продумать ротацию дежурств
- Пользоваться разницей во временных зонах
- Назначать "резервного дежурного"
- Прописывать пути эскалации
- Обеспечить достойную компенсацию

Суммируя, современный мониторинг...

- Состоит из многих компонентов, каждый из которых хорошо реализует свою часть.
- Постоянно улучшается
- Собирает не только инфраструктурные метрики, но и метрики приложения и бизнес-метрики
- Им пользуются не только системные администраторы, но и разработчики, тестировщики и менеджеры (бизнес)

Суммируя, современный мониторинг...

- Конфигурация хранится в коде. Ручные действия минимизированы. Новые сервисы/дашборды/алерты добавляются через коммиты в репозиторий
- Новые хосты и сервисы добавляются и удаляются автоматически через сервис обнаружения
- Оповещения настроены на критичные показатели, есть on-call дежурства, написаны runbooks, есть четкий механизм эскалации

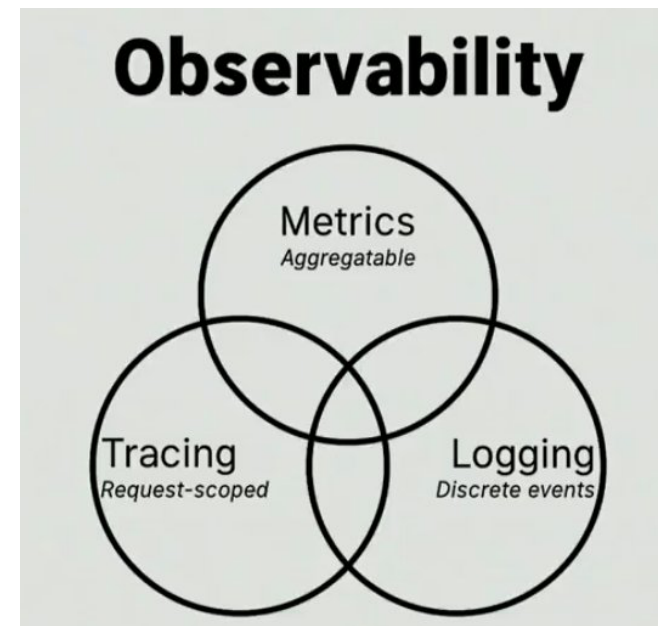
И в конце...

Сбор данных дешев, но отсутствие их в случае необходимости может обойтись дорого. Поэтому нужно обеспечить сбор всех полезных данных, которые разумно собирать.

"monitoring 101" - [Блог Datadog](#)

Что дальше?

- Что такое Observability (брошюра от Honeycomb)
- Observability Manifesto (они же)



Полезные ссылки

1. книга Brendan Gregg. [Systems Performance: Enterprise and the Cloud](#)
2. заметка про [USE и RED](#)
3. статья про [SRE Golden Signals](#)
4. статья от DataDog [Monitoring 101: Собираем правильные данные](#)
5. Алексей Иванов, Dropbox. [Практический опыт мониторинга распределённых систем. Слайды.](#)
6. Владимир Рычев, Google. [Как я научился не волноваться и полюбил пейджер](#) (про концепцию SRE, SLA/SLO/SLI и др.). [Слайды.](#)

Полезные ссылки

1. Владимир Иванов, Booking. [Graphite в booking.com](#)
2. Конференции и др.: [Monitorama](#), [FOSDEM](#), [Velocity](#), [Uptime community](#)
3. Примеры [Runbook](#) от [GitLab](#)