

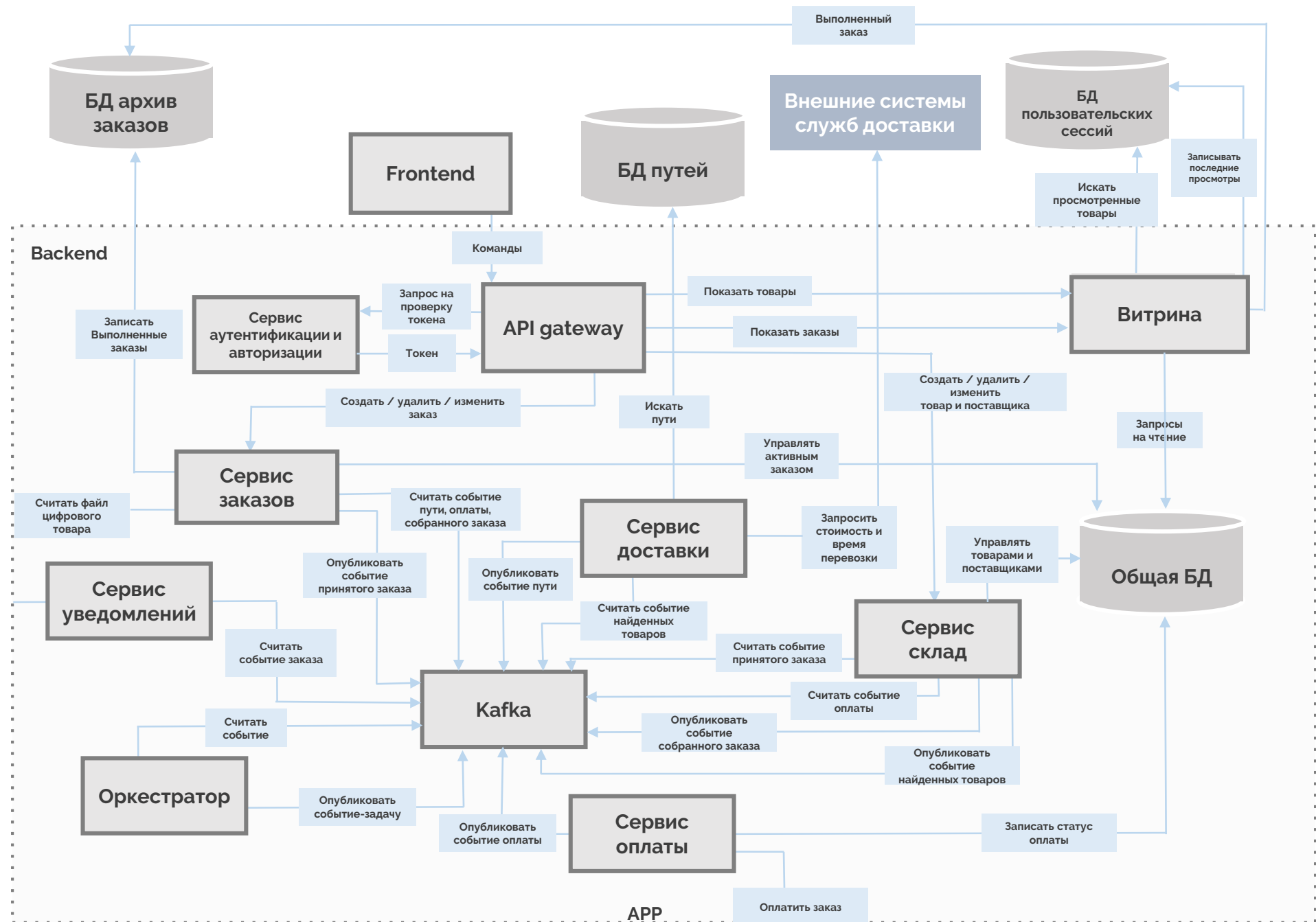
Observability - «наблюдаемость»

Логирование, трейсинг, мониторинг, алертинг

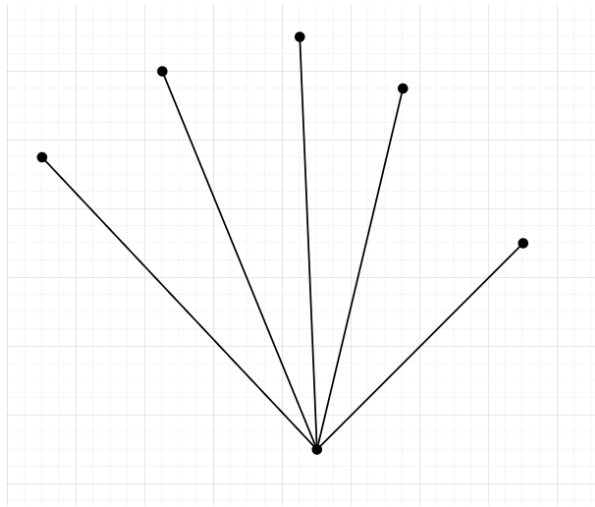
ФИТ Лекция №16. 25'

Типичная схема приложения из 5 сервисов

Если каждый соединяется с каждым, сколько будет связей?



Вспоминаем графы

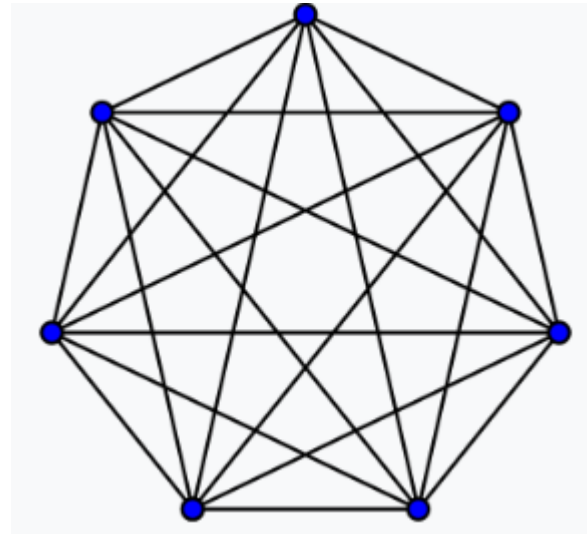


Одна вершина соединена с каждой

$$\text{Ребер} = n - 1$$

Линейная зависимость


Грубо, связей между микросервисами будет где то от «n», до «n²».
Каждая связь усложняет приложение и является точкой отказа.



Полный граф с n вершинами

$$\text{Ребер} = n * (n - 1) / 2$$

Квадратичная зависимость



Проблемы микросервисного приложения

- Каждый сервис и каждая связь — потенциальная точка отказа
- Сложно отлаживать - множество сервисов, распределенные ошибки, K8s как черный ящик
- Сложно отследить проблему в цепочке сервисов
- Зависимость от сети - 10 микросервисов могут дать задержку в 100мс
- Динамичность K8s - сервисы могут перезагружаться и масштабироваться
- Сложность сбора данных - логи разбросаны по нодам и контейнерам



Решение проблемы – Observability

Observability (наблюдаемость) — способность понимать внутреннее состояние системы на основе данных, которые она производит.

В контексте программного обеспечения, observability, с помощью логов, метрик, алертов и трейсов, помогает отслеживать работу системы, диагностировать проблемы и оптимизировать её производительность.

Observability – основные компоненты

Логирование (Logging)

Запись событий в текстовом виде. Логи могут содержать информацию о запросах, ошибках, предупреждениях или других важных действиях.

Трассировка (Tracing)

Способ отслеживания пути запроса через разные части системы, особенно в распределённых архитектурах. Трассировка показывает, как запрос перемещается между сервисами и где возникают задержки или сбои.

Метрики (Metrics)

Числовые показатели, которые отражают состояние и производительность системы. Примеры метрик: время ответа сервера, количество запросов в секунду, использование процессора или памяти. Метрики позволяют в реальном времени следить за работой системы и быстро замечать аномалии.

Оповещения (Alerting)

Способ автоматического оповещения о критических событиях или аномалиях в работе системы. Является ключевым компонентом систем мониторинга, позволяя быстро реагировать на сбои, угрозы безопасности или нарушения производительности.

Observability – инструменты

Инструментов много,
факторы успеха:

- Подобрать инструменты
- Согласовать метрики
- Настроить инфраструктуру
- Настроить процессы
- Обучить людей 😊

System		Logs		Application		Errors	
Nagios	Datadog	elastic	splunk	APPDYNAMICS	Scout24	Rollbar	RAYGUN
Sensu	opsview	sumologic	logz.io	solarwinds appops	INSTANA	bugsnag	Airbrake.io
Site24x7	zenoss	papertrail	logentries	GreenDash	New Relic	OverOps	Honeybadger
APIFORTRESS	LogicMonitor	XPLG	graylog	Wily	Stackify	Synthetic	
ConnectWise	ConnectWise	Logscale	fluentd	dynatrace	tideways		
WhatsUpGold	DataLoop	SCALYR	sematext	Time Series		gomez	solarwinds pingdom
icinga	spiceworks	LOGGLY	MICRO FOCUS	STATSD	librato	Panopta	Apica
Suites		Networks		collectd	coDT	SMARTBEAR	keynote
MONIT	Outlyer	SevOne	riverbed	signal fx	RRDtool	ThousandEyes	
Ganglia	ZABBIX	elastic	Corvil	Grafana	MUNIN	Mobile	
PAESSLER	ATERA	VIAXI	ca	Prometheus	graphite	New Relic MOBILE	splunk MINT
		ExtraHop	NETSCOUT	CherryTree	influxdata	APPDYNAMICS MOBILE	crashlytics
		Anomaly		MRTG	WAVEFRONT by VMware	dynatrace MOBILE	CRITTERCISM
		NETUITIVE	GRIOK	Specialized		Web	
		Etsy skyline	prelert	vmware	CloudWatch	MONIT	WebSitePulse
				Stackdriver	sysdig		

Логирование – история появления

Log – журнал событий, происходит от судовых журналов (**logbooks**), в него записывали маршрут и скорость.

Скорость измеряли с помощью веревки, узлов и полена (англ. «**log**»). К полену привязывали веревку с узлами, бросали за борт и определяли скорость по количеству узлов, разматывавшихся за определенное время. С тех пор скорость в морских узлах и меряют 😊.

Появление в ИТ (1950–1970-е) – в эпоху мейнфреймов возникла необходимость отслеживания выполнения пакетных заданий, диагностики сбоев и учета использования дорогостоящего оборудования.

Первые прототипы логов представляли собой распечатки на бумаге, где фиксировались:

- время начала и завершения задач
- коды ошибок
- использование процессорного времени
- обращения к периферийным устройствам

Логирование – появление syslog

Unix разработан в конце 1960-х — начале 1970-х годов в Bell Labs, его ранние версии, Version 6 (1975), не включали централизованную систему логирования.

Каждый процесс, самостоятельно записывал свои события в текстовые файлы или на консоль. Это создавало сложности при анализе логов, особенно в крупных системах.

В 1980-х годах на проекте Sendmail, Эрик Оллман из Беркли, столкнулся с необходимостью стандартизации логирования в распределенных системах. Как часть Sendmail появился сервис Syslog, который предлагал единый стандарт и протокол централизованной записи событий.

В 2001г публикуется RFC 3164, где впервые официально зафиксирован протокол и восемь уровней критичности.



Логирование – категории критичности из RFC

Уровень	Код	Описание
Emergency	0	Система неработоспособна
Alert	1	Требуются немедленные действия
Critical	2	Критические условия
Error	3	Ошибки выполнения
Warning	4	Предупреждения
Notice	5	Важные нормальные события
Informational	6	Информационные сообщения
Debug	7	Отладочная информация

Логирование – эпоха микросервисов

При переходе на микросервисы протокол syslog сталкивается проблемами:

- Логи разбросаны по разным микросервисам, это усложняет их сбор, корреляцию и анализ
- В микросервисах могут быть различные часовые пояса и форматы времени (ISO, Unix timestamp), это затрудняет упорядочивание событий
- Syslog традиционно использует UDP, это не гарантирует доставку сообщений, а TCP и TLS-расширения требуют дополнительной настройки
- Syslog-сообщения не содержат достаточного контекста (например, уникальных идентификаторов запросов), чтобы связать события между сервисами и проследить полный путь запроса (трассировку). Для микросервисов важна возможность связывать логи разных сервисов по единому Request ID, чего в классическом syslog нет
- Микросервисы генерируют значительно больше логов, чем монолиты, это создаёт нагрузку на систему логирования и требует масштабируемых решений



Логирование – что хочется получить

- **Быстрый поиск** – требуется БД с возможностью текстового поиска
- **Безопасность** – разные роли для разных аудиторий
- **Быстро увидеть общую картину** – визуализация и графики по событиям
- **Сохранить место** – ротация данных и архивирование устаревших данных
- **Связать разные данные** – корреляция логов разных микросервисов

Требуются новые продукты для работы с логами.

Логирование – основные инструменты

2006 - коммерческий релиз **Splunk**, создатели Erik Swan, Rob Das и Michael Baum, стал проприетарным стандартом для анализа больших объемом данных и безопасности, коммерческая лицензия

2010 - **ELK Stack** (Elasticsearch, Logstash, Kibana), создатели Shay Banon, Jordan Sissel, Rashid Khalilov, open-source-решение для централизованного управления логами

2012 - релиз **Graylog**, создатель Lennart Koopmann, open-source-решение для централизованного управления логами

Особенности решений:

1) На клиенте само приложение через готовую библиотеку отправляет логи на сервер, либо ставится агент, который собирает данные из текстовых логов, формирует json и отправляет их на сервер

2) На сервере есть middle level который производит фильтрацию, обогащение и трансформацию принятого json, и отправляет его в time series базу

3) На сервере есть web ui который позволяет делать запросы в базу, строить графики, отправлять алерты

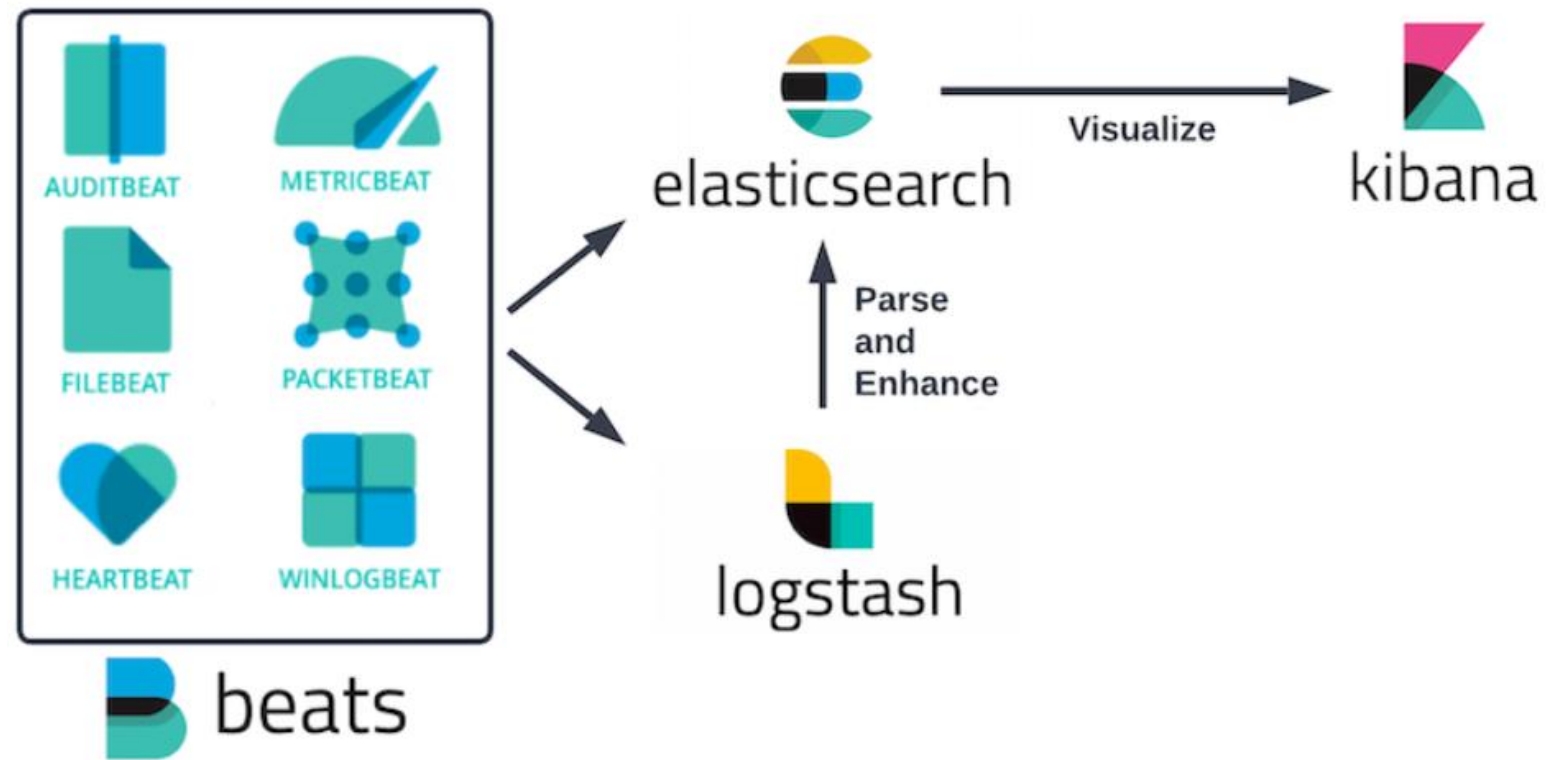
Логирование – ELK Stack

beats – семейство агентов сбора логов (>20 разных)

logstash – инструмент для обработки и преобразования данных

elasticsearch - распределенная поисковая и аналитическая NoSQL timeseries - база данных

kibana - веб-интерфейс для визуализации и анализа данных



Логирование – пример конфигурации winlogbeat

Winlogbeat

Запущен на сервере с windows

Собирает события входа

```
1 winlogbeat.event_logs:
2   - name: Security
3     event_id: 4624, 4625, 4648      # События входа, смены пароля
4     tags: ["auth"]                 # Тег для фильтрации
5   - name: Application
6     level: error, warning, information # Все уровни, кроме verbose
7   - name: System
8     ignore_older: 24h              # Игнорировать старые события
9   - name: Microsoft-Windows-PowerShell/Operational # Логи PowerShell
10
11 processors:
12   - add_host_metadata:             # Метаданные хоста
13     netinfo.enabled: true          # Сбор IP-адресов
14   - add_fields:                    # Кастомное поле
15     fields:
16       environment: "production"
17
18 output.elasticsearch:
19   hosts: ["https://es-node1:9200", "https://es-node2:9200"] # Отказоустойчивость
20   username: "winlog_user"
21   password: "${ES_PASSWORD}"      # Использование переменной среды
22   ssl:
23     verification_mode: "certificate" # Проверка сертификата
24     index: "winlog-%[agent.version]}" # Динамическое имя индекса
25
26 setup.ilm.enabled: true           # Включить управление жизненным циклом
27 logging.level: info              # Базовый уровень логирования
```

Логирование – пример конфигурации logstash

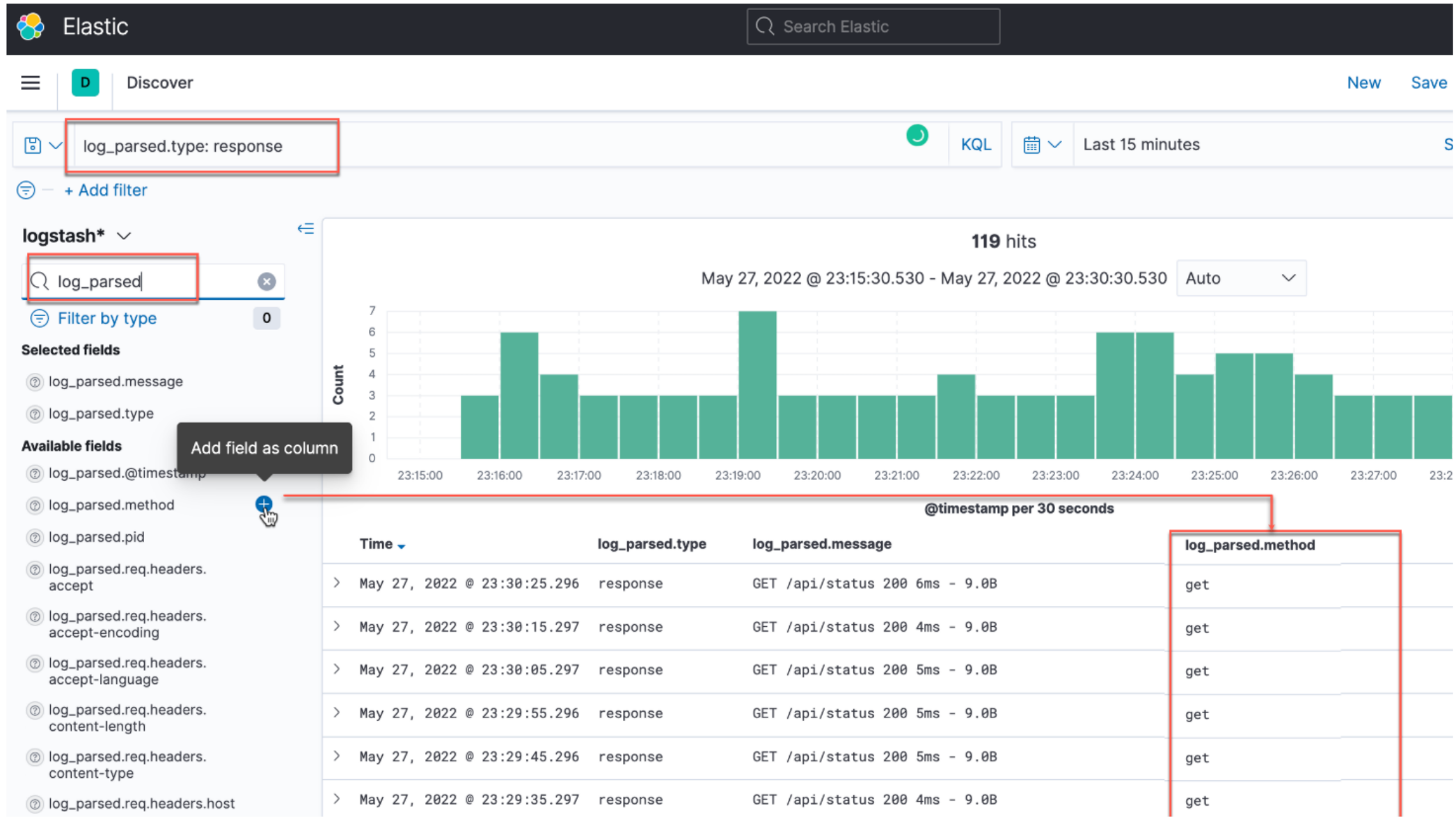
Logstash

Запущен на сервере ELK

Обогащает событие и удаляет
лишние поля

```
1 input {
2   beats {
3     port => 5044 # Winlogbeat отправляет сюда
4   }
5 }
6
7 filter {
8   # Обогащение: геоданные по IP, используется плагин geo для logstash с локальной GeoLite2 базой
9   if [winlog][event_data][IpAddress] {
10     geoip {
11       source => "[winlog][event_data][IpAddress]"
12       target => "geo"
13     }
14   }
15
16   # Трансформация: добавление метки для событий входа
17   if [winlog][event_id] == 4624 {
18     mutate {
19       add_field => { "event_type" => "Успешный вход" }
20     }
21   }
22
23   # Удаление лишних полей
24   mutate {
25     remove_field => ["@version", "tags"]
26   }
27 }
28
29 output {
30   elasticsearch {
31     hosts => ["http://elasticsearch:9200"]
32     index => "winlog-%{+YYYY.MM.dd}" # Индекс с датой
33   }
34   stdout { codec => rubydebug } # Для отладки (опционально)
35 }
```

Логирование – пример интерфейс kibana




Логи́рование – другие инструменты

- **Fluentd** - OpenSource CNCF коллектор логов с поддержкой JSON, буферизацией и 500+ плагинами. Используется в распределенных системах (Kubernetes, Docker)
- **Loggly** - облачный SaaS-сервис для анализа логов с готовыми дашбордами и алертами. Подходит для малых проектов
- **Loki** — OpenSource решение для сбора, хранения, стек Grafana Labs
- **Sumo Logic** - платного обучения. Фокус на безопасности (SIEM) и бизнес-аналитике
- **Datadog** - облачная платформа для мониторинга IT-инфраструктуры, объединяет метрики, логи и трейсы

Облака:

- **AWS** - CloudWatch - сбор, анализ, визуализация логов и метрик
- **Azure** - Azure Monitor - сбор логов и метрик, Application Insights - мониторинг производительности, Azure Log Analytics - анализ через KQL.
- **Google Cloud** - Cloud Logging - сбор логов и метрик, Cloud Monitoring - визуализация и алерты.



Логирование – типы и аудитория

Системное

- Инструменты – rsyslogd, journald
- Аудитория – admins, devops

Прикладное

- Инструменты – ELK \ Splunk \ Graylog и др.
- Аудитория – admins, devops, support, developers, testers, analytics, business

Логирование – best practices для разработчика

1. Реализация **централизованного хранения** логов
2. Лог должен содержать **обязательные данные**:
 - Время события
 - Привязка к пользователю
 - Имя приложение
 - Трейс
 - Сервер, на котором событие зафиксировано
 - Тело сообщения
 - Уровень логирования и имя логера
3. Лог должен быть **поведенческим или сценарным**. Он должен фиксировать:
 - Все шаги пользователя
 - Все шаги внутри системы
 - Все что приходит на вход и уходит в ответ (в том числе в сторонние системы)
4. Лог должен фиксировать **время выполнения запросов** пользователя на всех этапах
5. Лог должен фиксировать **факт релиза** (запуск новой версии микросервиса) **и перезапуск** после сбоя старой версии



Логирование - резюме

Логирование стало неотъемлемой частью современной практики DevOps и Observability, важно для аудита безопасности и детализации событий внутри сервиса.

Проблемы, которые помогло решить логирование:

- оперативная отладка и уменьшение простоев
- мониторинг производительности и своевременное выявление регрессий,
- повышение уровня безопасности и расследование инцидентов,
- соблюдение регуляторных требований и аудит,
- эффективная коллаборация и централизованное управление логами.

Трассировка – история появления

Вопрос – инструменты подобрали, логи настроили, людей обучили, **счастье настало?**

Ответ – проблемы остались и стали еще заковыристее:

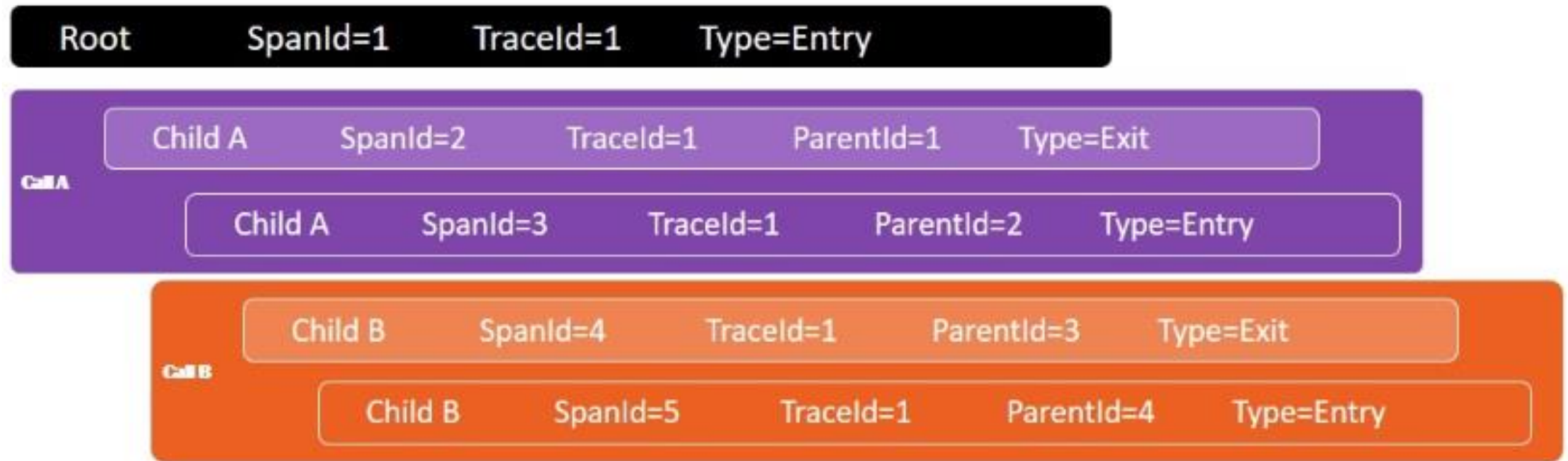
- запросы проходят через множество сервисов, на локализацию ошибки уходит много времени квалифицированных разработчиков, это плохо, хочется спихнуть такие проблемы на поддержку либо devops
- распределенные транзакции создают задержки, разработчик, даже грамотный, такие проблемы не сможет решить, привлекается такой же грамотный devops, опять плохо, опять хочется спихнуть такие проблемы, отгадайте куда?

Решение – Distributing Tracing (трассировка) – техника сбора и корреляции данных о выполнении запросов в распределённых приложениях.

Трассировка – история появления

- 2008** - Льюис Кирн (Lew Cirne), создает облачную платформу **New Relic**, специализирующуюся на APM (Application Performance Monitoring), сегодня платформа считается одной из ведущих в сфере observability
- 2010** - **Google Dapper** популяризирует концепции **trace-id** и **span-id** для мониторинга распределённых систем
- 2012** - **Twitter** запускает **Zipkin**, реализующий аналогичные идеи в open-source
- 2016** - **CNCF** создает стандарт **OpenTracing** независимый от поставщиков (vendor-neutral) для распределённой трассировки (distributed tracing)
- 2017** - **Uber** создает **Jaeger**, далее передан в **CNCF**
- 2018** - **Google** открывает проект **OpenCensus**, фреймворк и набор библиотек, для сбора метрик и реализации распределённой трассировки (distributed tracing) в приложениях, и передает в **CNCF**
- 2019** - **CNCF** создает **OpenTelemetry**, стандарт и фреймворк который объединяет **OpenTracing** и **OpenCensus**, устанавливает единые правила для генерации и передачи телеметрии, позволяет избежать привязки к конкретным продуктам и обеспечивает совместимость между различными системами мониторинга, стандартизирует форматы **trace-id** и **span-id** для корреляции запросов в микросервисах
- 2020** - **Grafana Labs** создает **Tempo** - лёгкий бэкенд трассировки с поддержкой **OpenTelemetry**

Трассировка – определения



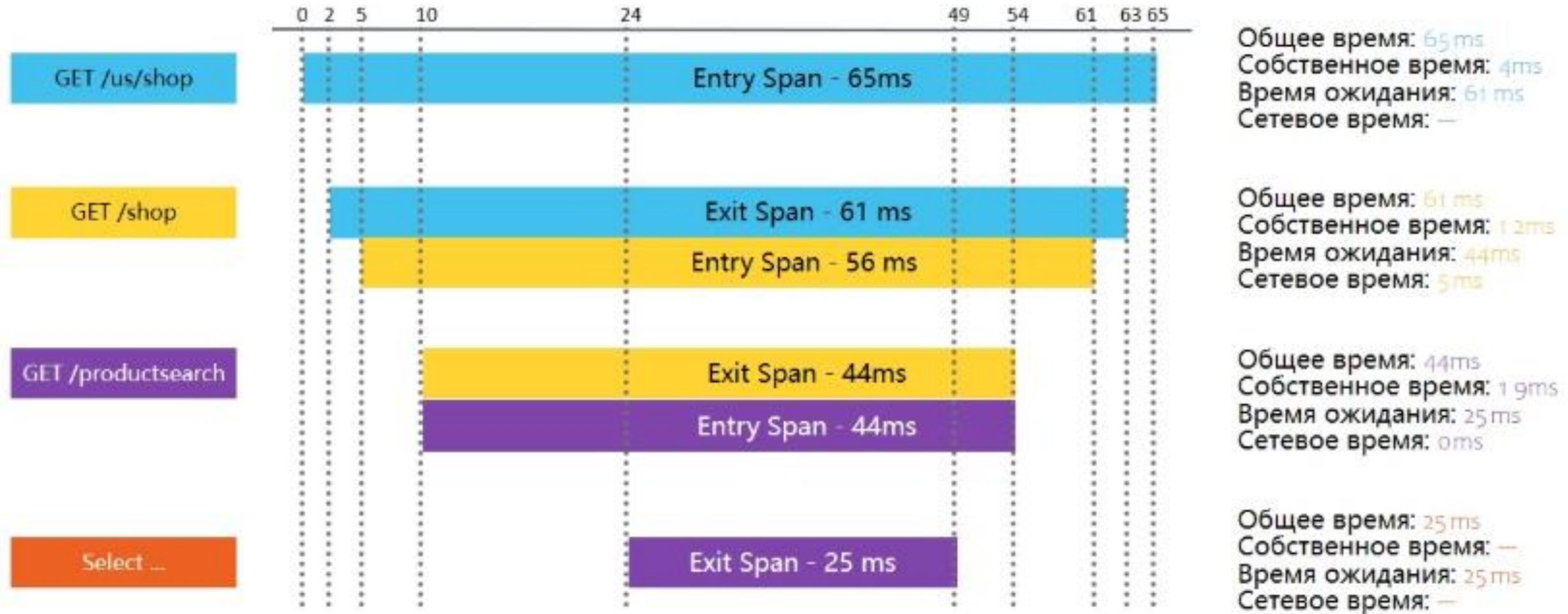
Trace (трасса) – цепочка вызовов (calls) состоит из одного или нескольких вызовов

Call (вызов) – обычно это запрос между двумя сервисами, но может быть и комбинация запросов

Span (сегмент) – это часть вызова, каждый вызов состоит из одного или нескольких spans, есть специализированные entry и exit сегменты, которые отмечают начало и конец вызова

TraceContext – описывает, какой из трейсов родительский, какой — дочерний, как они связаны друг с другом, склеив два разных спана, мы можем понять, что это один и тот же вызов, хранится и передается в HTTP заголовках

Трассировка – как работает



Трассировка – пример HTTP заголовков

- HTTP хидер traceparent содержит только trace-id и текущий span-id
- Связь между traceid, parent spanid и spanid хранится внутри Jaeger
- В UI Jaeger читает эти поля и выстраивает иерархию «деревя»
parentspanid → spanid

The diagram illustrates two HTTP requests with annotations for traceparent and tracestate headers. Red arrows point from Russian text labels to specific parts of the headers.

Request 1 (Parent):

```
GET /api/data HTTP/1.1
Host: parent.internal
traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-00f067aa0ba902b7-01
tracestate: vendorX=opaqueValue
```

Annotations for Request 1:

- запрос на родителя** (request to parent) points to the method and path.
- trace-id не меняется** (trace-id does not change) points to the first 32 hex characters of the traceparent header.
- span-id** points to the 16 hex characters following the hyphen in the traceparent header.
- версия протокола** (protocol version) points to the '00' at the start of the traceparent header.
- флаг семплирования** (sampling flag) points to the last 1 hex character of the traceparent header.

Request 2 (Child):

```
POST /process HTTP/1.1
Host: child.internal
traceparent: 00-4bf92f3577b34da6a3ce929d0e0e4736-7af7651916cd43dd-01
tracestate: vendorX=opaqueValue
```

Annotations for Request 2:

- запрос от родителя на дочерний** (request from parent to child) points to the method and path.
- span-id поменялся** (span-id changed) points to the 16 hex characters following the hyphen in the traceparent header.

Трассировка – что уходит в Jaeger

- Запрос через OTLP (OpenTelemetry Protocol) уходит в Jaeger Collector на стандартный порт 4318
- В качестве транспорта используется HTTP/JSON Protobuf (OTLP/HTTP), где тело запроса — JSON

```
{
  "resourceSpans": [
    {
      "resource": {
        "attributes": [
          { "key": "service.name", "value": { "stringValue": "child.internal" } }
        ]
      },
      "scopeSpans": [
        {
          "scope": {
            "name": "manual-instrumentation",
            "version": "1.0.0"
          },
          "spans": [
            {
              "traceId": "4bf92f3577b34da6a3ce929d0e0e4736",
              "spanId": "7af7651916cd43dd",
              "parentSpanId": "00f067aa0ba902b7",
              "name": "curl-span",
              "kind": 1,
              "startTimeUnixNano": "1678886400000000000",
              "endTimeUnixNano": "1678886401000000000",
              "attributes": [
                { "key": "http.method", "value": { "stringValue": "POST" } },
                { "key": "http.url", "value": { "stringValue": "/v1/traces" } }
              ]
            }
          ]
        }
      ]
    }
  ]
}
```

Трассировка – Jaeger Traces View

Find Traces

Service
frontend

all

Tags
http.status_code:400|http.status_code:200

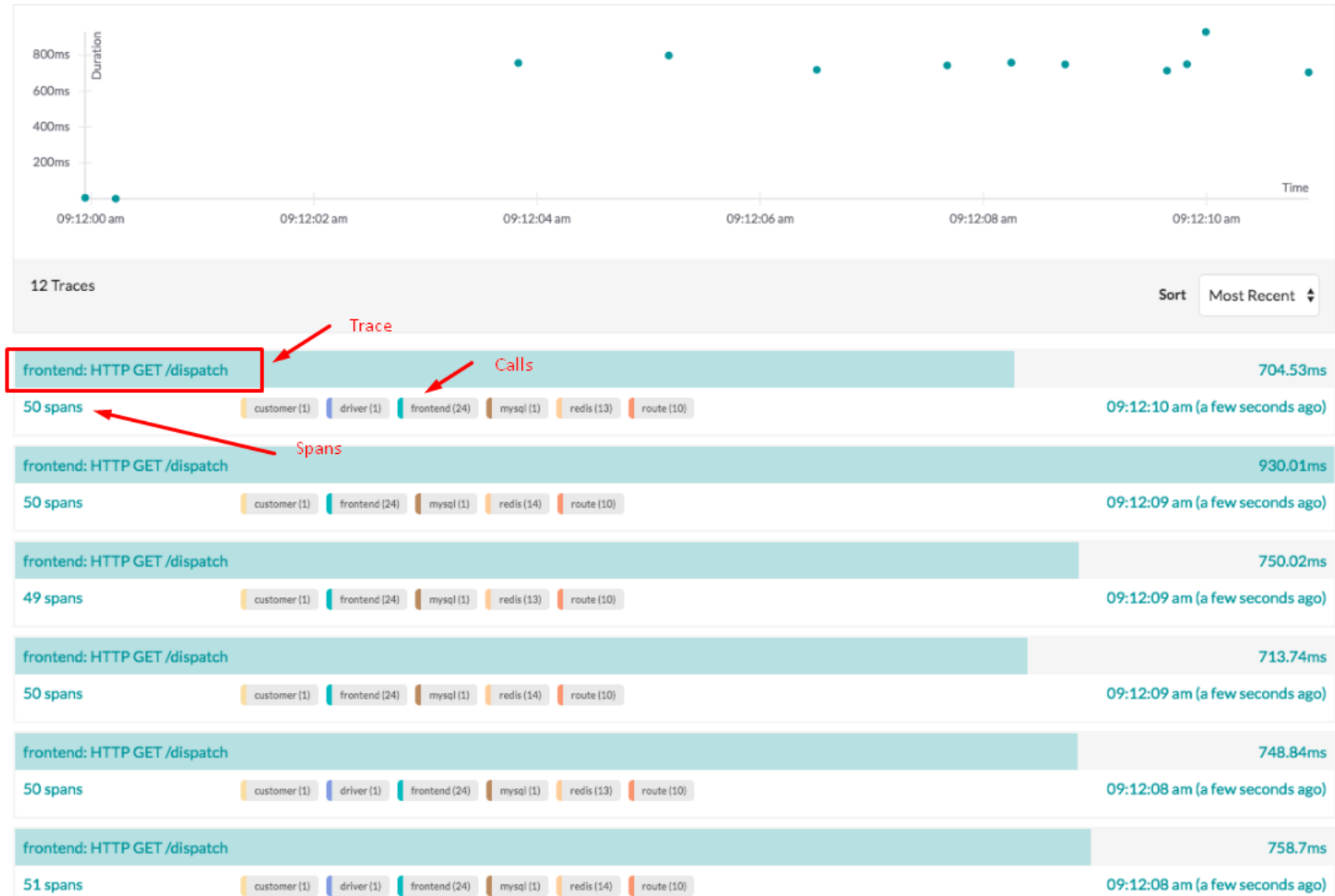
Lookback
Last Hour

Min Duration
e.g. 1.2s, 100ms, 500u:

Max Duration
e.g. 1.1s

Limit Results
20

Find Traces



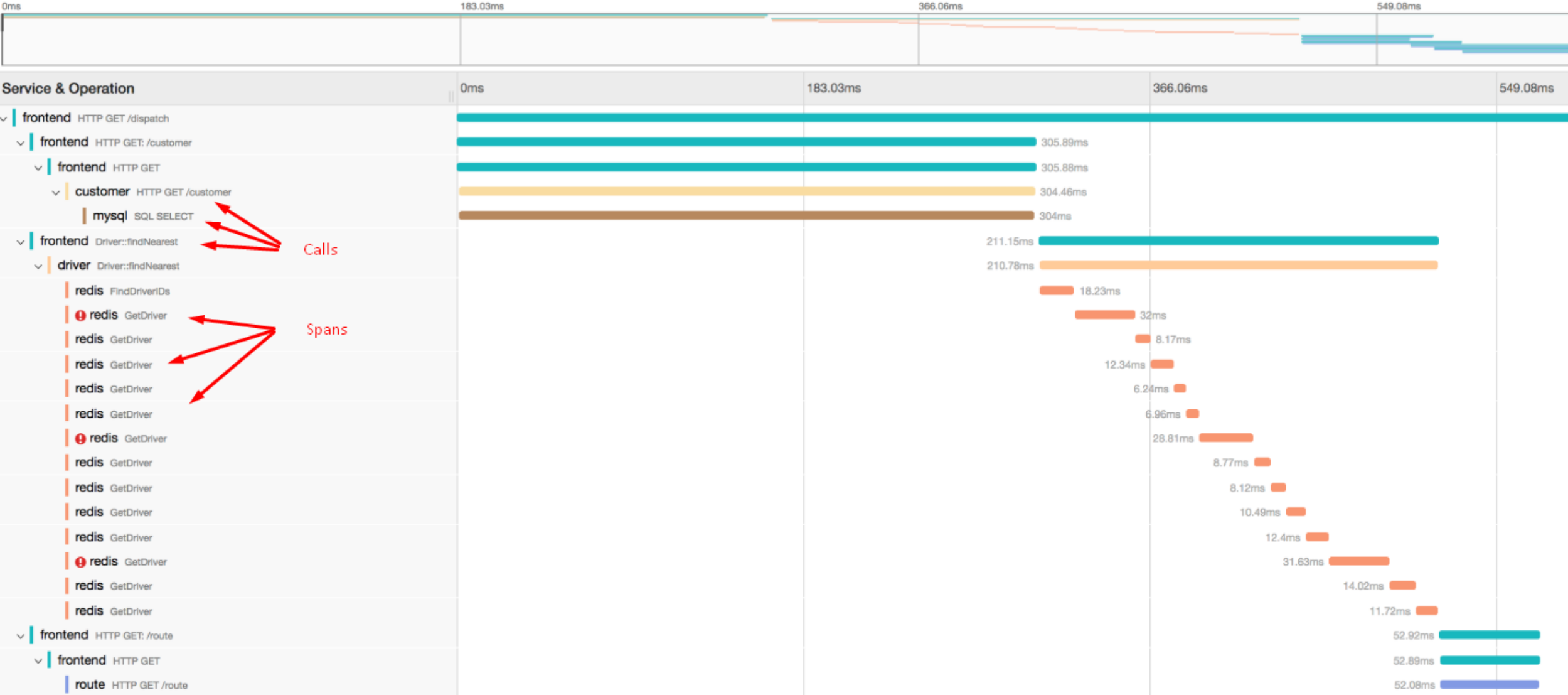


Трассировка – Jaeger Trace Detail View

▼ frontend: HTTP GET /dispatch



Trace Start: July 20, 2018 2:48 PM | Duration: 732.11ms | Services: 6 | Depth: 5 | Total Spans: 51





Трассировка - проблемы

Сложности при работе с distributed tracing:

- огромные объемы данных - в пять раз больше чем логов, частично можно снять проблему сэмплированием (фильтрацией не критичных) трейсов
- неполное покрытие - вы можете пропустить критические запросы
- нагрузка на процессор и сеть - 10-20% больше
- сетевой лаг - каждый запрос вносит задержку несколько мс
- необходимость обучения — настройка трейсов и анализ спанов требует новых навыков от инженеров, мало кто умеет
- архитектура - заранее прорабатывать архитектуру и инфраструктуру, мало кто умеет

Как вам такое решение проблемы?

Трассировка - резюме

Логирование важно для аудита и детализации событий внутри сервиса, но не даёт целостного представления о жизни запроса в системе.

Трассировка дополняет логирование, связывает события между сервисами и предоставляет анализ времени и ошибок.

Проблемы, которые помогает решать трассировка:

- Непрозрачность распределённых вызовов
- Диагностика латентности и узких мест
- Сложность корреляции ошибок
- Ускорение MTTR (времени восстановления)

Однако трассировка требует высокой квалификации и больших ресурсов, поэтому используем только на критичных продуктах и процессах.

Мониторинг – история появления

Вопрос – инструменты подобрали, логи настроили, трассировку настроили, людей обучили, **счастье настало** 😊?


Ответ – проблемы остались:

- Логирование и трейсинг помогают найти проблемы в системе, но не дают текущего актуального состояния системы
- Сложно сделать прогноз по истории либо трендам, например когда закончится место
- Сложно зафиксировать когда система выходит за замки SLA
- Сложно увидеть текущее состояние системы в целом

Решение – мониторинг: непрерывный сбор и визуализации метрик о работе системы (загрузка процессора, память, время ответа сервисов, количество ошибок и др.)

Задача мониторинга - обеспечить обзор состояния системы в реальном времени.

Мониторинг – инструменты

- 
- 1990** - **SNMP** и простейшие агенты, SNMPv1 (RFC 1157), ввёл модель «менеджер—агент» для удалённого сбора метрик с маршрутизаторов и коммутаторов
- 1995** - Тоби Оэтикер написал прототип скрипта для отображения загрузки выпустил **MRTG** 1.0 для графического отображения трафика
- 2001** - Алексей Владисев опубликовал первую версию **Zabbix**, который стал стандартом для мониторинга системных метрик серверов
- 2010** - Появление SaaS мониторинга **Datadog**, основанным Оливье Помелем и Алексисом Ле
- 2010** - появление **Icinga**, форка **Nagios** который эволюционировал в платформу с поддержкой REST API и графиков
- 2012** - **Prometheus**, проект начат в SoundCloud, закладывая модель pull запросов и встроенного TSDB
- 2014** - **Grafana** выпущена Тёркелем Эдегаардом как фронтенд для Graphite, OpenTSDB и InfluxDB, быстро став универсальным визуализатором метрик
- 2014** - **Google Stackdriver**, приобрёл одноимённый стартап и интегрировал его в Google Cloud Platform
- 2016** - **Microsoft Azure Monitor** встроенное решение для мониторинга ресурсов Azure

Мониторинг – умные фразы про метрики

«Если ты не можешь это измерить, значит, ты не можешь управлять этим», Питер Друкер, гуру менеджмента – все можно измерить, оценить и принять решение.

«Не бывает много или мало, бывает много или мало по сравнению с чем то», учат на ФФ – всегда есть эталон с которым можно сравнить.

«Сколько вешать в граммах?», из рекламного ролика – сравнивать нужно в одинаковых единицах измерения.

«И спросила кроха, что такое хорошо и что такое плохо?», Владимир Маяковский – говорить на одном языке, договариваться о единых критериях «хорошо \ плохо» с заказчиком.

«Все зависит от того с какого берега смотреть», пословица – что для разработчика хорошо, заказчику смерть.

Метрики служат основой для постановки целей, отслеживания прогресса и своевременного реагирования на отклонения от намеченного курса.

Мониторинг – дашборд пилота





Мониторинг – «чего же хотят женщины?»

Поразмышляем какие показатели интересны разным категориям пользователям продукта.

- Что хочет знать бизнес овнеры?
- Что хотят знать технологи?
- Что хотят знать аналитики?
- Чего хотят знать разработчики?
- Чего хотят знать тестировщики?
- Чего хотят знать devops?

Мониторинг – показатели

Что хочет знать бизнес овнеры? – показатели KPI (продажи, конверсия, посещения, лиды, доход на одного пользователя и др.)

Что хотят знать технологи? – показатели SLA (скорость продажи, скорость бронирования, удовлетворенность пользователей и др.)

Что хотят знать проектные менеджеры? – скорость вывода нового функционала, процент выполнения задач в спринте, количество багов за спринт и др.

Что хотят знать аналитики? – тоже что и менеджеры и технологи и овнеры

Чего хотят знать разработчики? – показатели java машины, статистика ошибок, количество повторных багов

Чего хотят знать тестировщики? – живы ли интеграционные сервисы, полнота покрытия тестами

Чего хотят знать devops? – загрузка систем, свободное место, живы ли интеграции

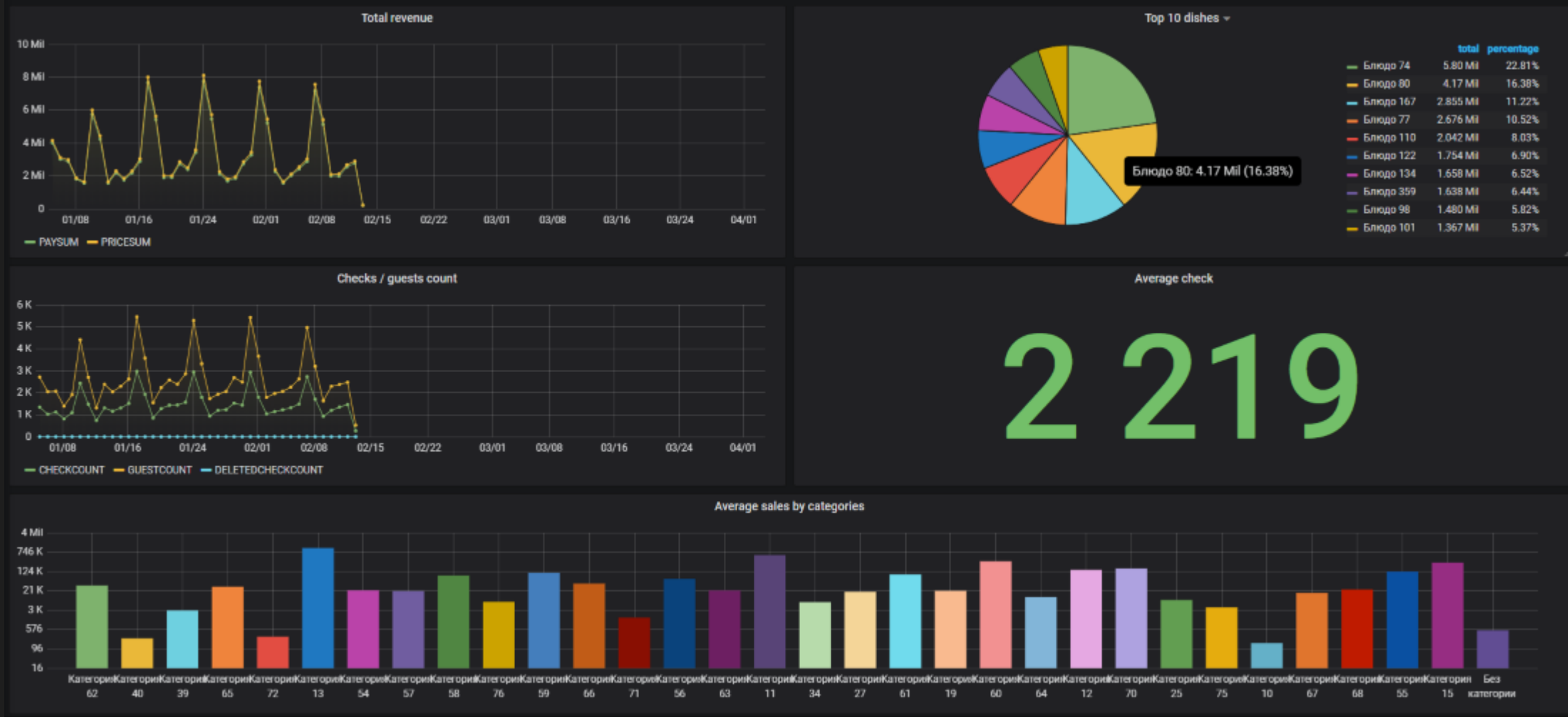
Метрик много, задача аналитика и менеджера на этапе MVP подобрать показатели для разных аудиторий, которые будут отображать актуальное состояние системы.

Мониторинг – бизнес метрики

Бизнес метрики – показатели KPI

Аудитория – бизнес, менеджеры, технологи, аналитики

Инструменты – Prometheus / Grafana

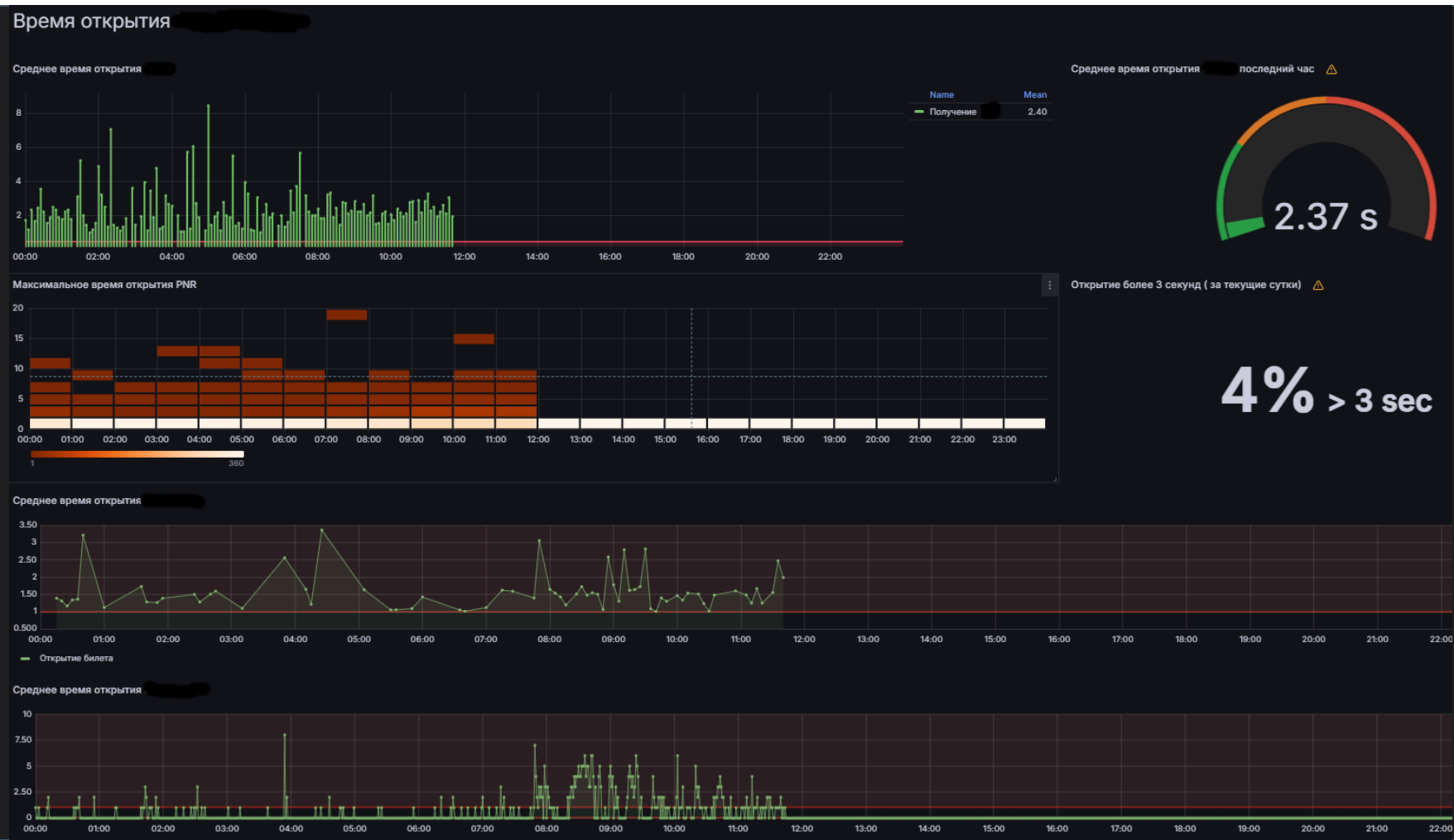


Мониторинг – технологические метрики

Технологические метрики – показатели SLA, интеграции

Аудитория – менеджеры, технологи, аналитики

Инструменты – Prometheus / Grafana



Мониторинг – технические метрики

Технические метрики – показатели микросервисов

Аудитория – разработчики, тестировщики

Инструменты – Prometheus / Grafana



Мониторинг – системные метрики

Системные метрики – показатели операционных систем и микросервисов

Аудитория – админы, devops, support

Инструменты – Prometheus / Grafana, Zabbix

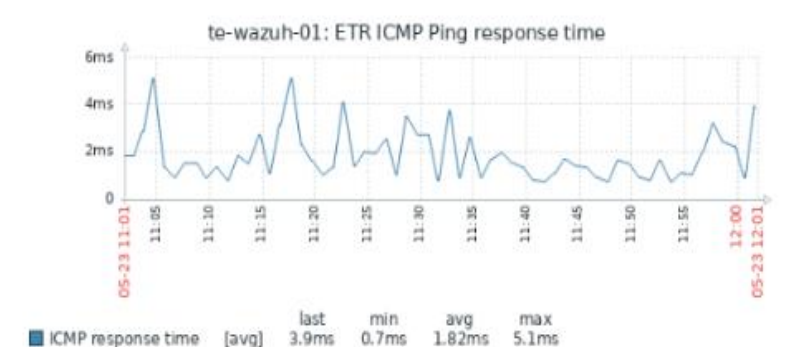
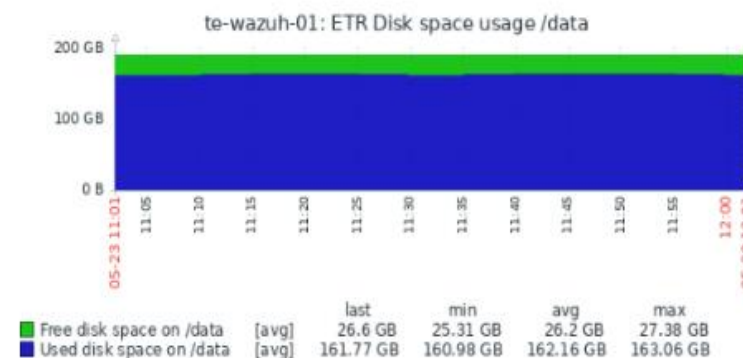
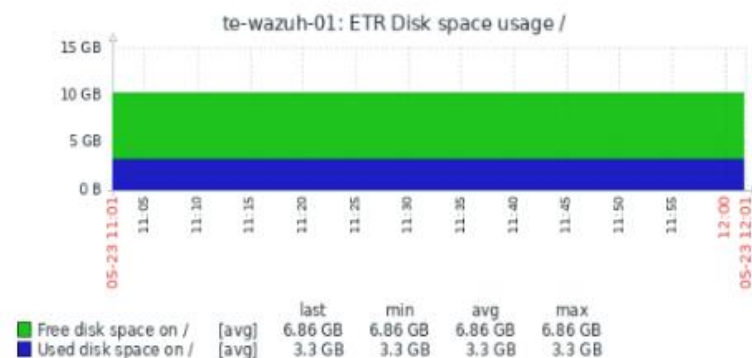
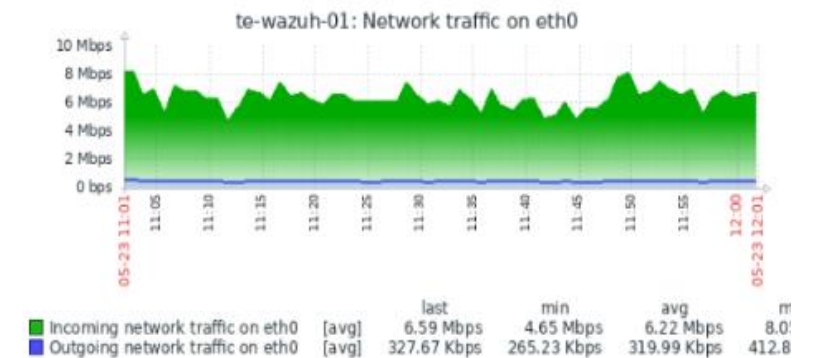
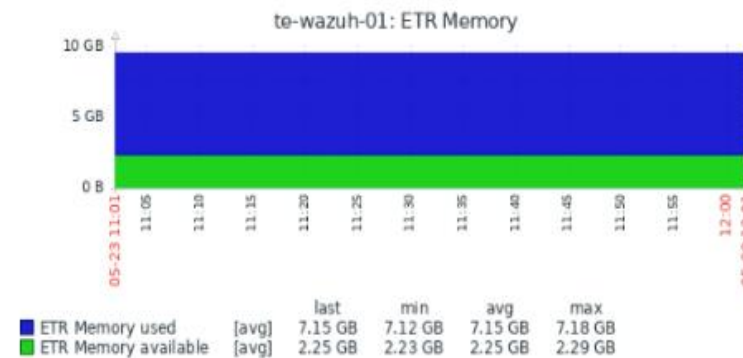
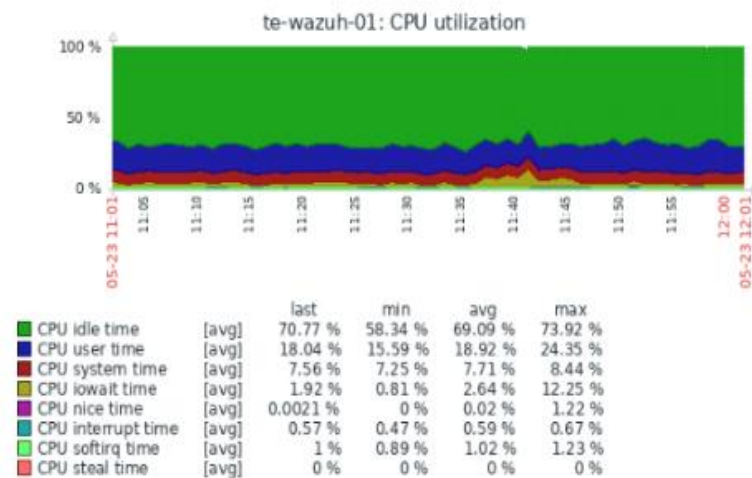


Мониторинг – системные метрики

Системные метрики – показатели операционных систем и микросервисов

Аудитория – админы, devops, support

Инструменты – Prometheus / Grafana, Zabbix



Мониторинг – как дотащить метрику в дашборд

Относится к бизнес метрикам и технологическим метрикам.

Любая метрика как фича – требует затрат и работы нескольких инженеров.

1. Аналитик общается с оwnerом, понимает какую бизнес цель несет метрика, формирует User Story
2. Разработчик дорабатывает микросервис чтобы данную метрику вынести в эндпоинт
3. Тестировщик убеждается что метрика соответствует бизнес смыслу и она есть в эндпоинте
4. Devops дорабатывает экспортер, чтобы метрика из эндпоинта появилась в prometheus и victoriametrics
5. Аналитик либо саппорт дорабатывает дашборд для отображения метрики в графана
6. Еще и дизайнера желательно подключить чтобы метрики выглядели читабельными, а дашборд не перегружал экран

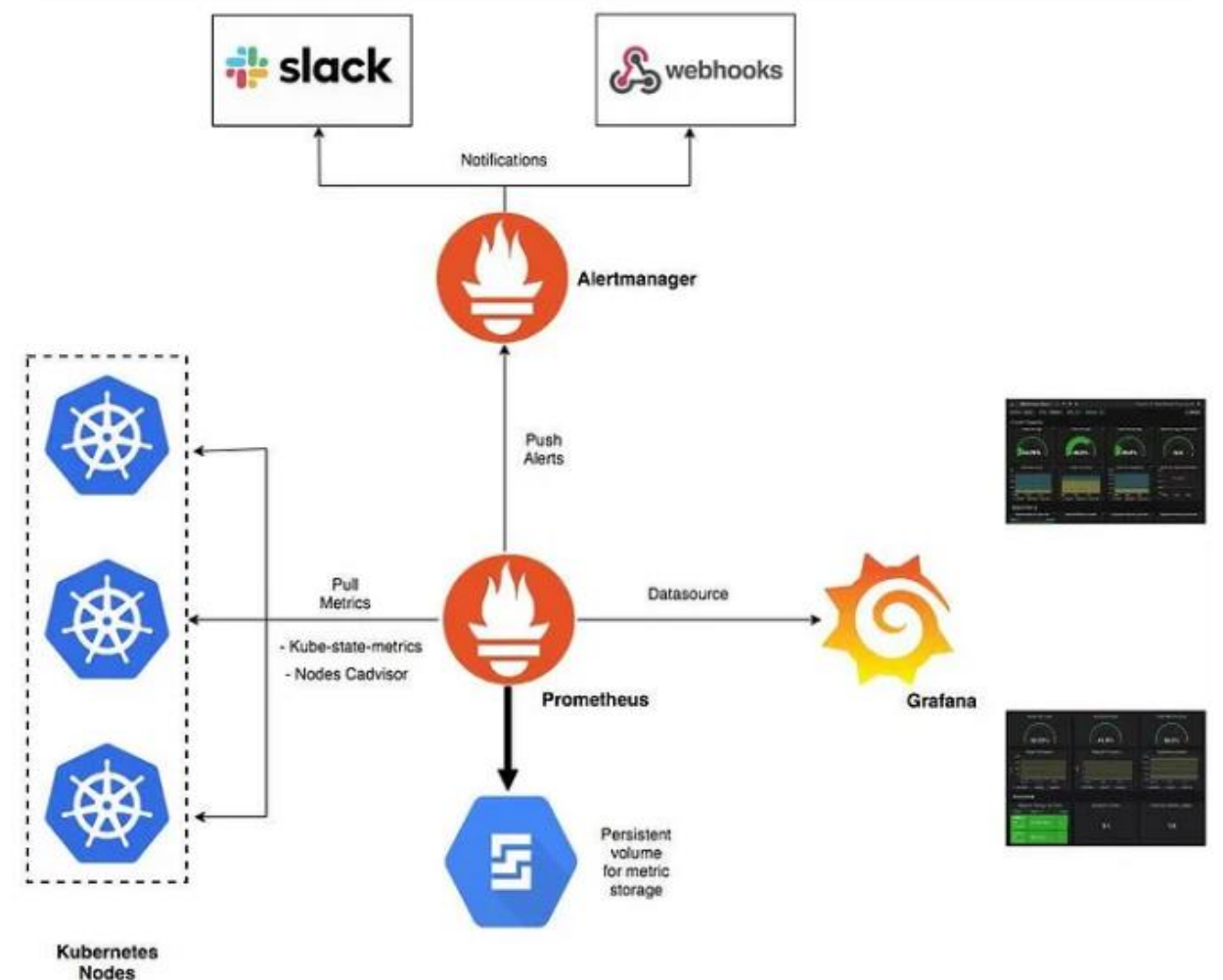
Мониторинг – как написать User Story метрики

Вопросы при разработке User Story для метрики:

- **Зачем, с точки зрения бизнеса?** - определяем несет ли метрика смысл, и готов ли бизнес нести затраты
- **Как человек может проверить это руками?** - определяем способ и место где можем достать метрику в ручном режиме
- **Как узнать, есть или нет проблемы (критерии проблем по метрике)?** - определяем при каких показателях считаем что есть проблема (что такое "хорошо \ плохо" для заказчика)
- **Как технически можно реализовать?** - решаем в какой микросервис добавить и как технически реализовать
- **Алерт?** - требуется ли реакция, кто и как будет реагировать, у исполнителя должны быть ресурсы и инструменты
- **Права?** - кто будет иметь доступ к метрике
- **Как представить в Графанах?** - на каком дашборде, в каком графике, и какой тип графика

Мониторинг – как работает Prometheus

- Prometheus регулярно отправляет HTTP-запросы к настроенным целям (targets), частота от 15 сек до 1 мин
- Цели (targets) - серверы, контейнеры, микросервисы, БД, оборудованы экспортерами (Exporters)
- Все метрики состоят из пары ключ-значение
- Собранные метрики сохраняются локально на сервере Prometheus в виде time-series database
- В K8s, Prometheus автообнаруживает сервисы через API
- Для анализа есть язык PromQL
- Визуализация обычно делается через Grafana
- Экспортеров несколько сотен, для различных продуктов <https://prometheus.io/docs/instrumenting/exporters>



Мониторинг – jmx exporter, запрос curl

Конфигурационный файл Prometheus в yaml джобы имеют вид:

```
- job_name: "jmx-exporter"

  static_configs:
    - targets: ["localhost:8080"]
```

Данные из экспортера можно посмотреть запросом

curl <http://localhost:8080/metrics>

```
# HELP jvm_gc_collection_seconds Time spent in a given JVM garbage collector in seconds.
# TYPE jvm_gc_collection_seconds summary
jvm_gc_collection_seconds_count{gc="G1 Young Generation",} 12.0
jvm_gc_collection_seconds_sum{gc="G1 Young Generation",} 0.086
jvm_gc_collection_seconds_count{gc="G1 Old Generation",} 0.0
jvm_gc_collection_seconds_sum{gc="G1 Old Generation",} 0.0
# HELP jvm_info VM version info
# TYPE jvm_info gauge
jvm_info{runtime="OpenJDK Runtime Environment",vendor="Private Build",version="17.0.9+9-Ubuntu-122.04",} 1.0
# HELP jmx_exporter_build_info A metric with a constant '1' value labeled with the version of the JMX exporter.
# TYPE jmx_exporter_build_info gauge
jmx_exporter_build_info{version="0.19.0",name="jmx_prometheus_javaagent",} 1.0
# HELP jvm_buffer_pool_used_bytes Used bytes of a given JVM buffer pool.
# TYPE jvm_buffer_pool_used_bytes gauge
jvm_buffer_pool_used_bytes{pool="mapped",} 0.0
jvm_buffer_pool_used_bytes{pool="direct",} 51199.0
jvm_buffer_pool_used_bytes{pool="mapped - 'non-volatile memory'",} 0.0
# HELP jvm_buffer_pool_capacity_bytes Bytes capacity of a given JVM buffer pool.
# TYPE jvm_buffer_pool_capacity_bytes gauge
jvm_buffer_pool_capacity_bytes{pool="mapped",} 0.0
jvm_buffer_pool_capacity_bytes{pool="direct",} 51199.0
jvm_buffer_pool_capacity_bytes{pool="mapped - 'non-volatile memory'",} 0.0
# HELP jvm_buffer_pool_used_buffers Used buffers of a given JVM buffer pool.
# TYPE jvm_buffer_pool_used_buffers gauge
jvm_buffer_pool_used_buffers{pool="mapped",} 0.0
jvm_buffer_pool_used_buffers{pool="direct",} 3.0
jvm_buffer_pool_used_buffers{pool="mapped - 'non-volatile memory'",} 0.0
# HELP jvm_classes_currently_loaded The number of classes that are currently loaded in the JVM
# TYPE jvm_classes_currently_loaded gauge
jvm_classes_currently_loaded 7164.0
```


Мониторинг - проблемы

Сложности при работе с Prometheus и Grafana:

- возможна высокая нагрузка на сеть и хранилище, если метрик много
- высокая стоимость поддержки, для надежности нужно развертывать в нескольких экземплярах
- работа с Grafana и PromQL это отдельная компетенция, как правило ни разработчики ни аналитики с ними работать не умеют
- создание метрики в Grafana требует участия нескольких инженеров из смежных областей

Как вам такое решение проблемы?

Мониторинг - резюме

Логирование фиксирует детальные события и ошибки внутри сервиса, позволяя разбирать конкретные инциденты и отлаживать код.

Мониторинг агрегирует ключевые метрики, отображает состояние системы, помогает быстро выявлять отклонения и потенциальные проблемы.

Проблемы, которые помогает решать трассировка:

- Позволяет оперативно увидеть отклонения от нормального состояния системы
- Контроль SLA и бизнес-показателей
- По историческим метрикам можно понять, когда началась проблема и как система вела себя в момент сбоя
- Отслеживание производительности сервисов
- Прогнозирование проблем

Однако настройка метрик требует отдельной компетентности и квалификации, а также и затрат на поддержку.

Алертинг – забытая функция observability

Вопрос – инструменты есть, логи есть, трейсинг есть, мониторинг есть, а кто будет следить и реагировать на показатели либо логи?

Ответ – проблемы остались:

- Могут ли разработчики следить и реагировать на показатели от технического мониторинга?
- Могут ли аналитики постоянно следить за бизнес метриками и реагировать?
- А админы или support следить и реагировать?
- А есть ли у тех кто реагирует инструменты и компетенция для обработки ошибки?
- А если событие ночью произошло?
- А сколько стоит содержать команду которая 24x7 дежурит?

Решение – Алертинг + настроенные процессы: оповещения о критических событиях.

Алертинг – инструменты ELK

ElastAlert – конфиг через yaml.

Настраивать может 1% инженеров.
Доступ к настройке как правило только у DevOps.

Каналы – email, slack, telegram, teams, pagerduty, jira, webhook

Проблемы:

- Сложность написания правил
- Сложность поддержки правил
- Сложность тестирования
- DevOps становится узким местом по написанию

```
# Настройки подключения к Elasticsearch
es_host: elasticsearch.example.com
es_port: 9200
es_username: "elastic" # опционально
es_password: "password" # опционально

# Правило алерта
name: "HTTP 500 Errors Alert"
type: frequency # Тип правила: алерт при превышении частоты событий
index: logs-* # Имя индекса (можно использовать шаблон)

# Условия срабатывания
num_events: 10 # Минимальное количество событий для триггера
timeframe:
  minutes: 5 # Окно времени для анализа

# Фильтр запросов в Elasticsearch
filter:
- query:
  query_string:
    query: "status_code:500 AND service:web-app"

# Настройки уведомлений
alert:
- email
email:
- "develeper@example.com"
smtp_host: "smtp.example.com"
smtp_port: 587
smtp_ssl: false
smtp_auth_file: /etc/elastalert/smtp_auth.yaml # Файл с логином/паролем SMTP

# Дополнительные параметры
realert:
  minutes: 5 # Минимальный интервал между уведомлениями
```

Алертинг – инструменты ELK

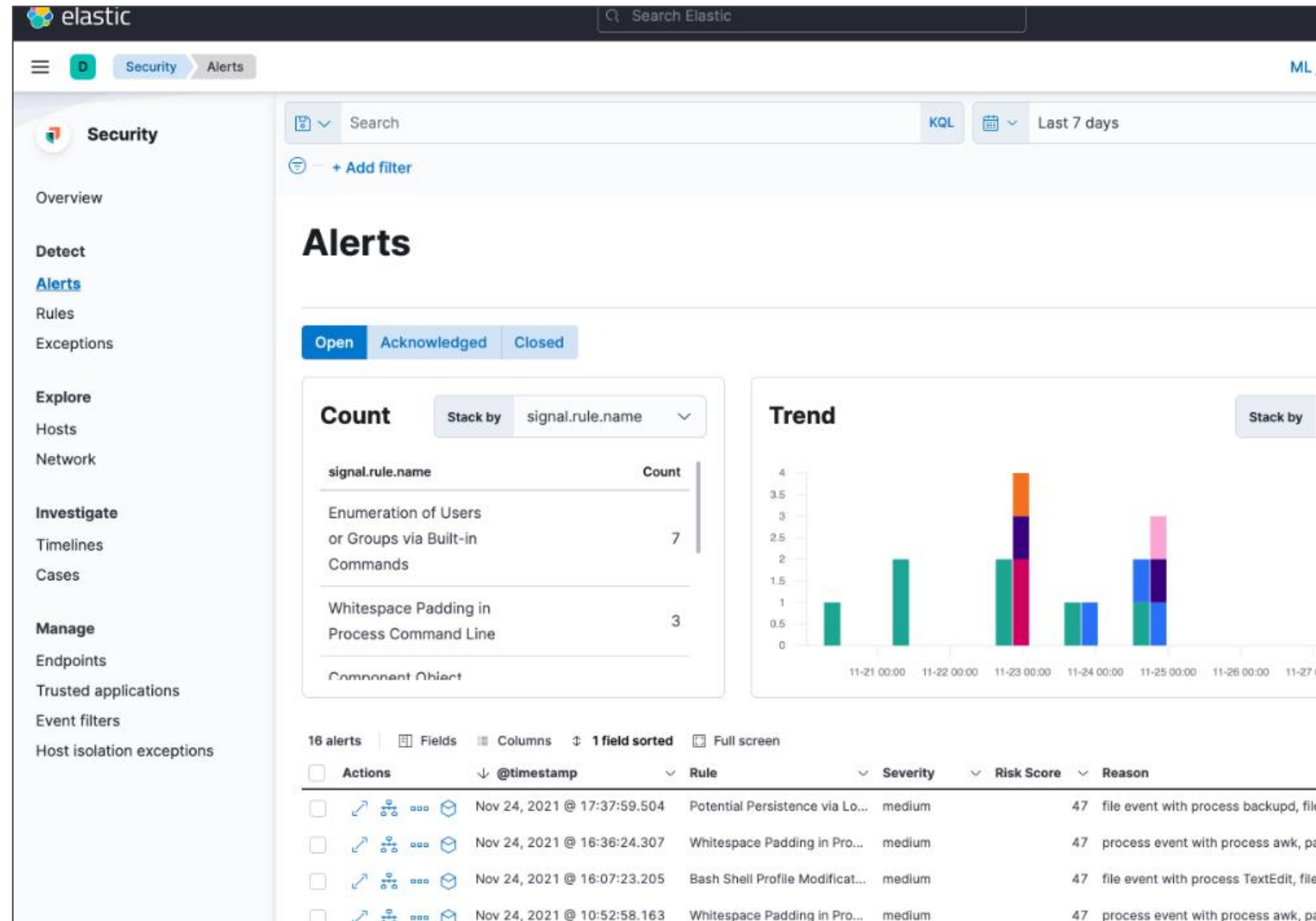
Kibana – в 7 версии

1% инженеров умеет

Каналы – email, slack, telegram, pagerduty, jira, webhook

Проблемы:

- Интерфейс запутанный
- Сложность настройки
- Сложность тестирования
- Базовые сценарии
- Отдельная компетенция



Алертинг – инструменты Prometheus

ALertManager – конфиг через yaml.

Настраивать может 1% инженеров.
Доступ к настройке как правило только у DevOps.

Каналы – email, slack, telegram, teams, pagerduty, jira, webhook

Проблемы:

- Сложность написания правил
- Сложность поддержки правил
- Сложность тестирования
- DevOps становится узким местом по написанию

```
global:
  resolve_timeout: 5m # Время ожидания перед сбросом алерта после исчезновения проблемы

route:
  group_by: ['alertname', 'severity'] # Группировка алертов по ключам
  group_wait: 30s # Задержка перед отправкой первого уведомления для группировки
  group_interval: 5m # Интервал между повторными уведомлениями для одной группы
  repeat_interval: 3h # Периодичность повторных уведомлений о нерешённых алертах

  receiver: 'email-notifications' # Получатель по умолчанию

receivers:
  - name: 'email-notifications'
    email_configs:
      - to: 'your-email@example.com' # Адрес получателя
        from: 'alertmanager@example.com' # Адрес отправителя
        smarthost: 'smtp.example.com:587' # SMTP-сервер
        auth_username: 'smtp-user' # Логин для SMTP
        auth_password: 'smtp-password' # Пароль для SMTP
        send_resolved: true # Отправлять уведомление о решении проблемы
```



Алертинг – инструменты Grafana

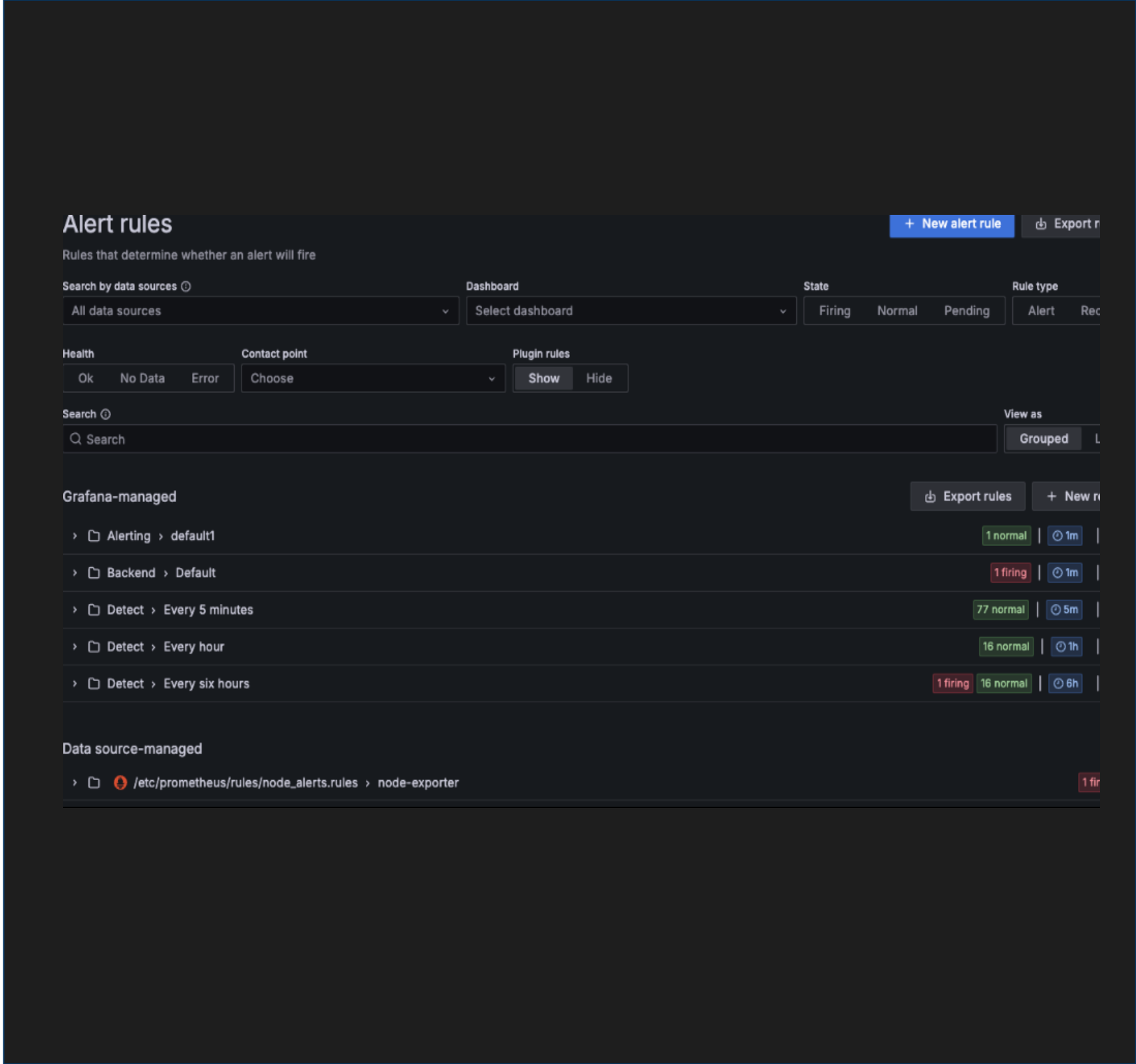
Grafana Alert

5% инженеров умеет

Есть мобильное приложение

Каналы – email, slack, jira,
webhook, telegram, pagerduty

Низкий порог вхождения





Алертинг – инструменты Zabbix

Zabbix – системные метрики

90% админов и devops умеют

Каналы – email, slack, jira, webhook, telegram, pagerduty, phone

Низкий порог вхождения

ActionOperationsRecovery operationsUpdate operations

* Default operation step duration1h

Default subjectProblem. {ACTION.NAME}. {EVENT.NAME}

Default message

Проверка "{EVENT.NAME}" в состоянии ошибки

Имя сервера: {HOST.NAME}

Время события: {EVENT.TIME} {EVENT.DATE}

Статус: {TRIGGER.STATUS}

Важность: {TRIGGER.SEVERITY}

Pause operations for suppressed problems☒

Operations

StepsDetailsStart inDurationAction

1Send message to user groups: ETR Infrastructure via all mediaImmediatelyDefaultEditRemove

Operation details

Steps

1

 -

1

 (0 - infinitely)

Step duration

0

 (0 - use action default)

Operation type

Send message

* At least one user or user group must be selected.

Send to User groups

User group

ETR Infrastructure

Add

Action

Remove

Send to Users

User

Add

Action

Send only to

- All -

Default message☒

Conditions

Label

Name

Action

New

Update

Cancel

* At least one operation, recovery operation or update operation must exist.

Update

Clone

Delete

Cancel

Алертинг – как сделать поддержку 24x7

24x7 – 3 инженера по 8ч в день, еще два выходных это +1 инженер, если кто то заболел либо отпуск это +1 инженер, итого минимум 5 инженеров требуется. Дорого или нет обеспечивать поддержку?

Частичное решение проблемы 24x7:

- гибкий график плюс инженеры из разных часовых поясов
- реакция только на дзастер в нерабочее время по звонку
- использовать менее дорогих и менее квалифицированных на первой линии
- аутсорсинг первой линии

Что делать с компетенцией инженеров, они же не могут править код?

- обучать и погружать в проекты
- написать готовые сценарии и сделать базу знаний
- подключать к команде тестирования и команде аналитиков
- обеспечить гибкий график и оплату за сверхурочную работу
- обеспечить инструментами и возможностью подключения инженеров devops и разработчиков

Алертинг – как сделать поддержку 24x7

Ответ – проблемы остались:

- Могут ли разработчики следить и реагировать на показатели от технического мониторинга?
- Могут ли аналитики постоянно следить за бизнес метриками и реагировать?
- А админы или support следить и реагировать?
- А есть ли у тех кто реагирует инструменты и компетенция для обработки ошибки?
- А если событие ночью произошло?
- А сколько стоит содержать команду которая 24x7 дежурит?

Решение – Алертинг + настроенные процессы: оповещения о критических событиях.

Алертинг – как сделать поддержку 24x7

Что делать с разработчиками и devops инженерами которых будут подключать?

- искать ответственных инженеров, которых можно вызвонить и подключить в случае проблем в нерабочее время (обычно тимлиды и ведущие инженеры обладают таким качеством)
- компенсировать затраты на работу во вне рабочее время
- обеспечивать график дежурств (когда можно вызванивать)
- и главное, снизить до минимума инциденты в не рабочее время

Как снизить число инцидентов?

- ревью кода
- периодический анализ произошедших инцидентов (управление проблемами)
- анализ и снижение количества багов
- постмортем анализ каждого инцидента и разбивание цепочки событий приведшее к инциденту

Алертинг неотъемлемая часть логирования и мониторинга, отвечает на вопрос кто и как будет реагировать. Настройка алертинга это в первую очередь настройка процессов и службы поддержки.

Вопросы?



N*

**Спасибо
за внимание**

ФИТ Лекция №16. 25'