# Меня хорошо слышно && видно?

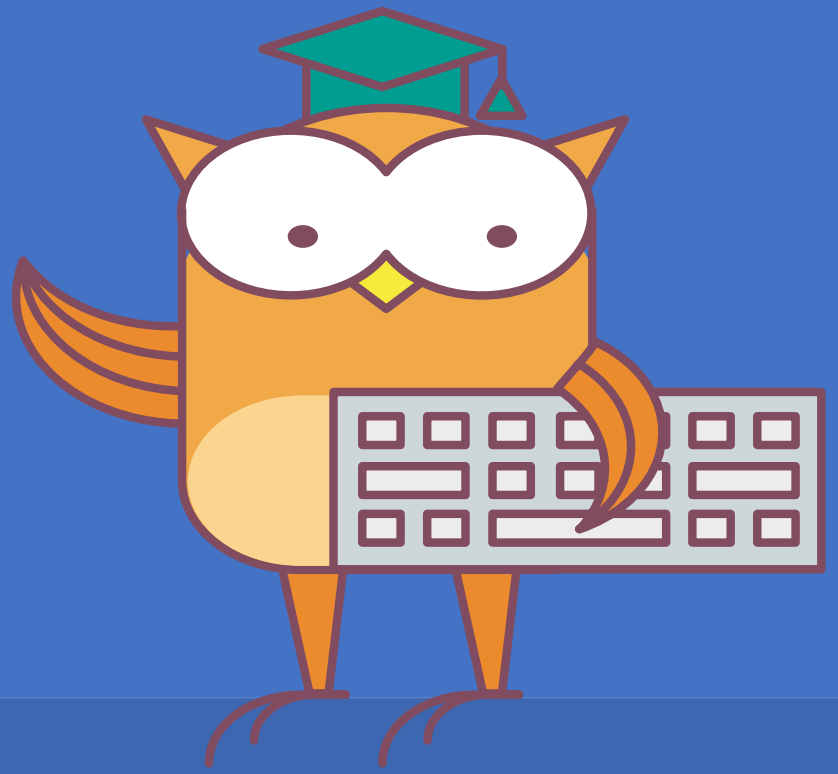
Напишите в чат, если есть проблемы!

Ставьте + если все хорошо

OTUS

# Мониторинг и алертинг

Архитектор высоких нагрузок



Щетинников Стас

# Карта вебинара

- Мониторинг и алертинг
- USE, RED и Four Golden Signals
- SLI, SLO, SLA и бюджет на ошибки
- Паттерны для сбора метрик
- Prometheus
- Grafana
- Sentry
- Управление инцидентами

# Зачем собирать метрики?

Метрики помогают принимать решения:

- Надо ли добавить на сервер оперативной памяти?
- Сколько еще времени мы можем жить на текущих мощностях?
- Какой сервис и в каком месте нужно оптимизировать прежде всего?

# Зачем собирать метрики?

Метрики помогают среагировать в случае непредвиденной ситуации

- Сервис начал долго отвечать
- Очередь переполнился

И т.д

# Мониторинг и алертинг

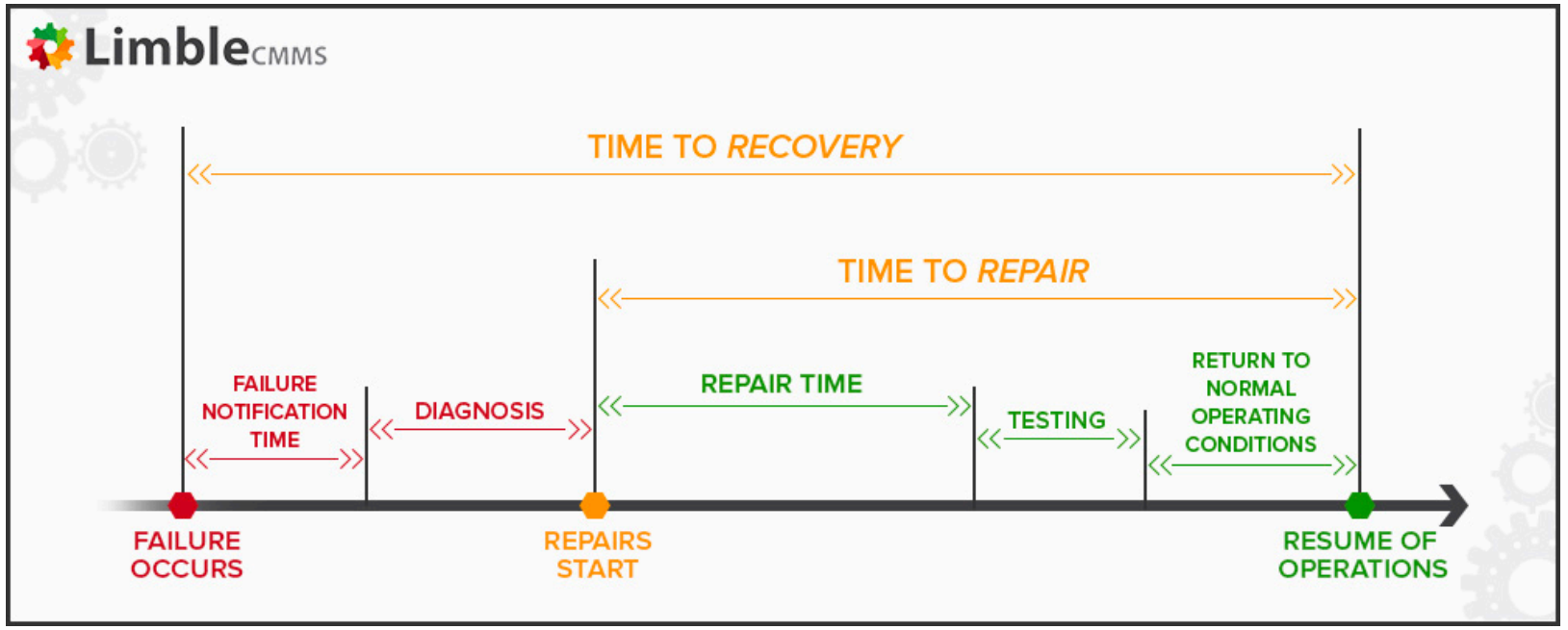
**Мониторинг** - это набор инструментов и сервисов, которые позволяют собирать, хранить, агрегировать и визуализировать метрики

**Алертинг –** это набор инструментов, которые позволяют узнавать о каких-то важных изменениях в поведении системы (с помощью метрик)

# Метрики и доступность

**Мониторинг и алертинг** напрямую влияют на доступность availability сервисов

- Правильный алертинг всегда сообщит команде о проблемах раньше, чем бизнес или пользователи
- Правильный мониторинг всегда поможет понять, где ошибка и что случилось.



# Какие метрики собирать?

- **Инфраструктурные** – состояние инфраструктуры: физические машины, сеть, системные сервисы: cpu/mem/disk
- **Сервисные** – состояние прикладных сервисов: rt/rps
- **Бизнес** – состояние бизнес процессов: сколько сделали заказов

# USE method

http://www.brendangregg.com/usemethod.html

- **Utilization** – время, которое затрачивает ресурс на выполнение задач
- **Saturation** – насколько ресурс близок к пределу, когда он начинает давать отказы
- **Errors** – количество ошибок

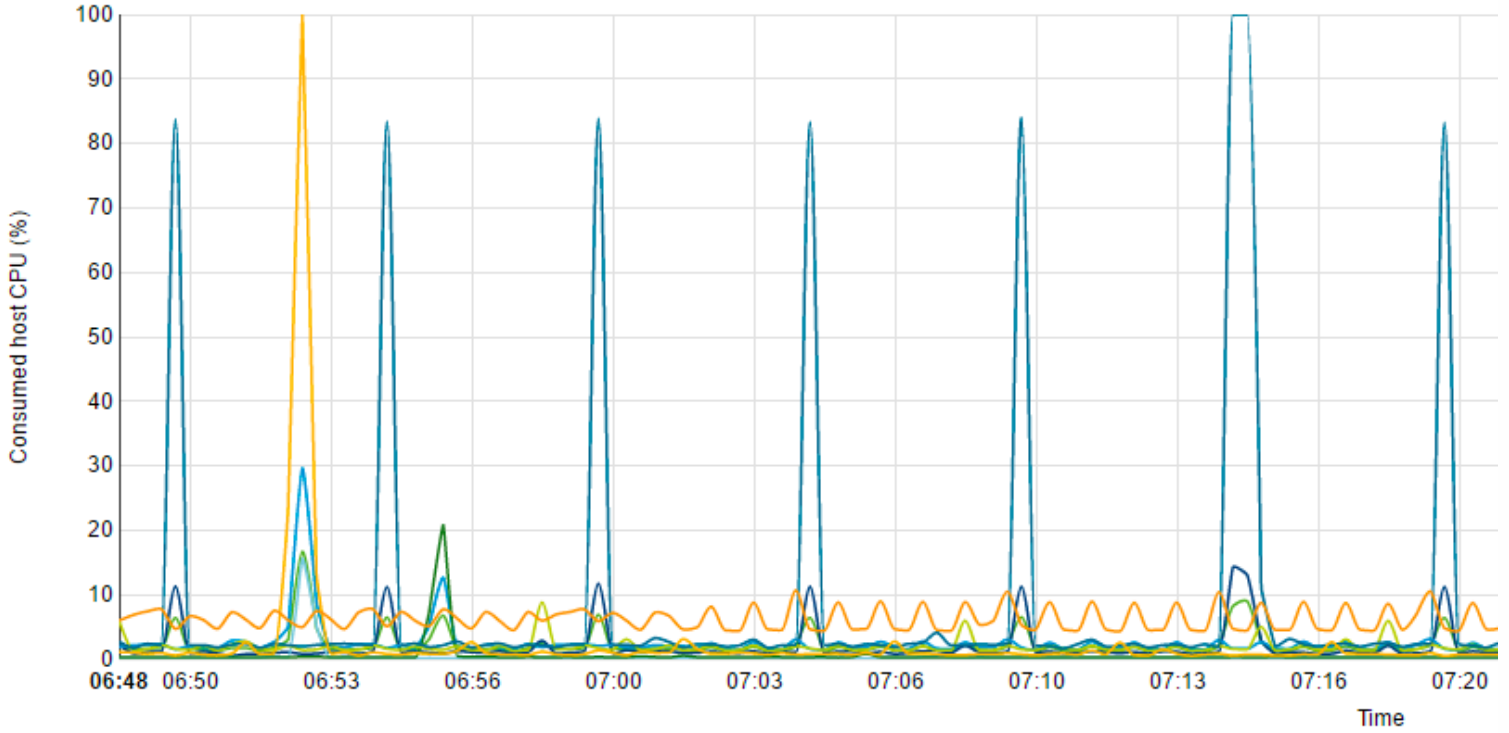
Используется в основном для инфраструктурных метрик: CPU, memory, IO, networking, но ими не ограничивается

# USE method

| Resource | Utilization | Saturation | Errors | Availability |
|---|---|---|---|---|
| Disk IO | % time that device was busy | wait queue length | # device errors | % time writable |
| Memory | % of total memory capacity in use | swap usage | N/A (not usually observable) | N/A |
| Microservice | average % time each request-servicing thread was busy | # enqueued requests | # internal errors such as caught exceptions | % time service is reachable |
| Database | average % time each connection was busy | # enqueued queries | # internal errors, e.g. replication errors | % time database is reachable |

# Saturation vs utilization

**Saturation** может быть высоким, в том время как average **utilization** низким.  Например, в CPU могут кратковременно уходить в 100%, при это большую часть времени оставаясь незагруженным.



# RED method

https://www.youtube.com/watch?v=TJLpYXbnfQ4 – подход озвучил Том Вилки

”RED метод для сервисов, USE для ресурсов”

- **Rate** – количество запросов в секунду
- **Errors** – количество ошибок
- **Duration** – время запроса

# The Four Golden Signals

https://landing.google.com/sre/books/

- **Latency** – время ответа
- **Traffic** - количество запросов
- **Errors** – количество ошибок
- **Saturation** – насколько ресурс близок к отказу

# Latency

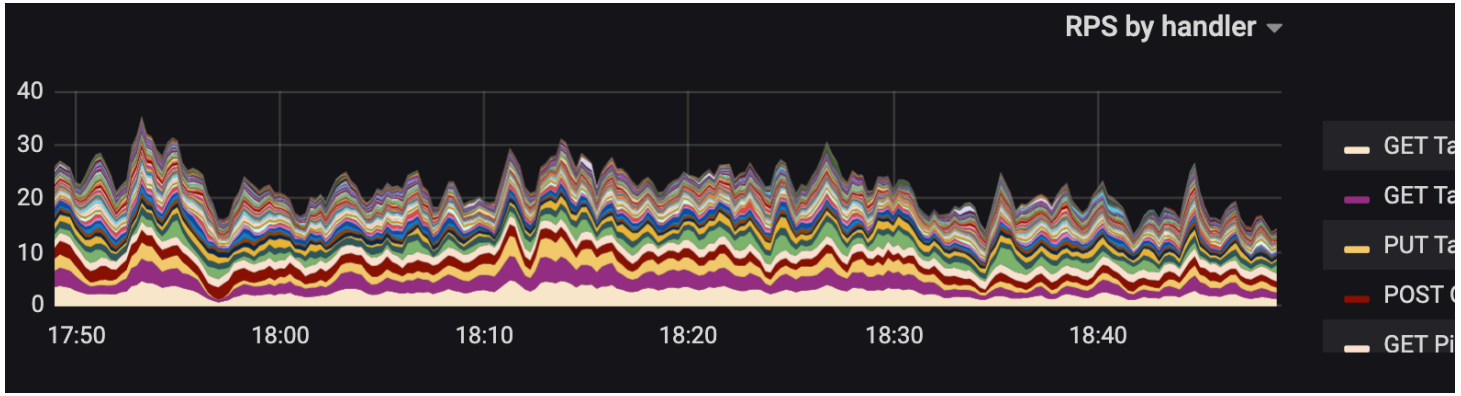
**Latency**, Response Time, Duration – время ответа сервиса

- Среднее (average) время зачастую может быть не очень показательно, потому что выбросы (очень долгие ответы) могут сильно на него влиять.
- 99, 95 и 50 квантиль более показательны и дают лучшее представление о том, как работает сервис
- Время ответа можно считать, как внутри приложения, так и снаружи. И важно понимать, входят ли в среднее время ответа: время ожидания во внутренних очередях, сетевые задержки и т.д.
- Время ответа из разных географических мест

# Traffic

**Traffic**, RPS – количество запросов в секунду

- Для веб-сервисов имеет смысл измерять не только общую производительность, но по конкретным API-методам



# Errors

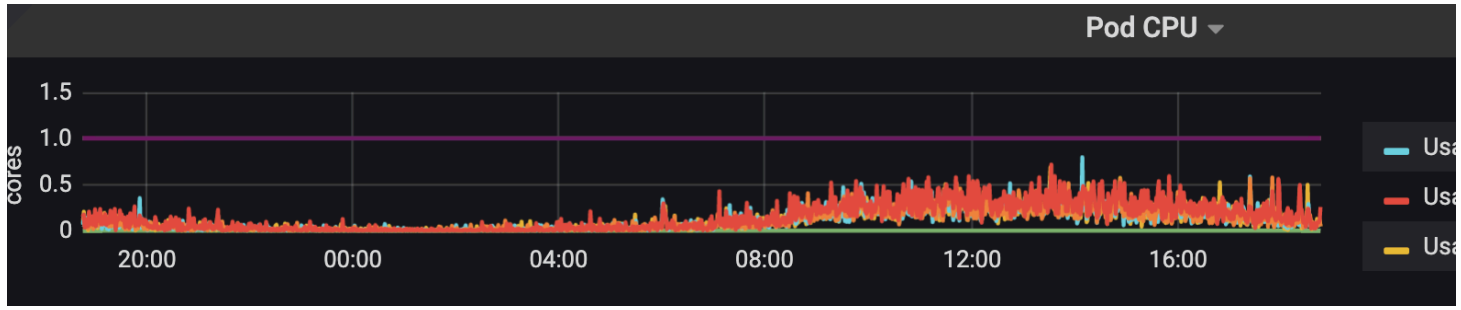
**Errors** – количество ошибок секунду

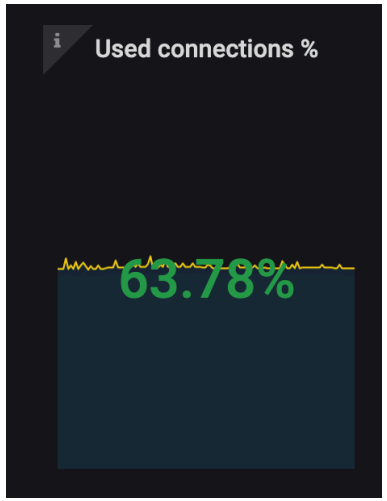
- Для REST веб-сервисов можно считать количество ответов по статус кодам
- Для не REST веб-сервисов нужно отделять ошибочные запросы от не ошибочных на основе протокола внутри
- Для фронтовых ошибок нужно использовать отдельную инструментацию (например, sentry)


# Saturation

**Saturation** – насыщенность, или насколько система близка к своему пределу

- Для CPU bound или IO bound задач можно считать насыщенность по ресурсу
- В сложных системах обычно тяжело сказать, что может оказаться ограничением, поэтому или нагрузочное тестирование, или экспертная оценка на основе предположения о том, что же является ограничением



# SLI, SLO, SLA

Один из способов управления сервисами являются SLI, SLO, SLA.

https://landing.google.com/sre/sre-book/chapters/service-level-objectives/

# SLI

**SLI** – service layer indicator – метрика, которой мы оцениваем работу сервиса.

Примеры:

- Для клиентской системы – availability, latency, throughput
- Процент ошибочных запросов
- Процент запросов, которые исполнились меньше чем 200ms

# SLO

**SLO** – service layer objective – целевые показатели метрики SLI:

lower bound <= SLI <= upper bound

Примеры:
- 99% процентов клиентских запросов должны выполняться меньше 100ms
- Доступность сервиса должна быть 99%
- Error rate должен быть не больше 1%

И т.д.

| Доступность % | Downtime в год | Downtime в месяц | Downtime в неделю | Downtime в день |
|---|---|---|---|---|
| 90% (одна 9) | 36.5 дней | 72 часа | 16.8 часов | 2.4 часа |
| 99% (две 9) | 3.65 дней | 7.20 часов | 1.68 часов | 14.4 минут |
| 99.9% (три 9) | 8.76 часов | 43.8 минут | 10.1 минут | 1.44 минут |
| 99.99% (четыре 9) | 52.56 минут | 4.38 минут | 1.01 минут | 8.66 секунд |
| 99.999% (пять 9) | 5.26 минут | 25.9 секунд | 6.05 секунд | 864.3 миллисекунд |

# SLA

**SLA** – service layer agreement –соглашение о том, какие последствия наступают, если SLO не выполнены.

# Бюджет на SLO

Например, есть SLO – процент удачных запросов 99.9%.

Бюджет на SLO = 100% - 99.9% = 0.1%
У сервиса 1 млрд запросов в месяц, поэтому он может «потратить» 1 миллион ошибок в месяц

https://landing.google.com/sre/workbook/chapters/slo-document/ - пример SLO документа

# На что тратиться бюджет?

Основной источник инцидентов – релизы.
Поэтому пока есть бюджет – можно спокойно релизиться.

Нет бюджета – никаких релизов, пока SLO не приведут в порядок и не проведут необходимых мероприятий.

# Алертинг

**Алертинг** – система предупреждения об ошибках

- Алертов должно быть достаточно, чтобы вовремя среагировать, но не слишком много, чтобы их тяжело было читать
- Если алерт никто не читает, то его можно спокойно убрать
- Следует разделять разные уровни алертов

**Критические** – обязательно нужна реакция человека и его обязательно должны прочитать и что-то сделать (или не сделать)
**Предупреждения** – реакция не обязательно нужна прямо здесь и сейчас

- Простые и понятные алерты лучше, чем со сложной и запутанной логикой
- Если вы о проблеме узнали от бизнеса или клиентов, значит, у вас плохой алертинг

# Self Test

Одной из вариацией алертинга и smoke-тестов являются self-check-и или self-test-ы.

С какой-то периодичностью запускается «скрипт», который проверяет на проде, что выполняются инварианты или ограничения.

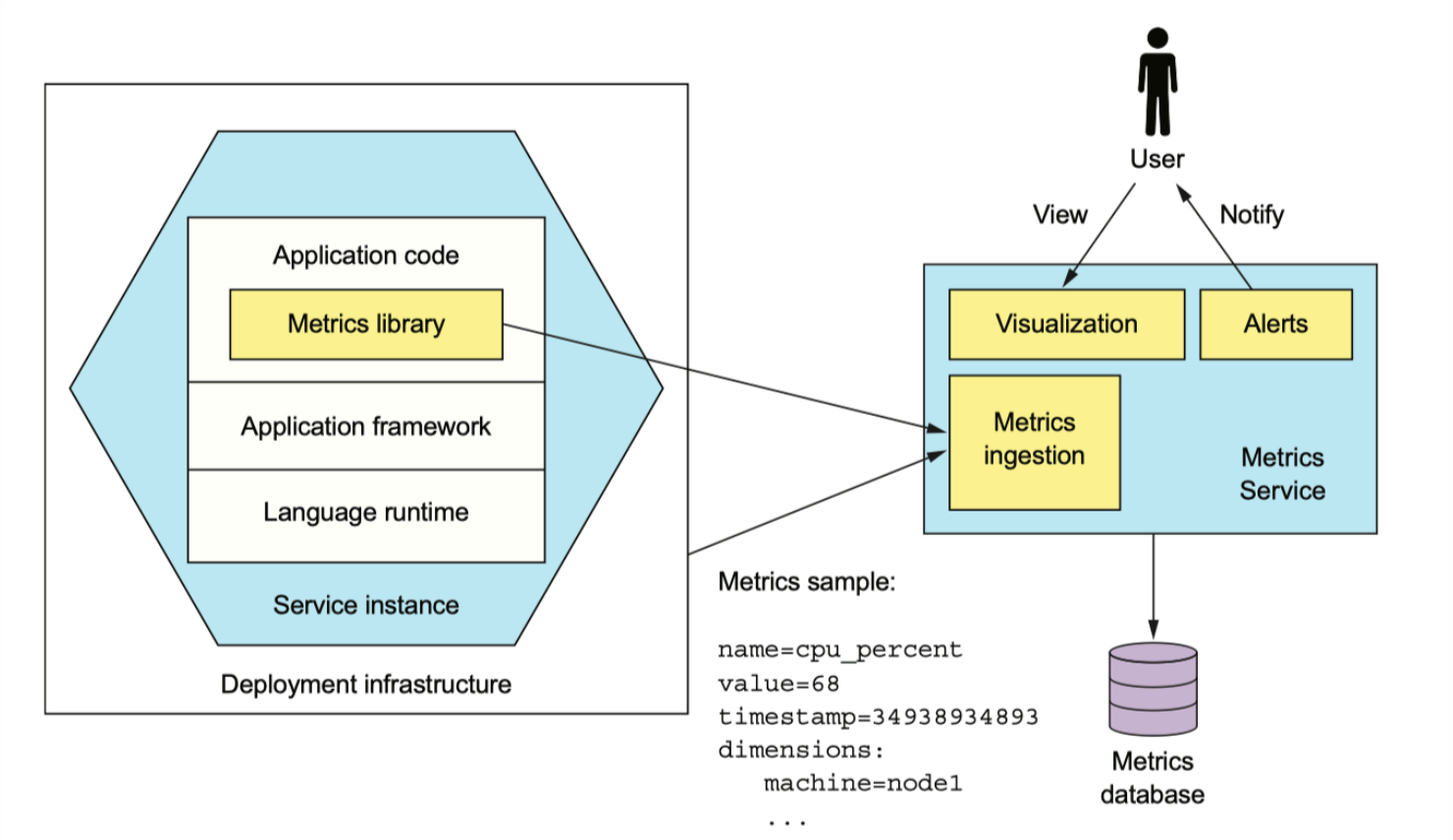
Например,

- Можно периодически ходить в базу и проверять различные constraints: нет заказов без пользователей, нет отрицательных балансов и т.д.
- Можно проверять на проде целые пользовательские сценарии (т.н. синтетическое тестирование) с помощью selenium-а.

# Инструменты

Существует различны системы мониторинга и алертинга.

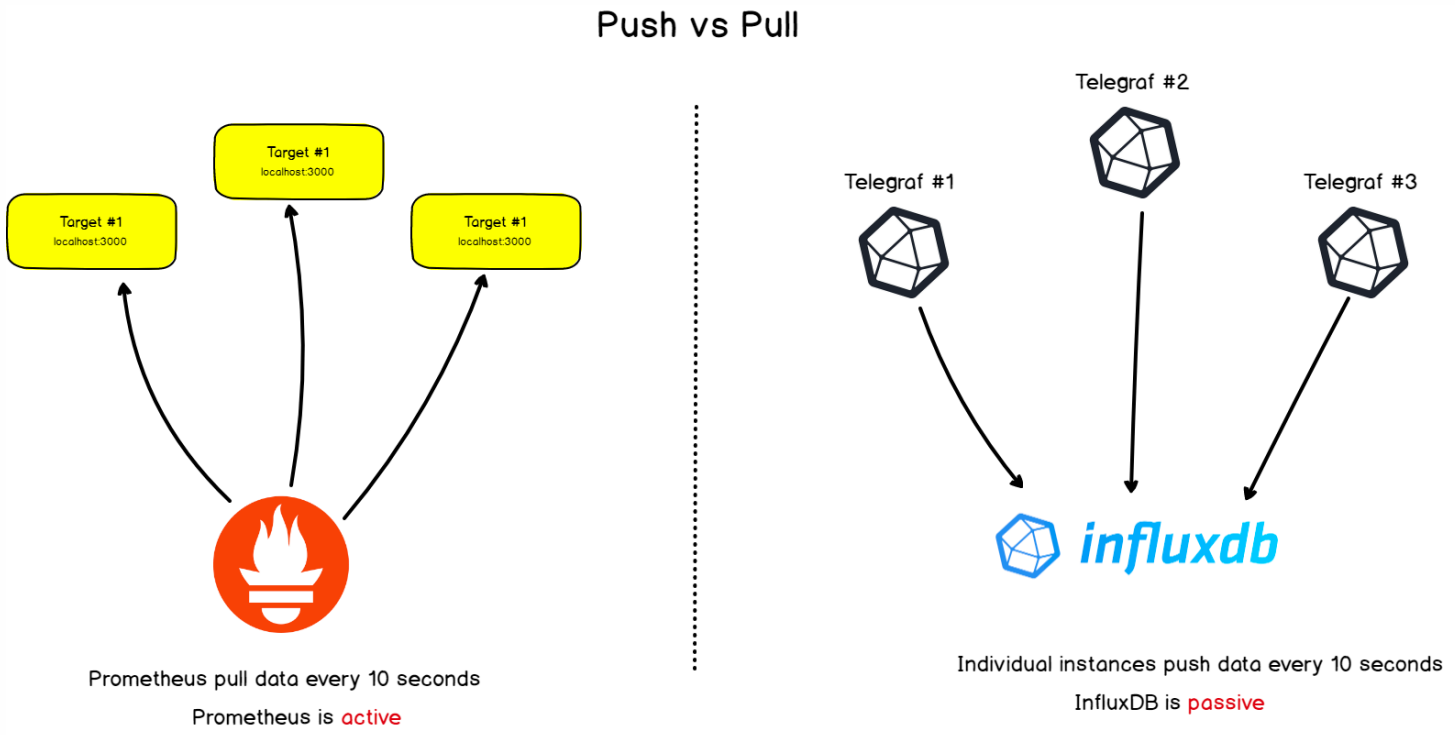
PRTG, Zabbix, Prometheus, Graphite, Grafana, Sentry и т.д.
Различные APM: newrelic, Elastic APM, DynaTrace, DataDog, Solar и т.д.



# Pull vs Push модель сбора метрик

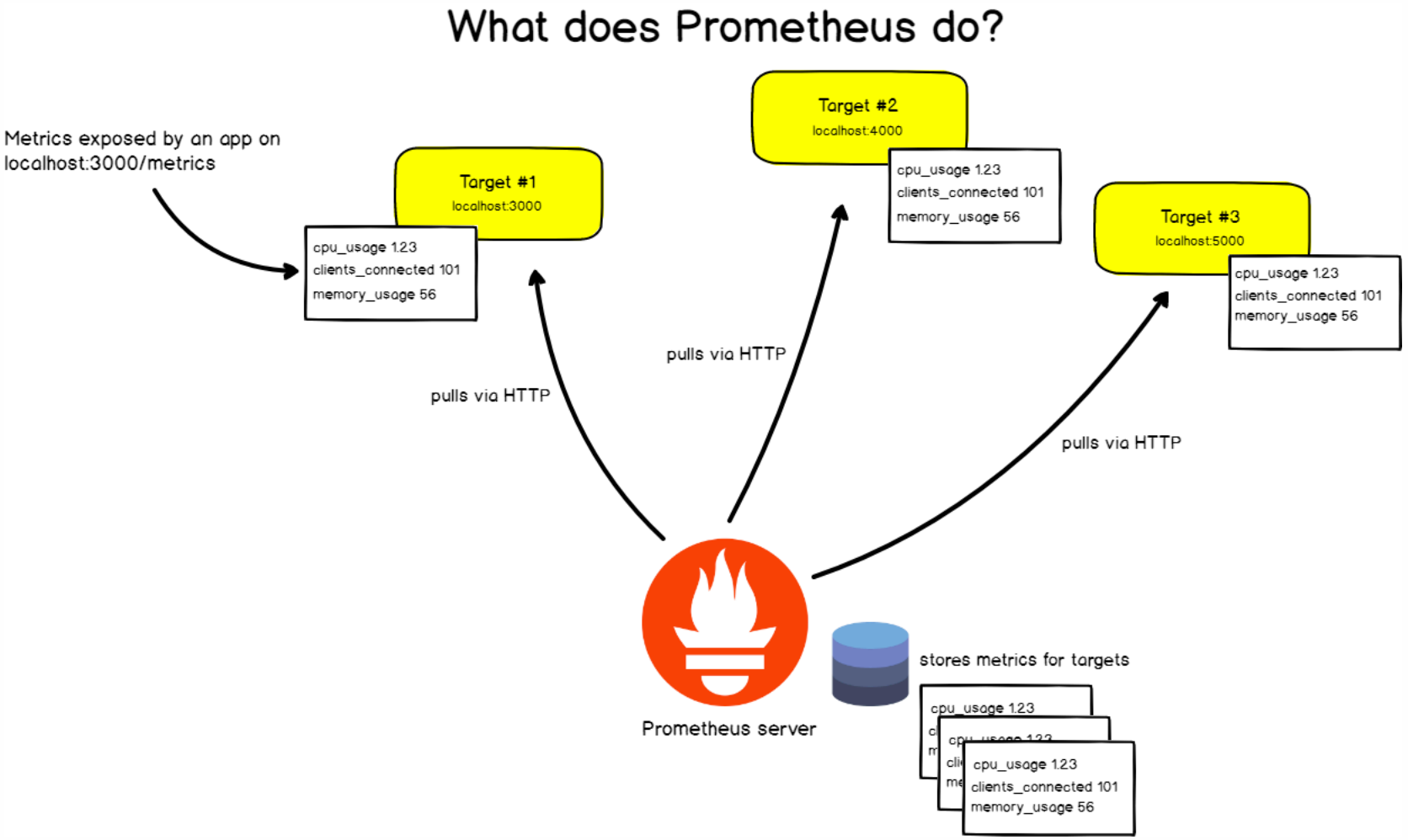
**Push модель**: приложение само ходит в сервис метрик и пушит туда все метрики

**Pull модель**: приложение выставляет урл (обычно /metrics), в котором все метрики, а сервис метрик забирает, и потом отображает.



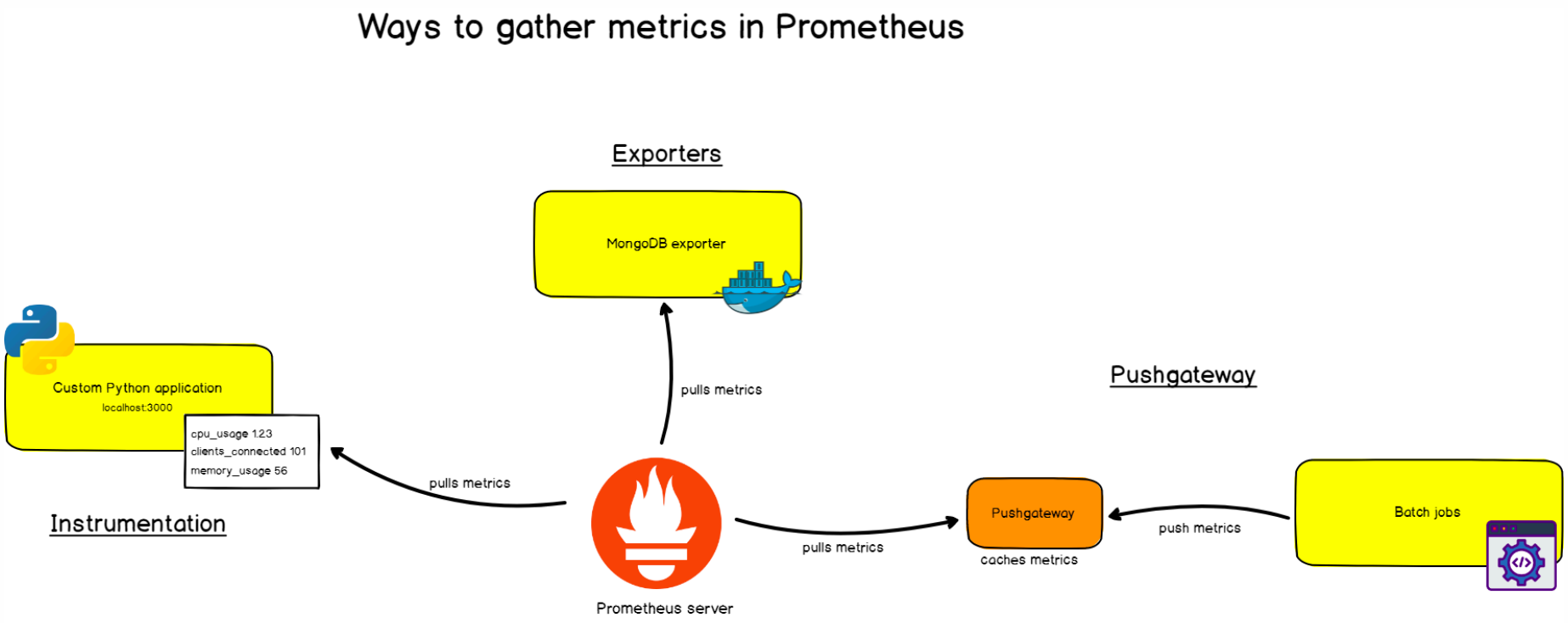
# Prometheus

Прометеус ходит по сервисам, забирает агрегированную статистику и складывает в базу.



# Prometheus

Прометеус может мониторить инфраструктуру с помощью экспортеров



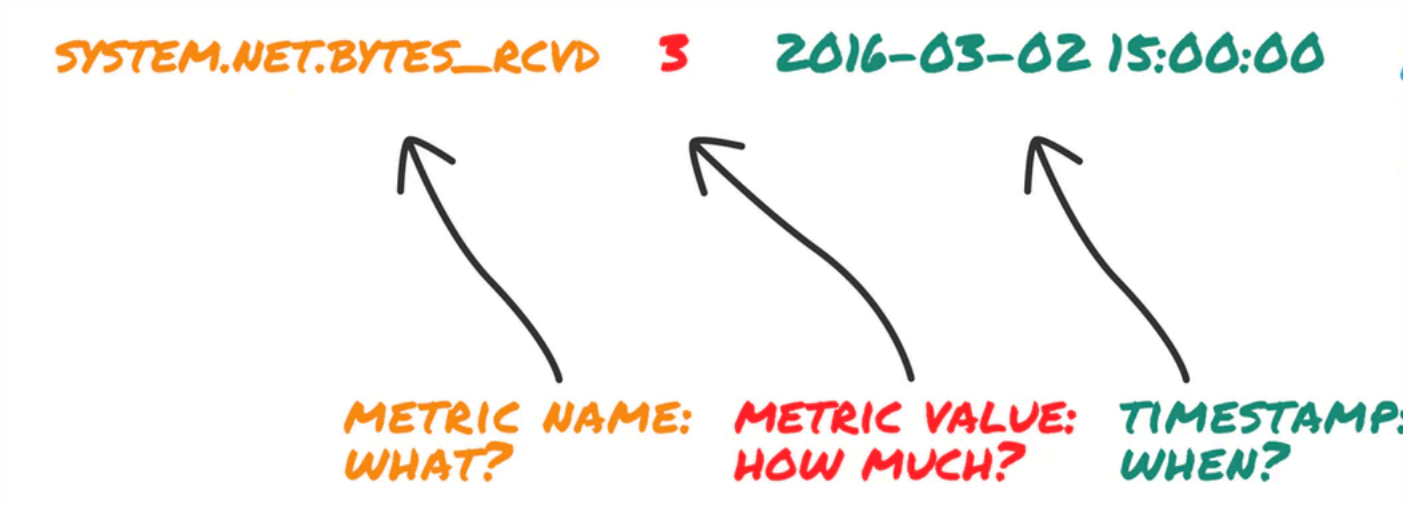
# Prometheus

- Использует TSDB для хранения метрик
- Есть язык запросов PromQL
- Есть свой алертинг на основе правил на языке PromQL
- Используется в Kubernetes

# Dot metrics

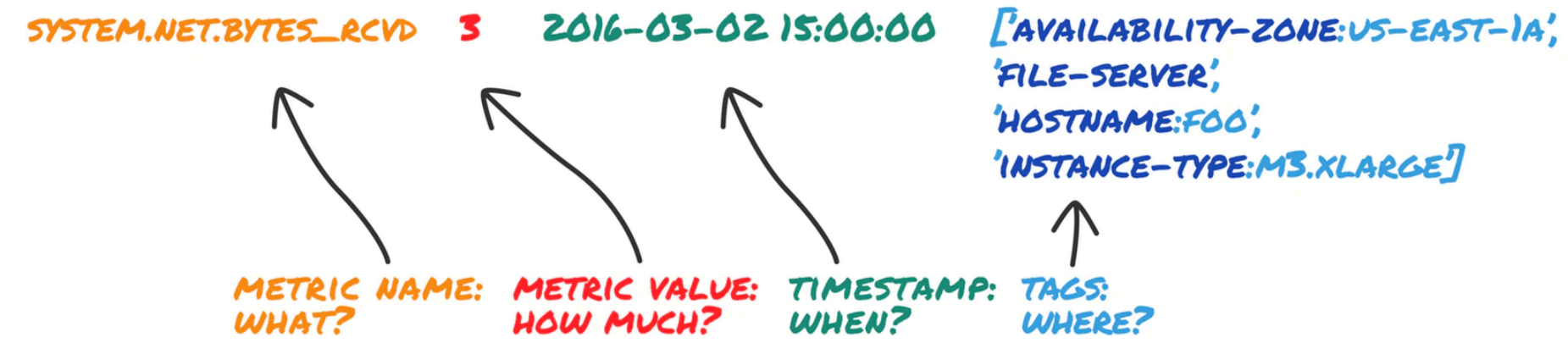
Формат хранения метрик, при котором метрики хранятся в формате «имя значение»:

Например, whisper формат в graphite



# Tagged metrics

Формат хранения метрик, при котором метрики хранятся в формате «имя(тэги) значение»:



# Openmetrics

Prometheus хранит данные в формате Openmetrics

```
http_requests_total{method="post",code="400"}   3    1395066363000
```

# Типы метрик

- **Counter** (счетчик)

Постоянно растущий счетчик

Например, количество запросов

```
# HELP go_memstats_alloc_bytes_total Total number of bytes
# TYPE go_memstats_alloc_bytes_total counter
go_memstats_alloc_bytes_total 3.7156890216e+10
```

- **Gauge** (мера)

Текущее состояние системы

Например, CPU

```
# HELP go_goroutines Number of goroutines that currently exist.
# TYPE go_goroutines gauge
go_goroutines 73
```

- **Histogramm (г**истограмма)

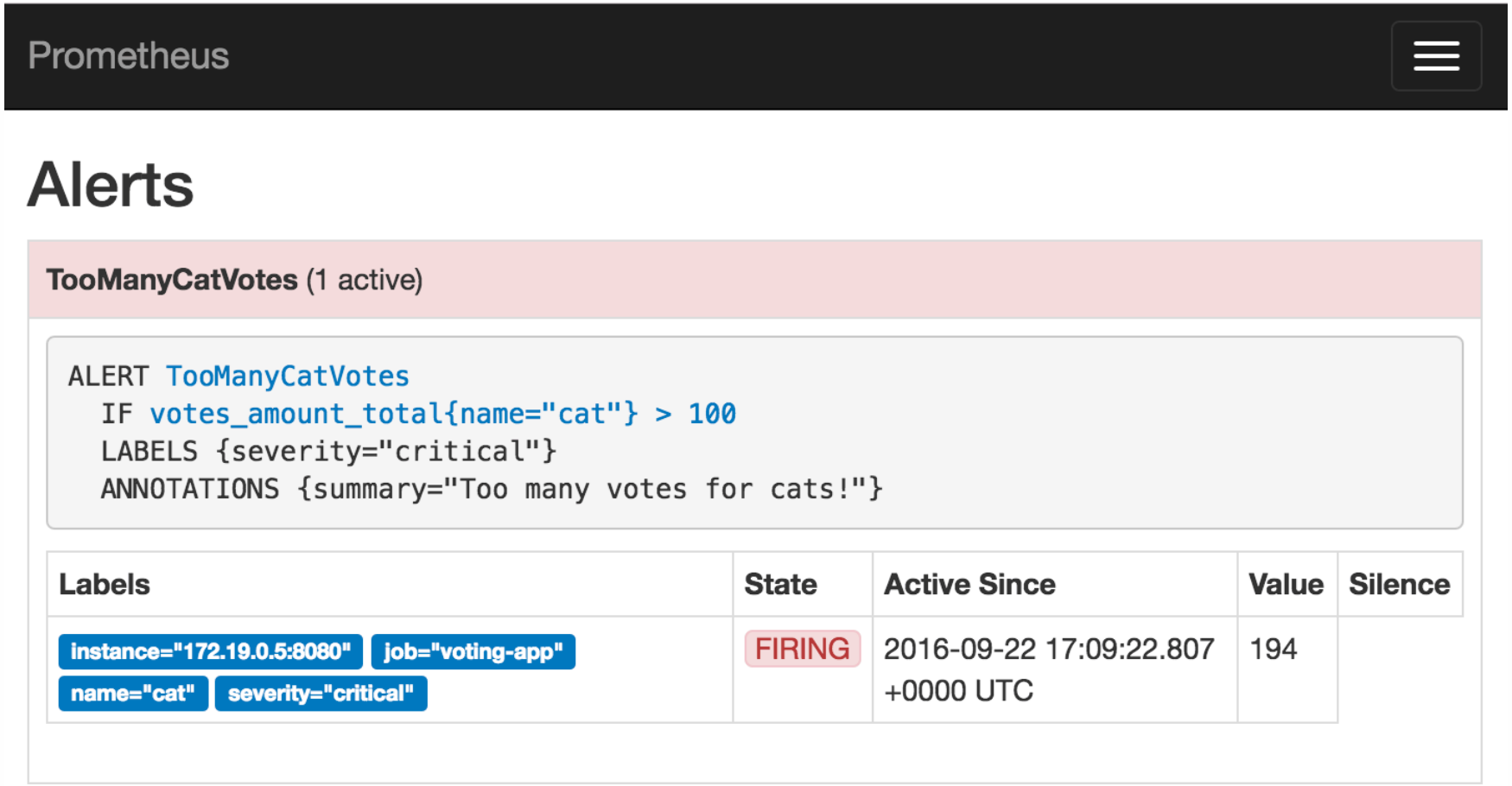
Распределение

Например, распределение времени ответа

```
# HELP http_request_duration_seconds request duration histogram
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.5"} 0
http_request_duration_seconds_bucket{le="1"} 1
http_request_duration_seconds_bucket{le="2"} 2
http_request_duration_seconds_bucket{le="3"} 3
http_request_duration_seconds_bucket{le="5"} 3
http_request_duration_seconds_bucket{le="+Inf"} 3
http_request_duration_seconds_sum 6
http_request_duration_seconds_count 3
```

# Alerting в Prometheus

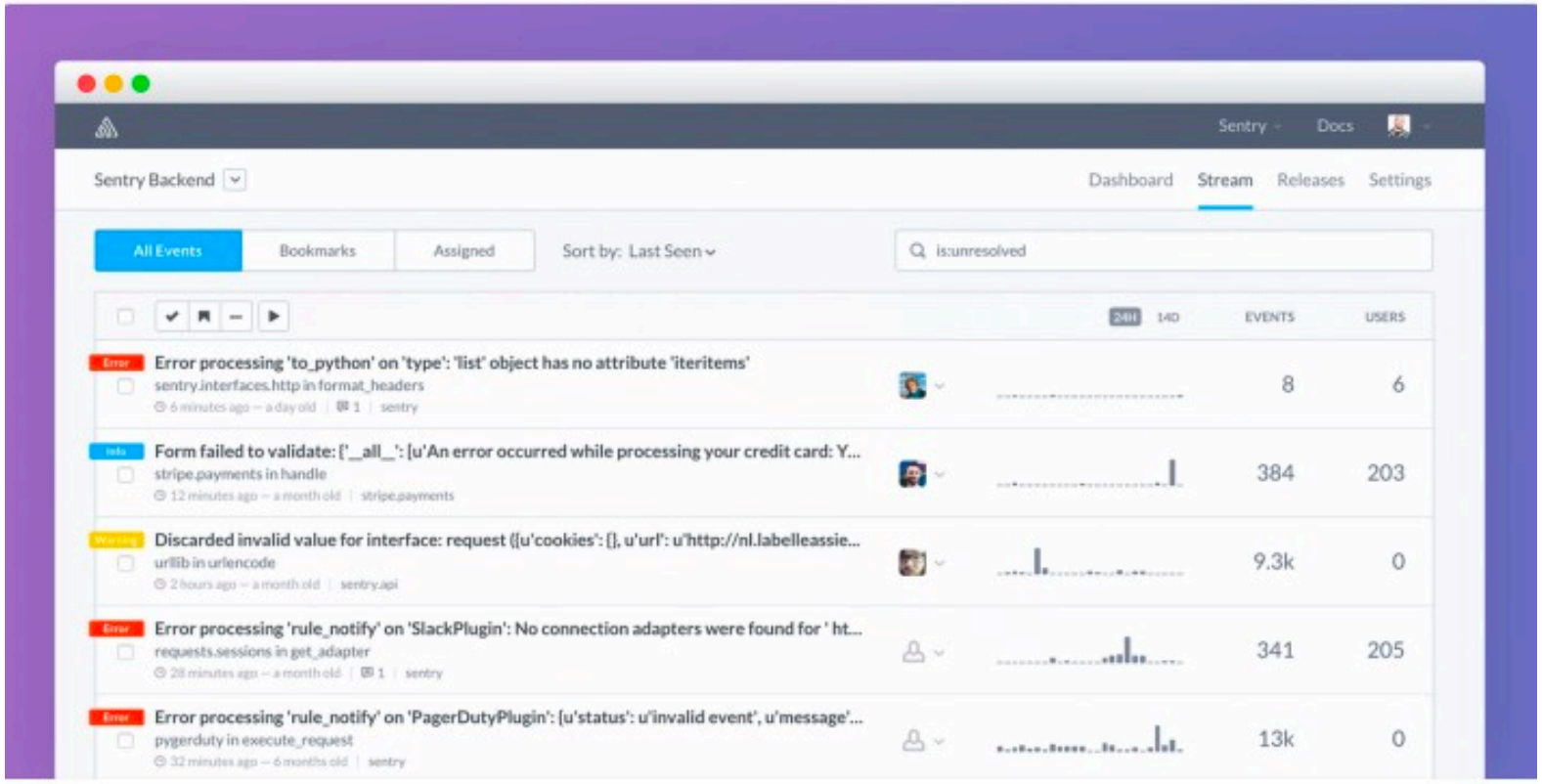
У прометеуса есть своя алертилка, которая позволяет задать алерт на любым правилам с использованием promQL



# Sentry

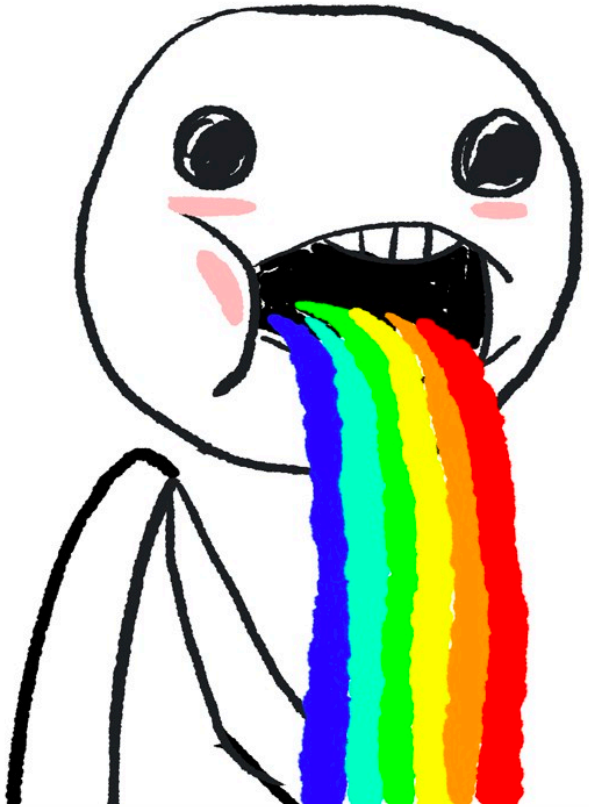
Sentry – инструмент для exception handling. Ловит, как фронтовые ошибки, так и бекенд.

- Группирует похожие ошибки
- Есть язык запросов для фильтрации



# Grafana

Grafana – дает возможность создавать богатые дашборды.
Есть свой алертинг



# Управление инцидентами

Инциденты всегда будут (к сожалению)

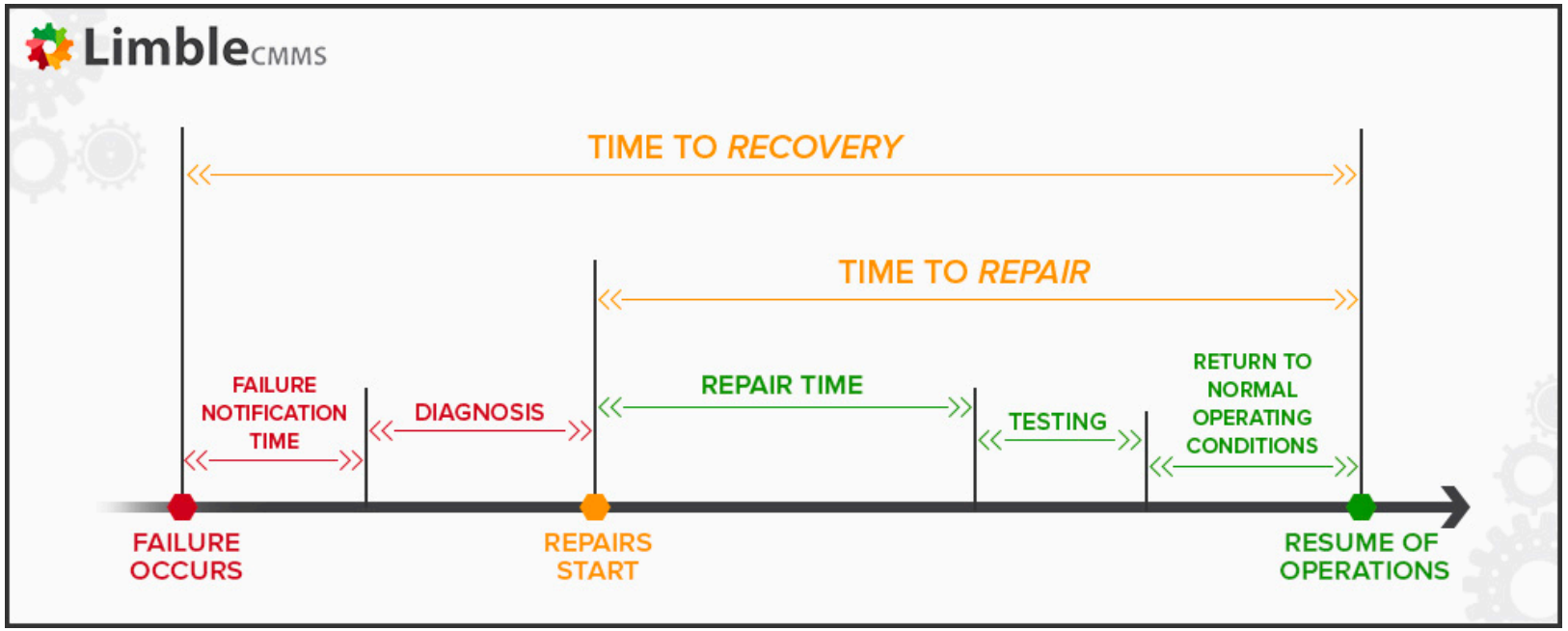
При каждом инциденты цели:

- Минимизировать ущерб
- Предотвратить повторение

# Минимизация ущерба

Минимизировать ущерб помогает:

- Быстрое обнаружение
- Развитые средства мониторинга и алертинга



# Чтобы не повторялись

Практика постмортем

- Собираем факты
- Собираем все артефакты и восстанавливаем, как все было
- Находим системную (корневую) причину, а не виновного
- Создаем задачи и не забываем пофиксить корневую причину

# Форма постмортема

Создание проблемы

**Заголовок** *

Коротко о том, что случилось

**Статус** *

Проявляется

**Проблемные сервисы** *

Добавьте сервисы причастные к данной проблеме

**Виновники** *

Добавьте пользователей причастных к данной проблеме

**Ссылка на задачу в JIRA** *

Ссылка на задачу в JIRA

**Создать задачу автоматически**

**Дата и время создания** *

дд.мм.гггг чч:мм

**Время недоступности (мин)**

0

**Влияние на пользователей** *

Выберите из списка

**Описание инцидента**

Расскажите в деталях о том, что случилось.

**Дата и время решения**

дд.мм.гггг чч:мм

**Ответственный за решение**

Выберите из списка

**Как решили проблему**

Сообщите о результатах работы. Предупредите о важном.

Прикрепите файл

# Управление инцидентами

Наши практики:

- Рассылка статуса доступности каждый день по всем сервисам
- Каждую неделю разбор инцидентов и сбор статуса
- Каждую неделю развод дежурных от прикладных команд
- Дежурный админ поднимает дежурного от прикладных команд по звонку
- Создает отдельный чат для обсуждения инцидента
- После завершения инцидента чат закрывается

# Книги


O'REILLY
Site Reliability Engineering
НАДЕЖНОСТЬ И БЕЗОТКАЗНОСТЬ КАК В GOOGLE
Бетси Бейер, Крис Джоунс,
Дженнифер Петофф,
Нейл Ричард Мёрфи
ПИТЕР

# Опрос

https://otus.ru/polls/5550/

# Спасибо
# за внимание!