Логирование

Автор: Бекмат Алибек

План

- Что такое логи и зачем нужны логи?
- Централизованная система логирования: основные компоненты, требования, примеры
- Elastic Stack
- Логирование приложения

Зачем нужны логи?

- Видимость и понимание того, как работают наши системы
- Поиск ошибок и их причин
- В отличие от метрик, содержат полный контекстработы системы или процесса

Пример применимости логов

Запустили ui сервис в фоновом режиме

```
$ docker-compose up -d ui
```

```
reddit_post_db_1 is up-to-dateStarting
reddit_post_1 ...
Starting reddit_post_1 ... doneStarting
reddit_ui_1 ...
Starting reddit_ui_1 ... done
```

Но приложение не работает :(Что делать?

This site can’t be reached

localhost refused to connect.

Search Google for localhost 9292

ERR_CONNECTION_REFUSED

Ищем ошибку

Смотрим логи и находим ошибку

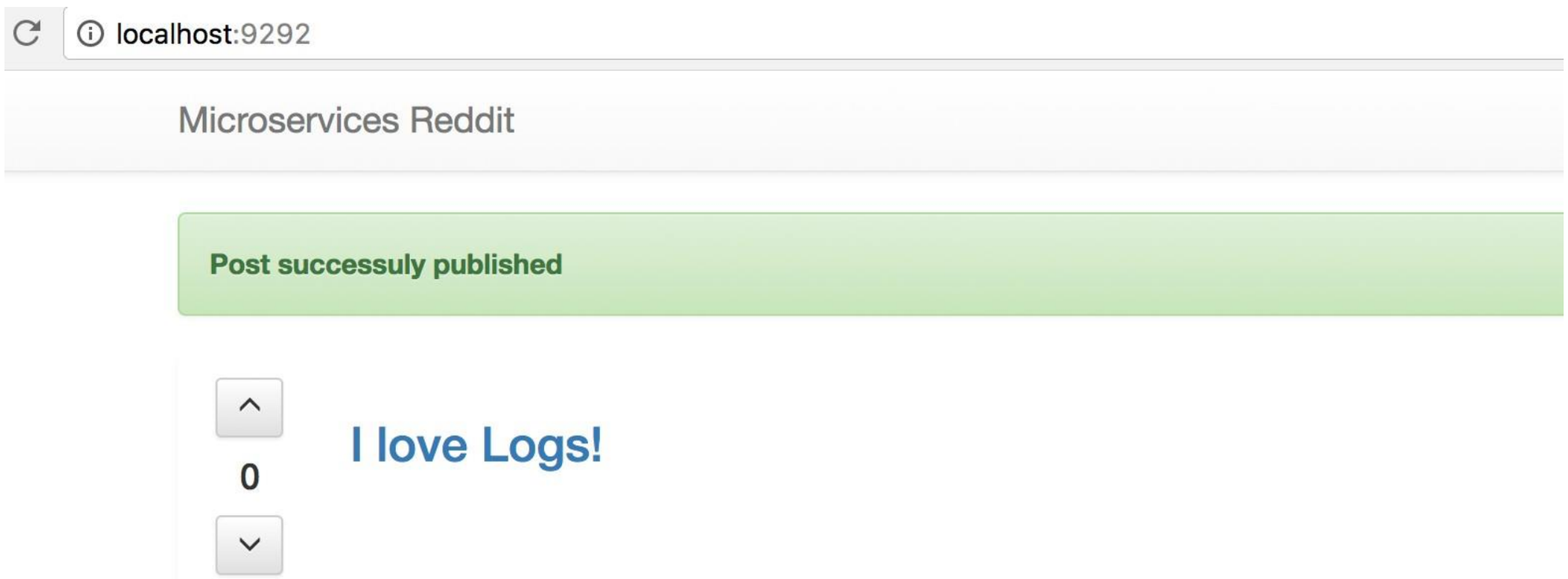
```
$ docker-compose logs ui
```

Attaching to reddit_ui_1

```
ui_1
ui_1   | * Min threads: 0, ..
ui_1   | * Environment: development
                          ui_1
| ! Unable to load application: SyntaxError: /app/ui_app.rb:60: syntax error,
unexpected keyword_end, expecting end-of-input
 ui_1                     | config.ru:1:in `require': /app/ui_app.rb:60: syntax error, unexpected
keyword_end, expecting end-of-input (SyntaxError)


`instance_eval'
 ui_1                     | from /usr/local/bundle/gems/rack-2.0.3/lib/rack/builder.rb:55:in
`initialize'

                          ui_1   | from config.ru:in `<main>'
                          ui_1   | from /usr/local/bundle/gems/rack-2.0.3/lib/rack/builder.rb:49:in `eval'

`new_from_string'
```

Правим код

Исправляем синтаксическую ошибку и приложение работает. Спасибо логам!



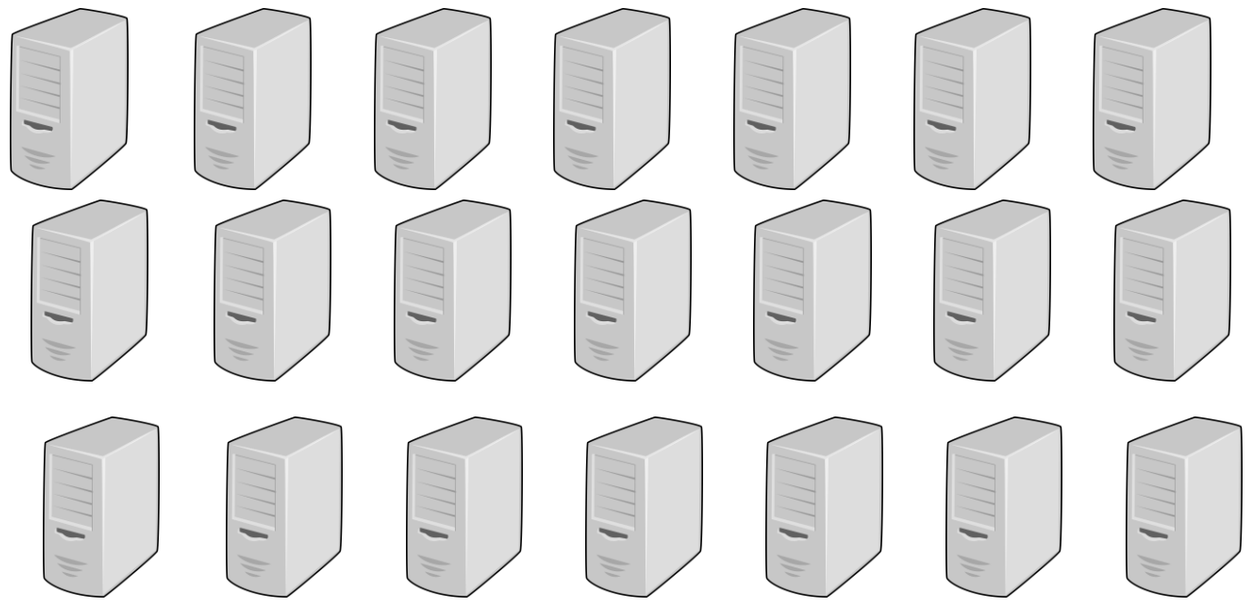
Что такое лог?

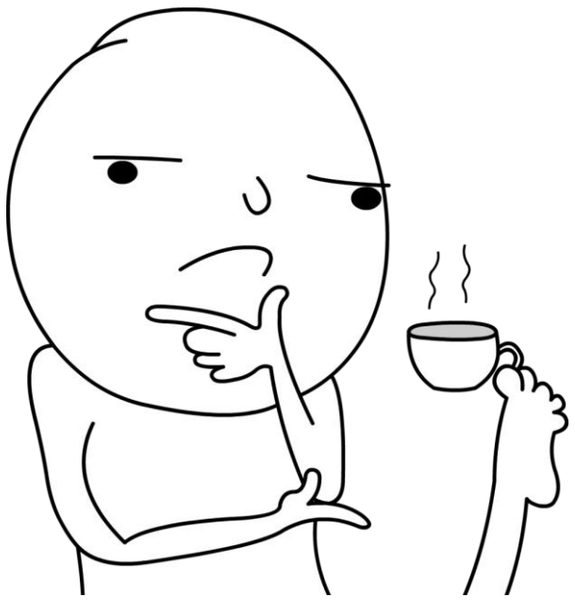
- Журнал событий происходящих во время работы системы или процесса

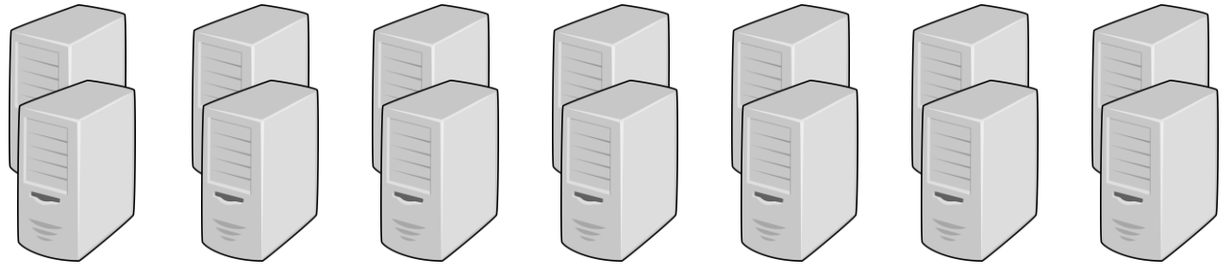
Храним локально?

- Не понимаем, что где происходит, пока сами не зайдем всистему и не посмотрим
- При большой ферме серверов не хватит рабочего времени на обход всех машин
- Нет возможности быстро локализовать проблему - следовательно, и ее решить

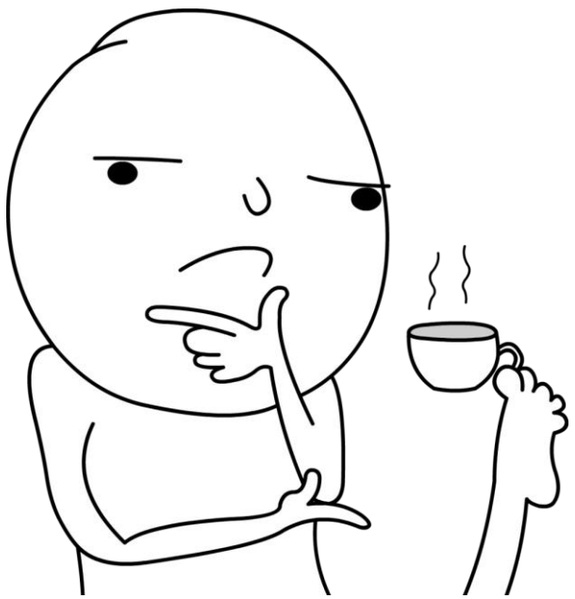
и что из них работает, а что нет?

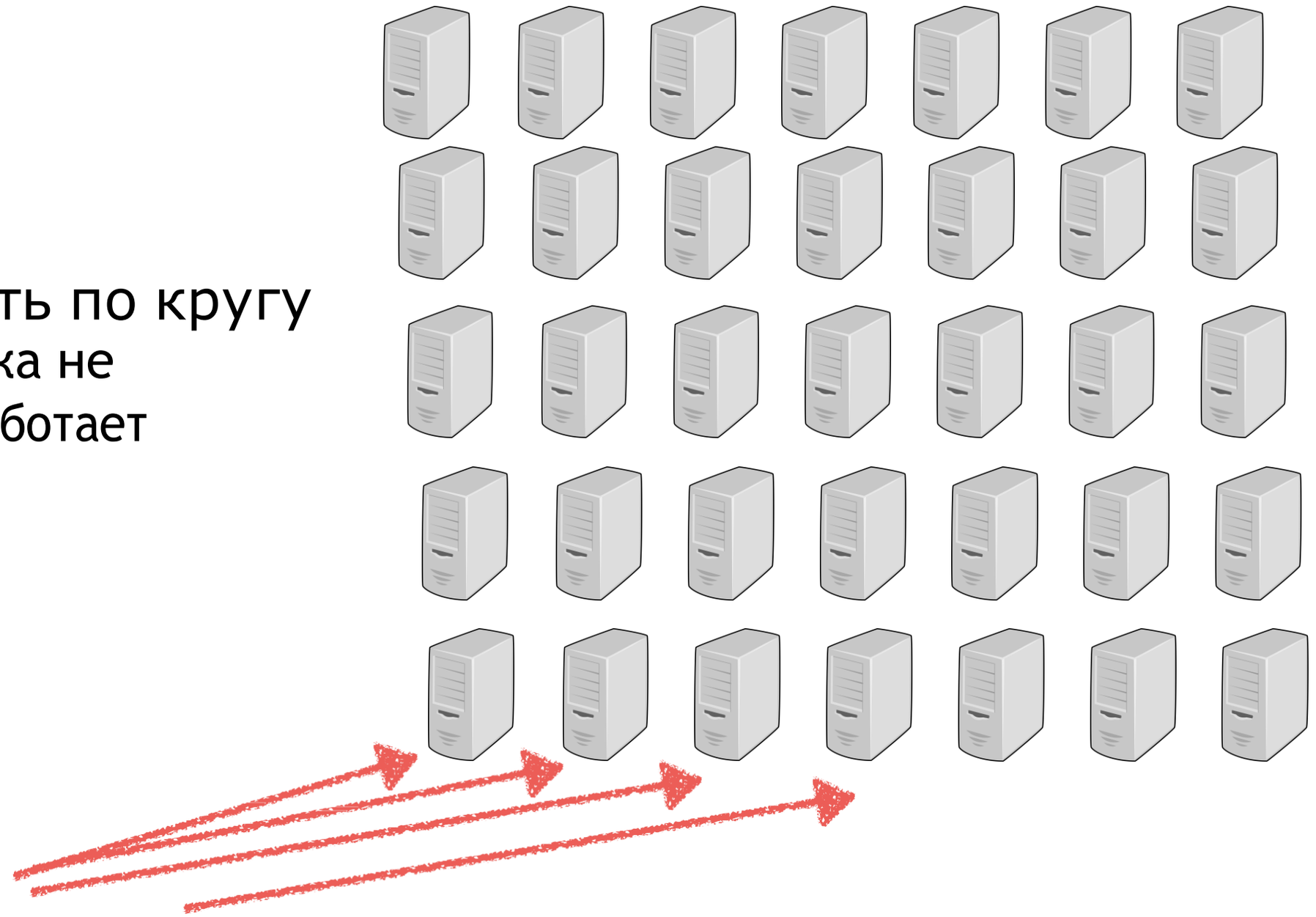




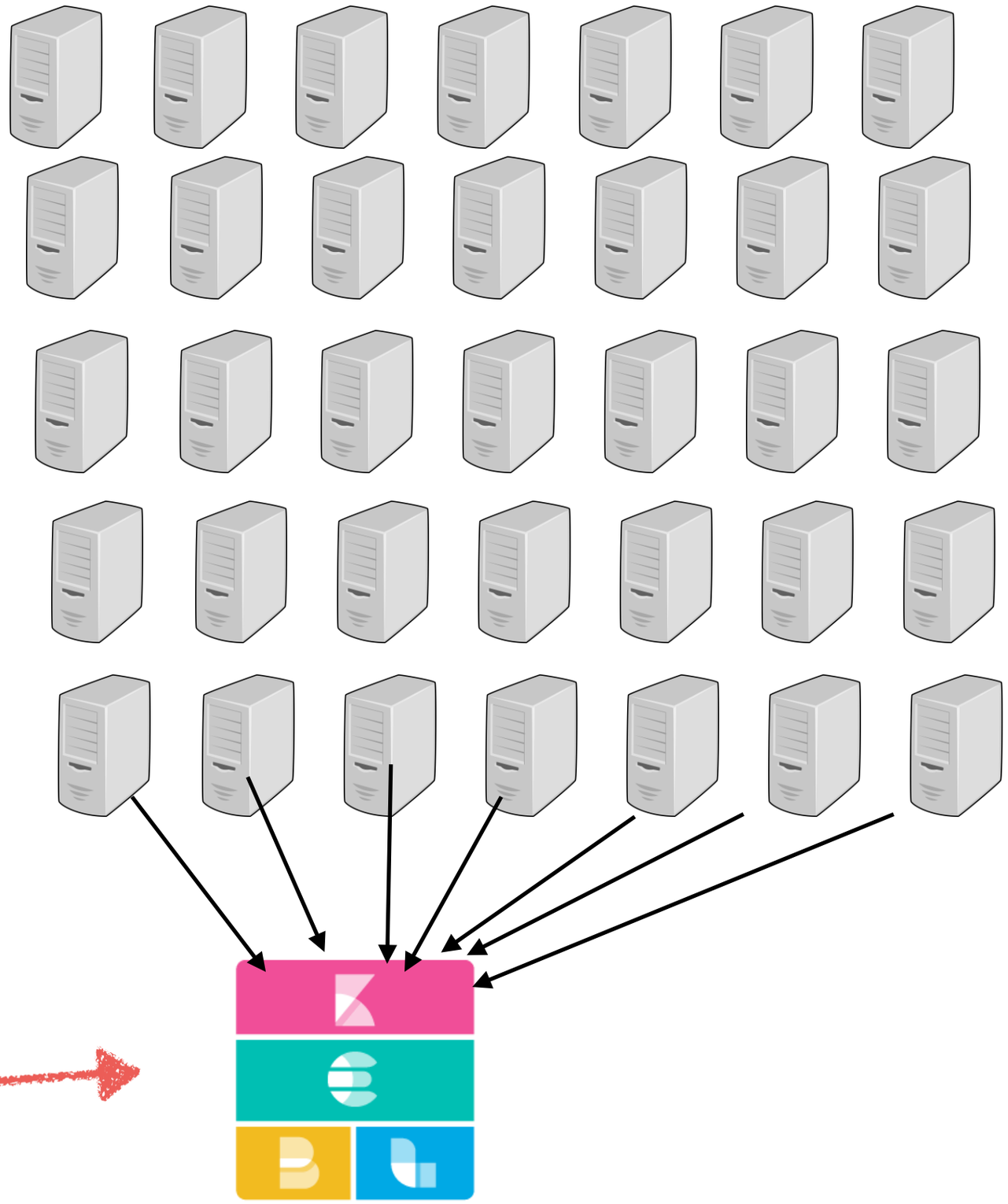


Начну обходить по кругу
все машины, пока не
найду, где не работает





Пускай хосты отдают всю информацию центральному серверу, буду обращаться только к нему



Централизованная система логирования (ЦСЛ)

- Центральный сервер(ы) агрегирует всю информацию по логам

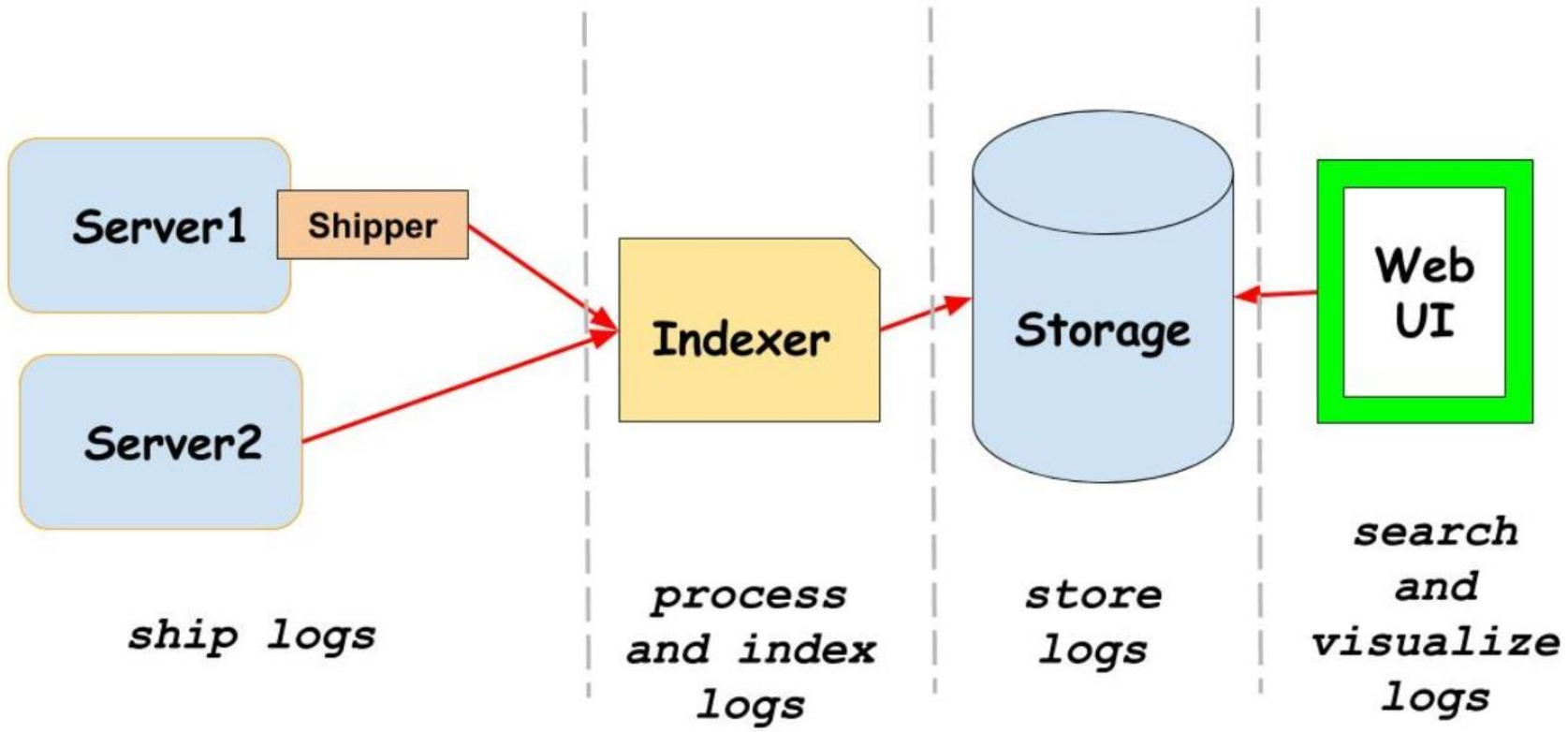
- Единая точка доступа ко все информации

- Возможность проведения анализа по всем системам

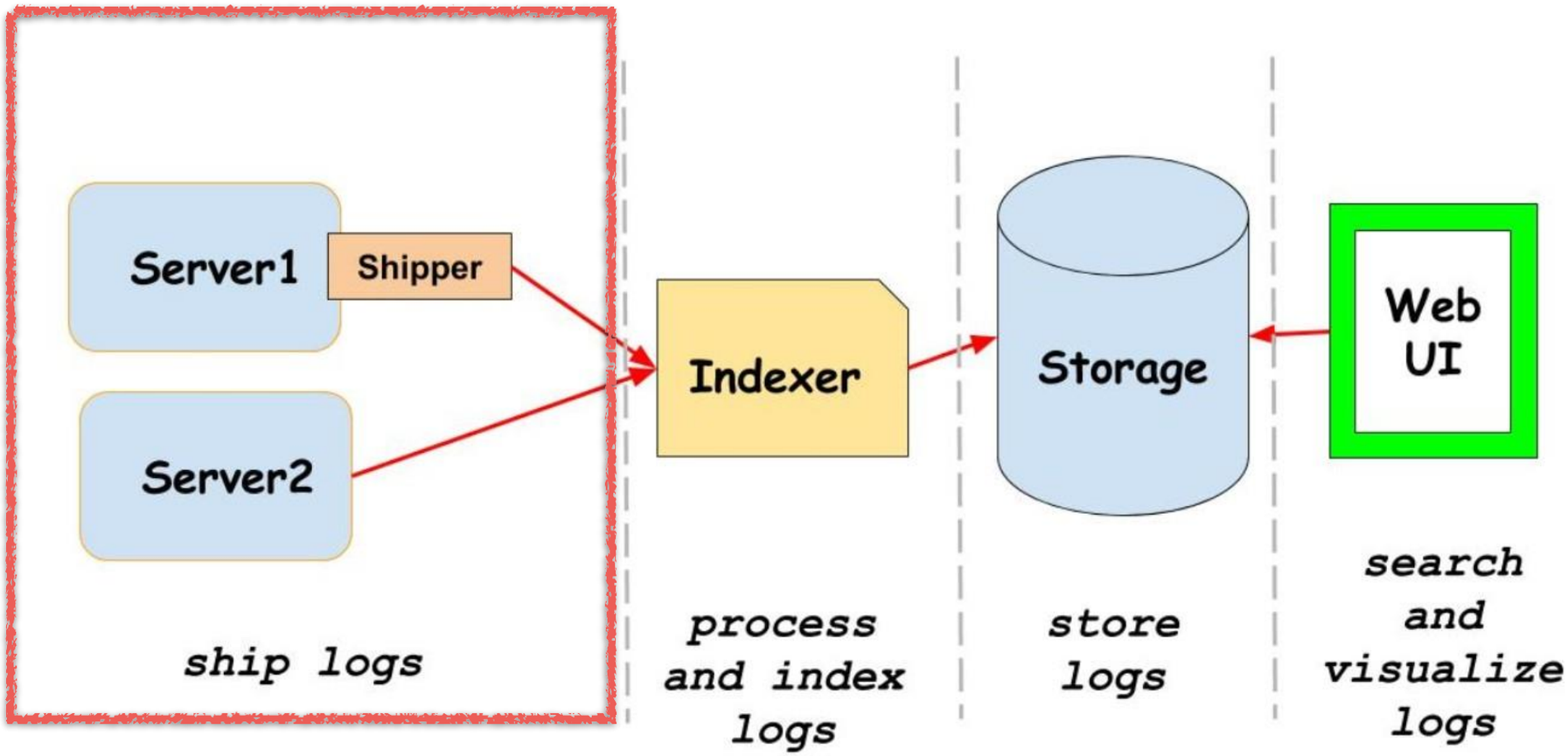
Но ...

- Централизованная система логирования не отменяет локального хранение логов
- Локальное хранение логов по-прежнему является самым надежным способом хранения
- Возможна потеря логов, если центральный сервер загружен или не доступен
- Практика показывает что центральный сервер может быть и не доступен

Основные компоненты ЦСЛ

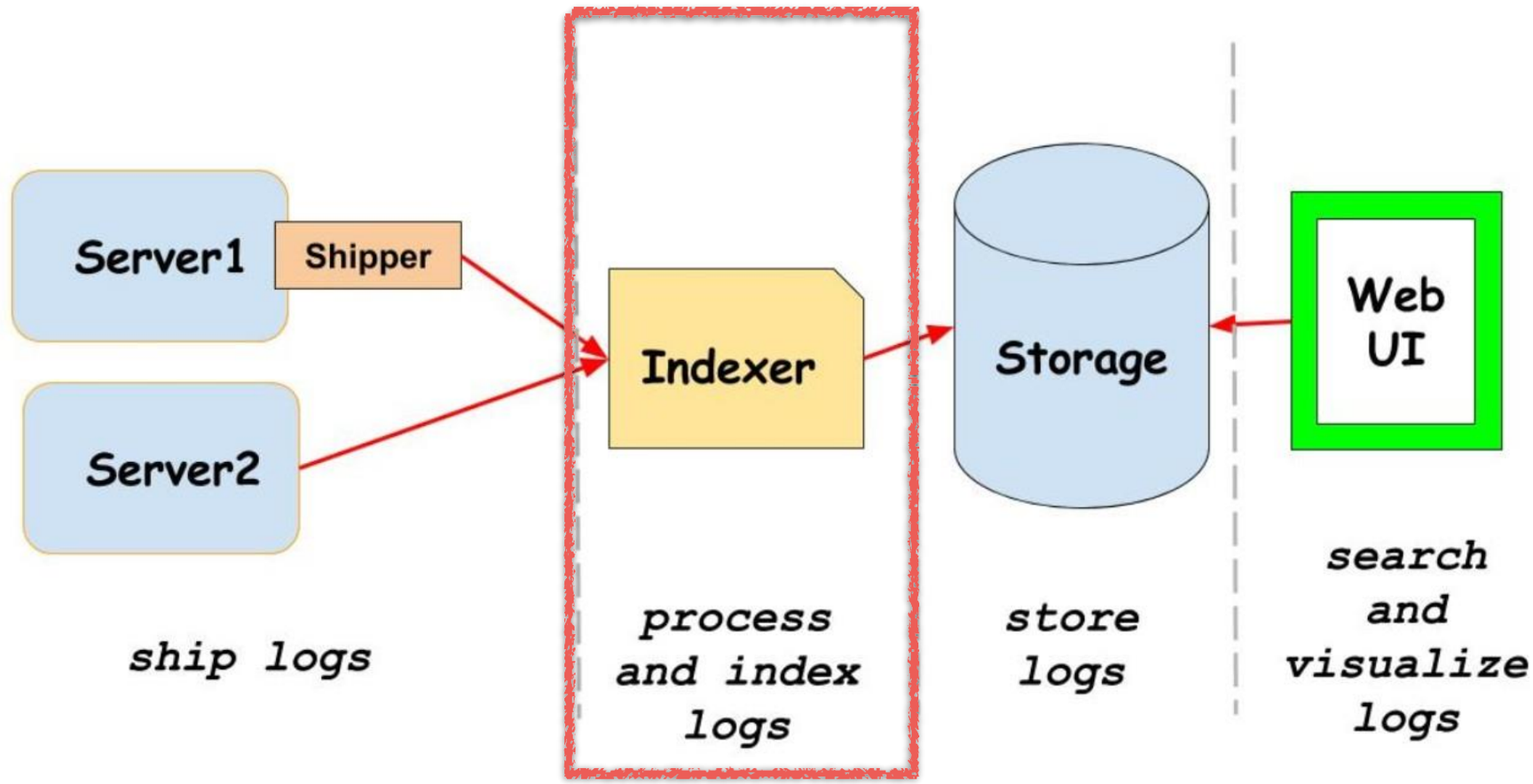


Отправка логов (shipping)



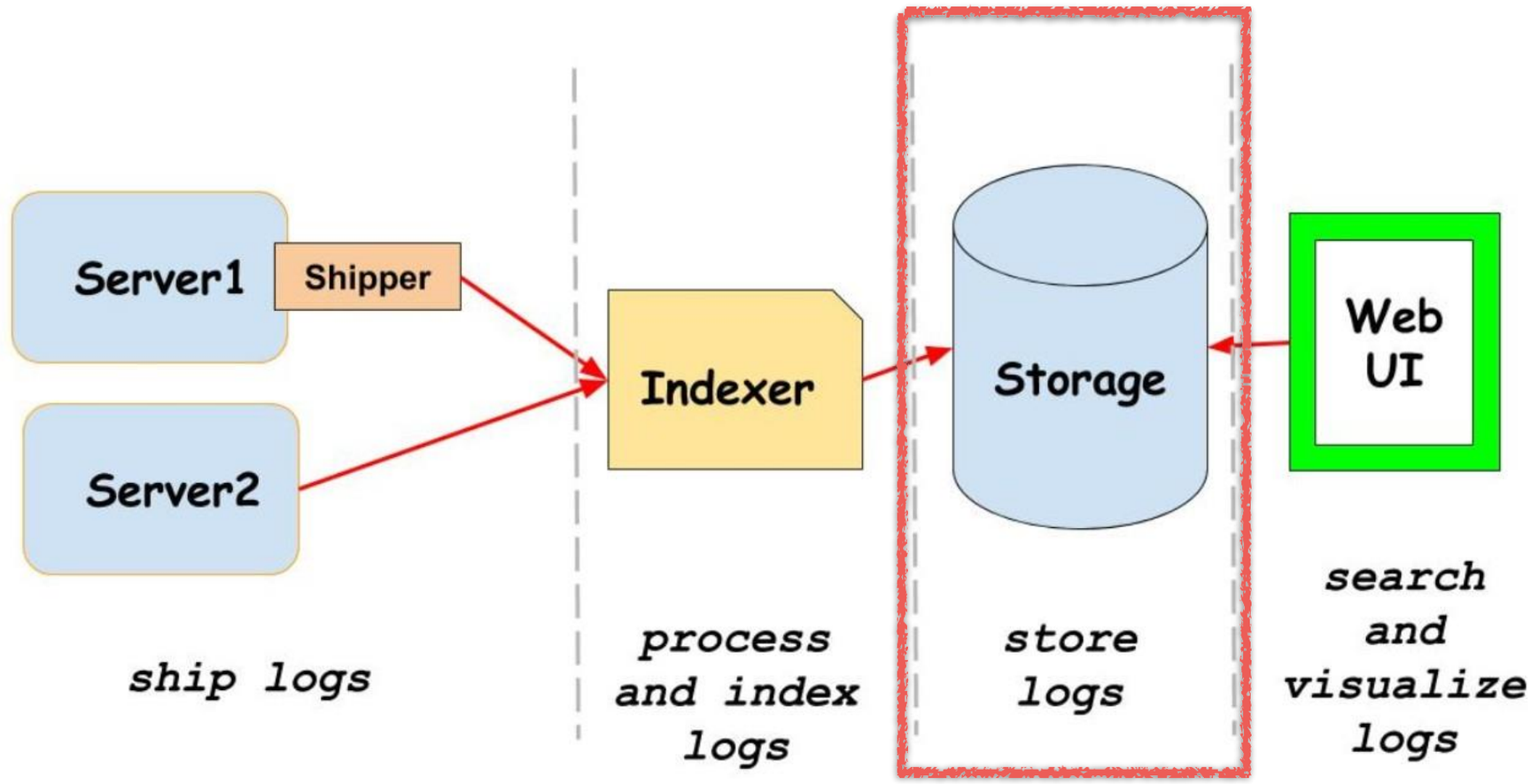
- Клиентские библиотеки
- Shippers: beats, nxlog, rsyslog, syslog-ng, fluentd, etc.

Агрегация и трансформация



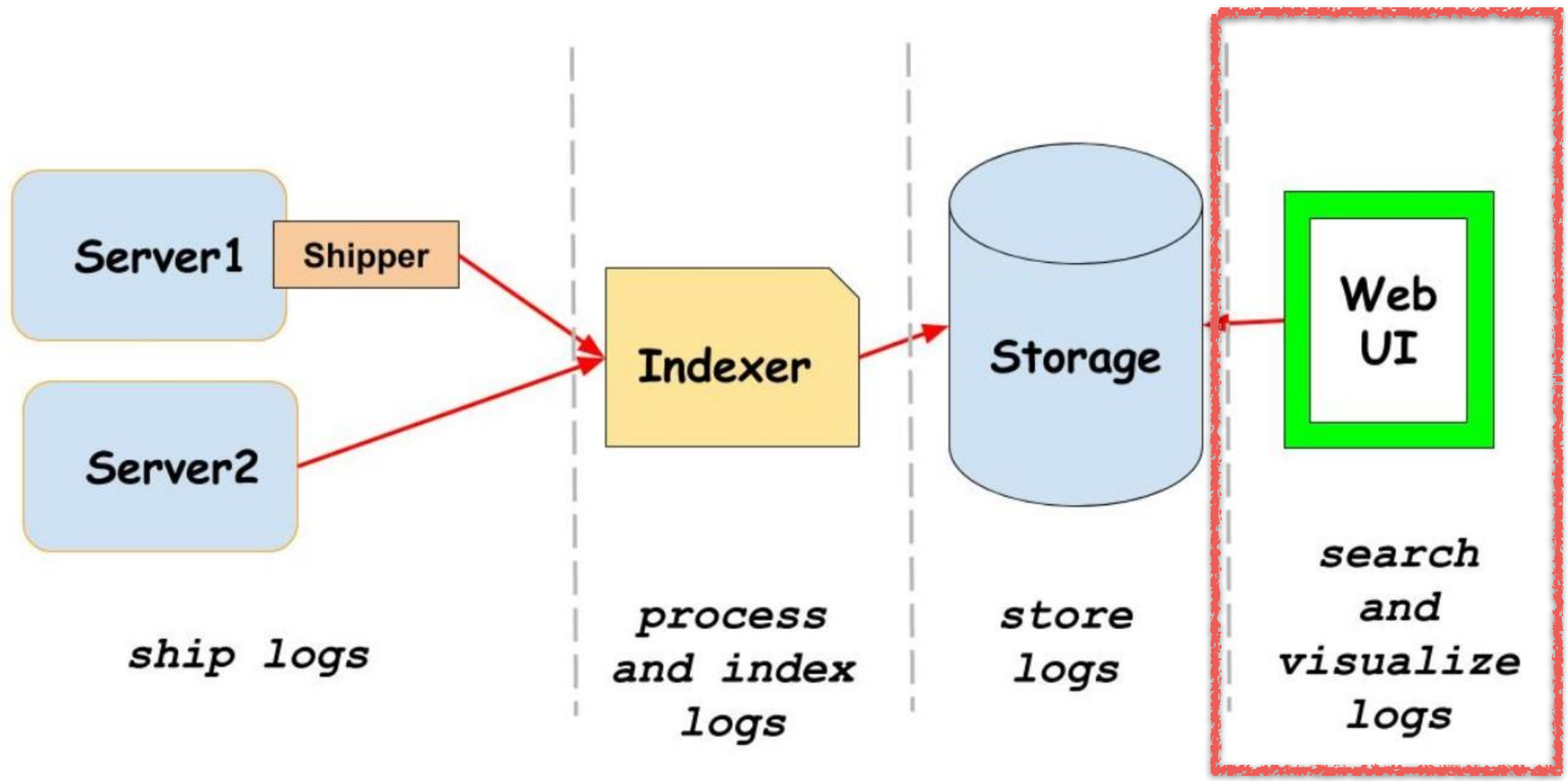
Logstash, Graylog 2, Splunk, etc.

Хранение логов



ElasticSearch, InfluxDB, MongoDB, S3, etc

Визуализация, анализ и алертинг



Kibana, Graylog 2, Grafana, etc.

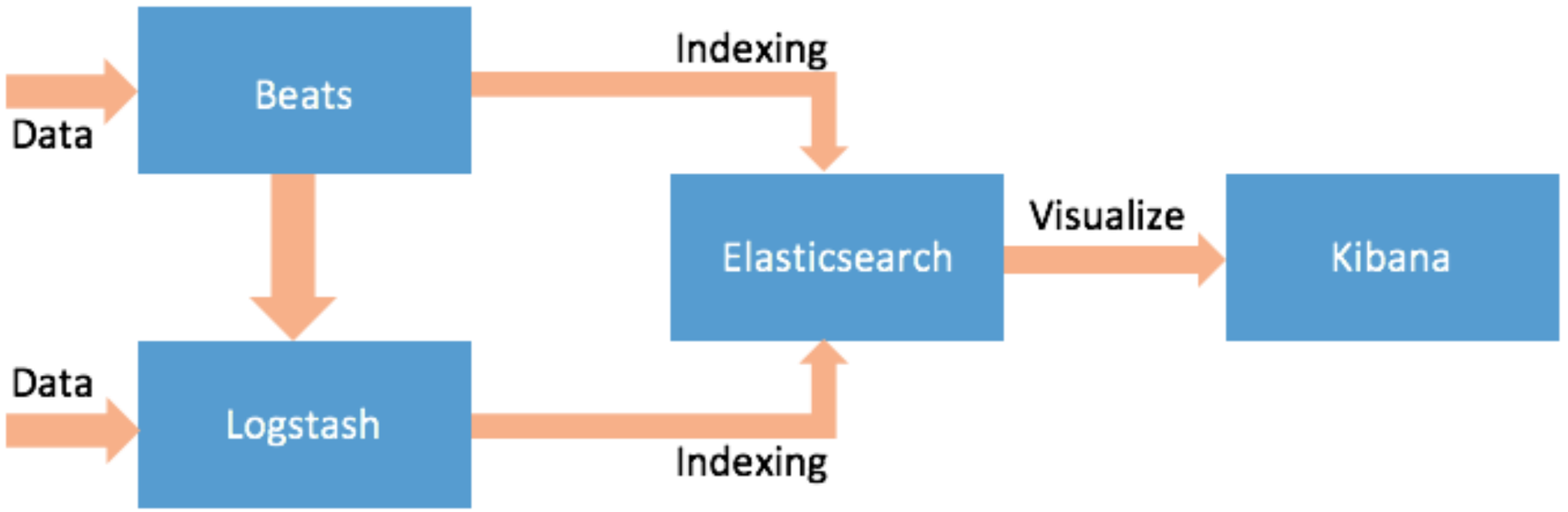
Требования к ЦСЛ

- Горизонтальная масштабируемость
- Надежность (отсутствие потери логов)
- Близость к real-time
- Должна быть недорогой

Примеры ЦСЛ

- Open source: Elastic Stack, Graylog 2
- SaaS: Splunk, Loggly, Papertrail
- Cloud Platform Service: Stackdriver Logging (GCP), CloudWatch (AWS)

Elastic Stack



Логирование приложения

Какую активность приложения логировать?

- Запросы и ответы
- Ошибки
- Вызовы ко всем внешним сервисам и API
- Бизнес события: создание пользователя, платеж
- Время чтения/записи к БД
- etc

Библиотеки для логирования

- Log4j
- Structlog
- Lograge
- etc

Логирование событияпример

```
$ docker-compose up post

post_1          | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)post_1 | * Restarting with stat
post_1          |   * Debugger is active! post_1   | * Debugger
PIN: 330-452-208
post_1                          | 172.21.0.5 - - [02/Nov/2017 16:03:27] "GET /healthcheck HTTP/1.1"  200  -
post_1                          | 172.21.0.5 - - [02/Nov/2017 16:03:30] "GET /healthcheck HTTP/1.1"  200  -
post_1                          | 172.21.0.5 - - [02/Nov/2017 16:03:33] "GET /healthcheck HTTP/1.1"  200  -
post_1                          | 172.21.0.5 - - [02/Nov/2017 16:03:36] "GET /healthcheck HTTP/1.1"  200  -
post_1                          | 172.21.0.5 - - [02/Nov/2017 16:04:06] "GET /healthcheck HTTP/1.1"  200  -
post_1          | 172.21.0.5 - - [02/Nov/2017 16:04:08] "POST /add_post HTTP/1.1" 200 -
```

Изменим формат лога в коде приложения

```
@app.route("/add_post", methods=['POST'])def
add_post():
        try:
            title = request.values.get("title")link =
            request.values.get("link")
            created_at = request.values.get("created_at")except
        Exception as e:
            log.warning('bad input data: {}'.format(request.values))return 'ERROR'
        try:
            mongo_db.insert({"title": title, "link": link, "created_at": created_at,"votes": 0})except Exception as e:
        log.error("post.created because of {}".format(str(e)), failed=True, title=title, link=link)return 'ERROR'
else:
        POST_COUNT.inc()
        log.info("post.created", failed=False, title=title, link=link)return 'OK'
```

Логирование событияпример

```
$ docker-compose up post

post_1                  | * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)post_1 | * Restarting
with stat
post_1                  |   * Debugger is active! post_1     | *
Debugger PIN: 330-452-208
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:23:48] "GET /healthcheck HTTP/1.1"  200 -
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:23:51] "GET /healthcheck HTTP/1.1"  200 -
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:23:54] "GET /healthcheck HTTP/1.1"  200 -
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:23:57] "GET /healthcheck HTTP/1.1"  200 -
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:24:00] "GET /healthcheck HTTP/1.1"  200 -
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:24:03] "GET /healthcheck HTTP/1.1"  200 -
post_1                                  | 172.21.0.5 - - [02/Nov/2017 16:24:06] "GET /healthcheck HTTP/1.1"  200 -
post_1                  | 2017-11-02 16:24.07 post.created failed=False link=https://github.com/hynek/structlog
title=Structlog is awesome!
```

Как получать нужную информацию из логов?

2) Использовать структурированный формат логовсогласно формату используемой системы логирования

3) Парсить существующущий формат логов иизвлекать нужную информацию

Примеры формата логов

Одна строка - требует парсинг для извлечения нужной информации:

"Started GET "/" for 127.0.0.1 at 2015-12-10 09:21:45 +0400"

Логи пишутся в формате JSON, который понимает системалогирования (парсинг поля не требуется)


```
{
"method":"GET",
"path":"/users",
"format":"html",
"controller":"users",
"action":"index",
"status":200,
"duration":189.35,
"view":186.35,
"db":0.92,
"@timestamp":"2015-12-11T13:35:47.062+00:00",
"@version":"1"
```

Логирование и Docker.Distributed tracing.

План

- Сбор логов с Docker контейнеров
- Distributed tracing
- Zipkin

Как собрать логи приложения в Docker?

- Docker logging drivers
  Sidecar контейнеры
- Писать из приложений в ЦСЛ

Logging drivers

- Собирают STDOUT, STDERR выводы контейнеров
- Добавляют метаинформацию
- Формируют сообщения для отправки в ЦСЛ
- Есть несколько встроенных, также подключаются в качестве плагинов
- По умолчанию используется Json-File driver

Logging drivers

Драйвер по умолчанию задается в файле:

/etc/docker/daemon.json

```
{
  "log-driver": "json-file"
}
```

Определение драйвера для отдельного контейнера --log-driver:

```
docker run \
    --log-driver json-file \
    alpine echo hello world
```

Json-file

- Являет драйвером по умолчанию
- Хранит логи о каждого контейнера в отдельном JSON файле
- Логи удаляются вместе с контейнером

```
>> tail -f $(docker inspect -f {{.LogPath}} dockerpuma_ui_1)
```


{"log":"* Min threads: 0, max threads: 16\n","stream":"stdout","time":"2017-11-06T12:24:39.094400172Z"}
{"log":"* Environment: development\n","stream":"stdout","time":"2017-11-06T12:24:39.094402562Z"}
{"log":"* Listening on tcp://0.0.0.0:9292\n","stream":"stdout","time":"2017-11-06T12:24:43.311598353Z"}
{"log":"Use Ctrl-C to stop\n","stream":"stdout","time":"2017-11-06T12:24:43.311883143Z"}
{"log":"89.106.198.71 - - [06/Nov/2017:12:25:02 +0000] \"GET / HTTP/1.1\" 200 -
0.0323\n”,"stream":"stderr","time":"2017-11-06T12:25:02.162951447Z"}
{"log":"89.106.198.71 - - [06/Nov/2017:12:25:02 +0000] \"GET /favicon.ico HTTP/1.1\" 404 471
0.0013\n","stream":"stderr","time":"2017-11-06T12:25:02.613096563Z"}

Journald-драйвер

Пишет в общий системный журнал

- бует тюнинга

>> journalctl CONTAINER_NAME=dockerpuma_ui_1

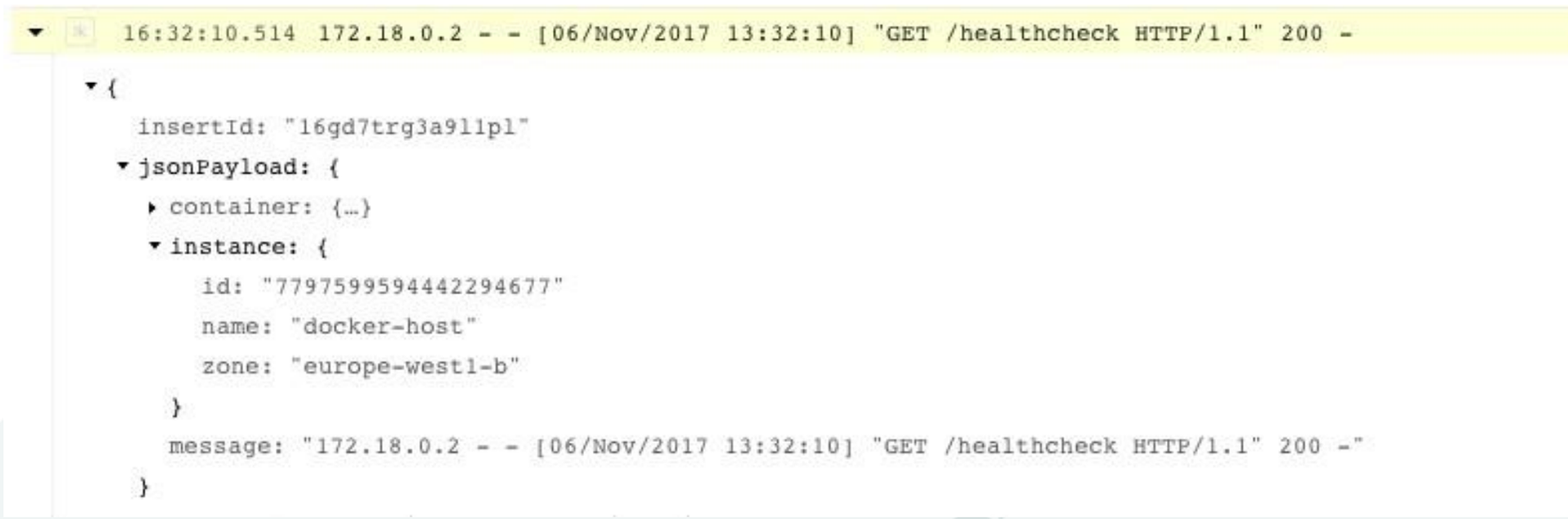
```
Nov 06 13:14:41 docker-host dockerd[1264]: * Min threads: 0, max threads: 16
Nov 06 13:14:41 docker-host dockerd[1264]: * Environment: development
Nov 06 13:14:45 docker-host dockerd[1264]: * Listening on tcp://0.0.0.0:9292
Nov 06 13:14:45 docker-host dockerd[1264]: Use Ctrl-C to stop
Nov 06 13:15:23 docker-host dockerd[1264]: 89.106.198.71 - - [06/Nov/2017:13:15:23 +0000] "GET / HTTP/1.1
200 - 0.0378
```

GCP-драйвер

- Работает только в GCP
- Автоматически пишет логи в Google Stackdriver
- Не совместим с чтением с помощью docker logs

/etc/docker/daemon.json

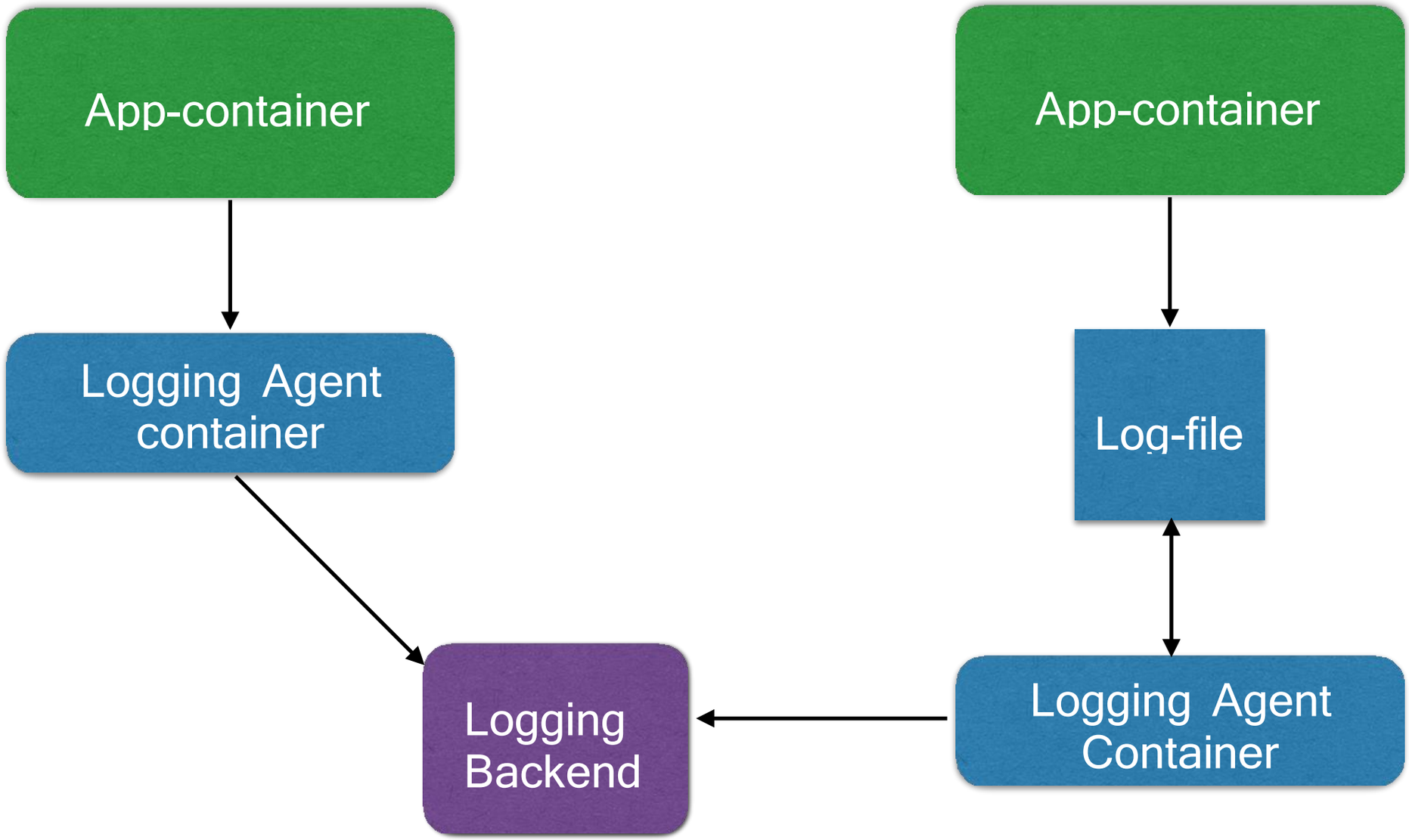
```
{
  "log-driver": "gcplogs",
  "log-opts": {
    "gcp-meta-name": “docker-project-name”
  }
}
```



Другие драйверы

- Syslog
- Gelf (Graylog Extended Format)
- Splunk
- Fluentd
- etc

Sidecar контейнеры



Логирование приложения

1) STDOUT, STDERR

4) Если приложение не умеет писать в STDERR, STDOUT:

```
ln -sf /dev/stdout /var/log/nginx/access.log \
&& ln -sf /dev/stderr /var/log/nginx/error.log
```

5) Напрямую в ЦСЛ (ELK, Graylog, Splunk, etc)

6) в Bind-Mount Volume (осторожно, логов может быть много):

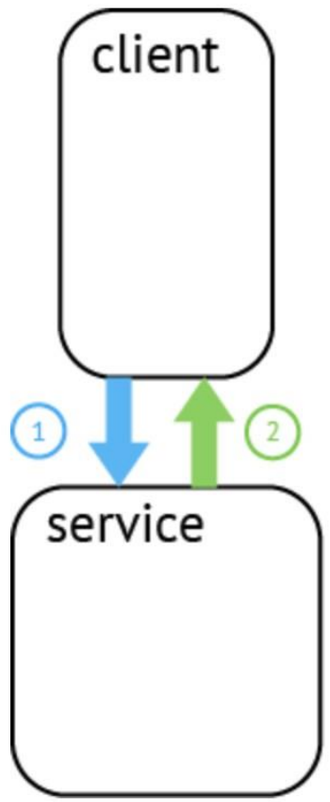
```
docker run -v /var/log/nginx.log:/var/log/nginx.log
```

7) Не пишем логи в файлы внутри контейнера!

Distributed tracing

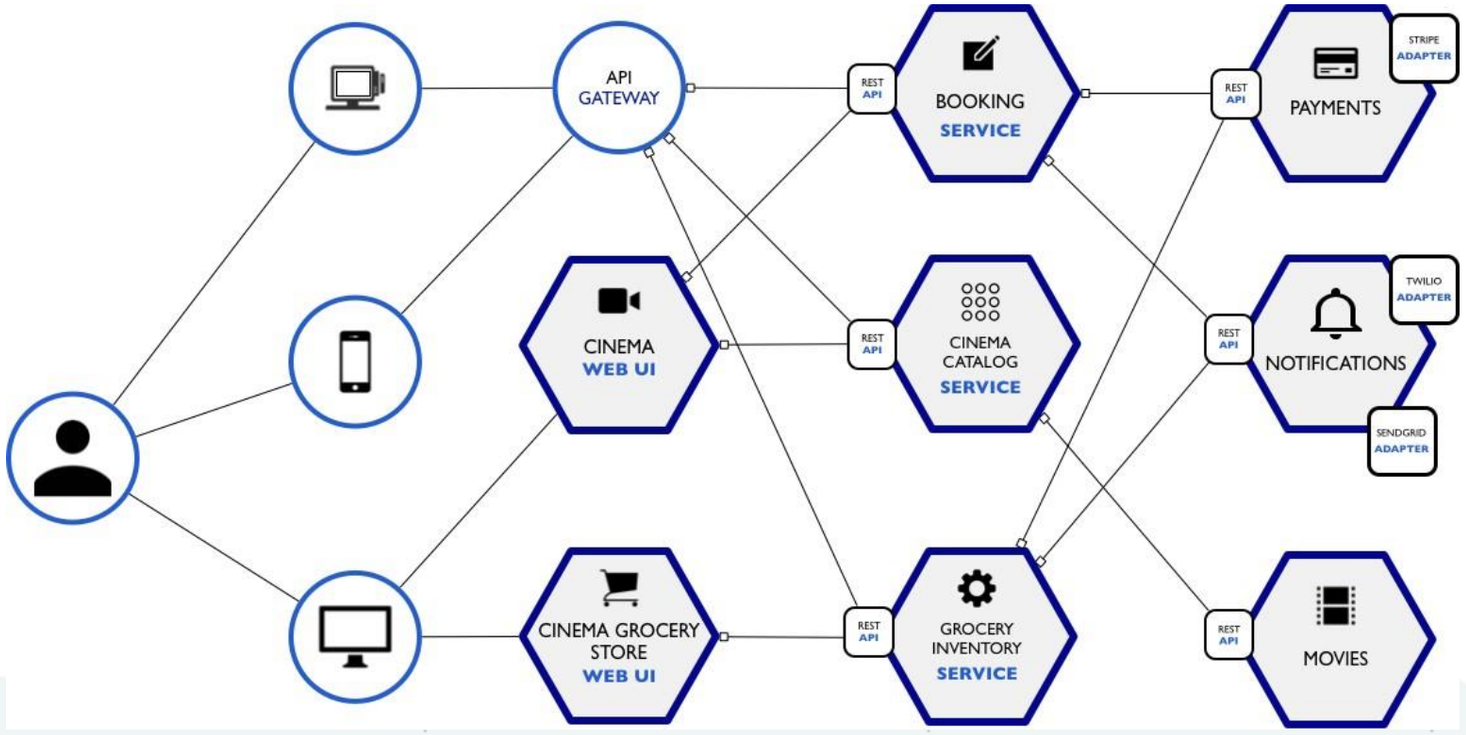
Trace

- Описывает историю одного события (в распределенной системе)



Проблемы микросервисов

- Запрос от клиента проходит через несколько микросервисов
- Нет видимости, как работает система в целом
- Трудности дебага Latency



Distributed tracing

- Позволяет представить графы задержки запросов (traces) в реальном времени
- Анализ графов помогает найти причины долгих запросов

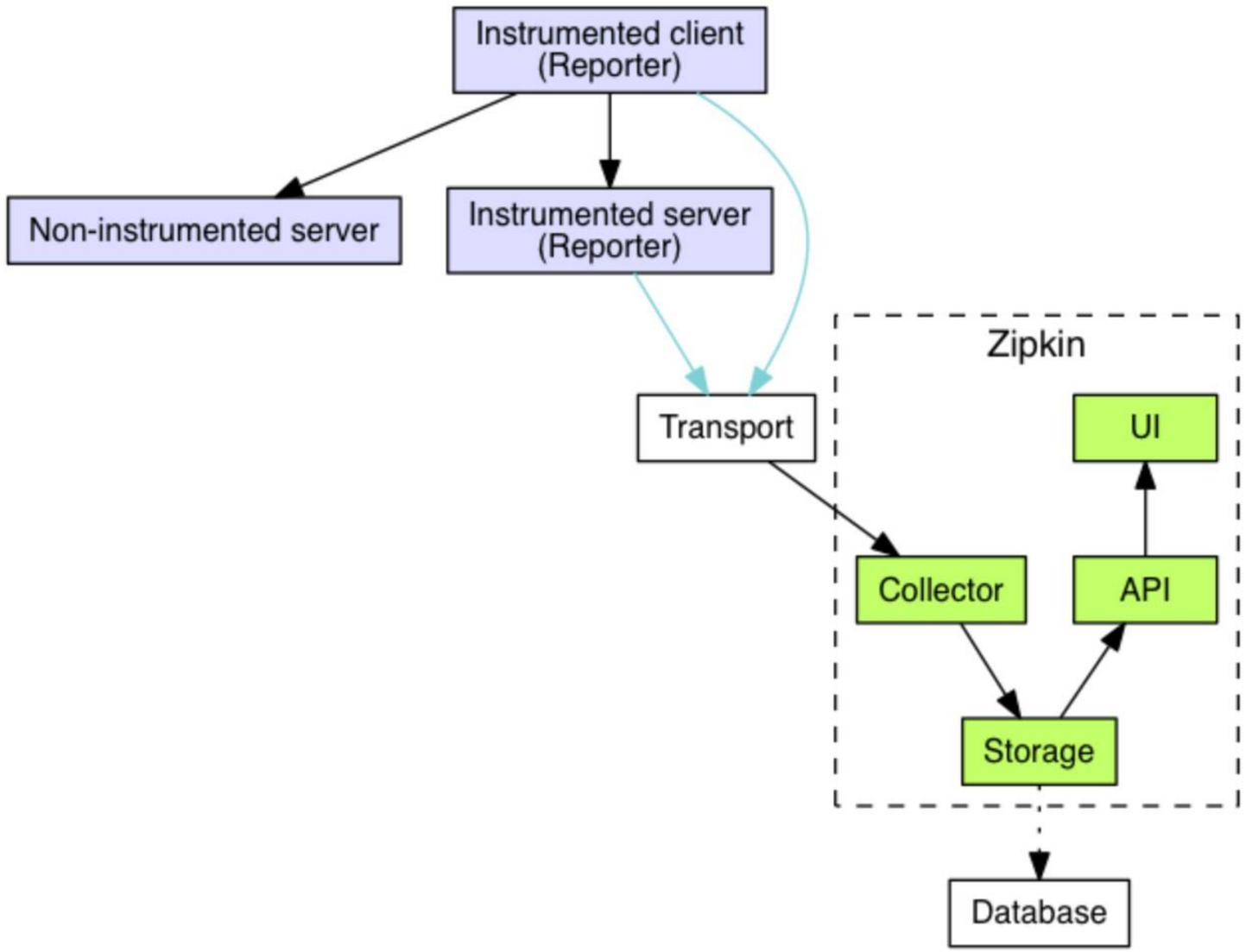
Системы трасировки

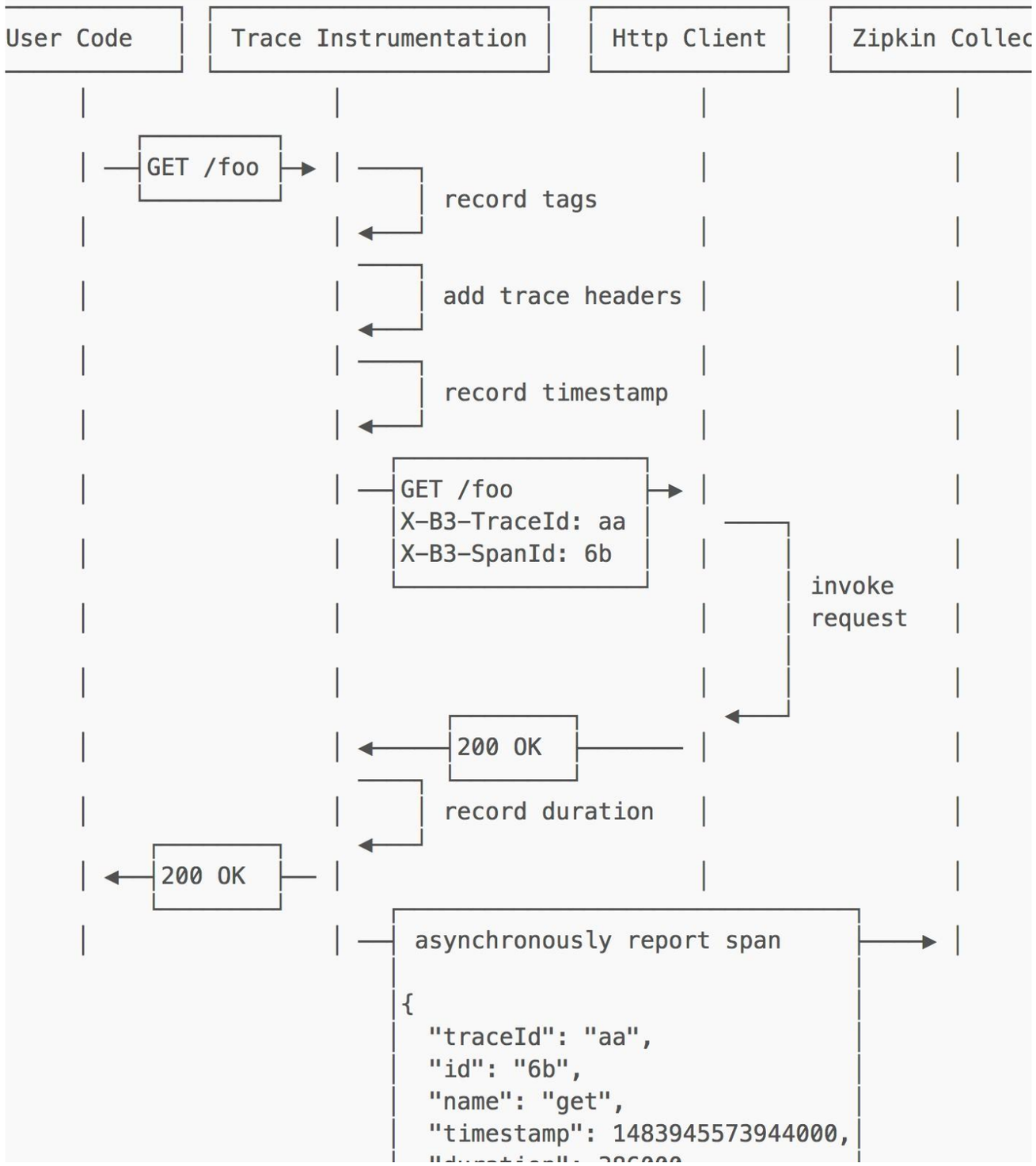
- Jaeger
- Appdash
- Zipkin

Zipkin

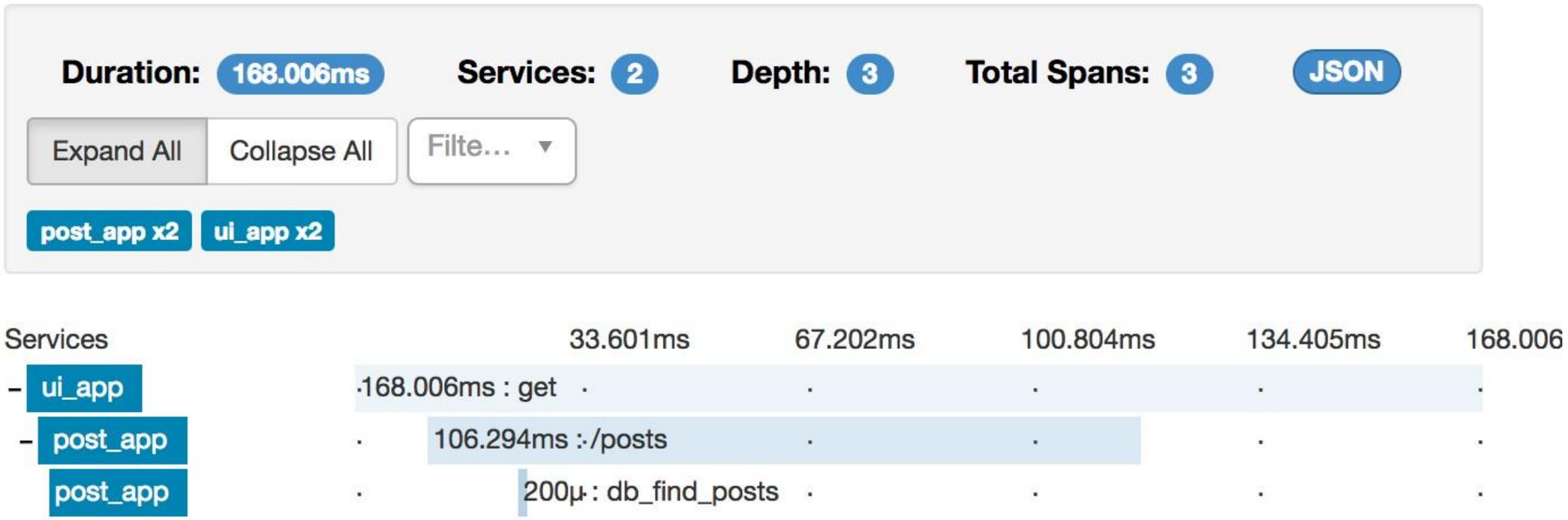
- 2012 год
- Разработан компанией Twitter
- Open source (https://github.com/openzipkin/zipkin)

Архитектура




User Code
Trace Instrumentation
Http Client
Zipkin Collec
GET /foo
record tags
add trace headers
record timestamp
GET /foo
X-B3-TraceId: aa
X-B3-SpanId: 6b
invoke
request
200 OK
record duration
200 OK
asynchronously report span
{
"traceId": "aa",
"id": "6b",
"name": "get",
"timestamp": 1483945573944000,

Пример Zipkin



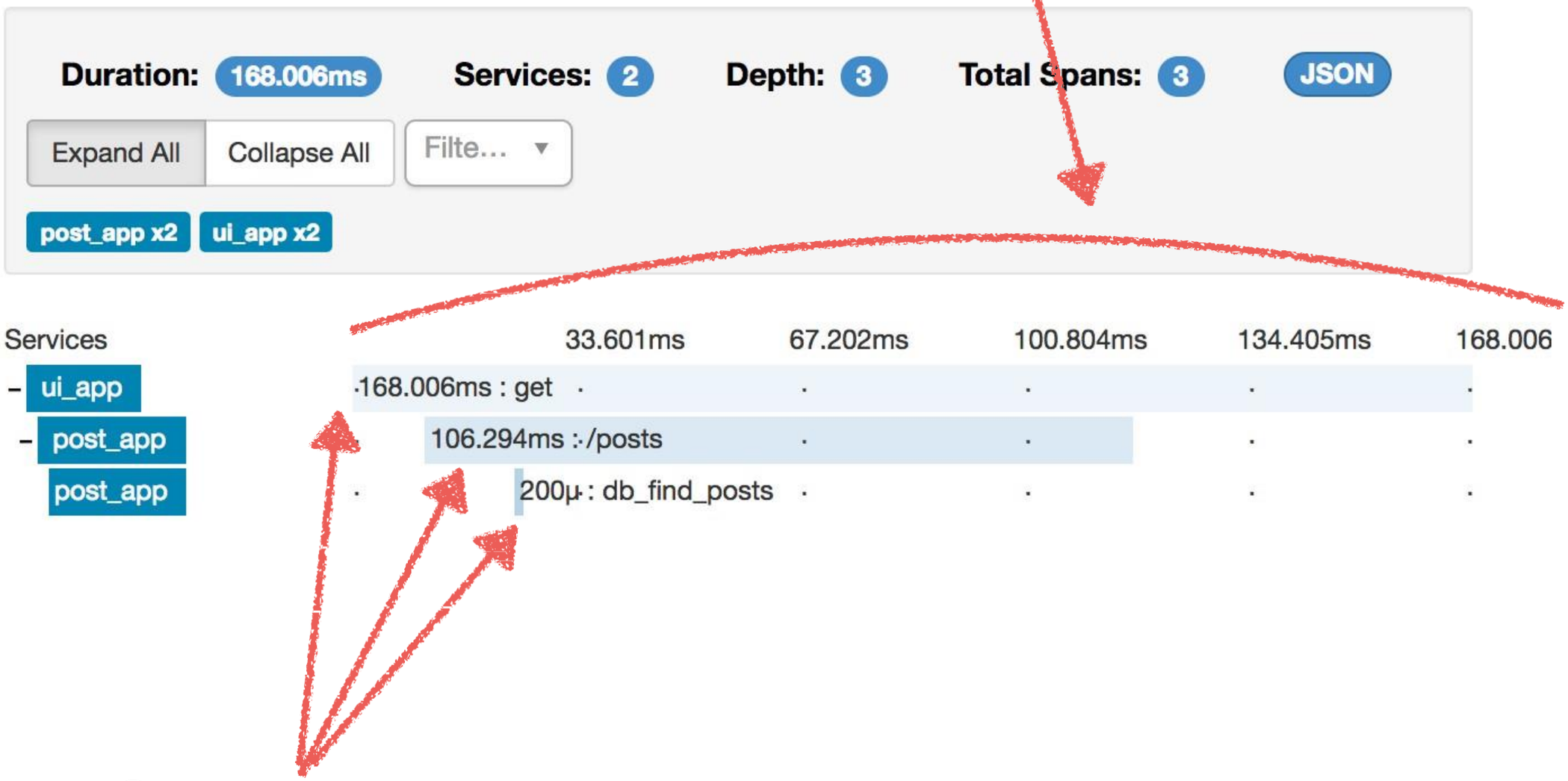
Основные понятия

- Span - одна завершившаяся операция в рамках запроса, содержит события и тэги

- Trace - граф задержки всего запроса, состоит из span-ов

- Tracer - библиотека в коде приложения, которые позволяют собирать и отправлять информацию о span-ах

Trace

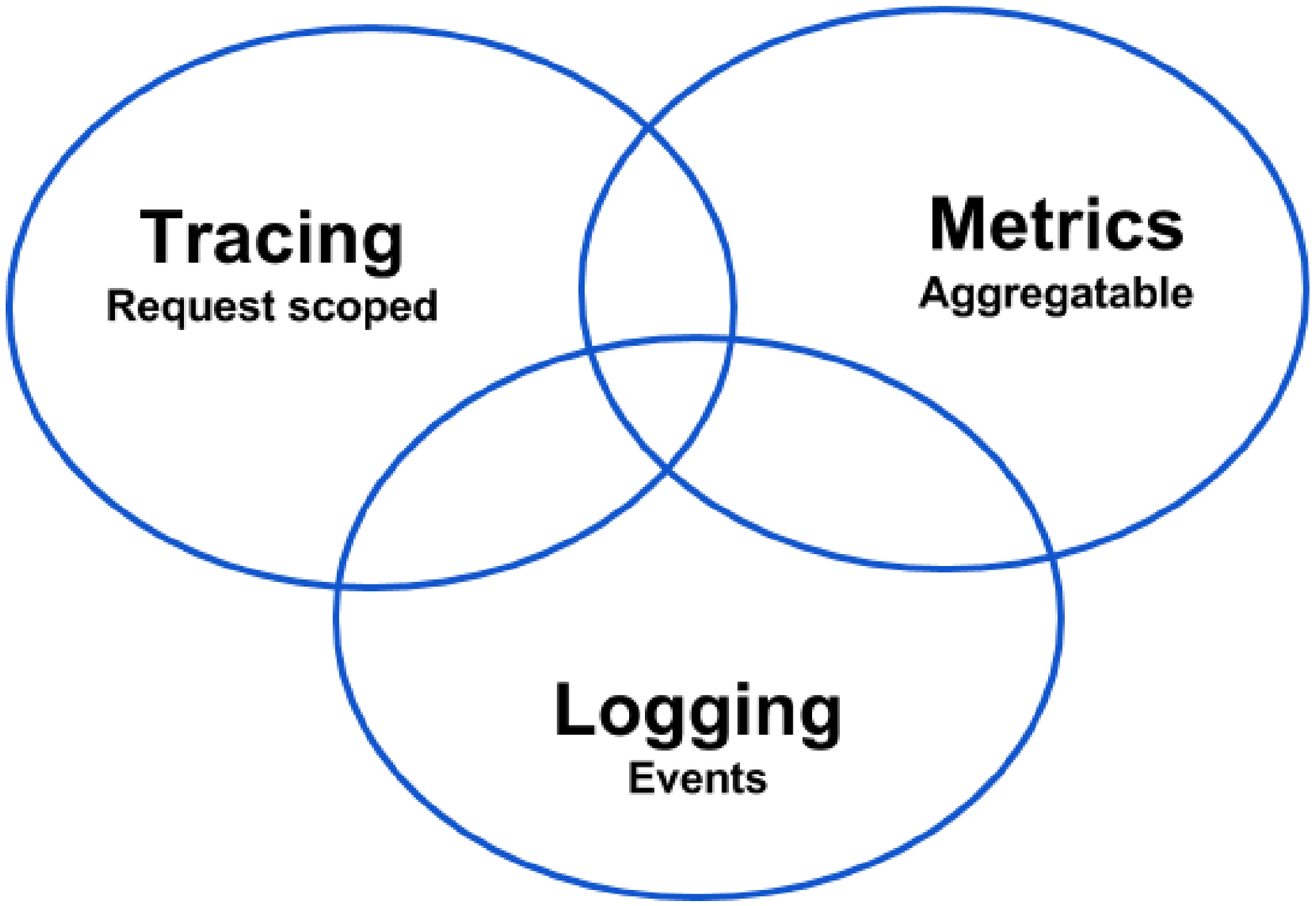

Duration: 168.006ms
Services: 2
Depth: 3
Total Spans: 3
JSON
Expand All
Collapse All
Filte…
post_app x2
ui_app x2
Services
33.601ms
67.202ms
100.804ms
134.405ms
168.006
ui_app
168.006ms : get
post_app
106.294ms : /posts
post_app
200µ : db_find_posts

Spans

Three pillars of observability

- Metrics - измерение параметров работы системы со временем
- Logging - журнал событий
- Distributed tracing - журнал событий с причино-следственной связью

Разные задачи, разные инструменты



источник: Peter Bourgon's post

Спасибо за внимания