

# Мониторинг в Додо

Павел Притчин

Dodo Pizza, Dodo IS Core Team





# Dodo IS

## Масштаб

- Сервисов: 20+
- VMs: 100+
- Dev infrastructure:  
~20%
- Monitoring  
infrastructure:  
~10%

## Технологии

- ASP.NET + IIS
- ASP.NET Core +  
Kubernetes
- WinServices, Quartz
- MySql, Redis,  
RabbitMQ

## Глобальная распределенная система, 24/7

- 50 разработчиков
- 12 стран
- 410+ пиццерий
- 3 Azure Datacenters  
(EU, US, CHN)

# .Net

## Раньше

1. Своя экосистема



## Сейчас

1. Появился dotnet core



# .Net

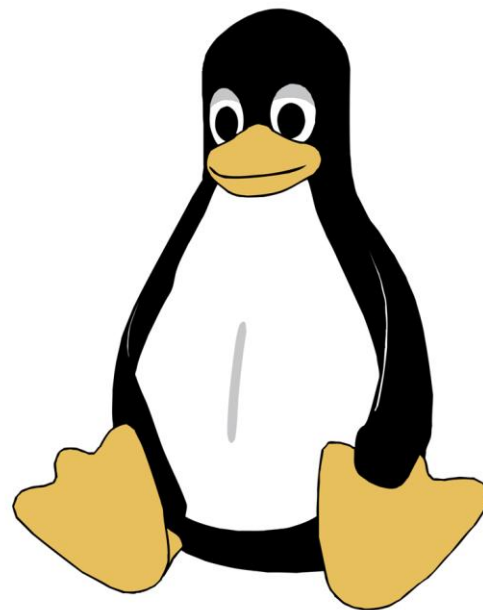
## Раньше

2. windows



## Сейчас

2. Не только windows



# .Net

Раньше

3. enterprise



Сейчас

3. Разные типы систем



# Проблемы

1. Как отслеживать производительность системы(и нефункциональные требования)?
2. Как обеспечить быстроту реакции на сбои?
3. Как обеспечить качество?



# Мониторинг. Требования

1. Поддержка распределенной и динамической системы
2. Разные стеки должны уживаться рядом
3. Простой, надежный,
4. Не влияет на наблюдения, действенный
5. Выгода для разработчиков

Мониторинг - это вынужденная мера



# Что наблюдаем

## По RED методу:

(Request) **Rate** - the number of requests, per second, you services are serving.

(Request) **Errors** - the number of failed requests per second.

(Request) **Duration** - distributions of the amount of time each request takes.

<https://www.weave.works/blog/the-red-method-key-metrics-for-microservices-architecture/>





# Что наблюдаем

По “4 golden signals” методу:

**Latency** - request duration.

**Traffic** - request count,

**Error** - count of failed requests, 5xx response status,

**Saturation** - CPU, memory, I/O, process threads etc.

[https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/#xref\\_monitoring\\_golden-signals](https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems/#xref_monitoring_golden-signals)



# 1. Метрики по сервисам

## а. Prometheus

## 2. Логи

### а. Nlog + ELK,

## 3. Распределенный трейсинг

### а. Zipkin



# Метрики. Prometheus.

<https://prometheus.io/>



Универсальное средство(mysql, redis,  
nginx, rabbit, k8s),  
Свое хранилище,  
Свой язык запросов,  
Хорошая интеграция,  
Есть клиенты и middleware

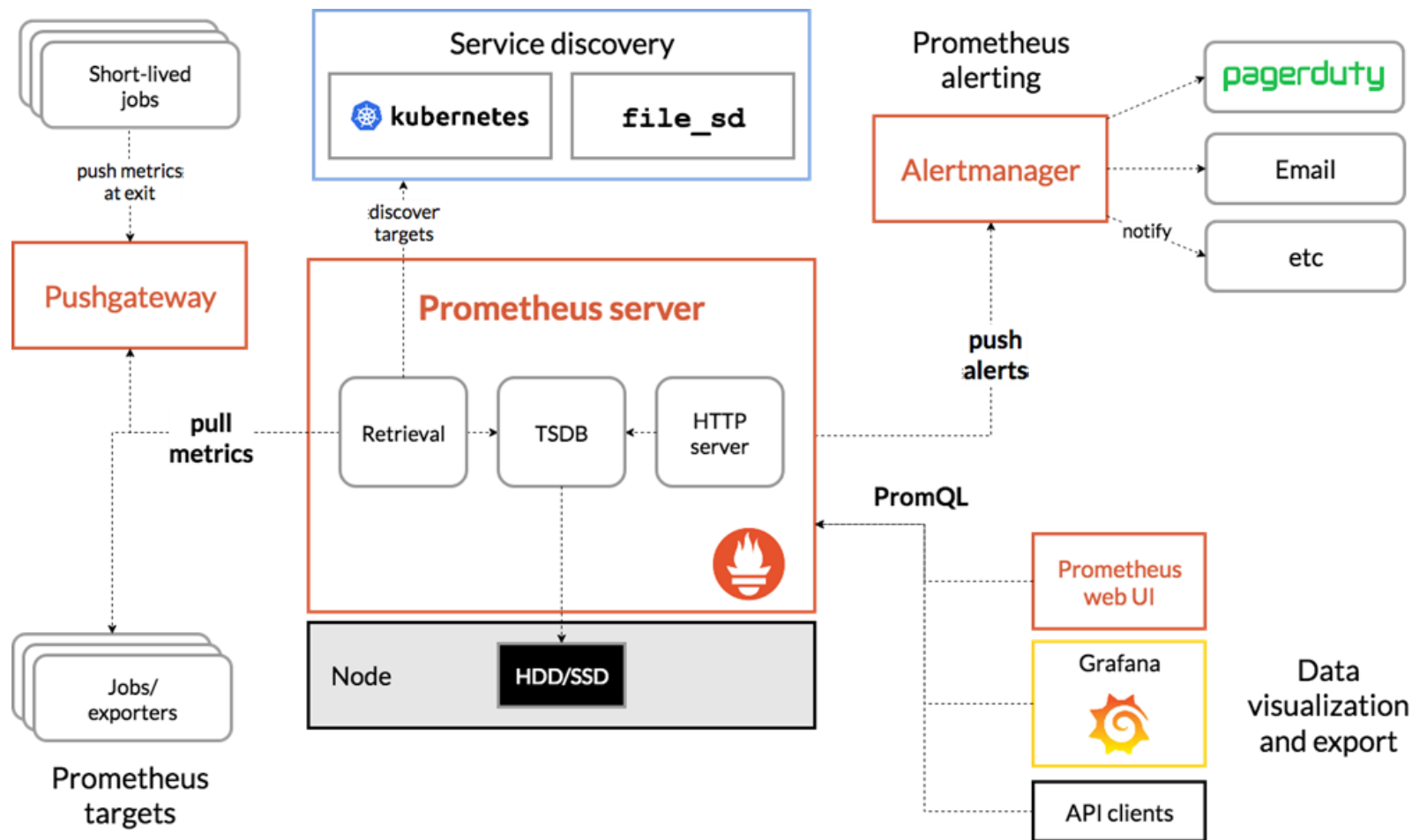
- RED: Rate + Duration. Errors (частично),
- 4 gs: Latency + Traffic + Saturation + Errors (частично)

*Альтернативы:*

- *graphite, influxdb, zabbix, riemann*



# Prometheus. Архитектура

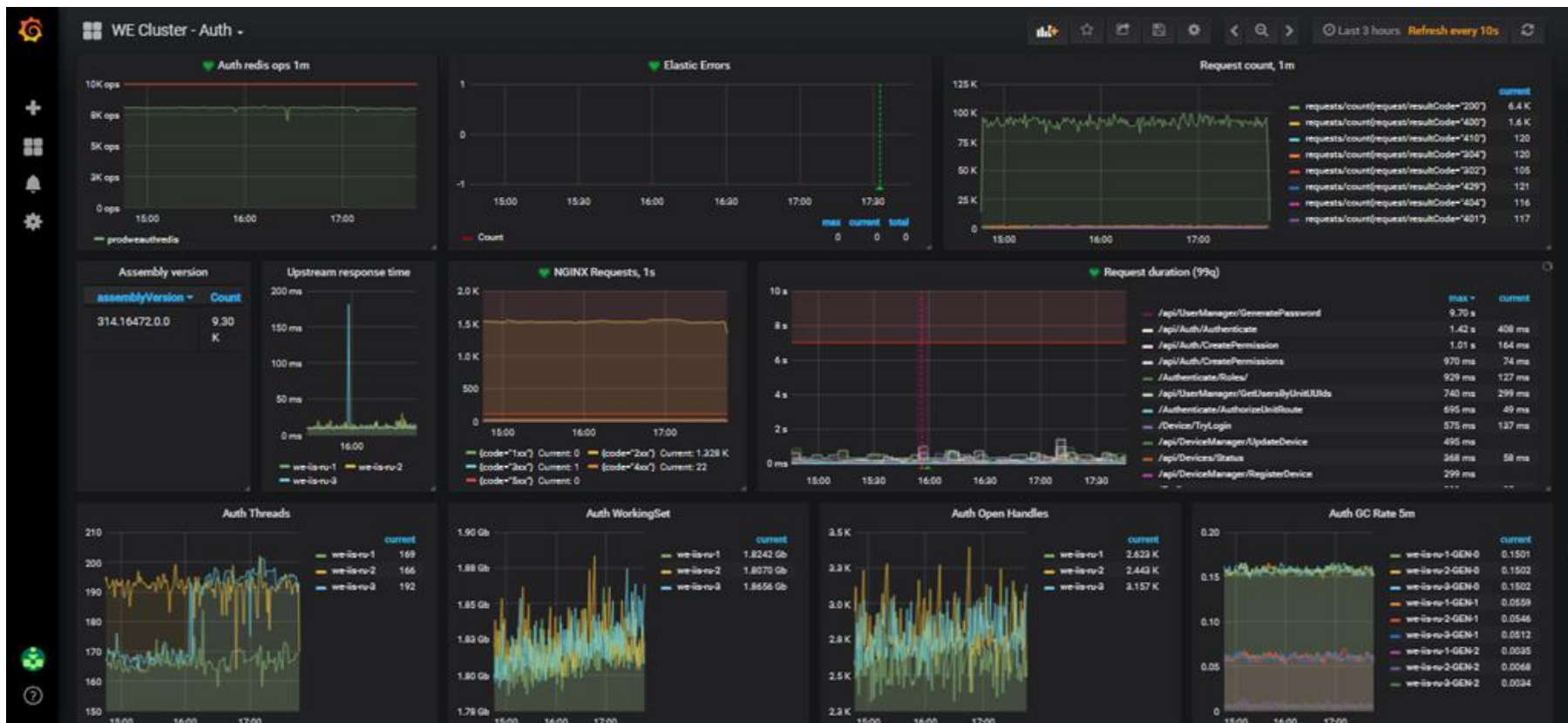


# Prometheus. Подключение

```
1 public async Task Invoke(HttpContext context)
2 {
3     var stopwatch = Stopwatch.StartNew();
4     var uriAbsolutePathWithoutId = FilterIdInAbsolutePath(new Uri(context.Request.GetDisplayUrl()));
5     try
6     {
7         await _next.Invoke(context);
8     }
9     finally
10    {
11        if (IncludeInStatistics(uriAbsolutePathWithoutId))
12        {
13            stopwatch.Stop();
14            PrometheusCounters.ObserveRequest(
15                uriAbsolutePathWithoutId,
16                stopwatch.ElapsedMilliseconds,
17                context.Response.StatusCode);
18        }
19    }
20 }
```



# Prometheus. Данные



# Prometheus. Вывод

- общая инфраструктура для всех типов сервисов
- типизированное использование
- сбор всех показателей по RED и 4gs моделям
- хорош как самостоятельное средство, так и в интеграции
- дублирование метрик и сбора между сервисами

## Минусы и проблемы:

- Надо уметь развернуть и поддерживать
- Хранение данных



1. Метрики по сервисам

а. Prometheus

## **2. Логи**

**а. Nlog + ELK,**

3. Распределенный трейсинг

а. Zipkin





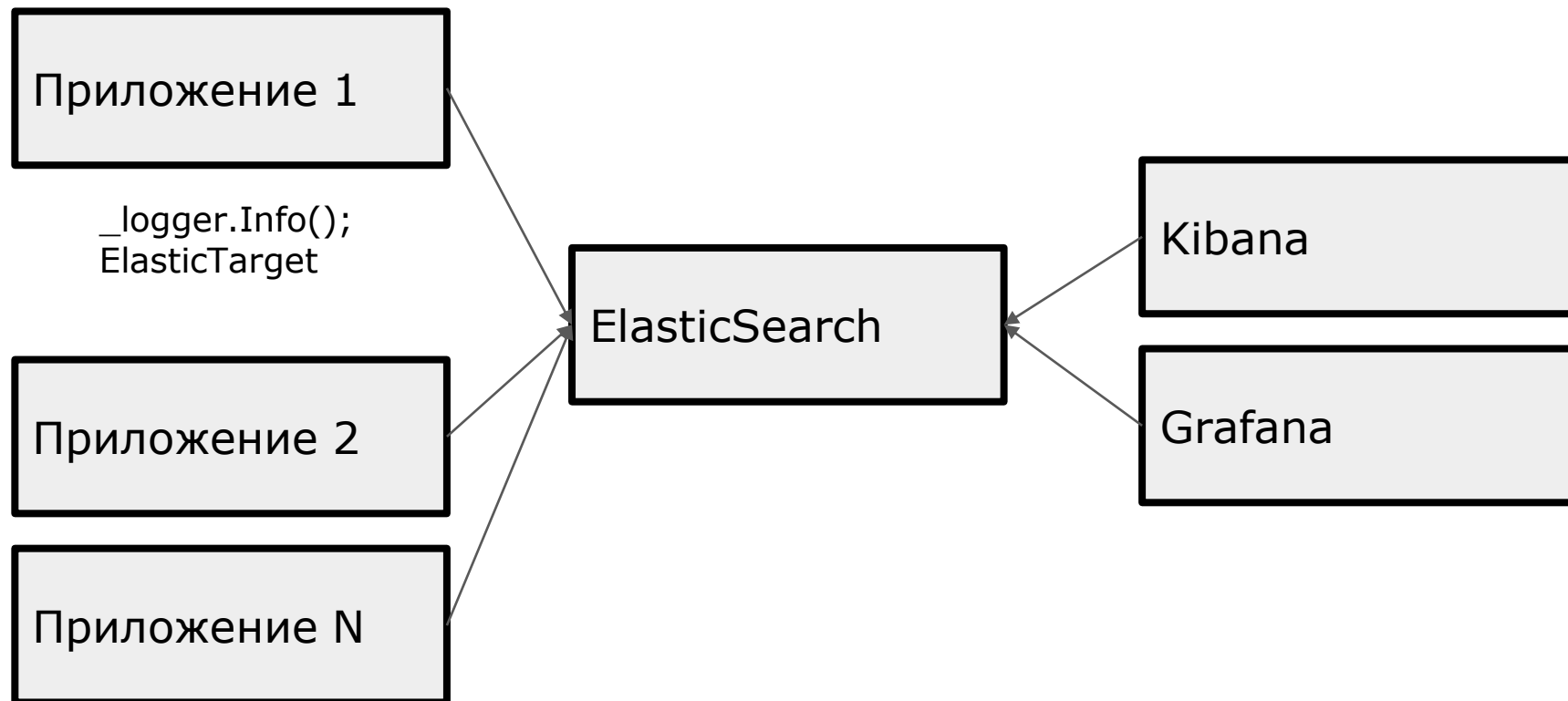
# Logging: Nlog + ElasticSearch + Kibana



- Пример для .net(но можно и другие),
- Структурированное логгирование
- Агрегация логов из разных систем,
- Дешевая визуализация,
- Интеграция
- Errors in RED and 4 golden signals
- Альтернативы:
  - Leg4net, serilog
  - Splunk, ClickHouse, bigquery, ms log analytics, stackdriver



# Logging. Архитектура

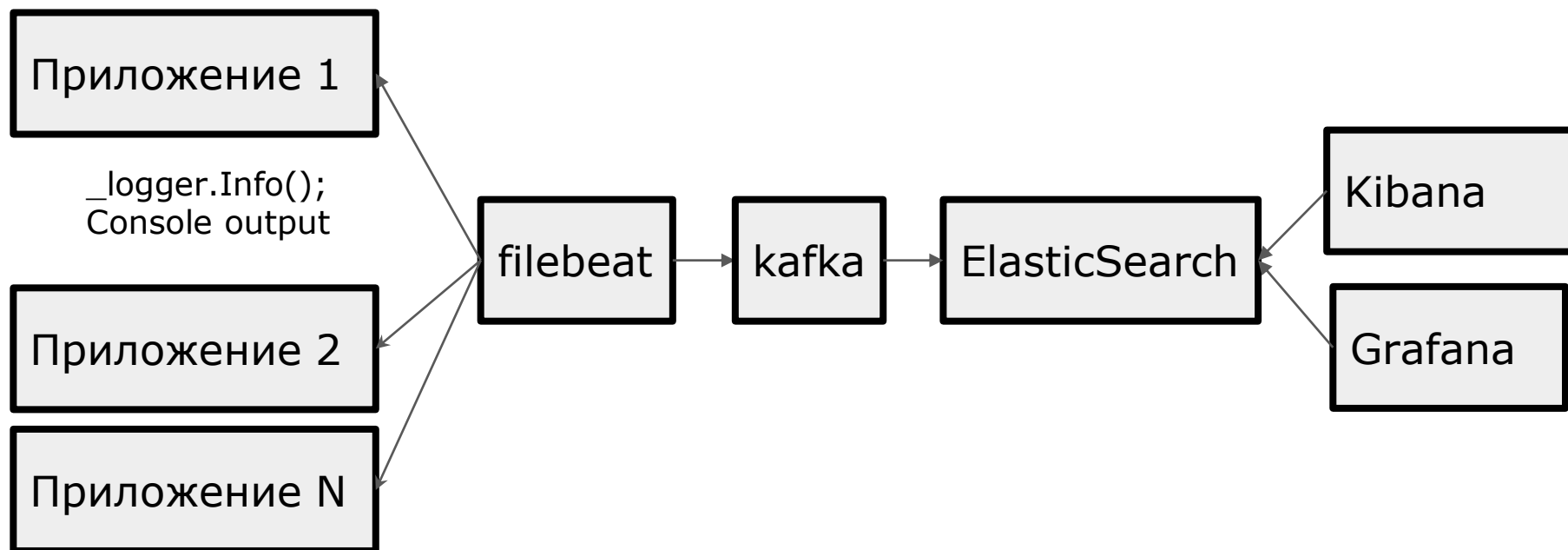


# Logging. Архитектура. Проблемы

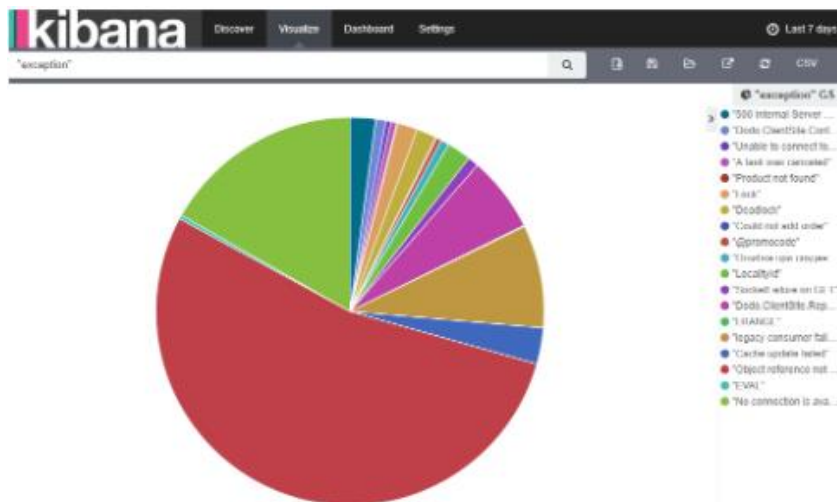
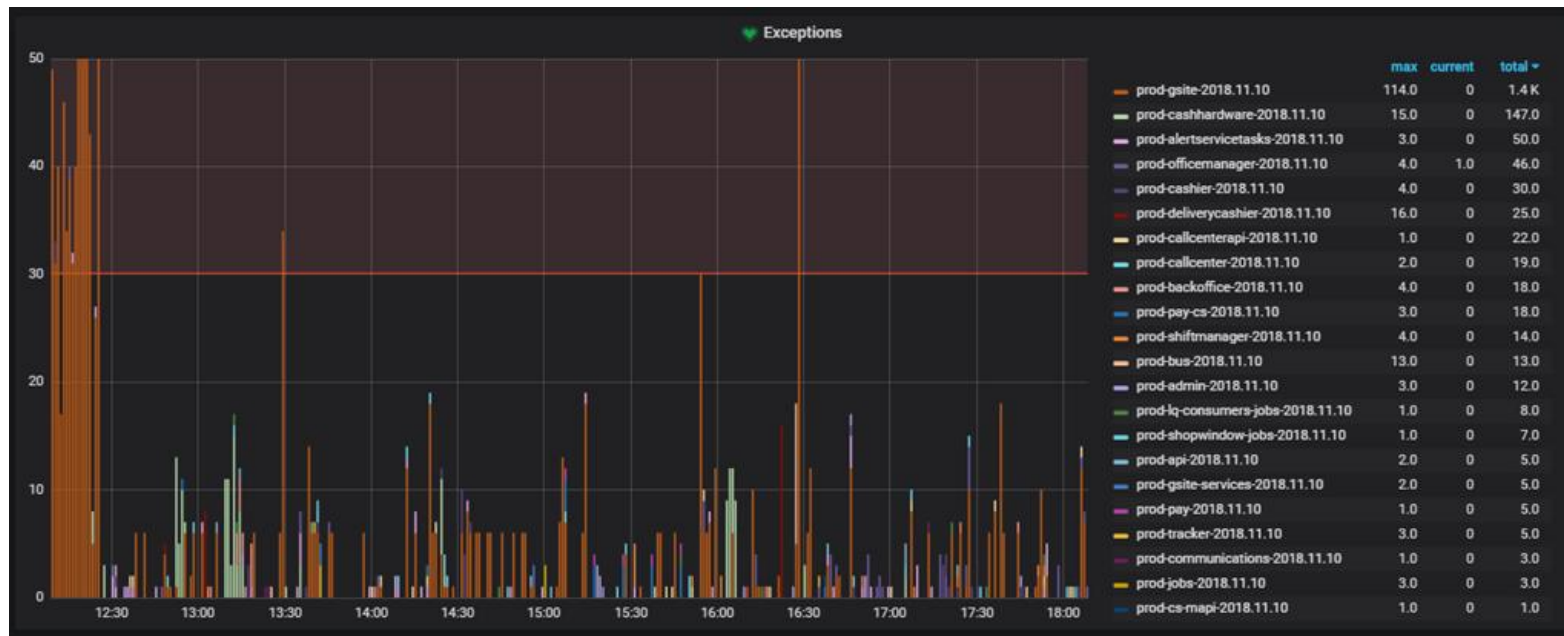
- Лишние зависимости,
- Дублирование ни к чему. Пишем в файл,
- Различные схемы сбора,
- Архитектура. Зависимости



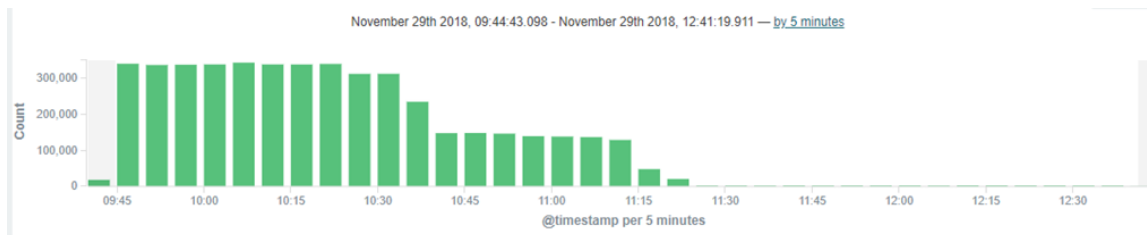
# Logging. Архитектура. v2



# Logging. Использование



# Logging. Производительность



# Logging. Вывод

- т.к. логи мы пишем все равно, то это очень дешевый мониторинг,
- при правильном структурном логгировании можно хорошо визуализировать,
- интеграция с общим мониторингом,
- выполняем важное правило мониторинга - резервирование

## Минусы и проблемы:

- Надо уметь развернуть и поддерживать,
- Хранение данных,
- Нужна сразу правильная архитектура



1. Метрики по сервисам

а. Prometheus

2. Логи

а. Nlog + ELK

**3. Распределенный  
трейсинг**

**а. Zipkin**





# Распределенный трейсинг: Zipkin



1. Помогает в поиске проблем.  
Находим узкие места,
2. Закрывает недостатки в отдельных мониторингах,
3. Видим потоки данных, в том числе при непрямах вызовах
4. Мониторинг внешних систем
5. Может работать как профайлер(частично)
6. Opentracing

RED - Duration(особенно с event-моделью),

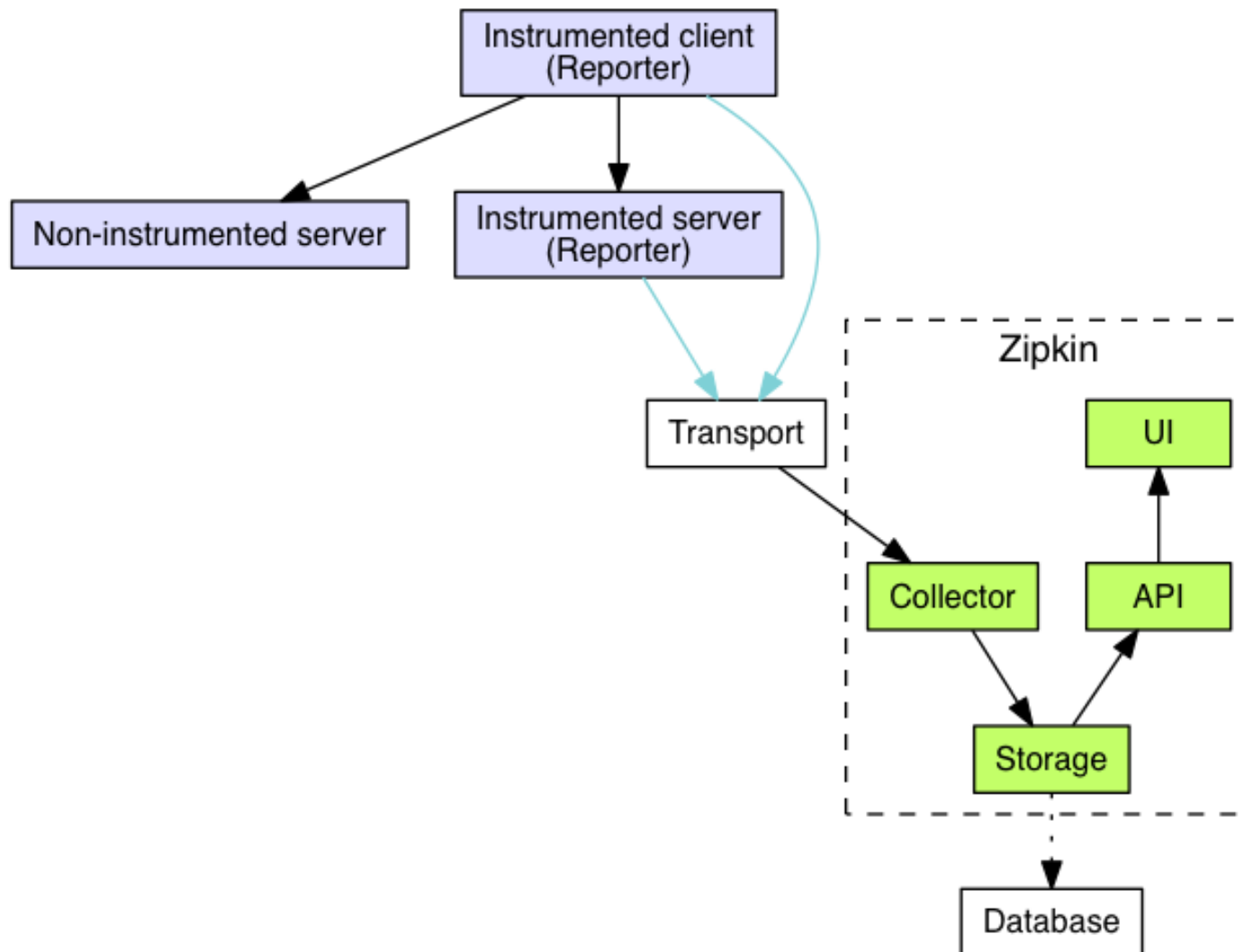
4 gs: Latency

Альтернативы:

- Jaeger



# Zipkin. Архитектура



# Zipkin. Подключение

1. Приходит запрос в приложение,
2. В middleware перед исполнением метода контроллера zipkin получает управление,
3. Если в header есть traceId, то мы устанавливаем его в контекст запроса. Если нет, генерим новый

```
1 public override void OnActionExecuting(ActionExecutingContext filterContext)
2 {
3     var context = filterContext.HttpContext;
4     var traceContext = _traceExtractor.Extract(context.Request.Headers);
5     var trace = traceContext == null ? Trace.Create() : Trace.CreateFromId(traceContext);
6
7     Trace.Current = trace;
8     ...
9     trace.Record(Annotations.Rpc(GetSpanName(filterContext)));
10 ...
11 }
```



# Zipkin. Подключение (продолжение)

4. В коде, если нужно используем трейсер, вставляя в нужных строчках. TraceId берется из контекста запроса.

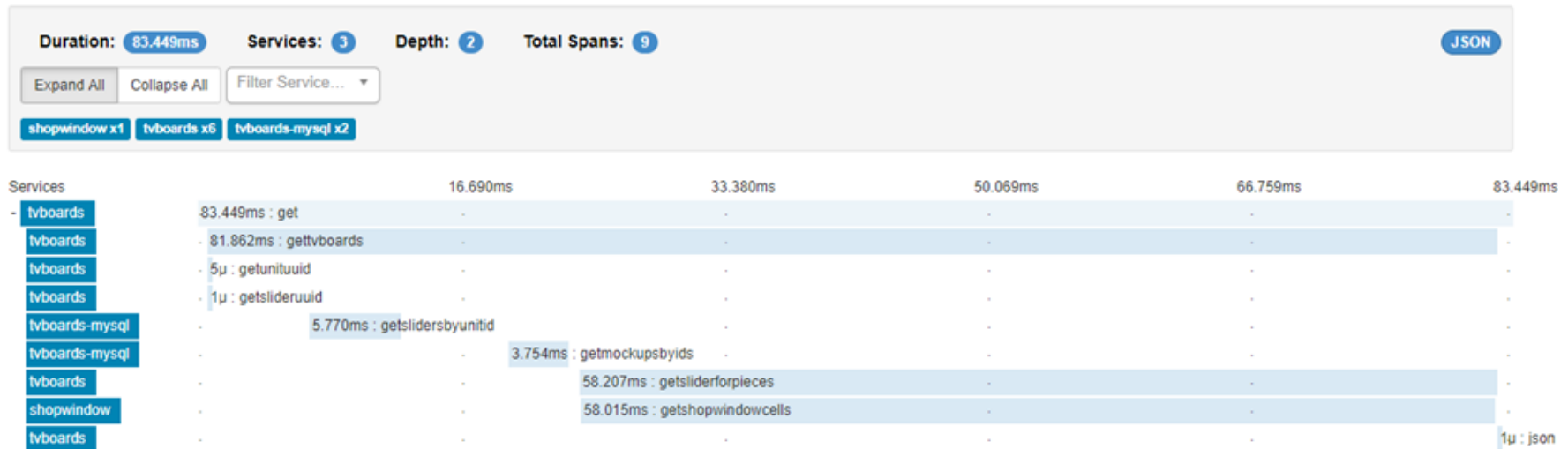
```
1  public async Task<CalculateOrderResponse> CalculateOrder(CalculateOrderRequest request)
2  {
3      return await _tracer.GetAsync(async () =>
4      {
5          ...
6
7          return new CalculateOrderResponse
8          {
9              CalculatedOrder = orderDto,
10         };
11     });
12 }
```

5. Обеспечиваем просовывание traceId в Http вызовы

```
16  serviceCollection
17      .AddHttpClient<IAuthApiClient, AuthApiClient>()
18      .ConfigurePrimaryHttpMessageHandler(() => new TracingHandler(clientName));
```



# Zipkin. Пример



# Zipkin. Вывод

- не всегда хватит Prometheus,
- можно с event-моделью, grpc etc

## Минусы и проблемы:

- Хранение данных. Лучше Jaeger,
- Атрибуты для трейсинга, старые версии asp.net,
- Архитектура. Отправка данных из приложения



# Zipkin. Производительность

```
dependencies  
| where data contains "http://we-mon-1:9411"  
| summarize avg(duration), max(duration), count(duration)|
```

Completed. Showing results from the last 24 hours.


























TABLE CHART | Columns ▾

Drag a column header and drop it here to group by that column

|   | avg_duration ▾ | max_duration ▾ | count_duration ▾ |           |
|---|----------------|----------------|------------------|-----------|
| > | 4.1356372510   | 405            | 63,515           | 2,114,472 |



# Производительность

| Label                                                                                                                                                   |  <a href="#">Show hot path</a> | Metric (ms)When |
|---------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|-----------------|
| ▼  Activity.AspNetCore.Hosting(//1/17729160/)                          |                                                                                                                 | 95.16           |
| ▼  1 Threads                                                           |                                                                                                                 | 95.16           |
| ▼  Thread (46448) CPU=0ms                                              |                                                                                                                 | 95.16           |
| ▼  OTHER <<RtlUserThreadStart>>                                        |                                                                                                                 | 94.93           |
| ▼  TracingMiddleware+<>c__DisplayClass0_0.<UseTracing>b__2             |                                                                                                                 | 94.61           |
|  OTHER <<AsyncTaskMethodBuilder.Start>>                                |                              | 94.61           |
| ▼  TracingMiddleware+<>c__DisplayClass0_0+<<UseTracing>b__2>d.MoveNext |                                                                                                                 | 94.61           |
| ▼  PrometheusMiddleware.Invoke                                         |                                                                                                                 | 93.79           |
|  OTHER <<AsyncTaskMethodBuilder.Start>>                                |                              | 93.79           |
| ▼  PrometheusMiddleware+<Invoke>d__3.MoveNext                          |                                                                                                                 | 93.79           |
| ▼  ErrorHandlingMiddleware.Invoke                                      |                                                                                                                 | 93.77           |
|  OTHER <<AsyncTaskMethodBuilder.Start>>                                |                              | 93.77           |
| ▼  ErrorHandlingMiddleware+<Invoke>d__3.MoveNext                      |                                                                                                                 | 93.77           |
| ▼  CollectRequestMetricsMiddleware.Invoke                            |                                                                                                                 | 93.76           |
|  OTHER <<AsyncTaskMethodBuilder.Start>>                              |                            | 93.76           |
| ▼  CollectRequestMetricsMiddleware+<Invoke>d__7.MoveNext             |                                                                                                                 | 93.76           |
| ▼  OTHER <<MapWhenMiddleware.Invoke>>                                |                            | 93.74           |
| ▼  dynamicClass.lambda_method                                        |                                                                                                                 | 92.96           |
|  AuthController.CreatePermissions                                    |                                                                                                                 | 92.96           |





# **История оптимизации**

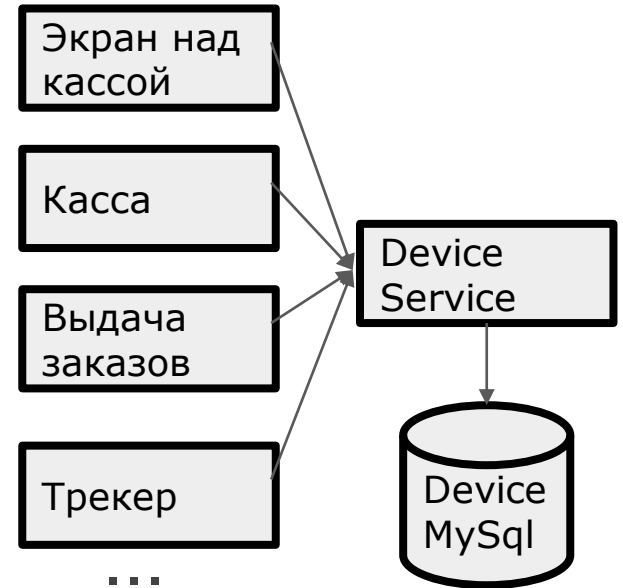




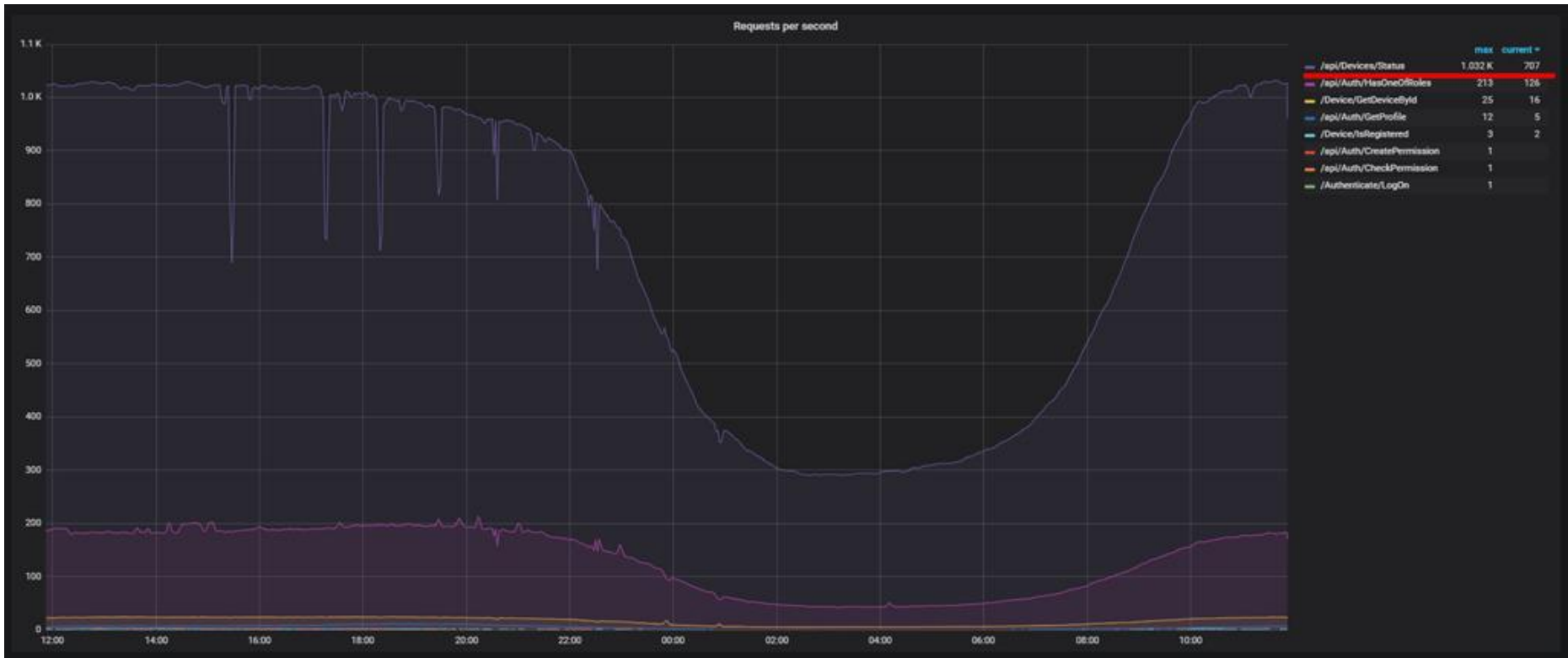
## Было



## Хочется



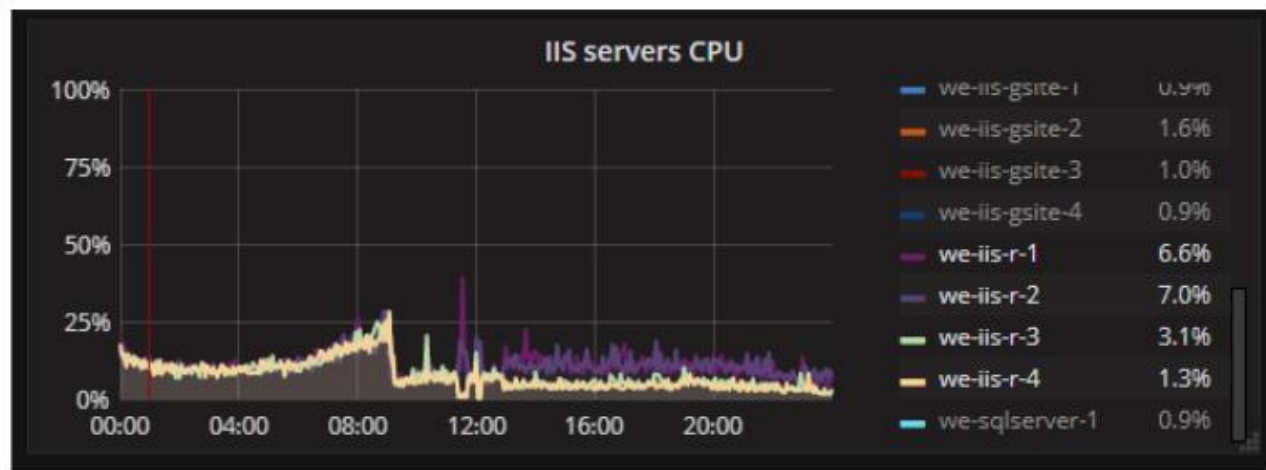
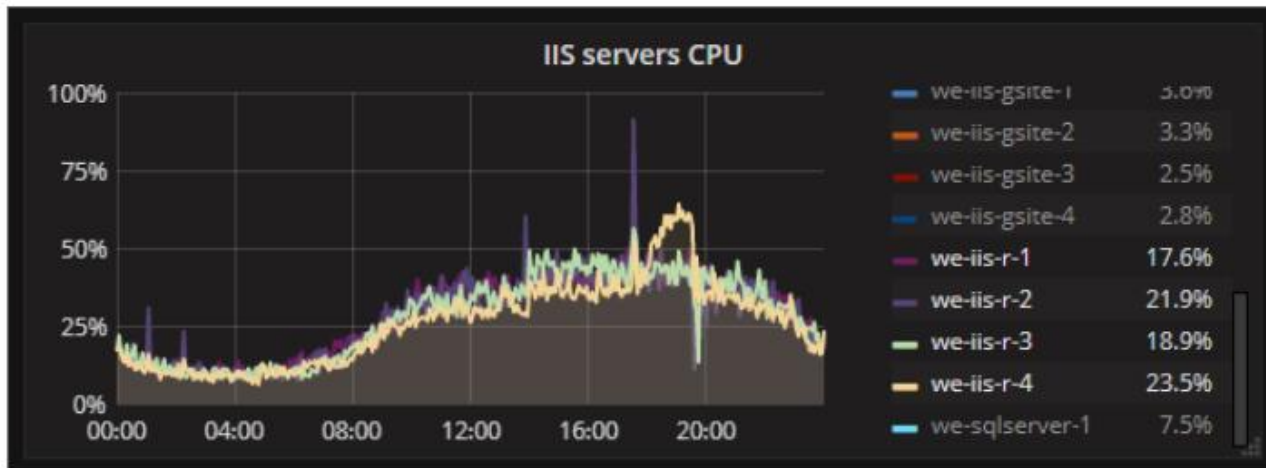
# Много запросов, 1k/sec



# Переключаем на сервис



# После выкладки



# Как узнаем?

Prometheus + Telegraf + Grafana

<https://github.com/influxdata/telegraf>

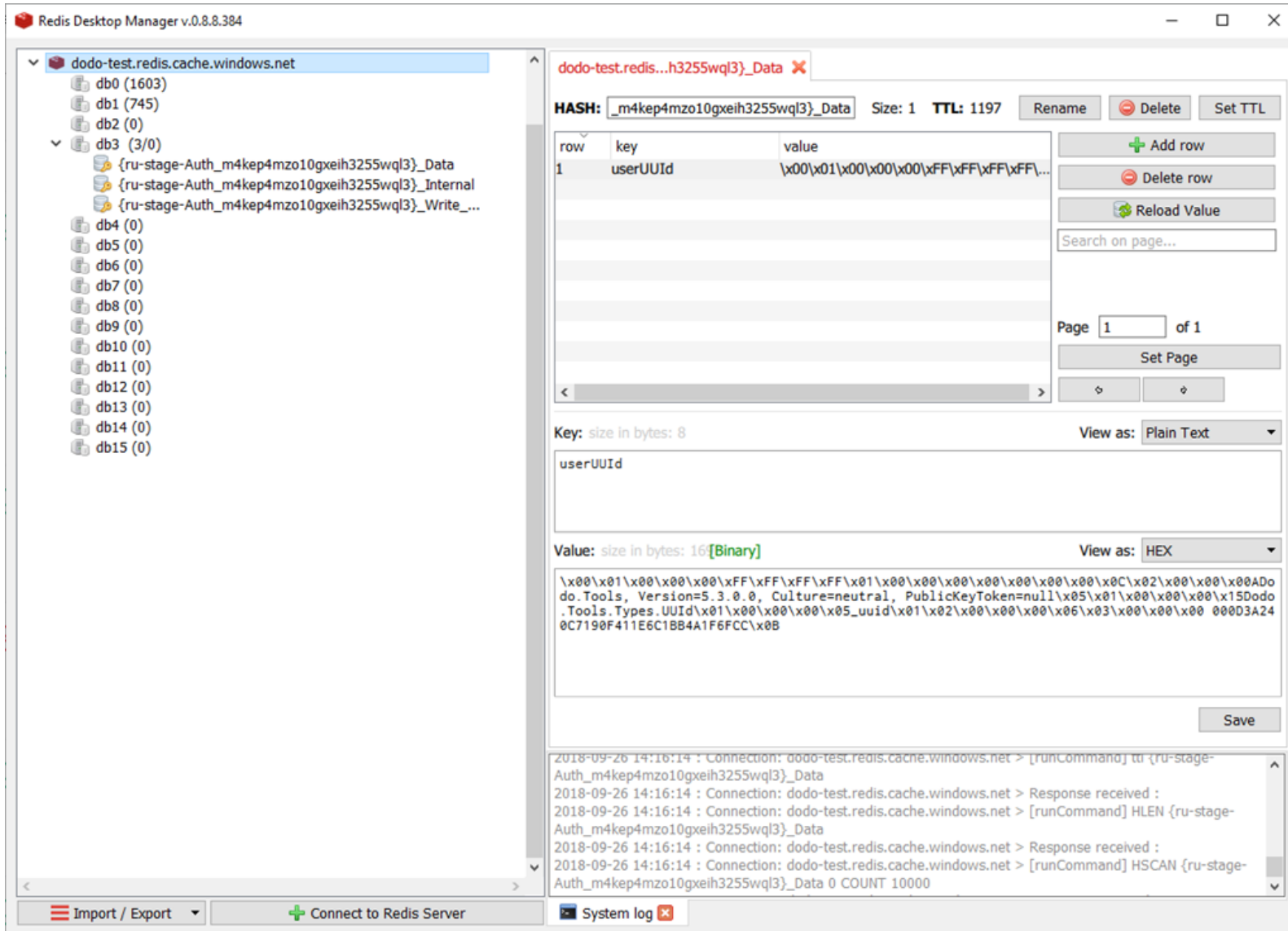


**В чем дело?**





# Сессия



```
1 private ISerializer GetSerializer()  
2 {  
3     string serializerTypeName = _configuration.RedisSerializerType;  
4     if (!string.IsNullOrEmpty(serializerTypeName))  
5     {  
6         var serializerType = Type.GetType(serializerTypeName, true);  
7         if (serializerType != null)  
8         {  
9             return (ISerializer)Activator.CreateInstance(serializerType);  
10        }  
11    }  
12    return new BinarySerializer();  
13 }
```



```

1 namespace Microsoft.Web.Redis
2 {
3     public class BinarySerializer : ISerializer
4     {
5         public byte[] Serialize(object data)
6         {
7             if (data == null)
8             {
9                 data = new RedisNull();
10            }
11            var binaryFormatter = new BinaryFormatter();
12            using (var memoryStream = new MemoryStream())
13            {
14                binaryFormatter.Serialize(memoryStream, data);
15                byte[] objectDataAsStream = memoryStream.ToArray();
16                return objectDataAsStream;
17            }
18        }
19
20        public object Deserialize(byte[] data)
21        {
22            if (data == null)
23            {
24                return null;
25            }
26            var binaryFormatter = new BinaryFormatter();
27            using (var memoryStream = new MemoryStream(data, 0, data.Length))
28            {
29                memoryStream.Seek(0, SeekOrigin.Begin);
30                object retObject = (object)binaryFormatter.Deserialize(memoryStream);
31                if (retObject.GetType() == typeof(RedisNull))
32                {
33                    return null;
34                }
35                return retObject;
36            }
37        }
38    }
39 }

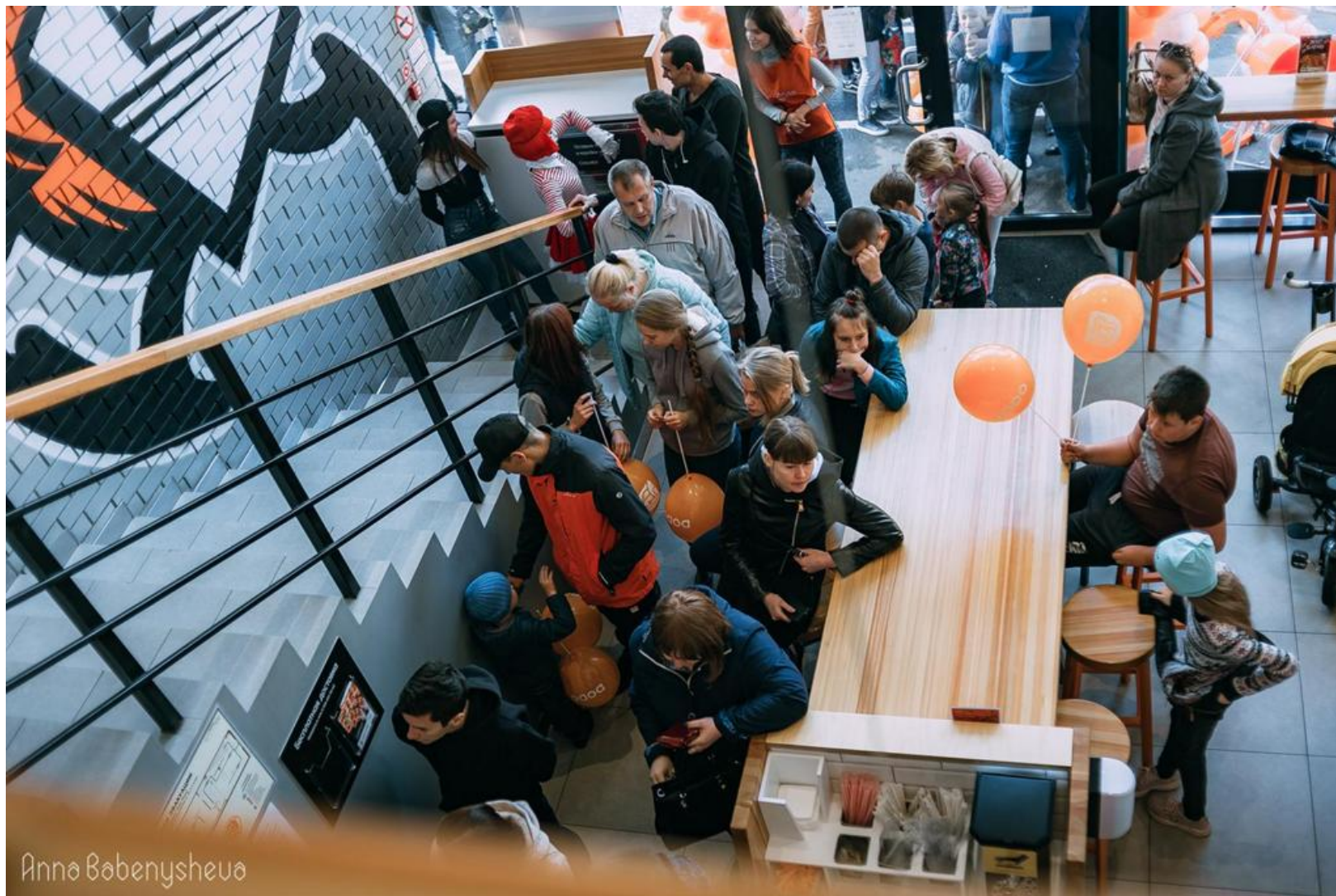
```



# История падения



# Пятница в пиццерии.



Anna Bobenysheva



# Начинаются проблемы...



Grafana Alerting APP 21:12

[Alerting] 5xx/min total alert

5xx more than 200/min

backoffice.dodopizza.ru

94.072362634334



Grafana v5.2.3 | Nov 10th



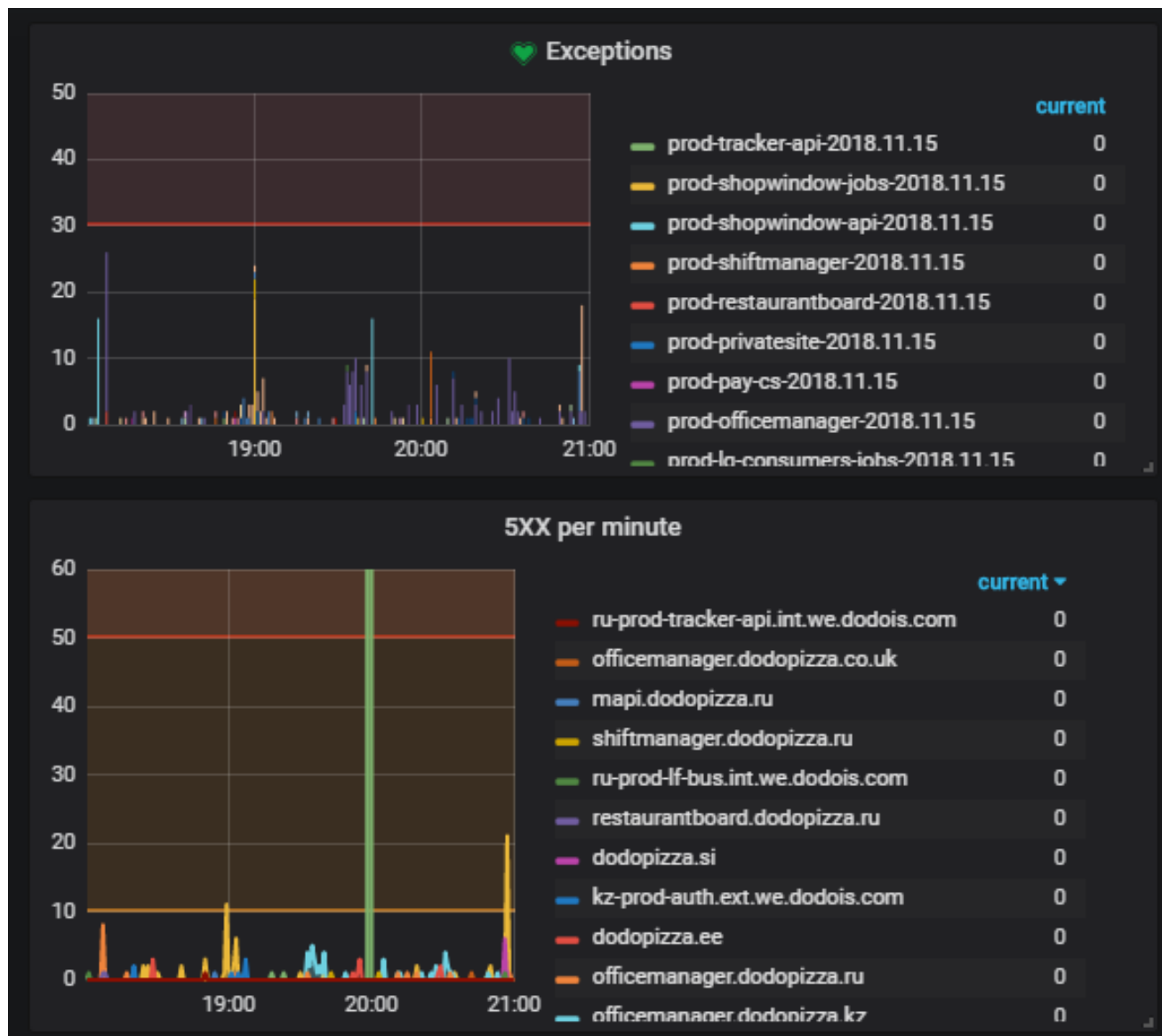
HAL 9000 APP 21:12

fRPettr8iHMG7F2yzls8 ▼





# Смотрим ошибки на мониторинге

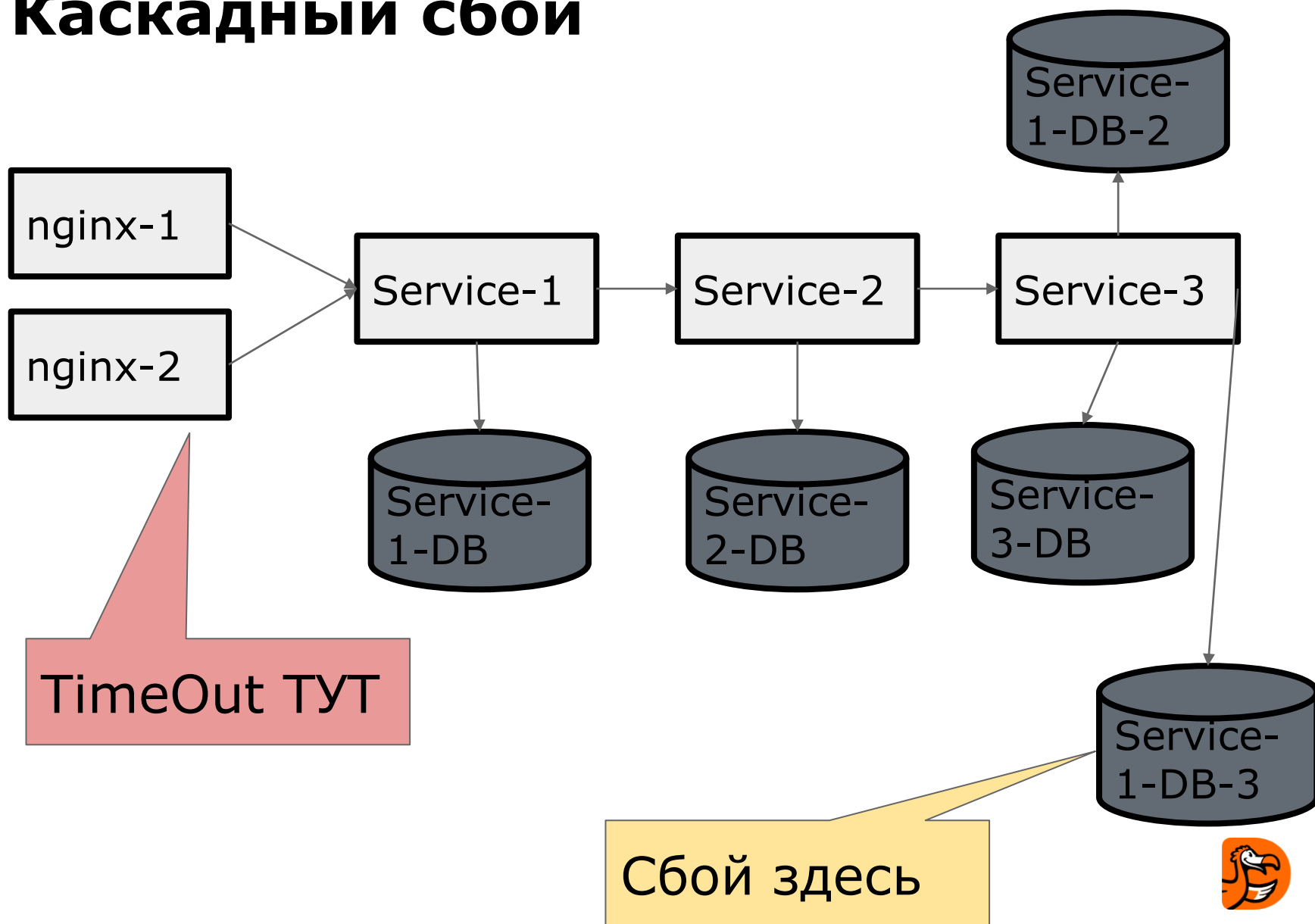


# ELK в действии

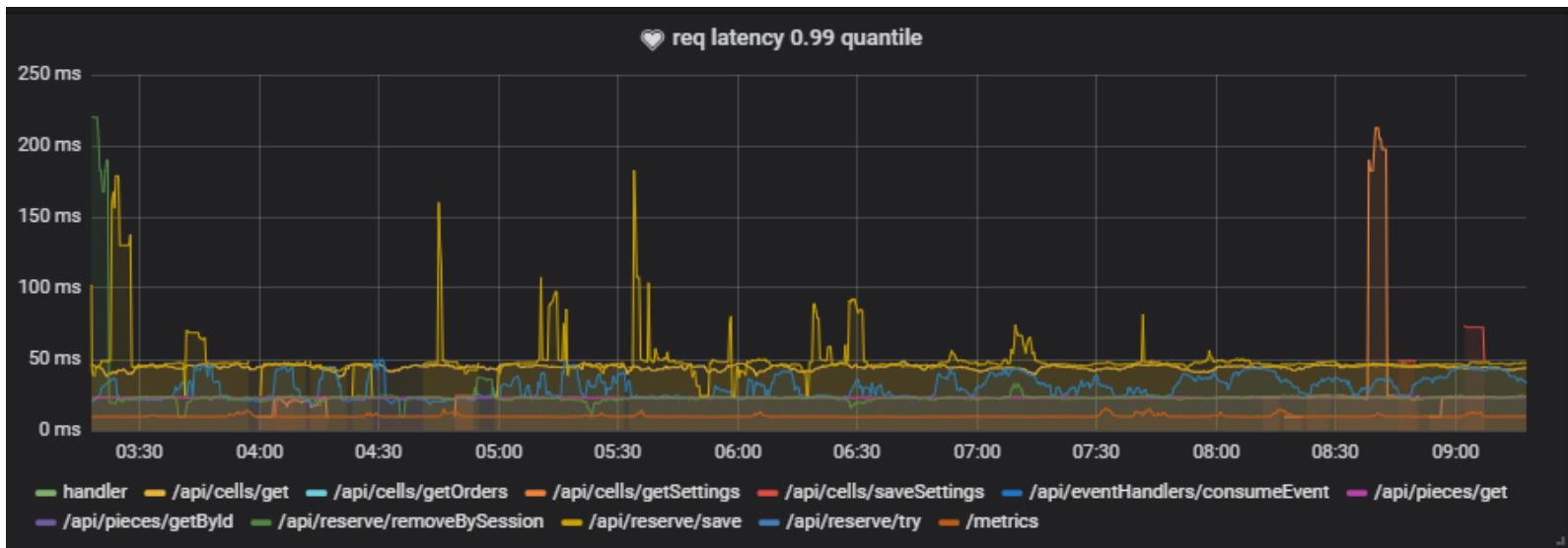
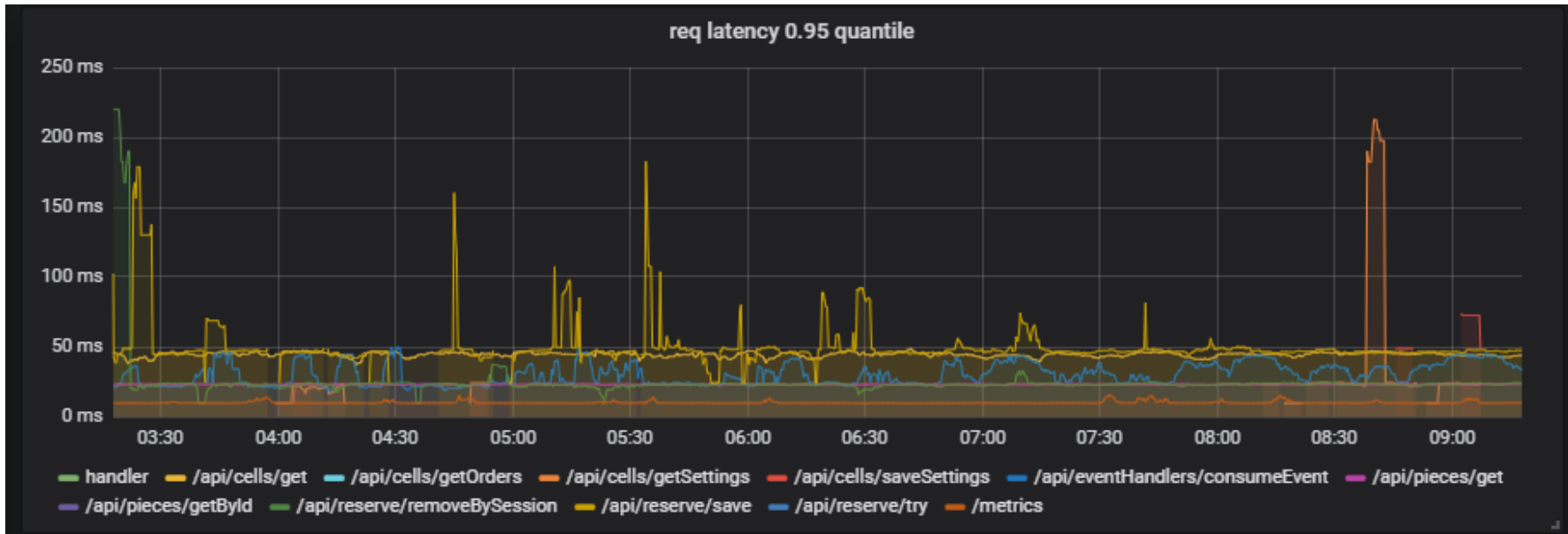




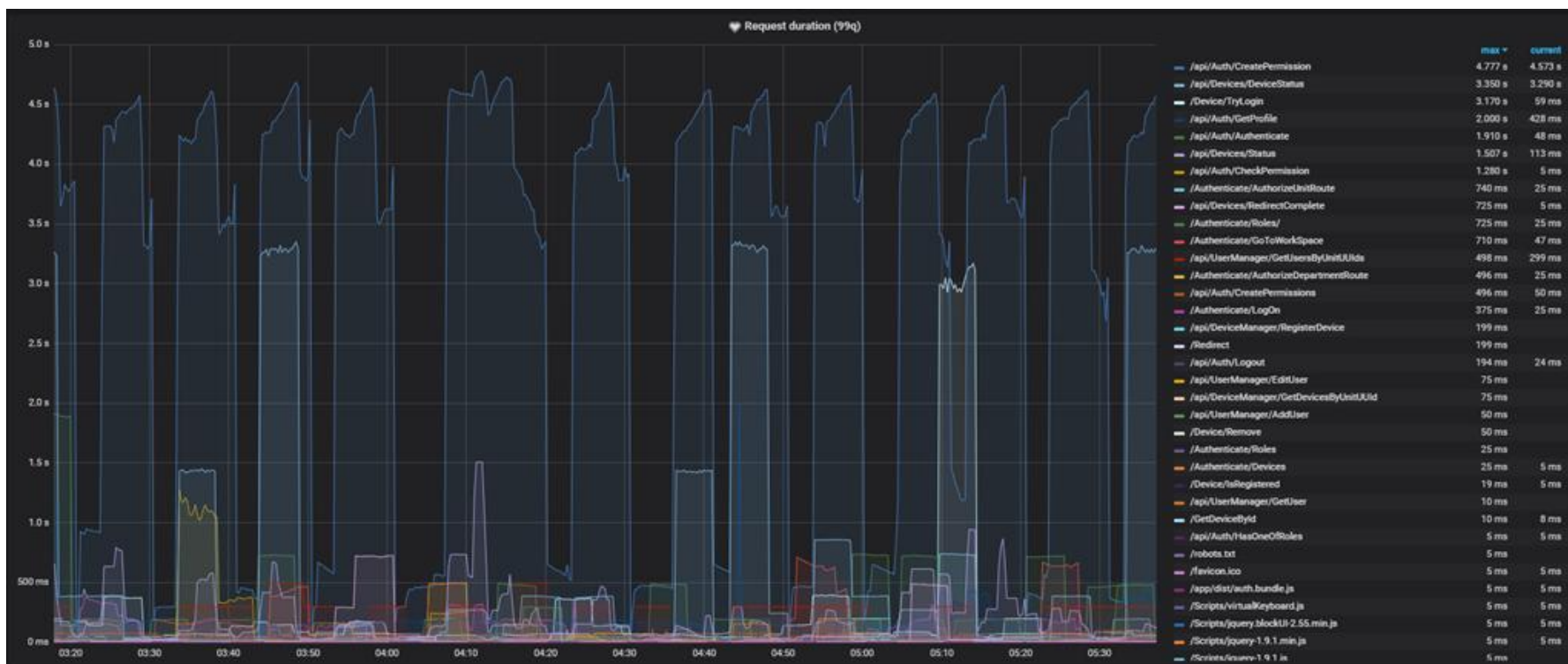
# Каскадный сбой



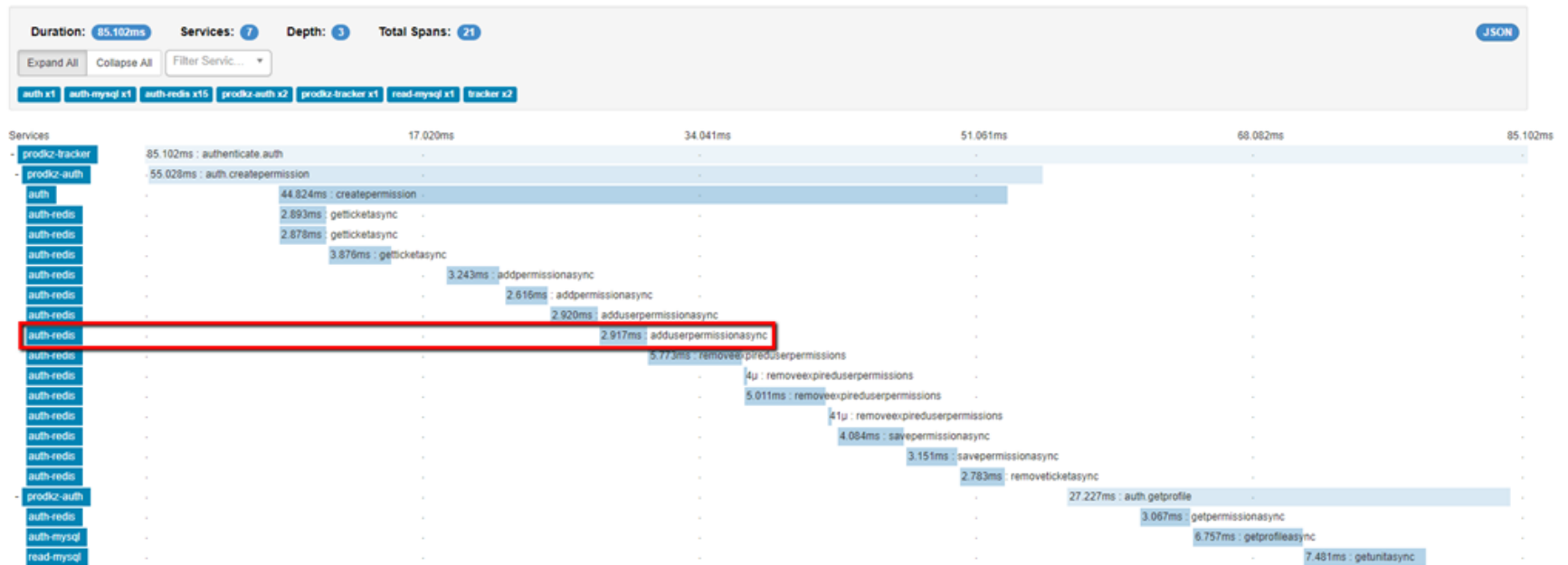
# Смотрим на request duration



# Сервис авторизации тормозит



# Zipkin



# В коде

```
1 private async Task AddUserPermissionAsync(string userId, Permission permission)
2 {
3     var userPermissions = await ExecuteAsync(
4         async () => (
5             await db.HashGetAllAsync(key)).ToList())
6         ?? new List<HashEntry>();
7     userPermissions.Add(permission.Token, 1);
8
9     await ExecuteAsync(async () =>
10         await db.HashSetAsync(key, userPermissions.ToArray()));
11     ...
12 }
```

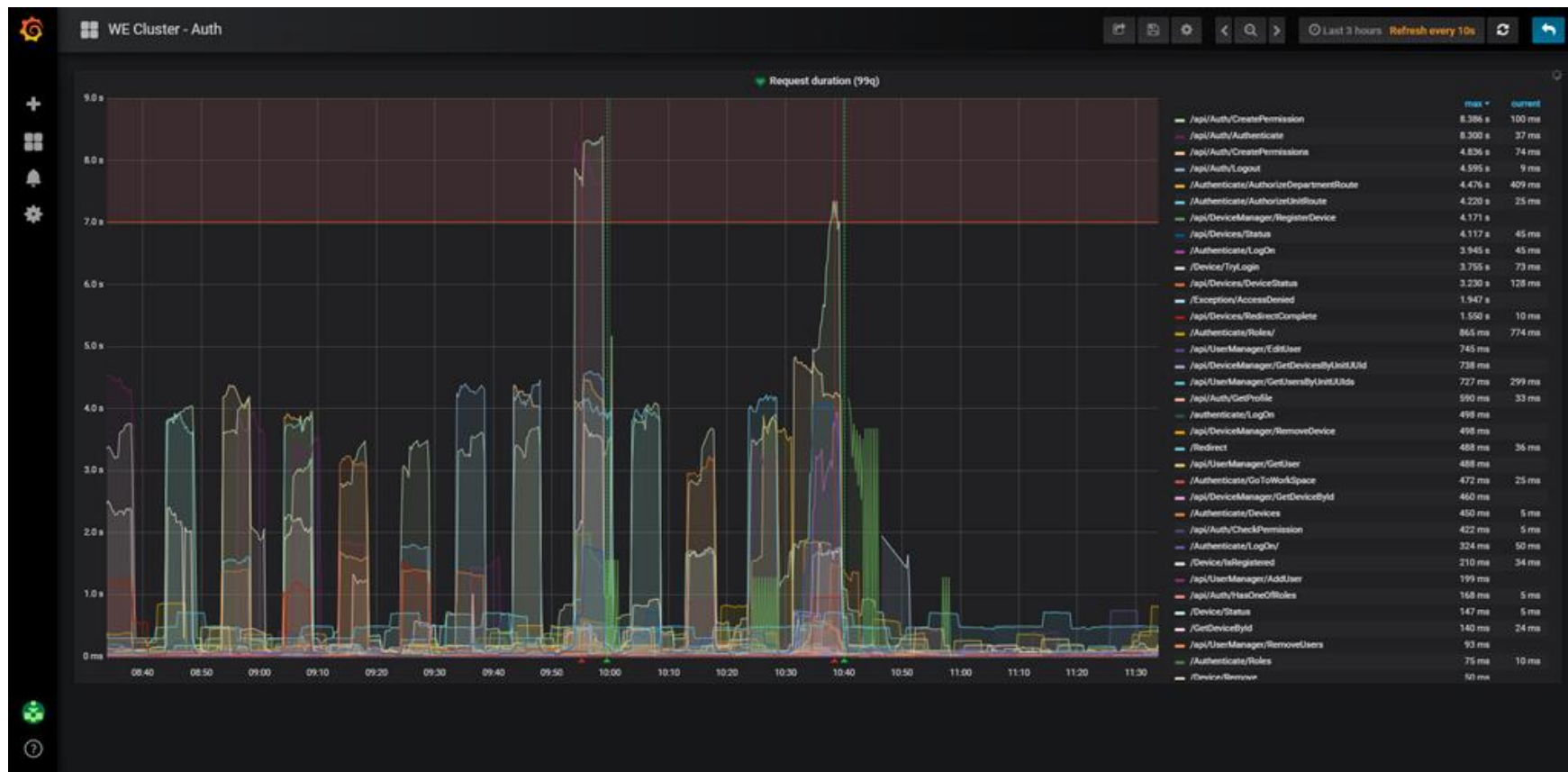


# Что делать?

- Сейчас: чистим редис руками.
- Ближайшее: время переписываем на не блокирующий алгоритм.



# Когда починили



# Правила хорошего мониторинга

1. Постепенное внедрение.
2. Разные части
3. Резервирование
4. Дисциплина в использовании
5. Договориться о названиях







# Выводы. Мониторинг нам помогает

1. Лучше контролировать и понимать систему,
2. Быстро реагировать на сбои
3. Выполнять нефункциональные требования к системе, критерий качества выполненных работ,
4. Искать и устранять неисправности, багов

# Вопросы

Павел Притчин

Dodo Pizza, Dodo IS Core Team

[p.pritchkin@dodopizza.com](mailto:p.pritchkin@dodopizza.com)

Telegram: @ppritchkin

