Автодискавери в мониторинге

Николай Сивко

okmeter.io

Кто говорит

- Очень долго работал админом
- Руководил эксплуатацией в hh.ru
- Основал и работаю в okmeter.io – сервис мониторинга

Задачи мониторинга

- Узнать о том, что что-то сломалось
- **Быстро узнать что именно и как именно сломалось**
- Прочие операционные задачи: оптимизации, capacity planning, итп

Метрики + графики + триггеры

- Самое главное в системе мониторинга — метрики = данные
- В 99% случаях их нельзя собрать задним числом или повысить детализацию
- **Хорошие** графики – оптимизация над метриками
- **Хорошие** триггеры – оптимизация над метриками

Как **обычно** устроен сбор метрик

- На каждом сервере запущен мониторинговый агент
- Агенту нужно объяснить, с каких сервисов снимать метрики:

```
[[inputs.redis]]
    servers = ["tcp://localhost:6379"]
```

- Конфигурацию каждого агента нужно поддерживать в актуальном состоянии

Поддерживать конфигурацию агентов сложно

- Написать конфиг агента для **КАЖДОГО** сервиса в ansible/puppet/chef — это работа
- Очень сложно понять, какие сервисы вы забыли замониторить
- О том, что кто-то сломал сбор метрик вы узнаете ровно в тот момент, когда эти метрики вам понадобятся

Пример из жизни #1

- **dmesg**: `[Fri Mar  2 17:13:31 2018] BUG: unable to handle kernel paging request at ffff881031607aa0`
- **PS**: `postgres  1724 13.4  0.0      0     0 ?        D     2017 33445:53 [pgbouncer]`

- Process state **D = uninterruptible sleep** (реально не прибивается)

- Это pgbouncer на master базе и он держит TCP порт

- Админ запускает рядом pgbouncer на другом порту и переключает всех клиентов на него

*** В этот момент мы бы потеряли метрики pgbouncer**

okmeter

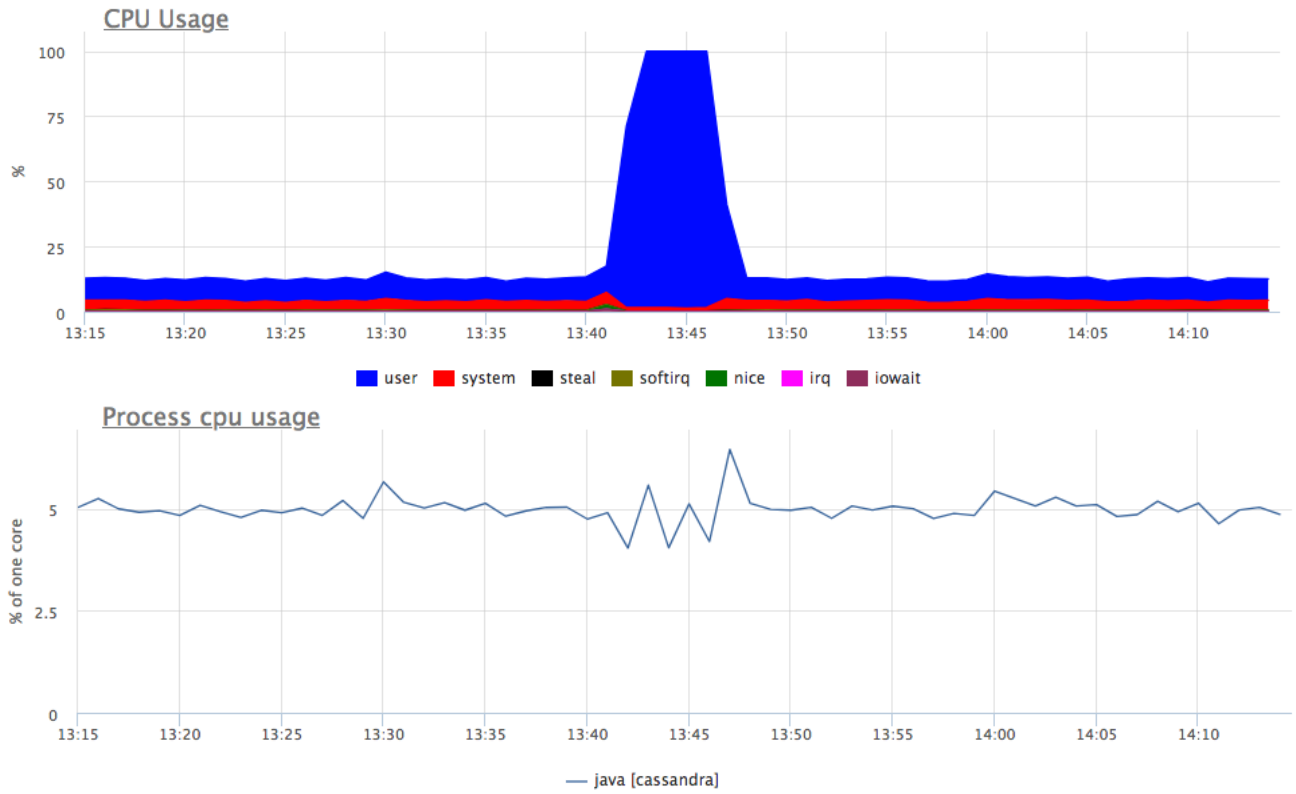
Пример из жизни #2

- Жил был memcached 1.4.x пользовательскими сессиями
- Вся память была занята слабом 304 байта
- Выпустили релиз с кэшированием еще одной сущности
- Она попадала в slab 120 байт
- mc1.4 не умеет перераспределять слабы, и по второй сущности были сплошные MISS
- Чтобы не сбрасывать сессии, развернули рядом в докере mc1.5

*** В этот момент никто бы и не подумал покрывать его мониторингом**

Полнота собираемых метрик

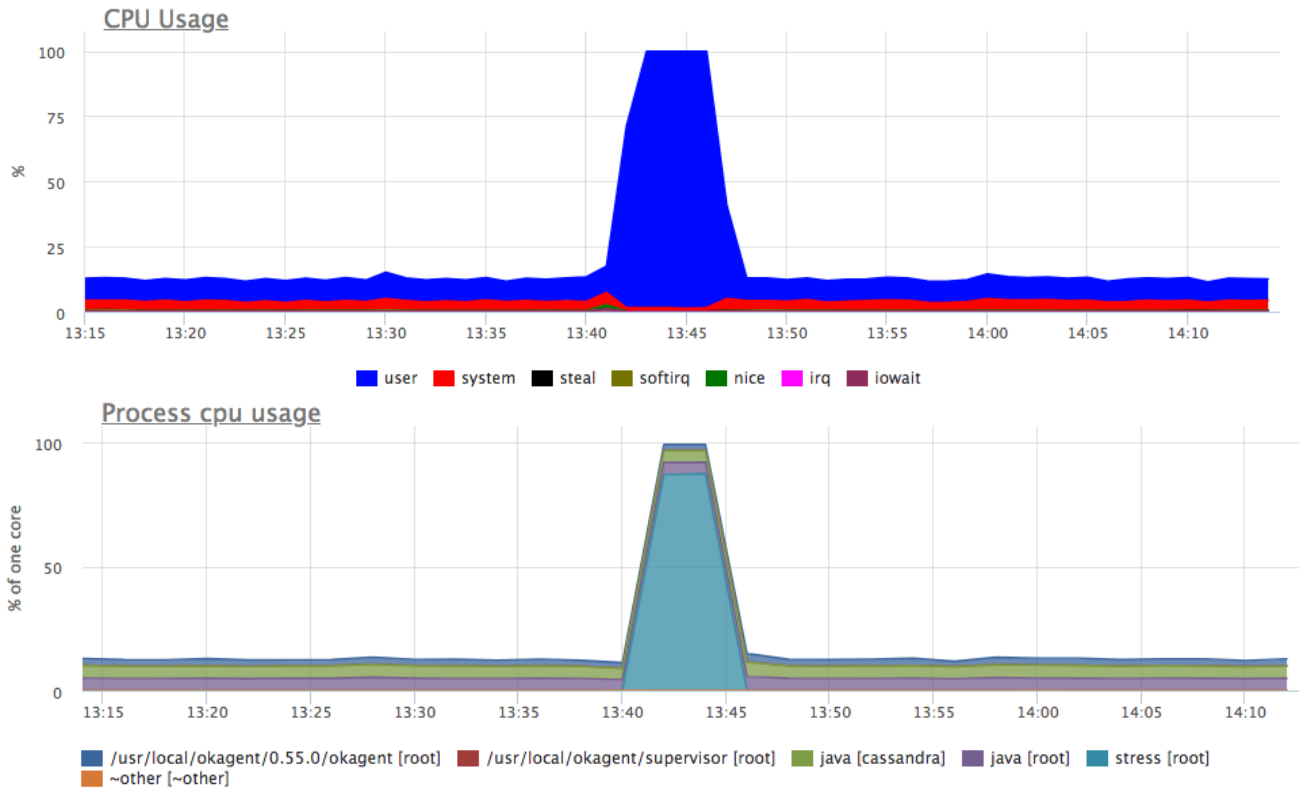
Недостаточно снять потребление только ”рабочих” процессов:



okmeter

Полнота собираемых метрик

Если снимать **ВООБЩЕ ВСЁ**



okmeter

Давайте всегда снимать **вообще ВСЁ**?

- Системные + “железные” метрики
- Метрики всех процессов/контейнеров/cgroups
- Метрики всех TCP соединений
- Метрики всех сервисов*

Давайте всегда снимать **вообще ВСЁ**?

- Системные + “железные” метрики
- Метрики всех процессов/контейнеров/cgroups
- Метрики всех TCP соединений
- Метрики всех сервисов*

… и назовем это АВТОДИСКАВЕРИ :)

Системные метрики

- CPU: usage, core_throttle_count, logical_cores
- Mem: usage, swap, errors, numa
- Disk: usage, inodes, io, latency, raid+bbu, SMART, fs errors
- Net: interfaces metrics+errors, TCP/IP metrics (retrans, overflows, …), conntrack table

Процессы/контейнеры/cgroup

- Берем список процессов, по каждому снимаем:
  - cpu (usage, quota, throttled)
  - mem (rss+cache+page faults)
  - disk io (fs+device),
  - open fds, threads, uptime
- Группируем (чтобы уменьшить количество метрик)
- Агрегируем значения в группе
- Навешиваем доп.метки на метрики: container, k8s_pod, k8s_ns, k8s_container (TBD: k8s_rs, k8s_deployment, …)

TCP соединения

- Все tcp соединения делим на inbound/outbound/listen
- Группируем по listen_ip, listen_port, remote_ip*
- Снимаем: states (count), rtt, rqueue/wqueue
- Listen: status, backlog current/max

#нифигасечобывает

- Ребята проксировали passive ftp через haproxy
- Открыли 64 тысячи tcp портов (for loop в шаблоне в руках админа – страшная вещь)
- С тех пор мы снимаем только topN listen ports

Сервисы #1

- Берем список процессов
- Видим знакомый сервис (nginx, pg, mysql, …)
- Ищем, где у него конфиг*
- Читаем конфиг
- Находим логи+форматы, listen ip+port, итд

Сервисы #2

- **Вводим идентификатор инстанса сервиса**: instance=“X”
- Пытаемся снять метрики
- Если не получается, отправляем спец.метрику с ошибкой/диагностикой
- По спец.метрике делаем алерт для пользователя (по каждому инстансу)*

Сервисы в контейнерах

- Нужно уметь попадать в fs (MntNs) контейнера
- Нужно уметь попадать в NetNs контейнера
- Instance=container/k8s…

#нифигасечобывает

- 100+ инстансов postgresql на сервере
- 100+ memcached на сервере
- 40 redis на сервере

Автодискавери 80 LVL

- Видим процесс postfix (есть такой MTA)
- Вычисляем его syslog facility
- Находим syslogd (rsyslog, syslog-ng,…) в том же контейнере*, читаем его конфиг
- Резолвим по правилам из конфига, куда пишется лог с нужным facility
- Запускаем парсер

Автодискавери 81 LVL (alfa)

- Видим процесс, получаем executable path
- Прочитав ELF header, понимаем, что это golang app
- Мержим (netstat listen socks inodes) X (proc fds)
- Определяем listen сокеты приложения
- Наобум пробуем, вдруг там http и есть expvar на стандартном URI
- Запускаем poller expvar метрик

Спец. метрики

- Numeric + text metric: {**name**: status, **plugin**: X, **instance**="Y"}
  - num value: 1|0
  - text value: "" | "error text"

Спец. метрики: конфигурационные алерты

- ”дядь, дай доступ в PG на сервере X”
- “не бойся, включи pg_stat_statements в проде”
- “$upstream_response_time в лог nginx добавить должен ты”
- “я тоже хочу доступ к rabbitmq vhost XXX”

Взаимодействие с человеком через алерты

- Амины лучше всего умеют закрывать алерты (встроенная геймификация в мониторинге:)
- Список “конфигурационных” алертов = TODO для админа
- Не надо держать состояние в голове
- Если что-то разломали, снова алерт

Проблемы

- Надо кодить

- **Очень-очень много** метрик:
  - Сложно хранить
  - Визуализировать
  - Дорого считать триггеры

- Часто приходится брать только TopN метрик, причем так, чтобы значимые были отдельно, а всякая мелочь схлопывалась

“авто*” бывают не только метрики

- Метрики не должны лежать мертвым грузом
- Автодашборды
- Автотриггеры

Автодашборды

- Detect expression: нужно ли показывать дашборд X?
- Группировки: дашборд elasticsearch на каждый cluster_name
- Корневой дашборд всегда верхнеуровневый

Hostpage

- Автодашборд с метриками хоста
- Все системные метрики хоста
- Все известные дашборды, ограниченные данным хостом

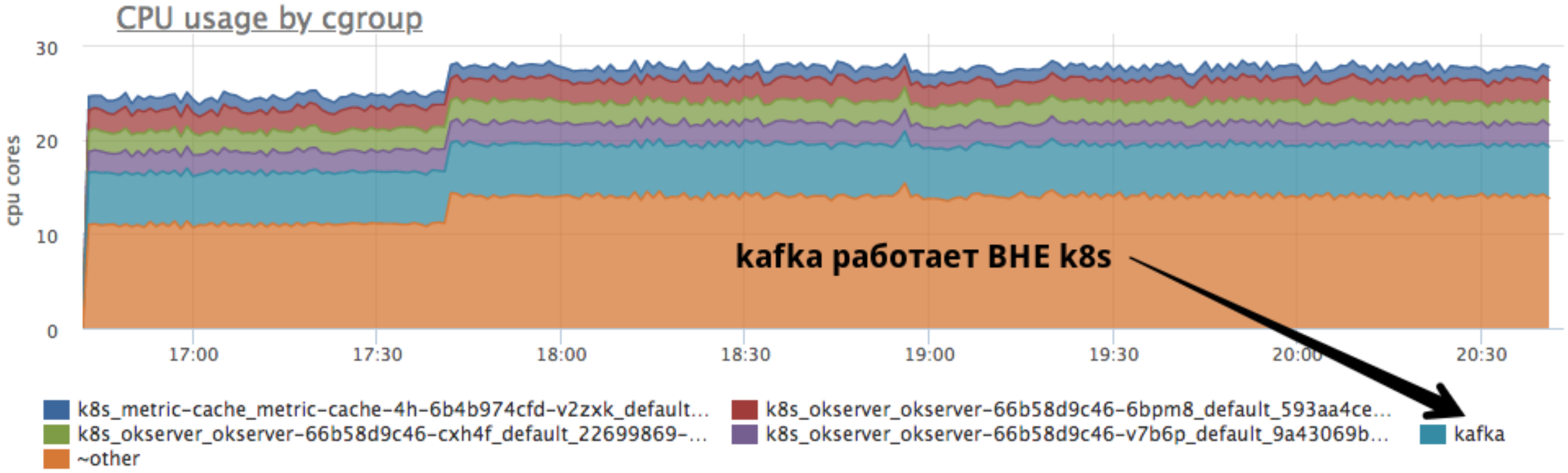
Дашборды

- Bird view: все графики cluster-wide (без привязки к серверам итд)
- Отправная точка: можно ”провалиться” в любой график за деталями или ограничить весь дашборд подмножеством метрик

Дашборды: навигация при факапе

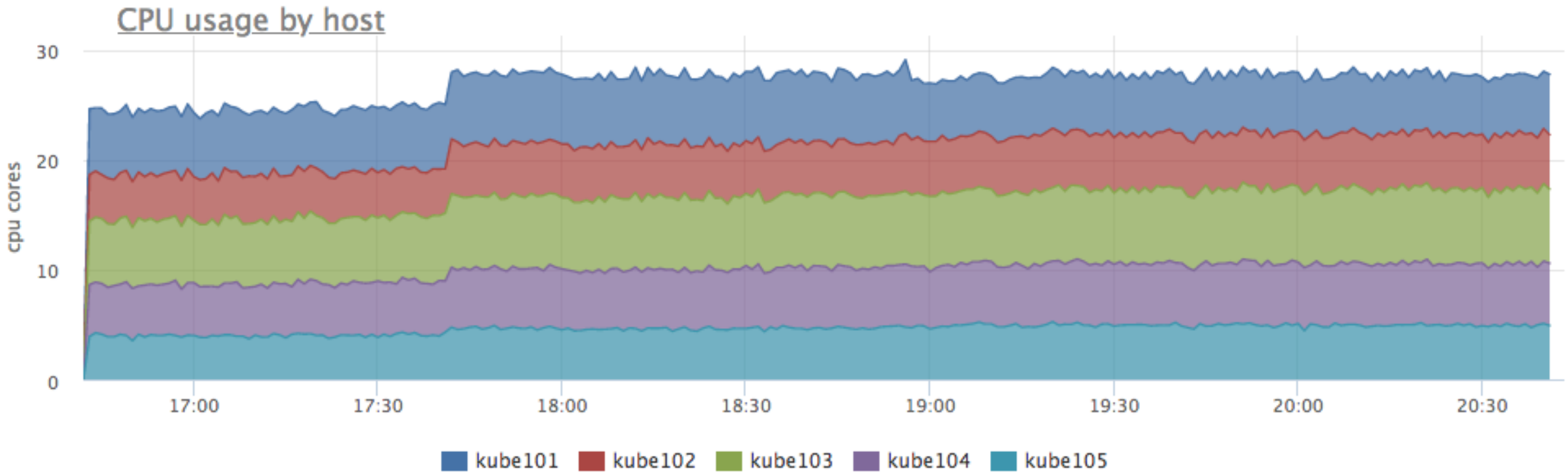
- Всегда начинаем с дашборда ”что сейчас видят пользователи” (есть проблема или нет)
- Потом summary по сервисам: topN по ошибкам, времени ответа итд (в каком конкретно сервисе проблема? Или во всех сразу?)
- Детализация каждого сервиса (в чем проблема: сам, база, соседний сервис, …)
- Детализация инфраструктуры под сервисом (базы, очереди, …)

top(5, sum_by(**cgroup**, counter_rate(
metric(name='cgroup.cpu.usage'))))



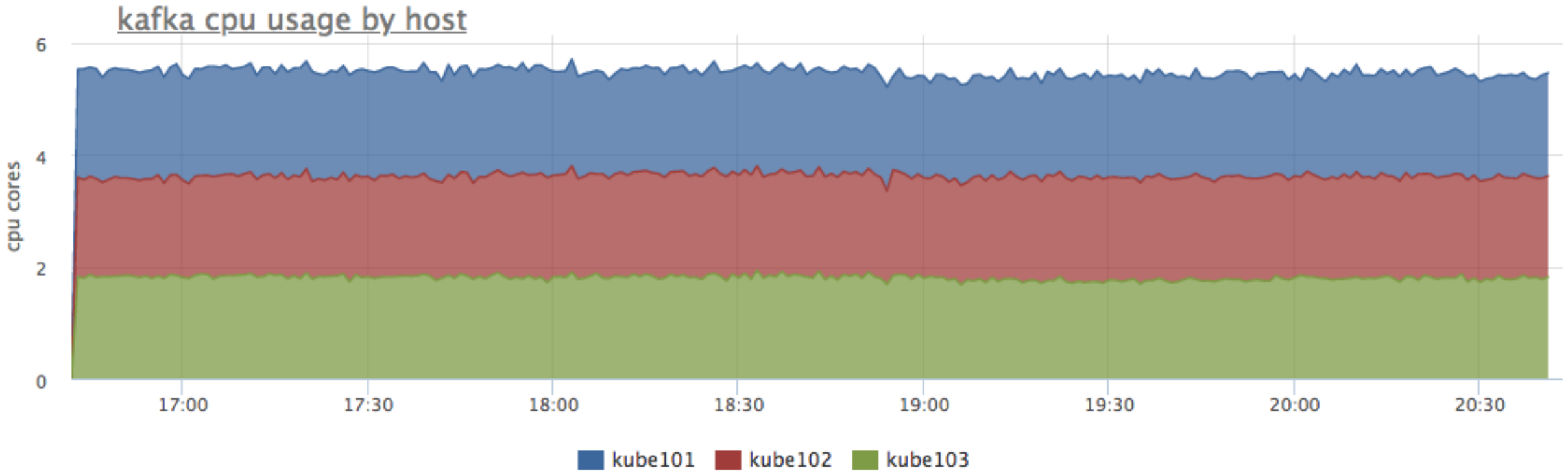
okmeter

```
top(5, sum_by(source_hostname, counter_rate(
                    metric(name='cgroup.cpu.usage'))))
```



okmeter

top(5, sum_by(source_hostname, counter_rate(
metric(name="cgroup.cpu.usage", **cgroup="kafka"**))))



okmeter

Автотриггеры

- Считаются по всем имеющимся метрикам
- Не нужны шаблоны, роли и подобные сущности
- Есть expression, все что нашлось — участвует
- Один expression может породить тысячи trigger instance (конкретная лампочка для subset метрик, зависит от группировок в expression)

Автотриггеры

- TCP: backlog/max > 90% -> **любое** приложение притупило и не берет новые соединения
- Proccess: max_cpu_per_thread > 90% -> тред уперся в ядро (pgbouncer, nginx, tarantool)
- Cgroup: cpu usage > 90% от лимита
- Nginx certs: (seconds_to_expire < N) || (revoke_status < 1) -> мониторим абсолютно все сертификаты, известные nginx
- …

Автотриггеры

- Правило “чистого мониторинга”: если у вас сейчас все в порядке, то в мониторинге не должно быть горящих алертов (как минимум critical)

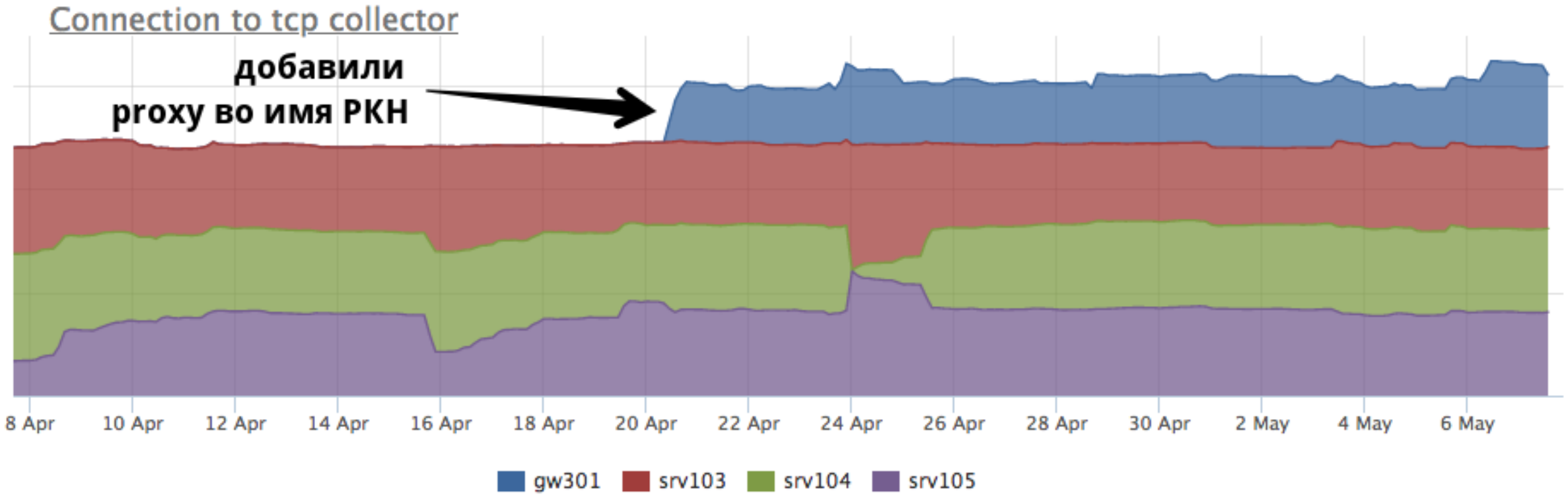
- “force resolve” + добавление исключений

Play

- Пишем expression –> рисуется график (доступна простенькая математика над метриками)

- Играет новыми красками когда ”внизу” тысячи метрик ”про всё”

```
sum_by(source_hostname,
        metric(name="netstat.connections.inbound.count", listen_port="443", listen_ip=[...]))
```



okmeter

Anomaly detection

- У нас **пока** нет, так как были заняты метриками:)
- Обычно все делают ML по “load average” и “cpu usage”, но практического смысла в этом почти нет
- Мы исследуем, как повторить роботом поведение человека при факапе
- Метрики очень ”шумные”, гарантировано будет много false positive, вешать алерты на это нельзя

okmeter

Итого

- Метрики – самое главное в мониторинге
- Снимайте метрики про все подсистемы всегда
- Чтобы человек ничего не забывал, не нужно ему ничего поручать
- Если что-то поручаете человеку, нужно проверять за ним роботом
- Лучше написать код, чем регламент для людей:)

Спасибо за внимание!

Вопросы?

Николай Сивко

nsv@okmeter.io