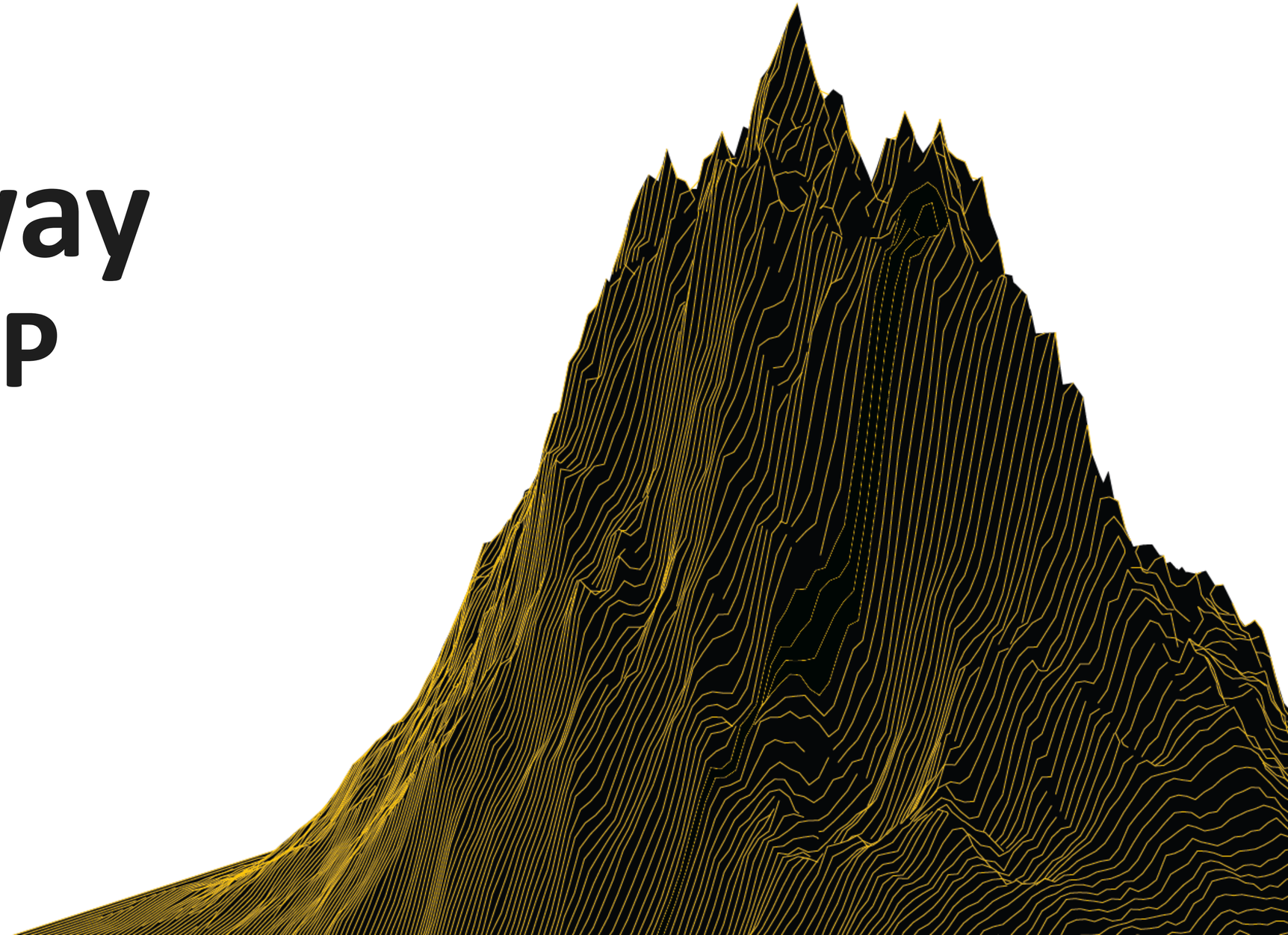
beetech

API Gateway на базе YARP



Алексей Галлямов

2024, DotNext

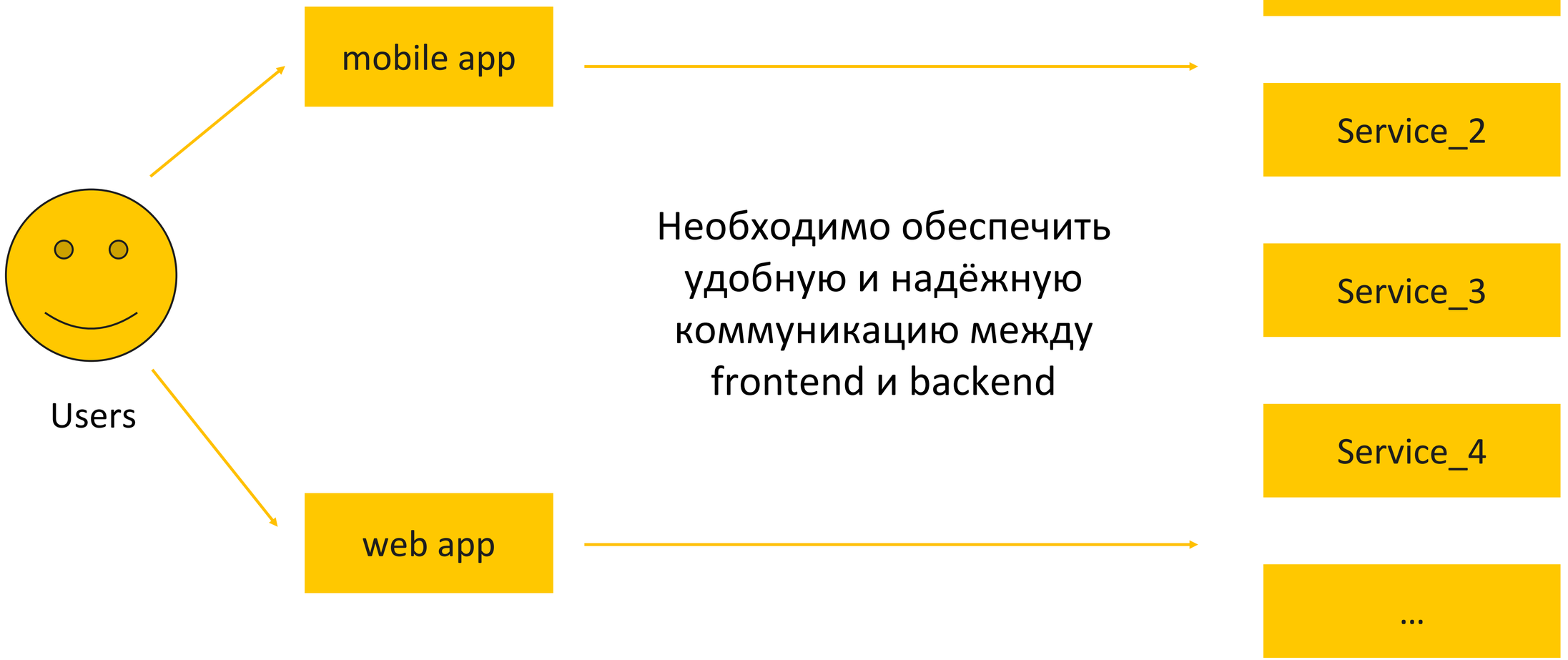
О себе

Алексей Галлямов

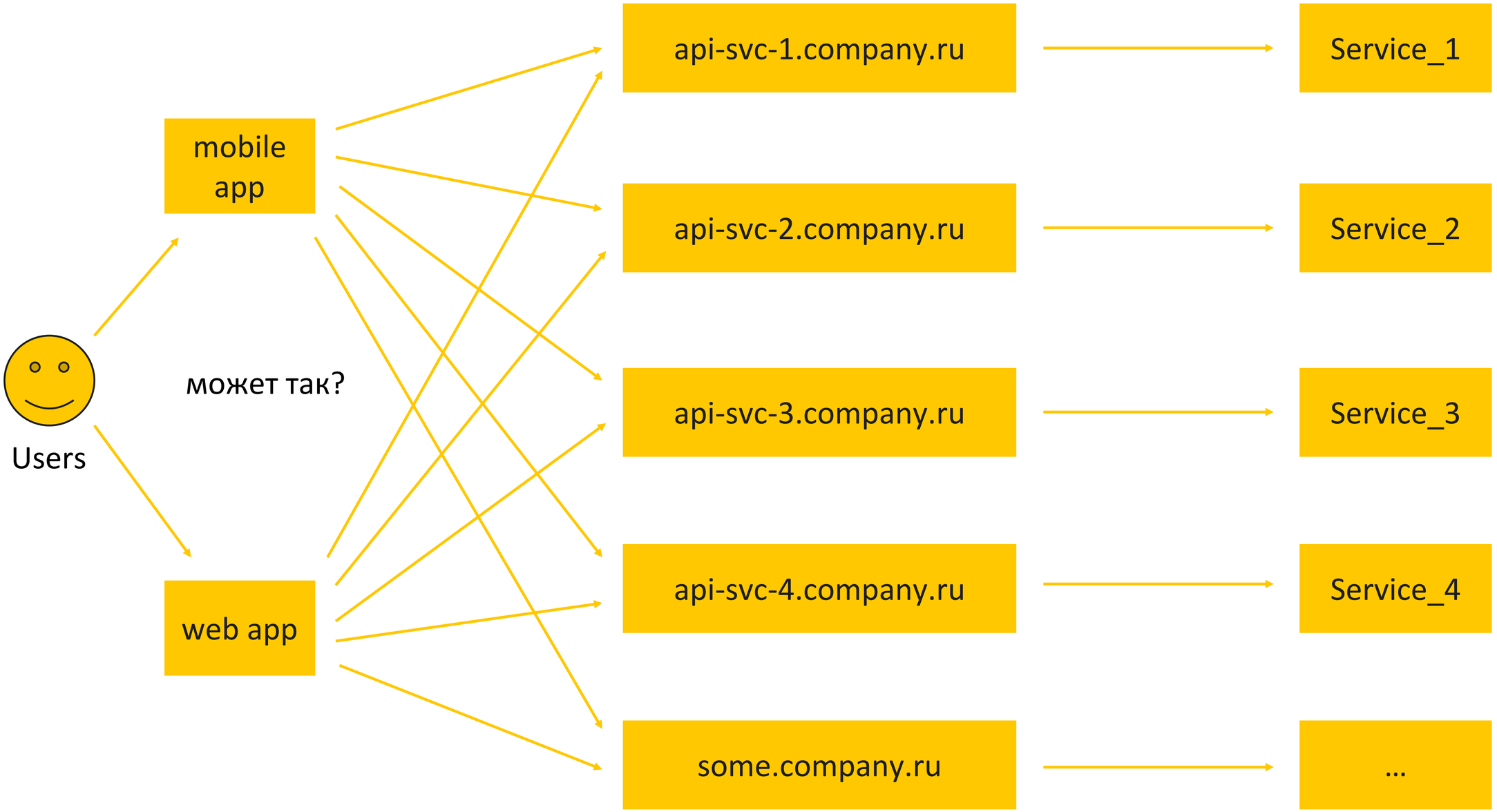
руководитель управления разработки в **beeline**

- Занимаемся продуктовой разработкой ПО:
  - мобильные приложения,
  - веб-приложения,
  - бекенд и микро-сервисы
- Автоматизируем процессы продаж и обслуживания наших клиентов, в том числе в партнёрских каналах

Задача


mobile app
web app
Users
Необходимо обеспечить удобную и надёжную коммуникацию между frontend и backend
Service_1
Service_2
Service_3
Service_4
...

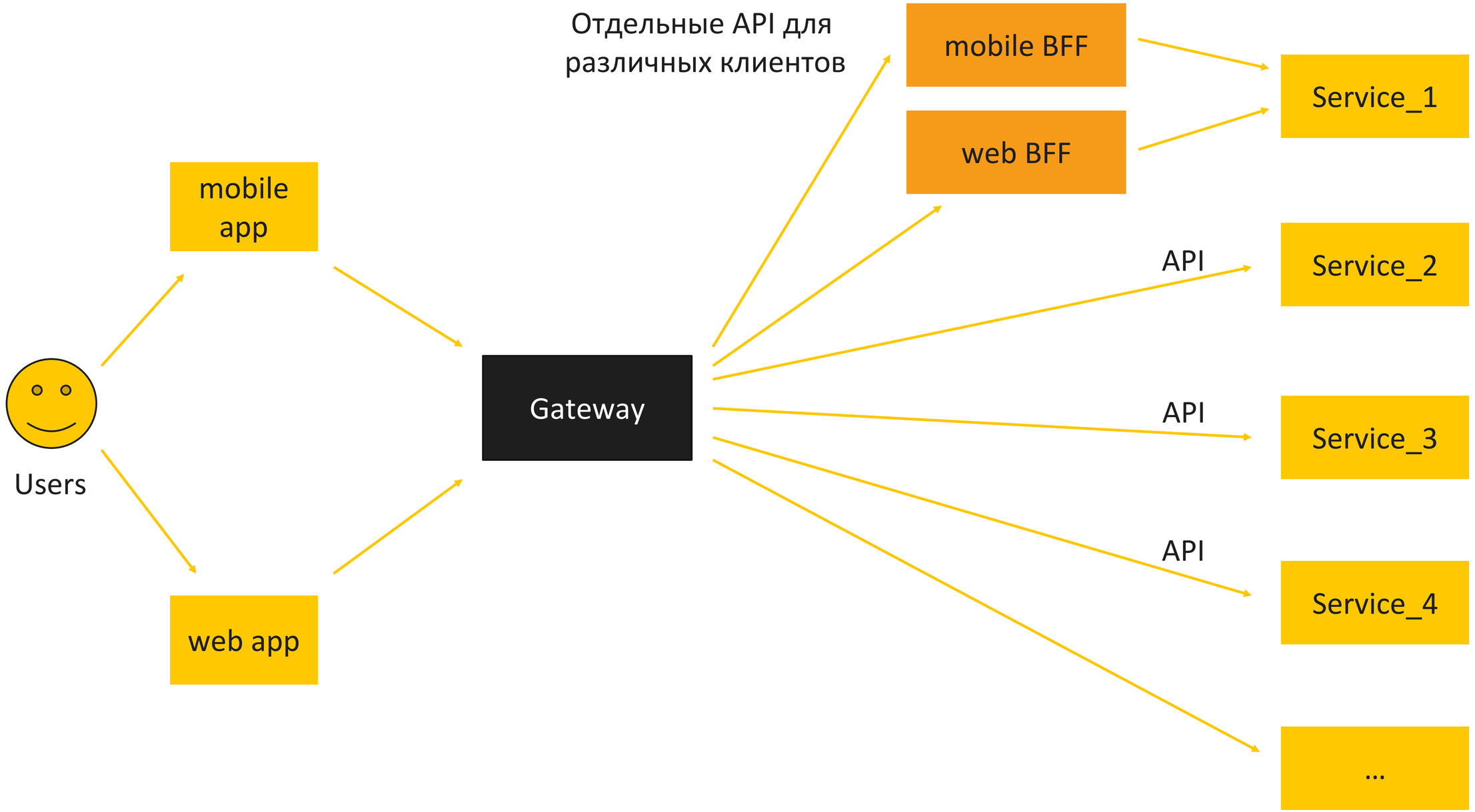
Задача


mobile app
web app
Users
может так?
api-svc-1.company.ru
api-svc-2.company.ru
api-svc-3.company.ru
api-svc-4.company.ru
some.company.ru
Service_1
Service_2
Service_3
Service_4
...

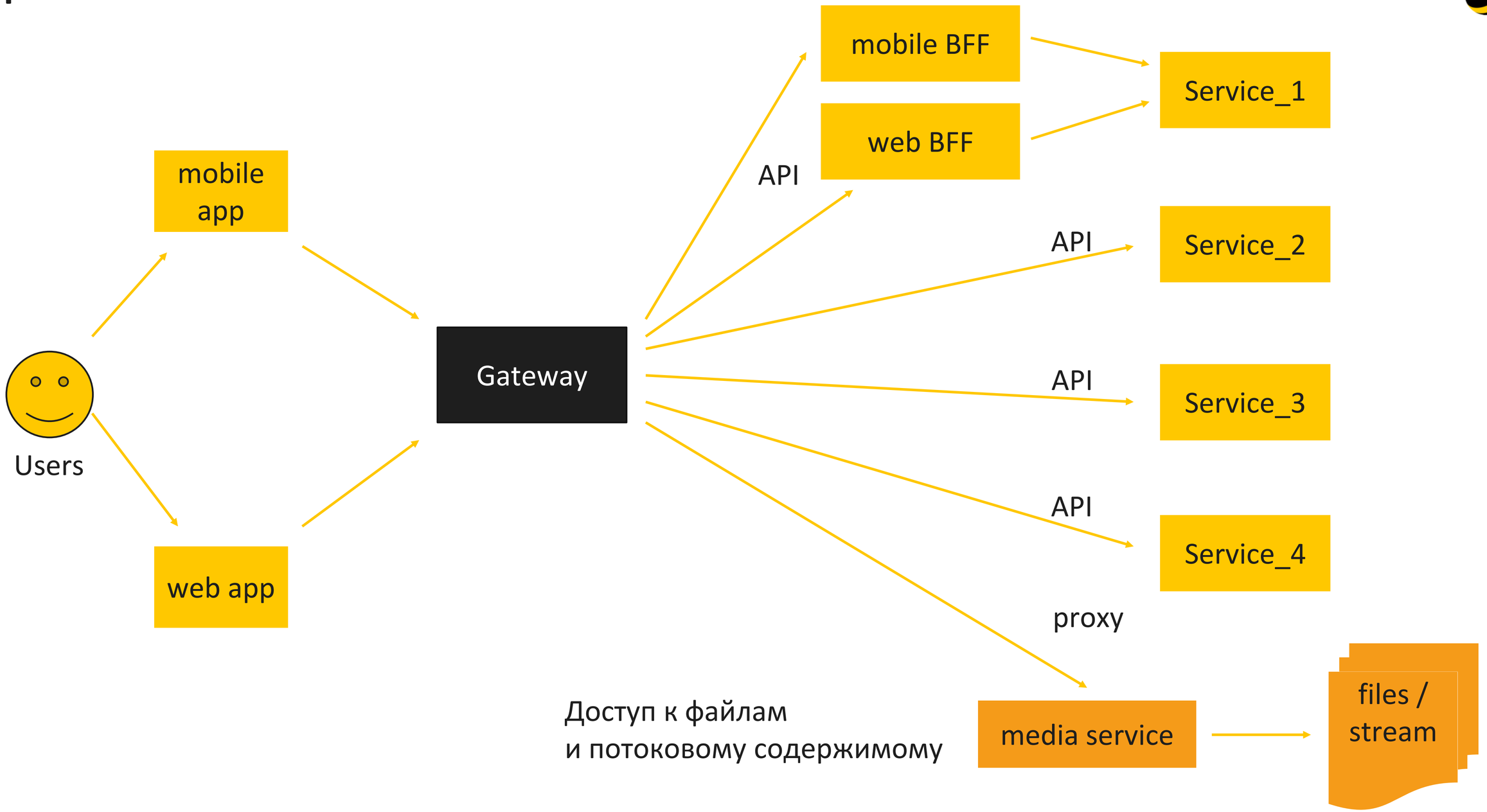
Задача


Напрашивается промежуточный
узел, который будет принимать
все запросы и распределять
их по backend-ам
mobile
app
web app
Users
Gateway
API
API
API
API
Service_1
Service_2
Service_3
Service_4
...

Задача

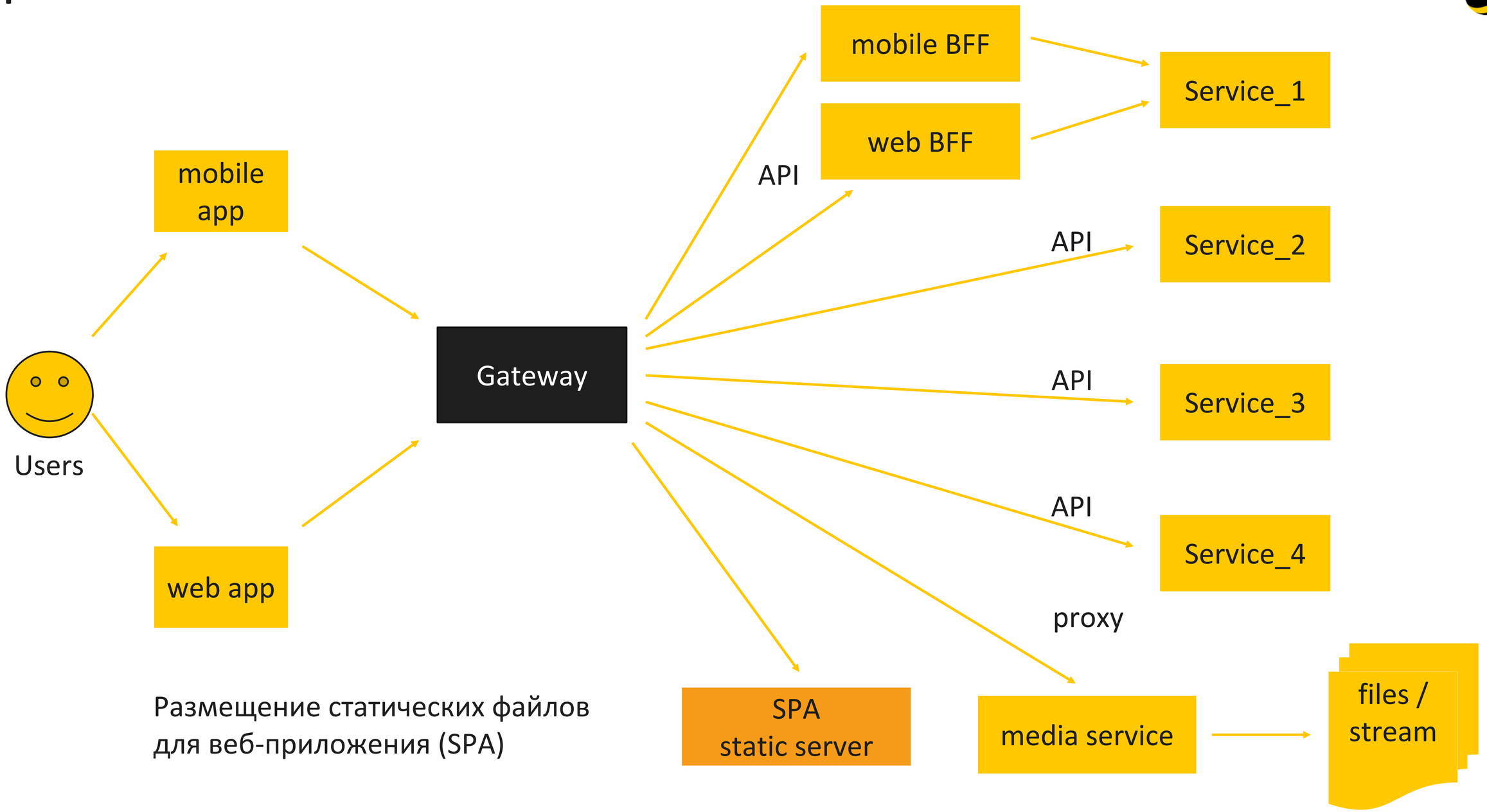

Отдельные API для различных клиентов
mobile BFF
web BFF
Service_1
mobile app
API
Service_2
Users
Gateway
API
Service_3
API
Service_4
web app
...

Задача



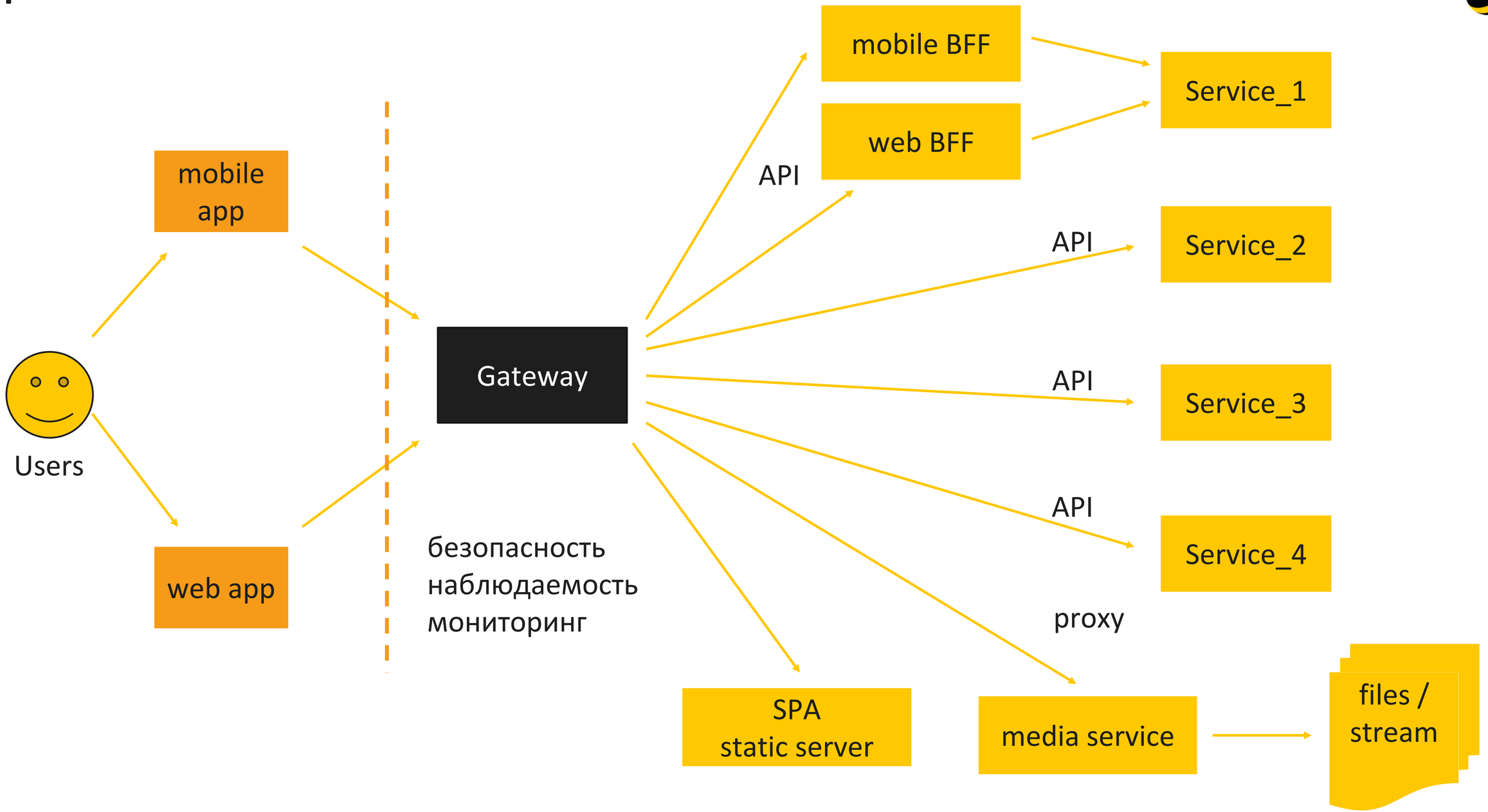
Доступ к файлам
и потоковому содержимому

Задача



Размещение статических файлов для веб-приложения (SPA)

Задача


mobile BFF
web BFF
Service_1
mobile app
API
API
Service_2
Gateway
API
Service_3
API
Service_4
Users
web app
безопасность
наблюдаемость
мониторинг
proxy
SPA
static server
media service
files / stream

Паттерны для решения

Паттерны для решения

Паттерны для решения

частный случай

частный случай

reverse proxy	API Gateway	BFF (backend for frontend)
• TLS терминация (HTTPS => HTTP) • балансировка • кеширование • сжатие • ограничение скорости • маршрутизация • наблюдаемость • протоколы gRPC, WebSocket	• авторизация запросов • параметрическая маршрутизация • проверка содержимого • трансформация запросов/ответов • прикладная наблюдаемость	• авторизация пользователя • формирование API для конкретного клиента • агрегация запросов

Существующие решения

Существующие решения

HAPROXY

Kong

IIS

Слишком низкоуровневые

NGINX

envoy

krakenD

Существующие решения

Слишком избыточны

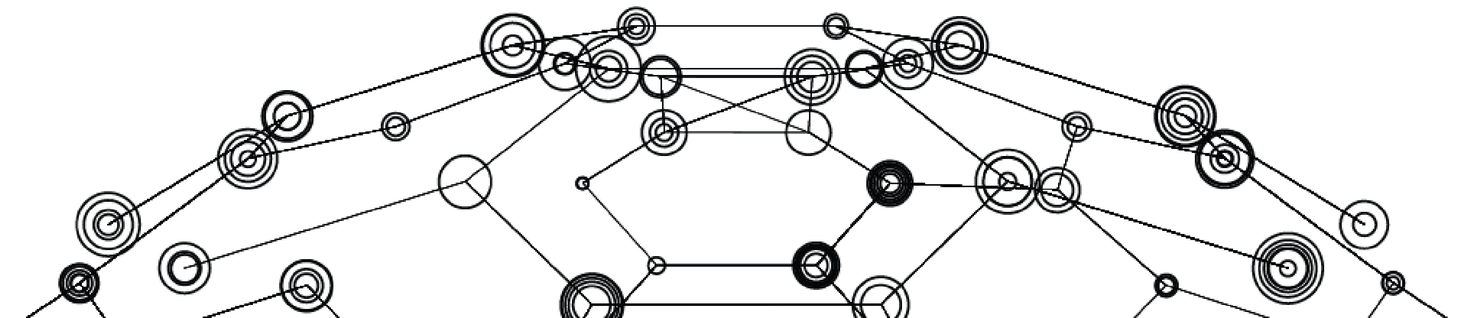
Существующие решения в мире .NET

ProxyKit

OCELOT

Microsoft YARP

(Yet Another Reverse Proxy)



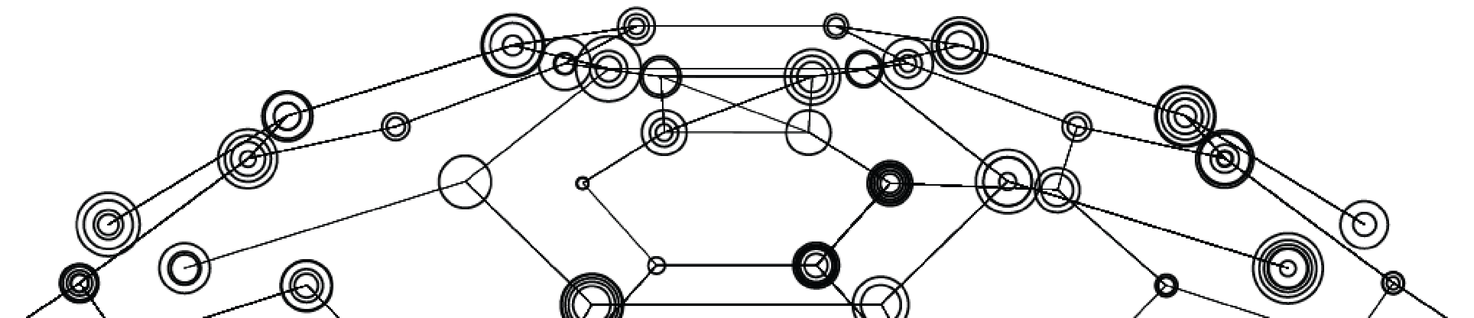
Существующие решения в мире .NET

ProxyKit

Разработка прекращена
в июле 2021 года

OCELOT

Microsoft YARP
(Yet Another Reverse Proxy)



Ocelot

- https://github.com/ThreeMammals/Ocelot
- Open Source решение, развивающееся с 2016 года
- Позиционирует себя непосредственно как API Gateway
- Представляет из себя набор библиотек и расширений
- Встраивается в приложение ASP.NET
- middleware, выстроенные в определённом порядке
- Реализует собственную маршрутизацию
- Поддерживает WebSocket
- Поддержка агрегации запросов
- Поддержка динамической конфигурации
- Поддержка Service Discovery

Microsoft YARP

(Yet Another Reverse Proxy)

- https://github.com/microsoft/reverse-proxy
- Open Source решение от Microsoft, развивающееся с 2020 года
- Позиционирует себя как Reverse Proxy (level 7 HTTP proxy)
- Представляет из себя библиотеку, встраиваемую в ASP.NET, как endpoints
- Интегрируется в маршрутизацию ASP.NET
- Поддерживает WebSocket, gRPC и прямую потоковую переадресацию запросов
- Имеет декларативную расширяемость
- Полная поддержка динамической конфигурации

Ограничения и минусы Ocelot

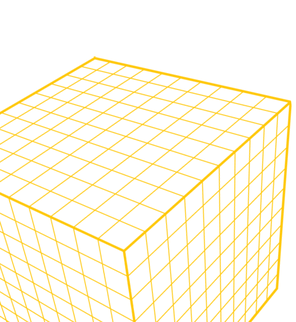
- Нет поддержки gRPC
- При определении маршрутов требуется многократно указывать адреса сервисов (Downstream***)
- Избыточная конфигурация (ocelot.json)
- Возможности расширяются компонентами (плагинами), отдельно от ASP.NET
- Не предоставляет прямую потоковую переадресацию
- Нет поддержки крупного вендора

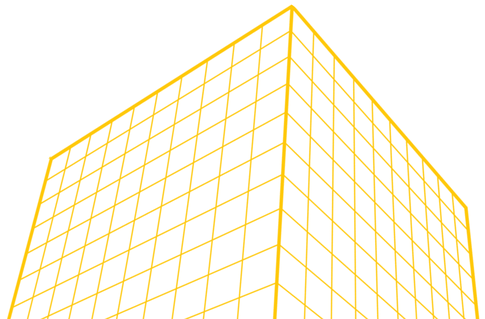
Ограничения и минусы YARP

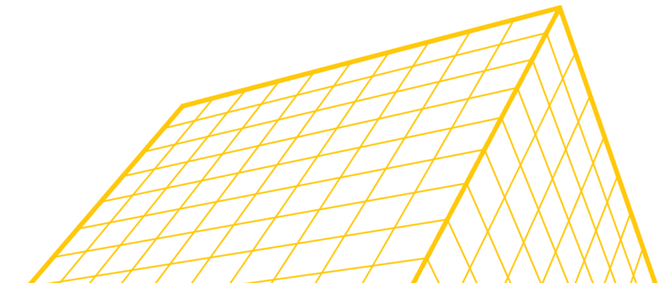
- Нет поддержки агрегации запросов
- Нет прямой поддержки Service Discovery
- Достаточно молодой проект

Сравнение YARP vs Ocelot

Возможность	Ocelot	YARP
Гибкая маршрутизация запросов	Да	Да
Поддержка авторизации, CORS	Да	Да
Ограничение скорости	Да	Да
Балансировка	Да	Да
Поддержка gRPC и потоковой переадресации	**Нет**	**Да**
Агрегация запросов	Да	Нет (ручная)
Мониторинг и трассировка	Да	Да
Динамическая конфигурация	Да	Да
Разделение конфигурации маршрутов и назначения	**Нет**	**Да**
Поддержка вендора (Microsoft)	**Нет**	**Да**







Наш выбор YARP

Microsoft YARP

(Yet Another Reverse Proxy)

С чего начать?

- Создать пустой проект ASP.NET

```
dotnet new web -n MyProxy -f net8.0
```

- Подключить пакет Yarp.ReverseProxy

```
<ItemGroup>
  <PackageReference Include="Yarp.ReverseProxy" Version="2.1.0" />
</ItemGroup>
```

- Зарегистрировать сервис и добавить в pipeline обработки запросов:

```
var builder = WebApplication.CreateBuilder(args);

builder.Services
    .AddReverseProxy()
    .LoadFromConfig(builder.Configuration.GetSection("ReverseProxy"));

var app = builder.Build();

app.MapReverseProxy();

app.Run();
```

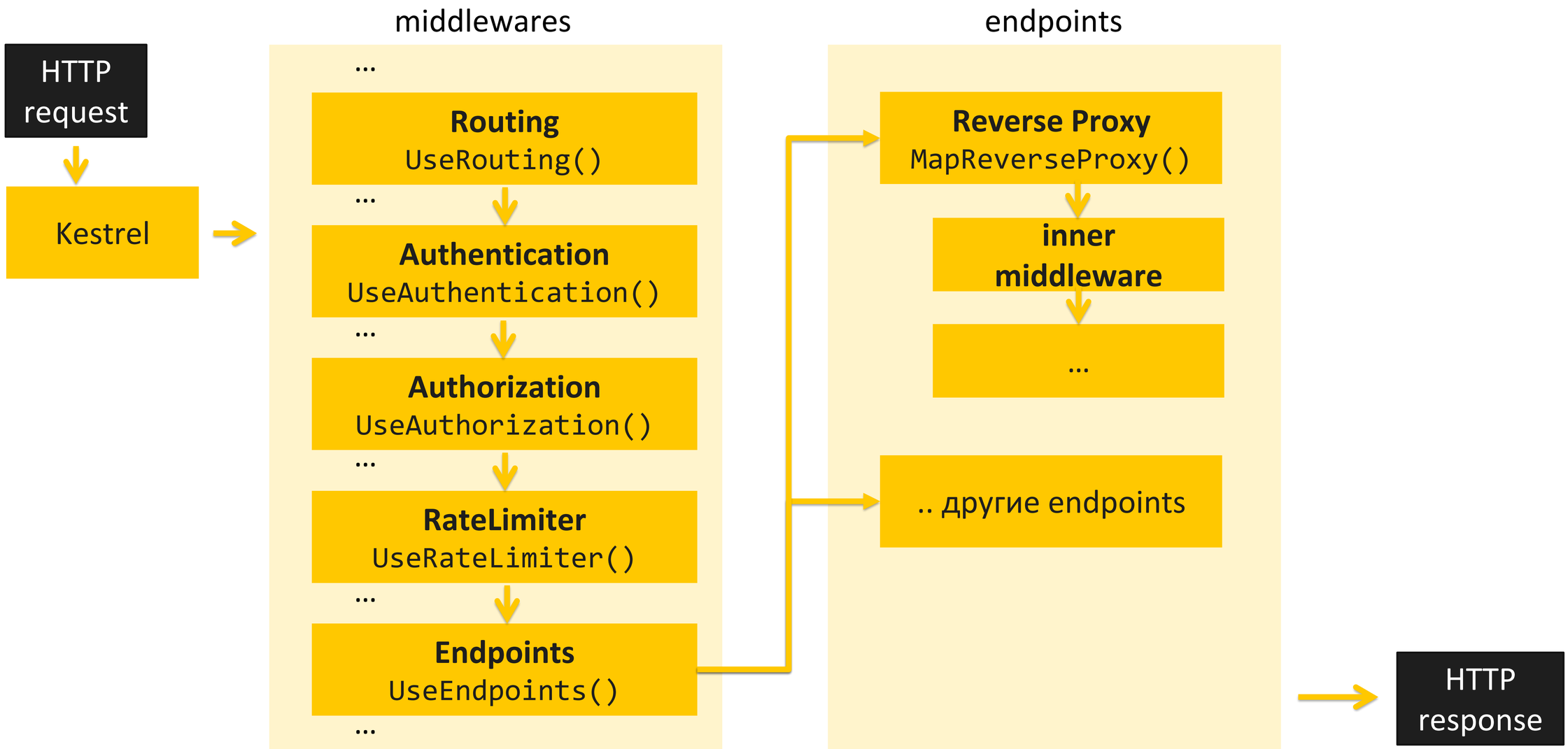
С чего начать?

Минимальная конфигурация в **appsettings.json**

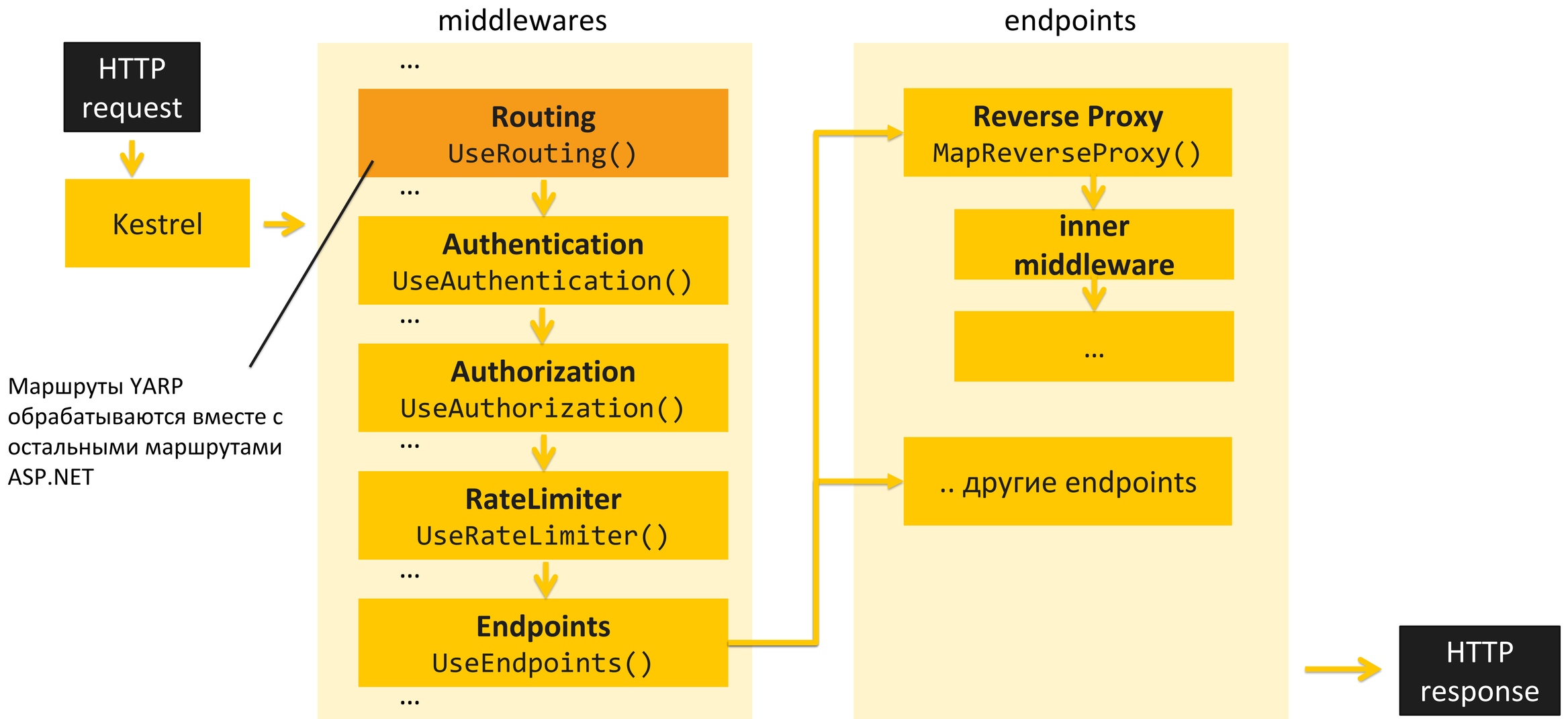
В данном примере, все пришедшие запросы будут переадресованы на сервер https://some.backend.ru/ как есть, без изменений.

```
{
  "ReverseProxy": {
    "Routes": {
      "route1": {
        "ClusterId": "cluster1",
        "Match": {
          "Path": "{**catch-all}"
        }
      }
    },
    "Clusters": {
      "cluster1": {
        "Destinations": {
          "destination1": {
            "Address": "https://some.backend.ru/"
          }
        }
      }
    }
  }
}
```

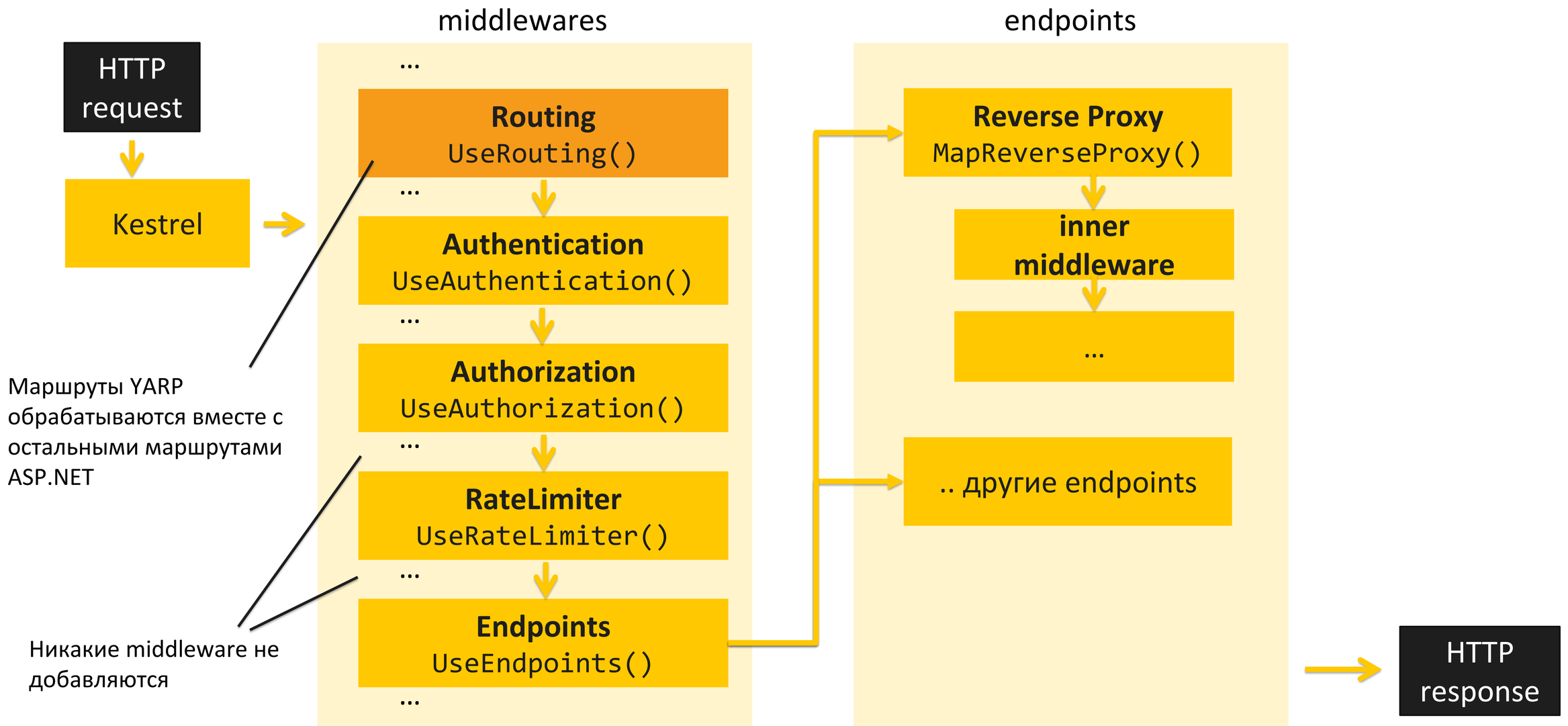
Принцип работы YARP



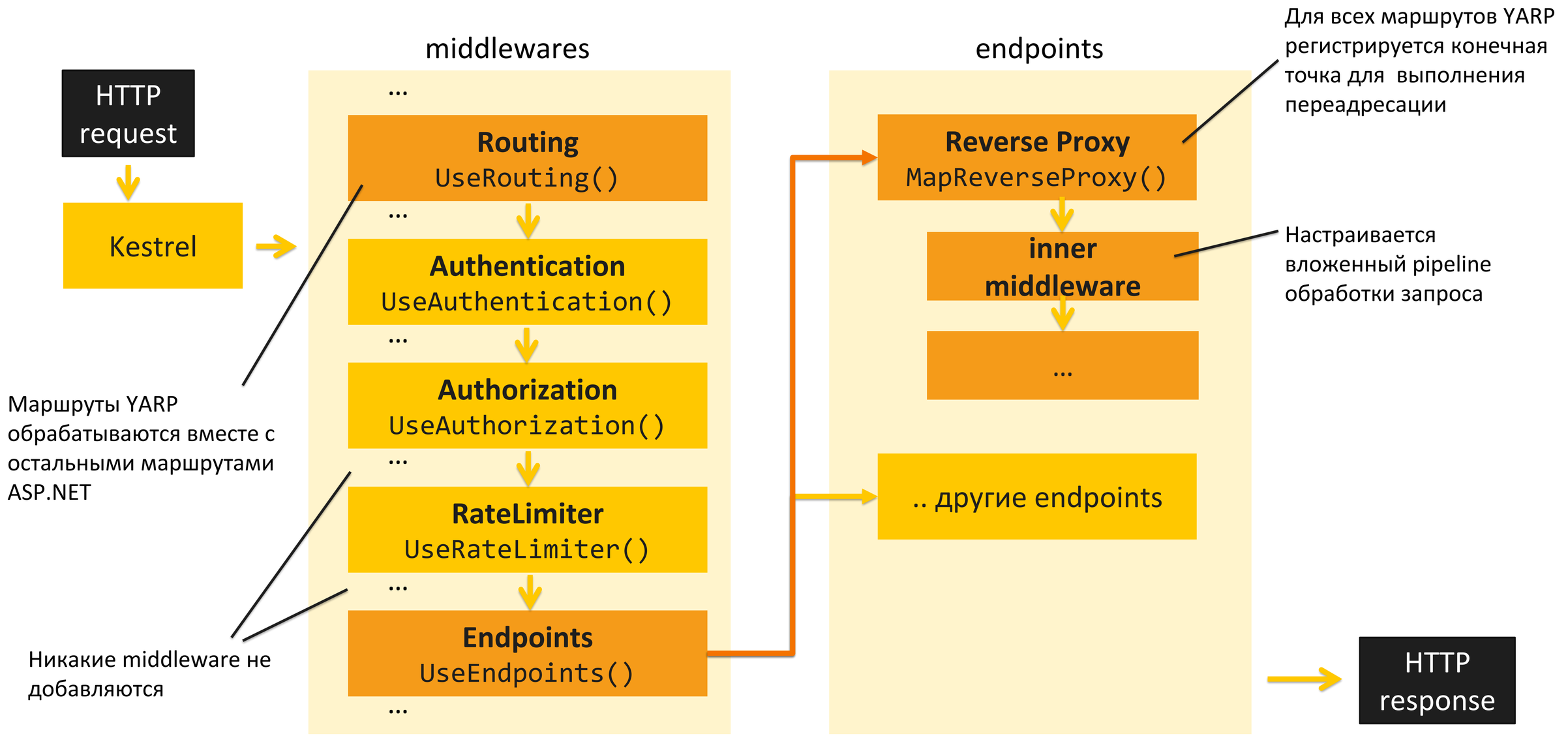
Принцип работы YARP



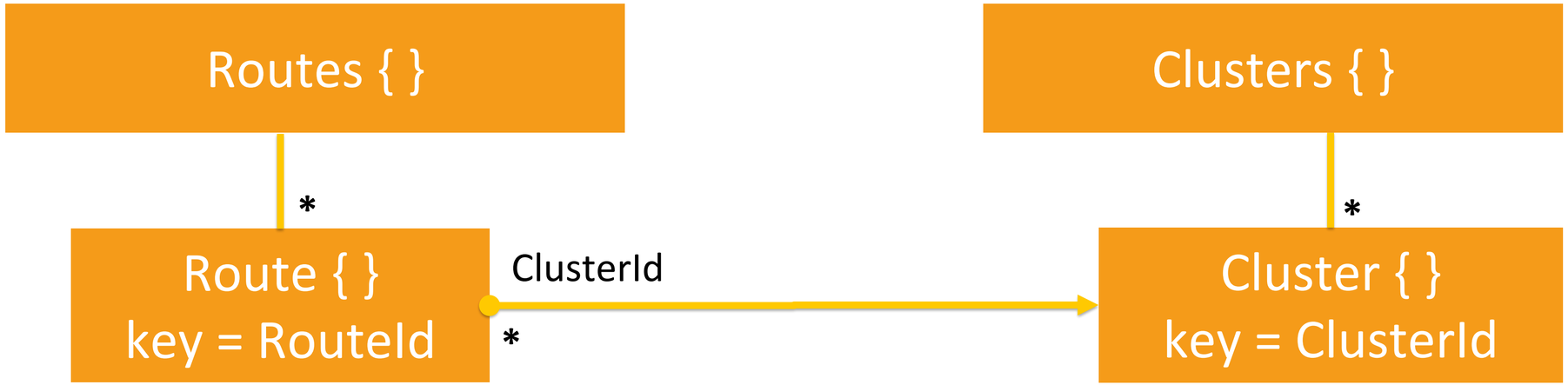
Принцип работы YARP



Принцип работы YARP



Структура конфигурации YARP



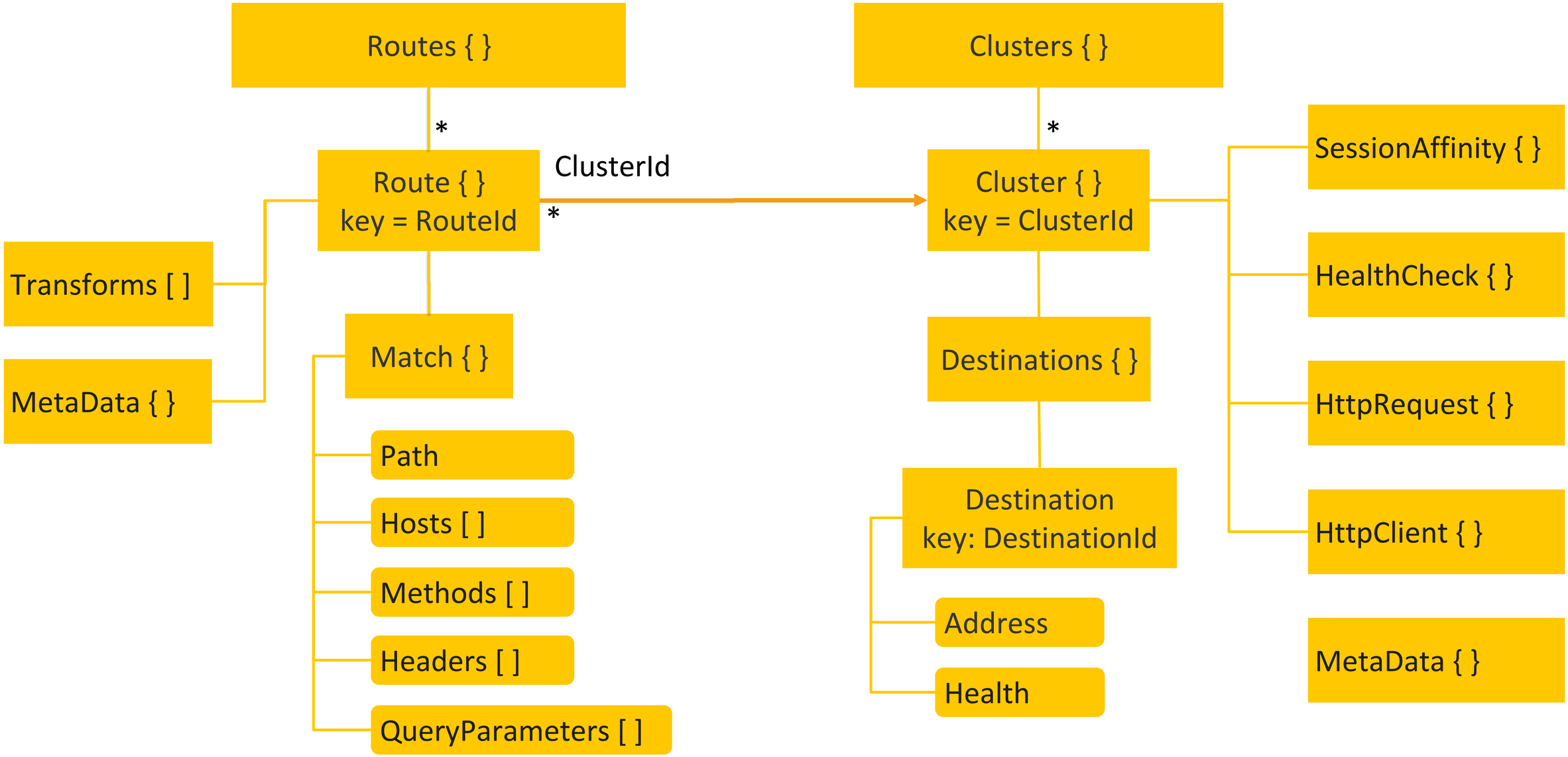
Маршруты определяют:

- правила сопоставления
- трансформации
- безопасность:
  – авторизация, CORS
  – кеширование
  – ограничение скорости
  – размер запроса

Кластера определяют:

- конечные адреса
- политики выбора адреса
  – проверка доступности
  – балансировка
  – прилипание сессий
- настройки исходящих запросов

Структура конфигурации YARP



Структура конфигурации YARP

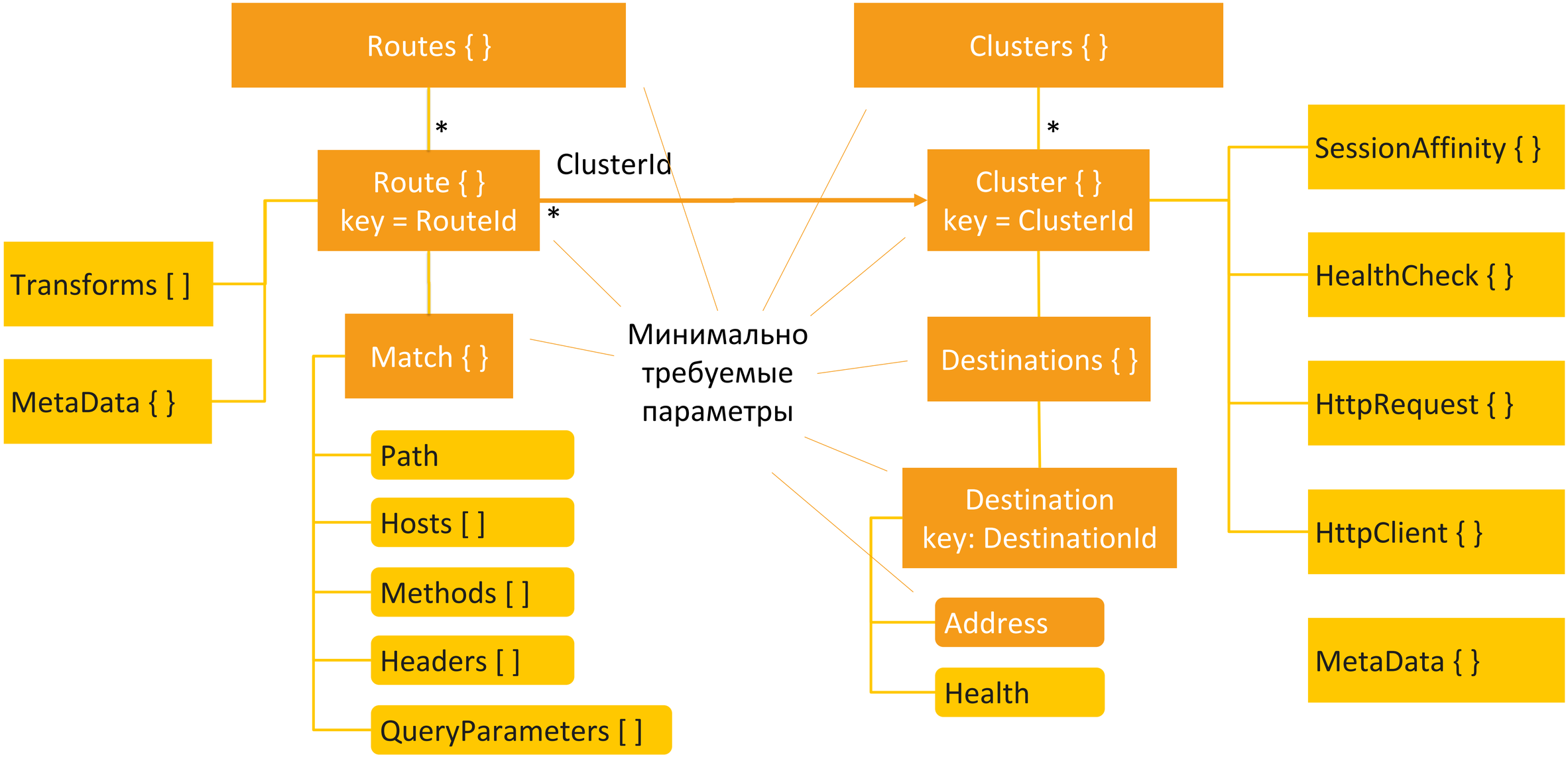
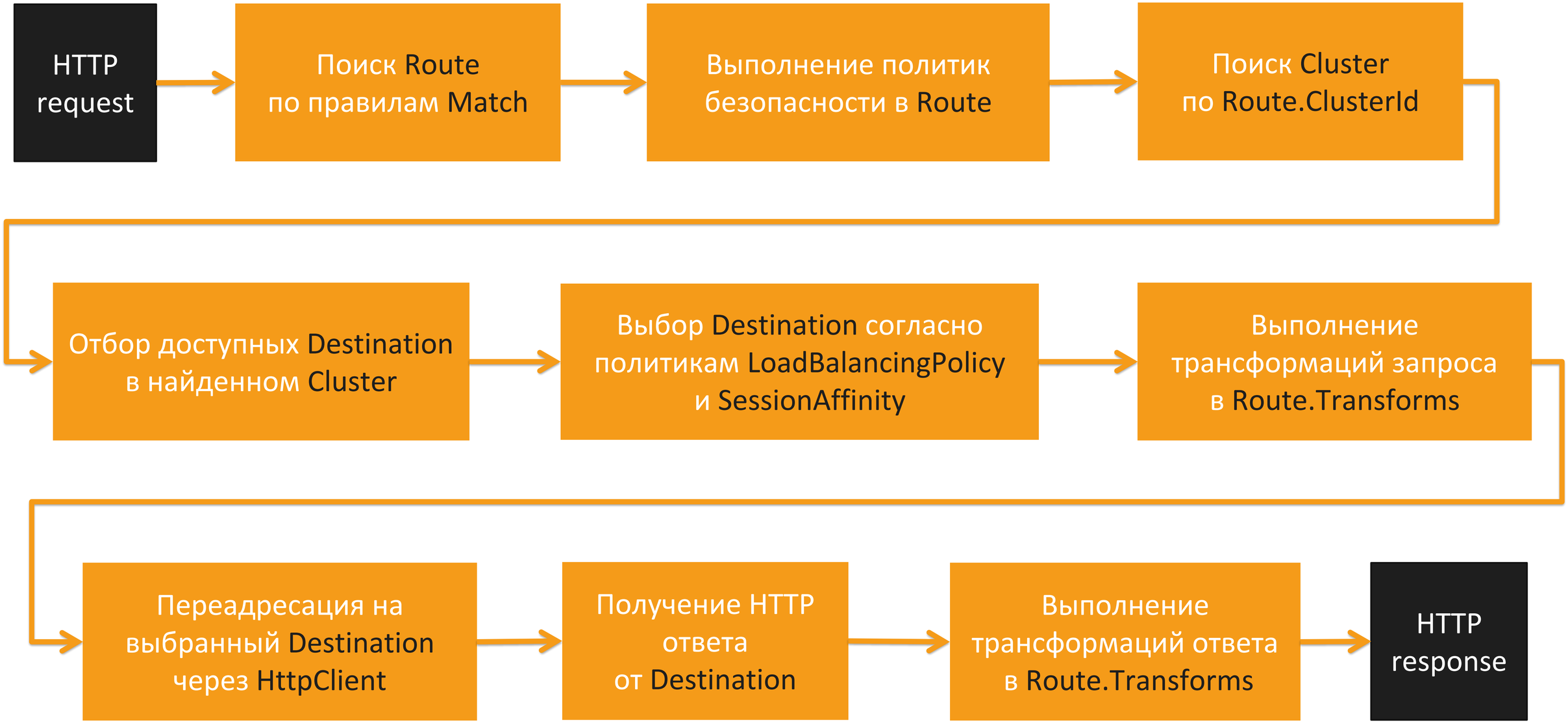


Схема переадресации запроса



Сопоставление маршрута

Параметры сопоставления в секции **Match:**

- **Path** – шаблон маршрута, используется фундаментальный механизм маршрутизации ASP.NET, см. https://learn.microsoft.com/en-US/aspnet/core/fundamentals/routing
- **Hosts** – по списку разрешённых хостов (заголовок HOST)
- **Headers** – по списку заголовков
- **QueryParameters** – по параметрам в строке запроса

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "Match": {
     "Path": "/something/{**catch-all}",
     "Hosts": [ "www.aaaaa.com", "www.bbbbb.com:8080" ],
     "Methods": [ "GET", "PUT" ],
     "Headers": [
      {
       "Name": "MyCustomHeader",
       "Values": [ "value1", "value2", "another value" ],
       // "HeaderPrefix", "Exists" , "Contains", "NotContains", "NotExists"
       "Mode": "ExactHeader",
       "IsCaseSensitive": true
      }
     ],
     "QueryParameters": [
      {
       "Name": "MyQueryParameter",
       "Values": [ "value1",  "value2",  "another value"  ],
       // "Prefix", "Exists" , "Contains", "NotContains"
       "Mode": "Exact",
       "IsCaseSensitive": true
      }
     ...
...
```

Расширение маршрутизации

Что делать, если правил не хватает?

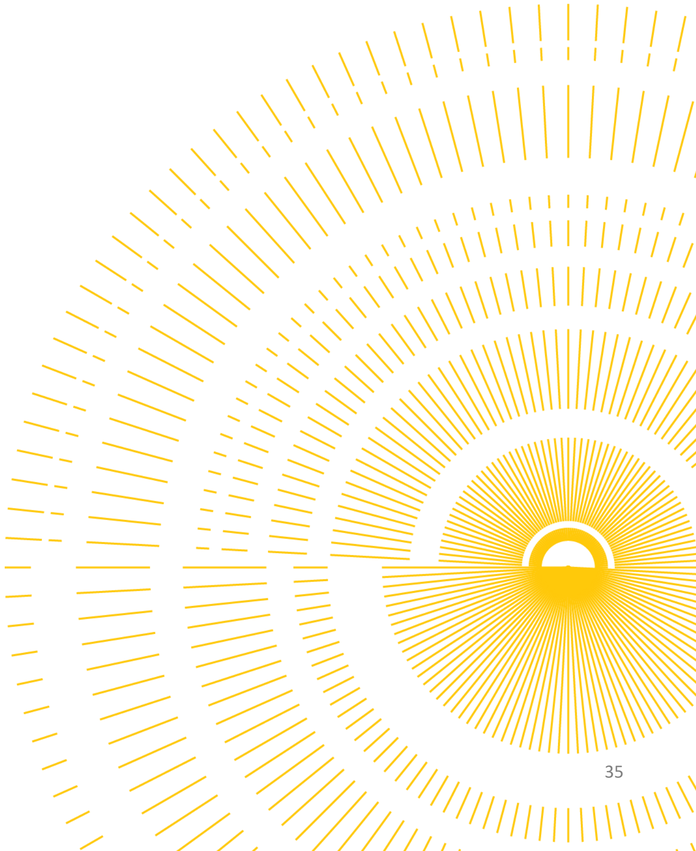
Попробуйте механизм ограничений: route constraints!

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing#route-constraints

Ограничение маршрута – это простая функция

Декларативно указывается в шаблоне через двоеточие

```
[Route("users/{id:min(1)}")]
public User GetUserById(int id)
{ … }
```



Пример: исключение маршрута

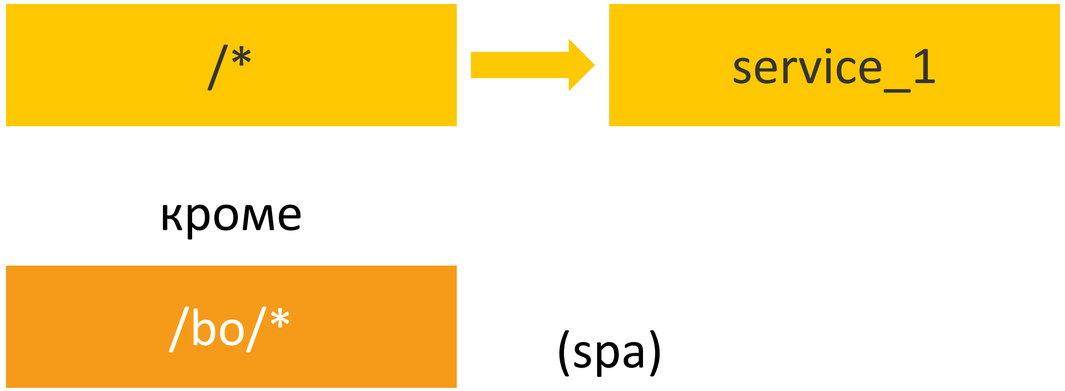
кроме

(spa)

Как сделать?

```
{
  "ReverseProxy": {
    "Routes": {
      "route_1": {
        "ClusterId": "cluster_1",
        "Match": {
          "Path": "/{**catch-all}"
        }
```

Пример: исключение маршрута



Воспользуемся ограничением маршрута:

constraint parameter

"/{**catch-all:ignore(bo/)}"

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "Match": {
     "Path": "/{**catch-all}"
    }
```

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "Match": {
     "Path": "/{**catch-all:ignore(bo/)}"
    }
```

Пример: исключение маршрута

IgnoreRouteConstraint.cs

```
internal class IgnoreRouteConstraint(string pathPrefix)
    : IRouteConstraint
{
    public bool Match(HttpContext? httpContext,
        IRouter? route,
        string routeKey,
        RouteValueDictionary values,
        RouteDirection routeDirection)
    {
        if (values.TryGetValue(routeKey, out var routeValue)
            && routeValue is string routeStringValue)
        {
            if (routeStringValue.StartsWith(pathPrefix,
                    StringComparison.OrdinalIgnoreCase)) return false;
        }

        return true;
    }
}
```

Program.cs

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.Configure<RouteOptions>(options =>
{
    options.ConstraintMap.Add("ignore", typeof(IgnoreRouteConstraint));
});
```

Имя ограничения

Тип, реализующий ограничение

Стандартные ограничения маршрута

https://learn.microsoft.com/en-us/aspnet/core/fundamentals/routing#route-constraints

```
{id:int}
{active:bool}
{dob:datetime}
{price:decimal}
{weight:double}
{weight:float}
{id:guid}
{ticks:long}
{username:minlength(4)}
```

```
{filename:maxlength(8)}
{filename:length(12)}
{filename:length(8,16)}
{age:min(18)}
{age:max(120)}
{age:range(18,120)}
{name:alpha}
{ssn:regex(^\\d{{3}}-\\d{{2}}-\\d{{4}}$)}
{name:required}
```

Можно комбинировать => "users/{id:int:min(1)}"

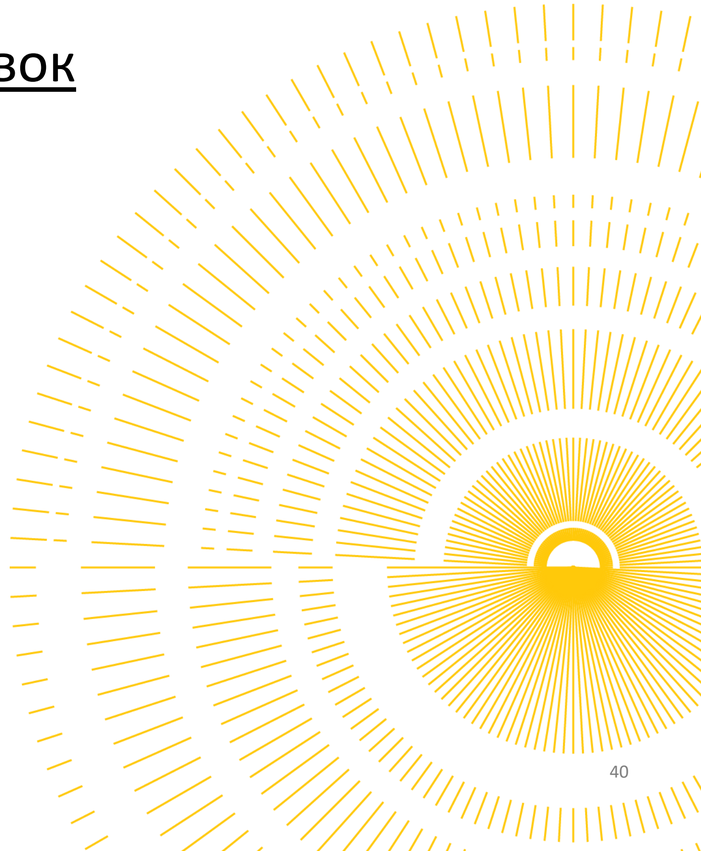
Расширение маршрутизации

А если правило с очень сложной программной логикой?

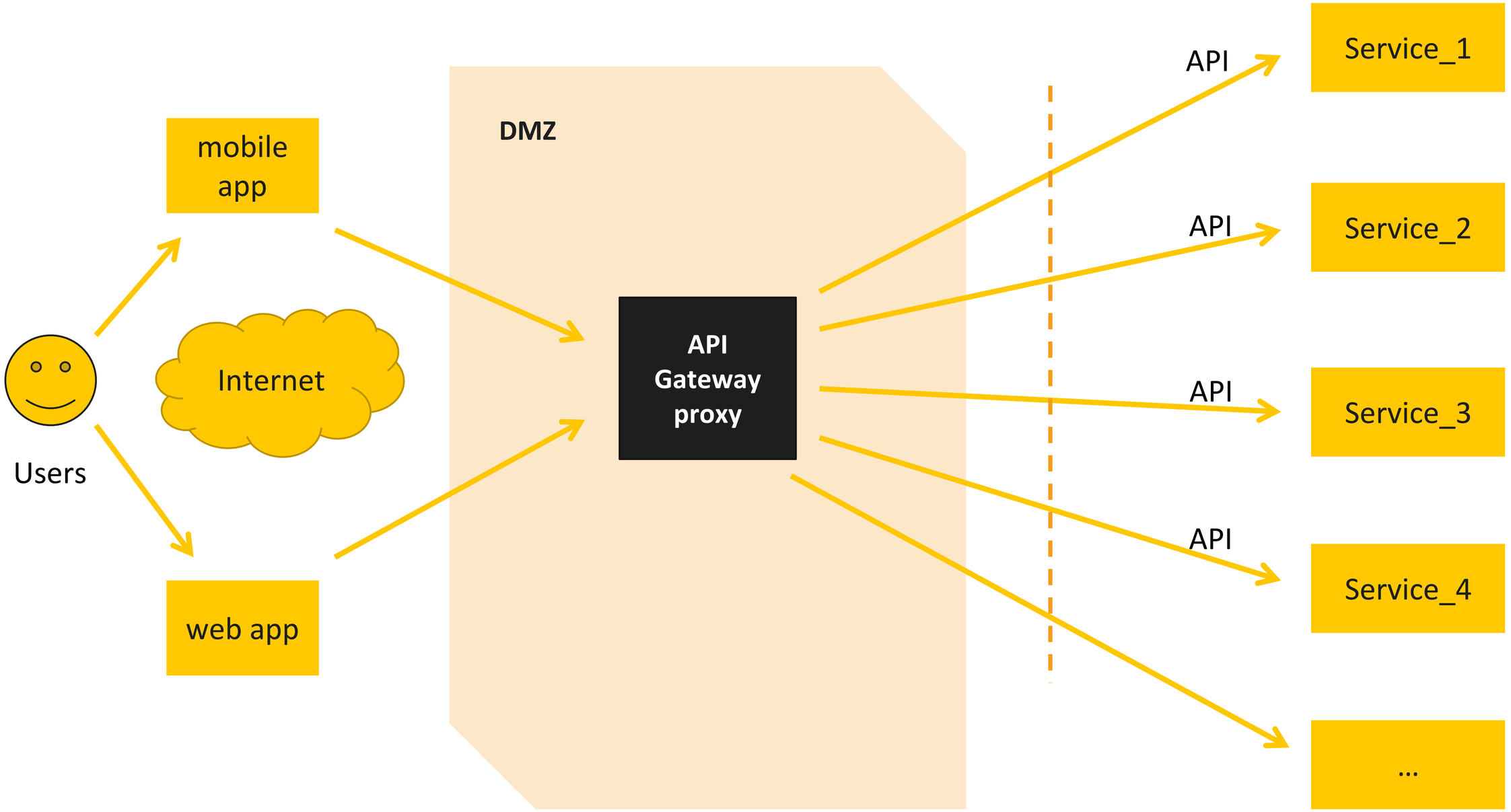
Можно воспользоваться трюком:

- Реализуем собственный **middleware**, который регистрируется ДО вызова **UseRouting()**
- Выполняется необходимая логика
- К списку заголовков запроса добавляется специальный заголовок
- В правилах маршрутизации используется сопоставление по заголовку

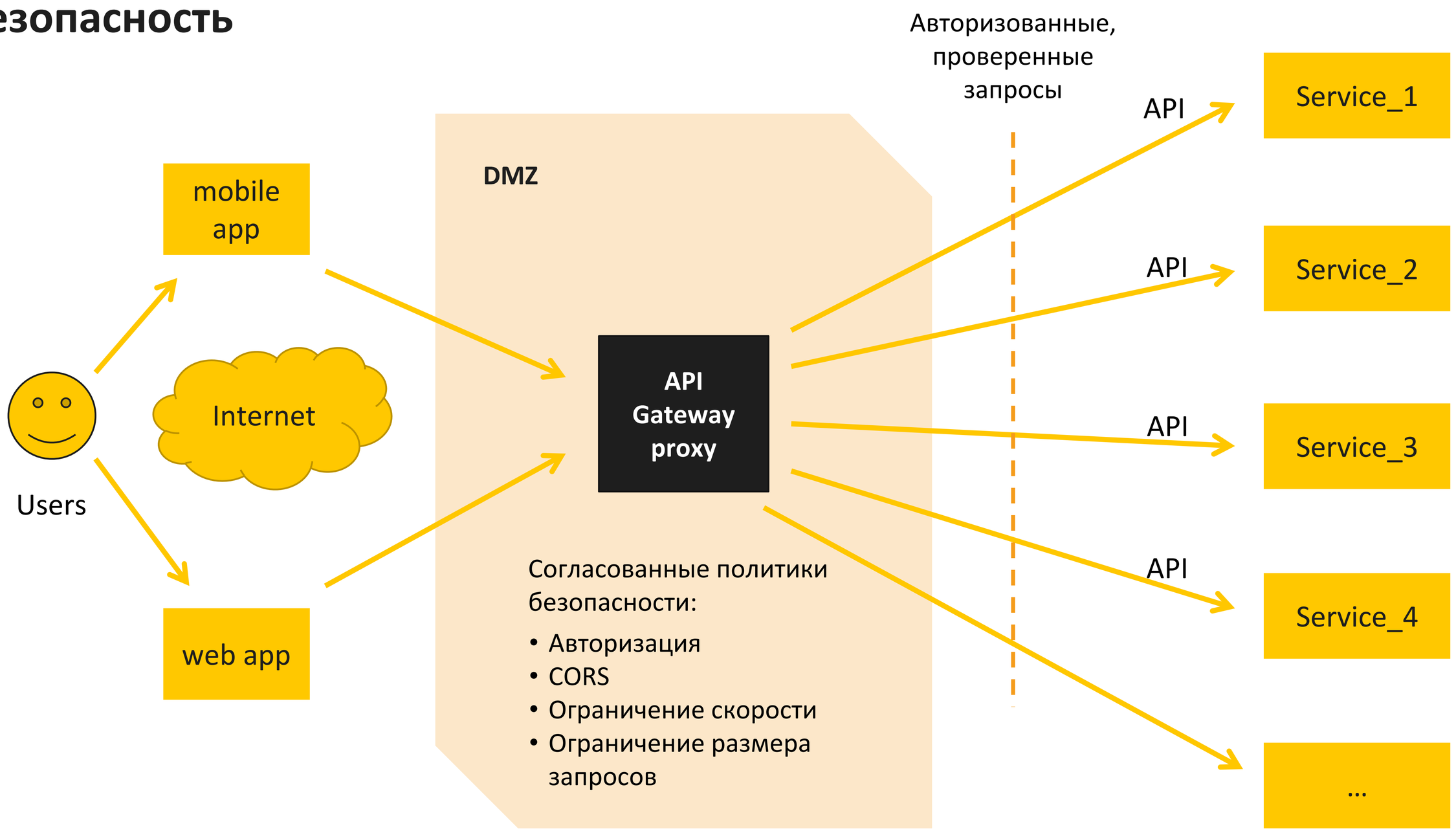
P.S. Сложных правил маршрутизации лучше избегать ☺



Безопасность



Безопасность



Безопасность

- Используются интегрированные механизмы ASP.NET
- Задаются в каждом маршруте:
  - **Политика авторизации**
  - Политика CORS
  - Политика Rate Limiter
  - Ограничение размера

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "AuthorizationPolicy": "AuthPolicy_1",
    "CorsPolicy": "CorsPolicy_1",
    "RateLimiterPolicy": "RatePolicy_1",
    "MaxRequestBodySize": 1000000,
    "Match": {
     "Path": "/api/{**catch-all}"
    }
   }
  },
```

- Сами политики настраиваются программным образом

Безопасность

- Используются интегрированные механизмы ASP.NET
- Задаются в каждом маршруте:
  - Политика авторизации
  - **Политика CORS**
  - Политика Rate Limiter
  - Ограничение размера

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "AuthorizationPolicy": "AuthPolicy_1",
    "CorsPolicy": "CorsPolicy_1",
    "RateLimiterPolicy": "RatePolicy_1",
    "MaxRequestBodySize": 1000000,
    "Match": {
     "Path": "/api/{**catch-all}"
    }
   }
  },
```

- Сами политики настраиваются программным образом

Безопасность

- Используются интегрированные механизмы ASP.NET
- Задаются в каждом маршруте:
  - Политика авторизации
  - Политика CORS
  - **Политика Rate Limiter**
  - Ограничение размера

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "AuthorizationPolicy": "AuthPolicy_1",
    "CorsPolicy": "CorsPolicy_1",
    "RateLimiterPolicy": "RatePolicy_1",
    "MaxRequestBodySize": 1000000,
    "Match": {
     "Path": "/api/{**catch-all}"
    }
   }
  },
```

- Сами политики настраиваются программным образом

Безопасность

- Используются интегрированные механизмы ASP.NET
- Задаются в каждом маршруте:
  - Политика авторизации
  - Политика CORS
  - Политика Rate Limiter
  - **Ограничение размера**

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "AuthorizationPolicy": "AuthPolicy_1",
    "CorsPolicy": "CorsPolicy_1",
    "RateLimiterPolicy": "RatePolicy_1",
    "MaxRequestBodySize": 1000000,
    "Match": {
     "Path": "/api/{**catch-all}"
    }
   }
  },
```

- Сами политики настраиваются программным образом

Аутентификация и авторизация

Настройка схемы аутентификации (например, JwtBearer)

```
builder.Services
   .AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
   .AddJwtBearer(options =>
   {
      options.TokenValidationParameters =
         new TokenValidationParameters
         {
            ValidateIssuer = true,
            ValidIssuer = "http://sso.company.ru/",
            ValidateLifetime = true,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey = new JsonWebKey("...json web key...")
         };
   });
```

* рекомендуется использовать IssuerSigningKeyResolver и JWKS сервер

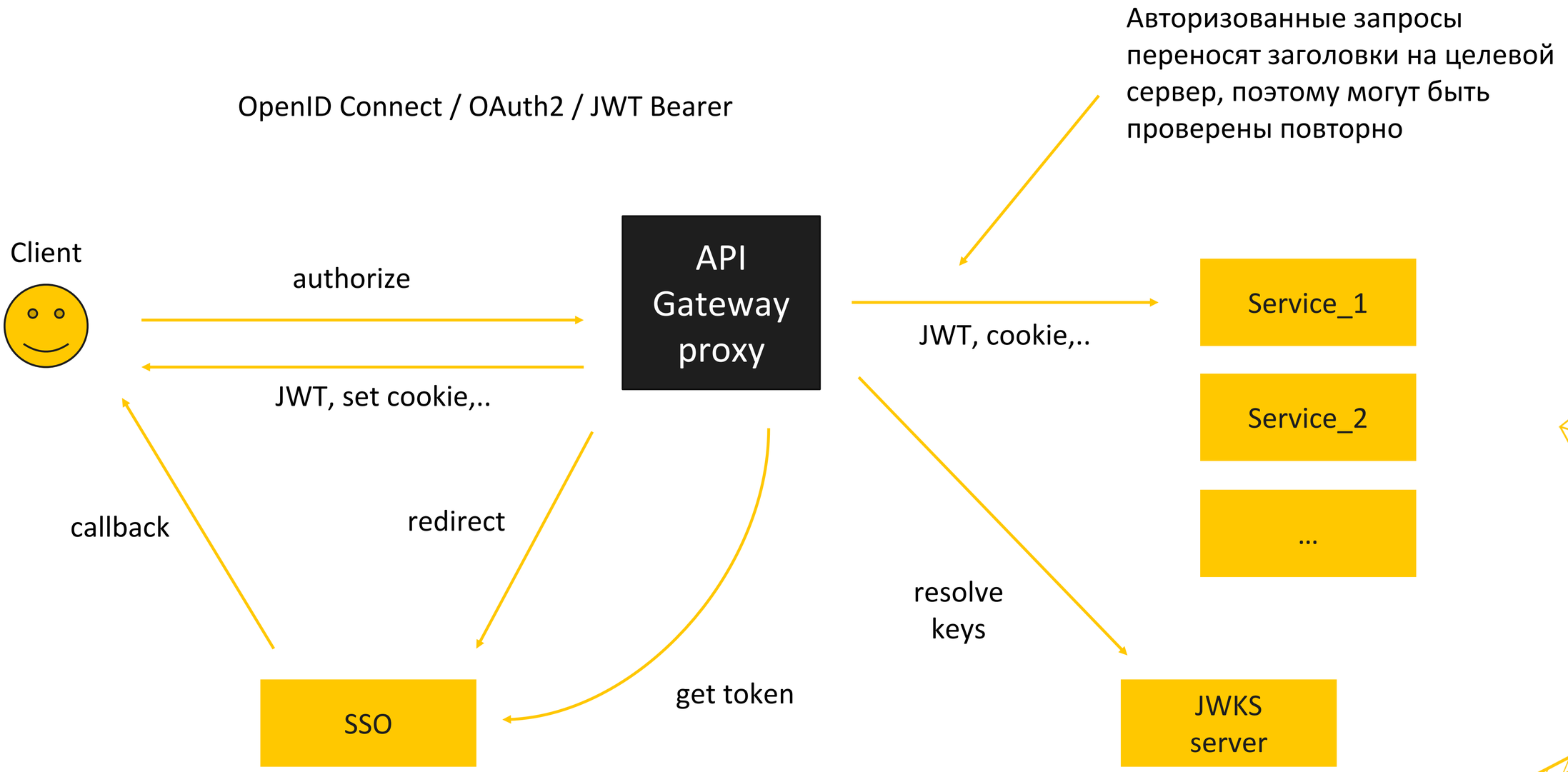
Настройка политик авторизации

```
builder.Services
   .AddAuthorization(options => options
      .AddPolicy("AuthPolicy_1",
         policy => policy
            .AddAuthenticationSchemes(
                  JwtBearerDefaults.AuthenticationScheme)
            .RequireAuthenticatedUser()
            .RequireRole("client_api")));
```

Обязательно включаем в pipeline

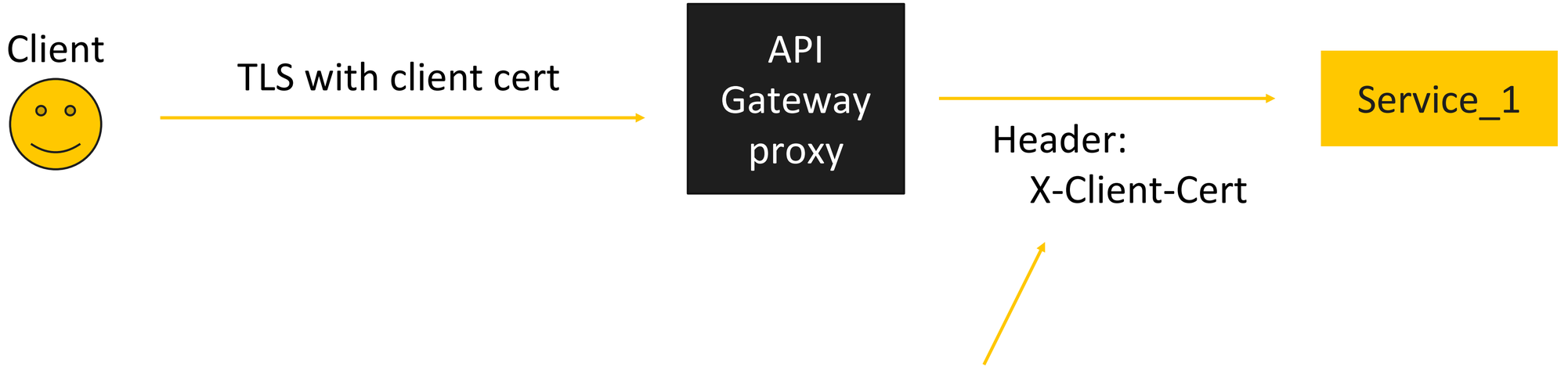
```
app.UseAuthentication();
app.UseAuthorization();

app.MapReverseProxy();
```

Схемы авторизации



Схемы авторизации

Client Certificates



Документация ASP.NET:
https://learn.microsoft.com/en-us/aspnet/core/security

Cors

Cross-Origin Requests

Настройка политик CORS

```
builder.Services
    .AddCors(options =>
    {
        options
            .AddPolicy("CorsPolicy_1", policy =>
                policy.WithOrigins(
                    "cdn-1.company.ru", "cdn-2.company.ru"));
    });
```

Включаем в pipeline, а то не заработает

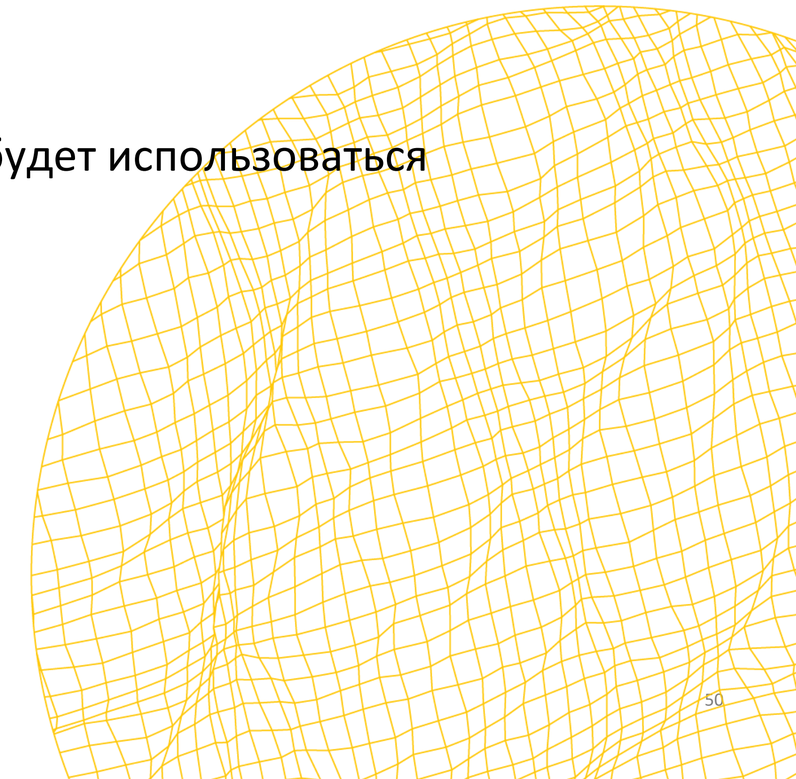
```
…
app.UseCors();
…
app.MapReverseProxy();
```

Использование политики по умолчанию

```
builder.Services
    .AddCors(options =>
    {
        …
        options.DefaultPolicyName = "CorsPolicy_1";
    });
```

Теперь эта политика будет использоваться для всех запросов

Подробней про CORS:
https://learn.microsoft.com/en-us/aspnet/core/security/cors



Ограничение скорости (троттлинг)

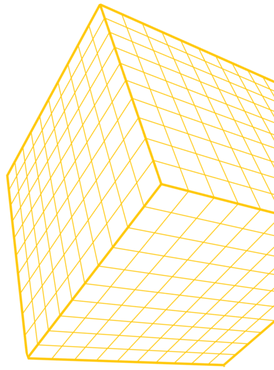
Настройка политик ограничения скорости

```
builder.Services.AddRateLimiter(options =>
{
    options.AddFixedWindowLimiter("RatePolicy_1", opt =>
    {
        opt.PermitLimit = 4;
        opt.Window = TimeSpan.FromSeconds(12);
        opt.QueueProcessingOrder =
            QueueProcessingOrder.OldestFirst;
        opt.QueueLimit = 2;
    });
});
```

Включаем в pipeline, а то не заработает

```
app.UseRateLimiter();
…
app.MapReverseProxy();
```

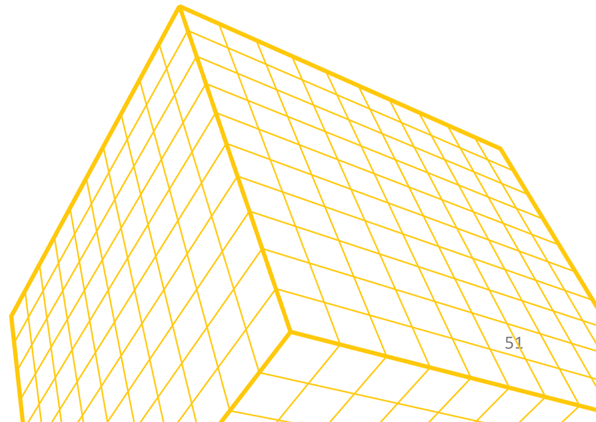
* Для установки ограничения на все запросы нужно задать значение **options.GlobalLimiter**



Избегайте глобальных ограничений!

Подробная документация:
https://learn.microsoft.com/en-us/aspnet/core/performance/rate-limit

Рекомендую выступление Евгения Пешкова «Алгоритмы троттлинга запросов»



Трансформация маршрутов

Возможность декларативно изменять запрос, который переадресуется, и ответ, возвращаемый от бекенда:

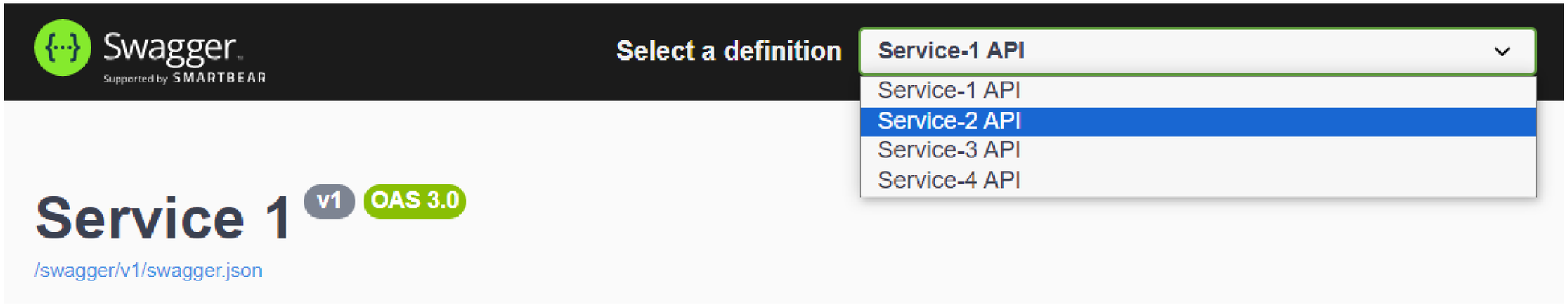
- путь запроса
- состав заголовков
- параметры строки запроса
- значение заголовка HOST
- тело запроса и ответа
- формирование специальных заголовков (сертификат клиента, X-Forwarded-* и др.)

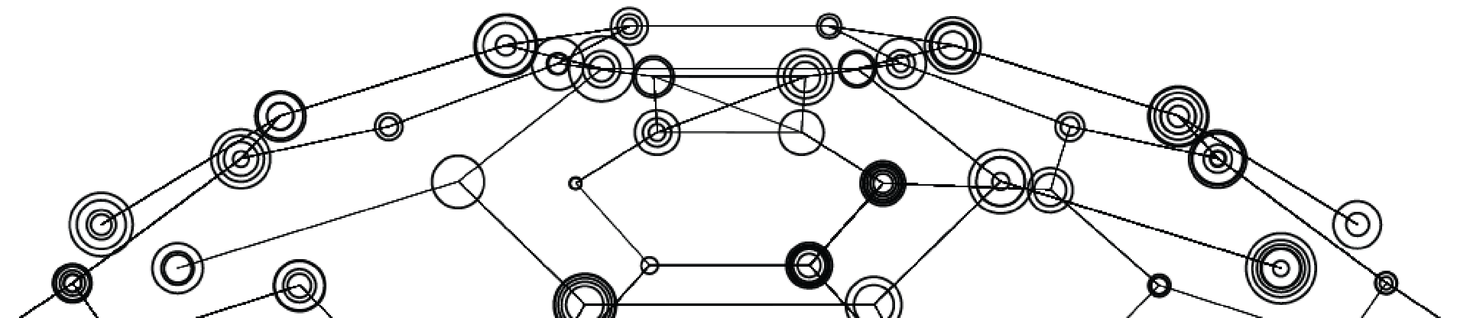
Набор доступных трансформаций можно расширять программным образом.

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "Match": {
     "Hosts": [ "my.company.ru" ]
    },
    "Transforms": [
     { "PathPrefix": "/api" },
     {
      "RequestHeader": "header_1",
      "Append": "bar",
      "When": "Always"
     },
     { "ClientCert": "X-Client-Cert" },
     { "RequestHeadersCopy": "true" },
     { "RequestHeaderOriginalHost": "true" }
    ]
   },
```

Пример: изменение путей в Swagger

- Есть желание объединить все Swagger на одну страницу
- При переадресации, пути для каждого сервиса меняются:
  /api/**svc-1**/{**catch-all} => /api/{**catch-all}
- Простое проксирование **swagger.json** работать не будет
- Нужно добиться, чтоб работало вот так:





Пример: изменение путей в Swagger

Добавляем ссылки на swagger-документы каждого сервиса

```
app.UseSwaggerUI(c =>
{
   c.SwaggerEndpoint("/swagger/svc-1/swagger.json", "Service-1 API");
   c.SwaggerEndpoint("/swagger/svc-2/swagger.json", "Service-2 API");
   c.SwaggerEndpoint("/swagger/svc-3/swagger.json", "Service-3 API");
   c.SwaggerEndpoint("/swagger/svc-4/swagger.json", "Service-4 API");
});
```

Регистрируем два маршрута для сервиса:
1. доступ к API
2. доступ к документу swagger

Будем делать собственную трансформацию

```
{
 "ReverseProxy": {
  "Routes": {
   "route_1": {
    "ClusterId": "cluster_1",
    "Match": {
     "Path": "/api/svc-1/{**catch-all}"
    },
    "Transforms": [
     { "PathPattern": "/api/{**catch-all}" }
    ]
   },
   "route_1_swagger": {
    "ClusterId": "cluster_1",
    "Match": {
     "Path": "/swagger/svc-1/swagger.json"
    },
    "Transforms": [
     { "PathSet": "/swagger/v1/swagger.json" },
     { "Swagger": "svc-1" }
    ]
   }
  },
```

Пример: изменение путей в Swagger

```
internal class SwaggerTransformFactory : ITransformFactory
{
    public bool Validate(
        TransformRouteValidationContext context,
        IReadOnlyDictionary<string, string> transformValues)
    {
        if (!transformValues.TryGetValue("Swagger", out var value)) return false;
        if (string.IsNullOrEmpty(value))
            context.Errors.Add(
                new ArgumentException("A non-empty Swagger transform value is required"));
        return true;
    }

    public bool Build(
        TransformBuilderContext context,
        IReadOnlyDictionary<string, string> transformValues)
    {
        if (!transformValues.TryGetValue("Swagger", out var value)) return false;
        if (string.IsNullOrEmpty(value))
            throw new ArgumentException("A non-empty Swagger value is required");
        context.AddResponseTransform(async ctx
            => await TransformSwaggerLinks(ctx, value));
        return true;
    }
```

Создаём новую фабрику трансформаций

Трансформация добавляется здесь

Наша реализация

Пример: изменение путей в Swagger

Реализация трансформации тела ответа:

```
private static async ValueTask TransformSwaggerLinks(
    ResponseTransformContext context,
    string value)
{
    if (context.ProxyResponse is null) return;
    var ct = context.HttpContext.RequestAborted;
    context.SuppressResponseBody = true;
    var body = await context.ProxyResponse.Content
        .ReadAsStringAsync(ct);

    // нужно изменить ссылки внутри документа на проксируемые
    var newBody = body.Replace("/api/", $"/api/{value}/");
    var length = Encoding.UTF8.GetByteCount(newBody);

    context.HttpContext.Response.Headers.ContentLength = length;
    await context.HttpContext.Response
        .WriteAsync(newBody, Encoding.UTF8, ct);
}
```

Регистрация трансформации:

```
builder.Services
    .AddReverseProxy()
    .LoadFromConfig(builder.Configuration
            .GetSection("ReverseProxy"))
    .AddTransformFactory<SwaggerTransformFactory>();
```

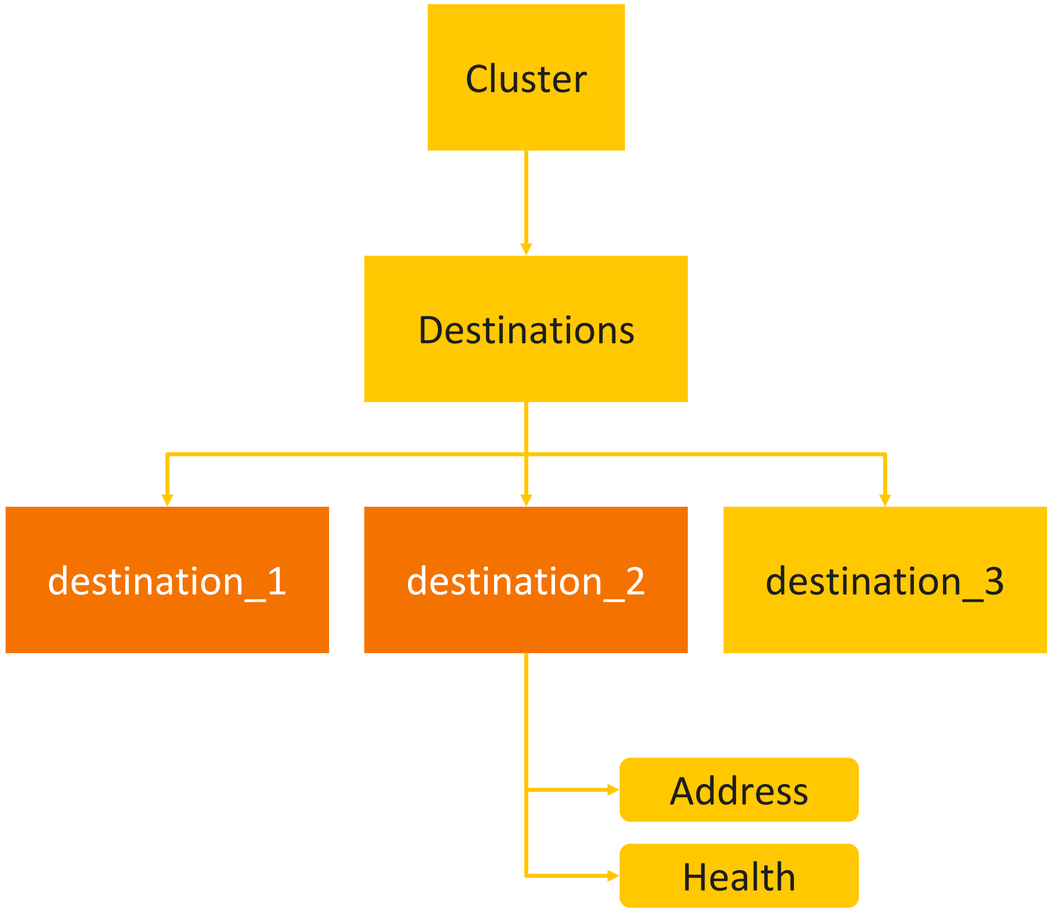
Подавляем вывод исходного ответа

Вычитываем ответ для обработки

Обязательно! Перезаписываем новый размер содержимого

Записываем содержимое изменённого ответа

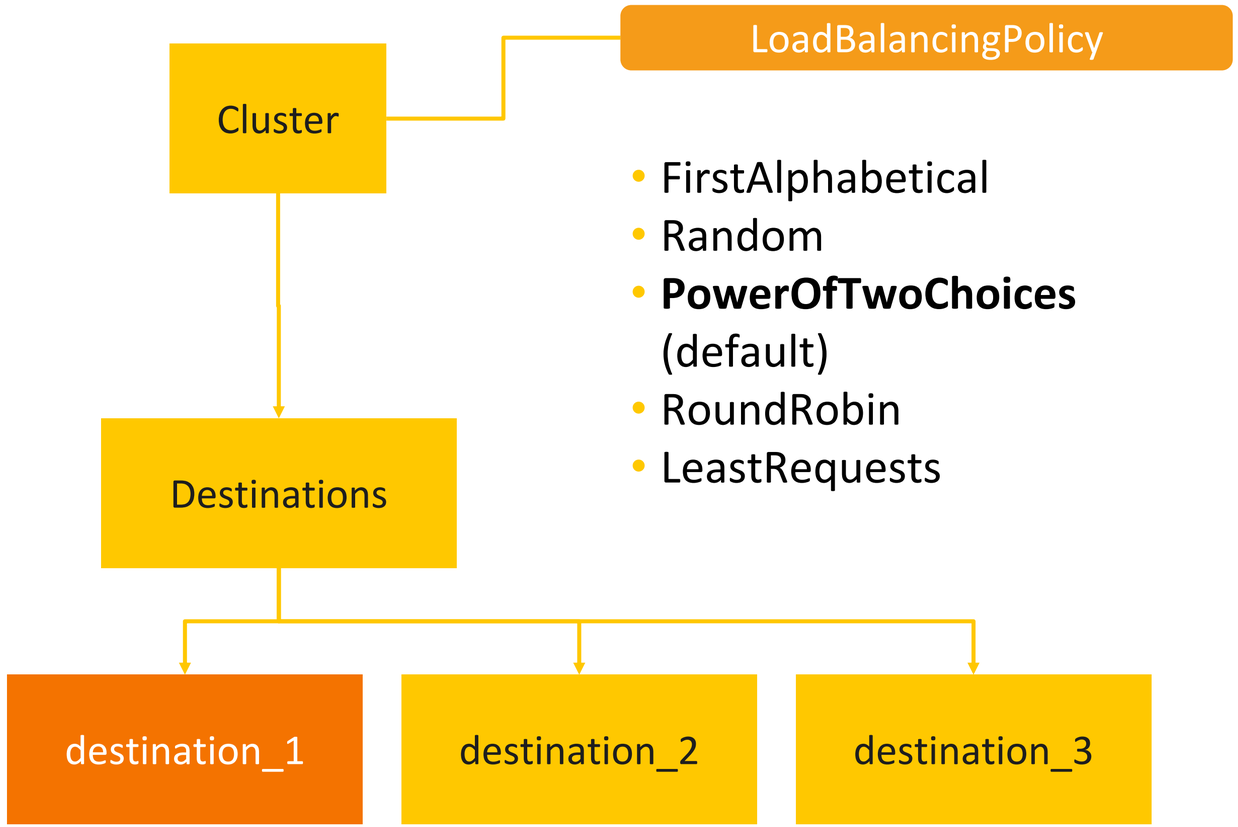
Проверка доступности (health check)



- Два режима проверки доступности:
  - Активный
  - Пассивный
- Из коробки доступны политики:
  - "ConsecutiveFailures" – активная
  - "TransportFailureRate" – пассивнаяя
- В активном режиме проверка осуществляется периодически
- В пассивном ведётся учёт успехов и неудач при переадресации запросов на указанный адрес

```
app.MapReverseProxy(p =>
{
    p.UseLoadBalancing();
    p.UsePassiveHealthChecks();
});
```

Проверка доступности (health check)

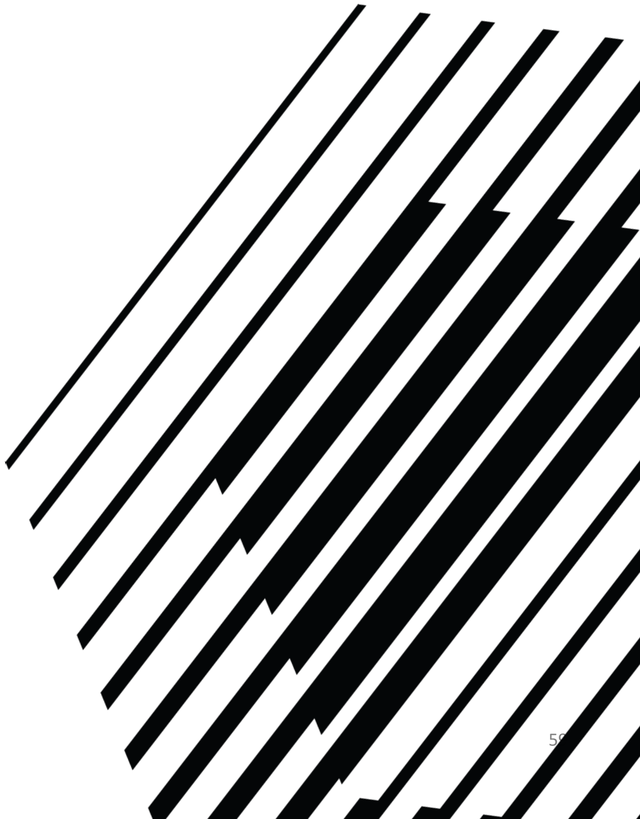


- FirstAlphabetical
- Random
- **PowerOfTwoChoices** (default)
- RoundRobin
- LeastRequests

```
{
 "ReverseProxy": {
  "Clusters": {
   "cluster_1": {
    "LoadBalancingPolicy": "RoundRobin",
    "Destinations": {
     "cluster_1/destination_1": {
      "Address": "https://serv-1.host.ru:10000/"
     },
     "cluster_1/destination_2": {
      "Address": "https://serv-2.host.ru:10010/"
     },
     "cluster_1/destination_3": {
      "Address": "https://serv-3.host.ru:10000/"
     }
    }
   }
  }
 }
}
```

Известные подводные камни

- Строгое следование HTTP-стандартам
- Ошибки 502
  - Кодировка UTF-8 в заголовках (Cookie)
  - Не указано ограничение на размер содержимого
  - Прерывание соединения по инициативе клиента
- Не буферизируются запросы по умолчанию
- Размещение в docker-контейнерах
  - Монтирование эфемерного тома для временных файлов в /tmp
  - Настройка культуры
- Ошибки 504 и управление таймаутами
- Отдельный порт для вывода методов проверки доступности, сбора метрик (Prometheus)



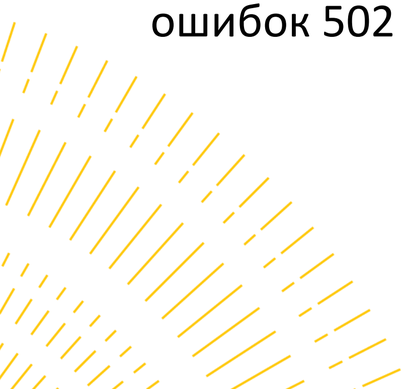
Настройка HttpClient

Ограничить версии используемых TLS

Отключить проверку сертификата при установке HTTPS соединения (опасно! не делайте так!)

Ограничить максимальное количество соединений

Разрешить UTF-8 в заголовках запросов и ответов (обычно, это Cookie) для решения проблемы ошибок 502



```
{
  "ReverseProxy": {
    "Clusters": {
      "cluster_1": {
        "Destinations": {
          "dest_1": {
            "Address": "https://serv1.host.ru"
          },
          "dest_2": {
            "Address": "https://serv2.host.ru"
          }
        },
        "HttpClient": {
          "SSLProtocols": "Tls12,Tls13",
          "DangerousAcceptAnyServerCertificate": true,
          "MaxConnectionsPerServer": 1024,
          "EnableMultipleHttp2Connections": true,
          "RequestHeaderEncoding": "utf-8",
          "ResponseHeaderEncoding": "utf-8"
        }
      }
    }
  }
}
```

Настройка HttpRequest

Настроить таймаут, на ожидание ответа или проявление активности

Задать версию протокола HTTP

Политика выбора версии протокола

Разрешить буфферизацию (не стоит!)

```
{
  "ReverseProxy": {
    "Clusters": {
      "cluster_1": {
        "Destinations": {
          "dest_1": {
            "Address": "https://serv1.host.ru"
          },
          "dest_2": {
            "Address": "https://serv2.host.ru"
          }
        },
        "HttpRequest": {
          "ActivityTimeout": "00:00:30",
          "Version": "2",
          "VersionPolicy": "RequestVersionOrLower",
          "AllowResponseBuffering": "false"
        }
      }
    }
  }
}
```

Размещение в контейнерах

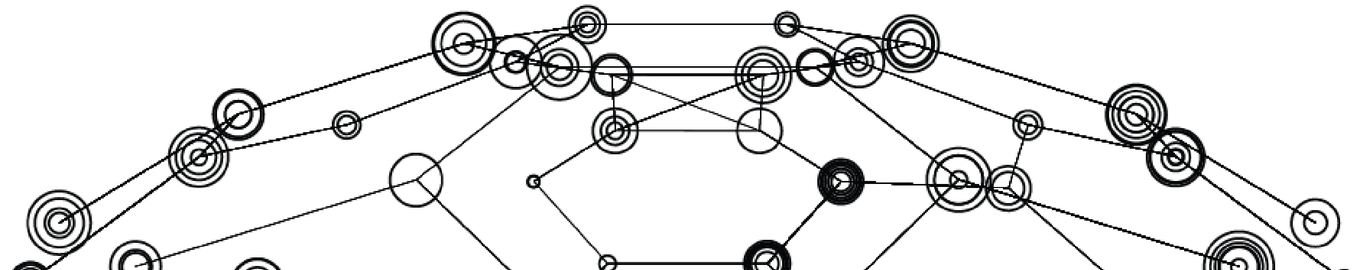
Выведите отдельный порт
для собственного API

dockerfile

```
FROM local-proxyhub/mcr/dotnet/aspnet:8.0 AS base
ENV ASPNETCORE_URLS=http://+:8080,https://+:8081,http://+:8090
EXPOSE 8080
EXPOSE 8081
EXPOSE 8090
WORKDIR /app
COPY . .
ENTRYPOINT ["dotnet", "MyApp.dll"]
```

Для собственных API, используйте
привязку к отдельному хосту и порту

```
app.MapHealthChecks("/healtz")
   .RequireHost("localhost:8090");
```

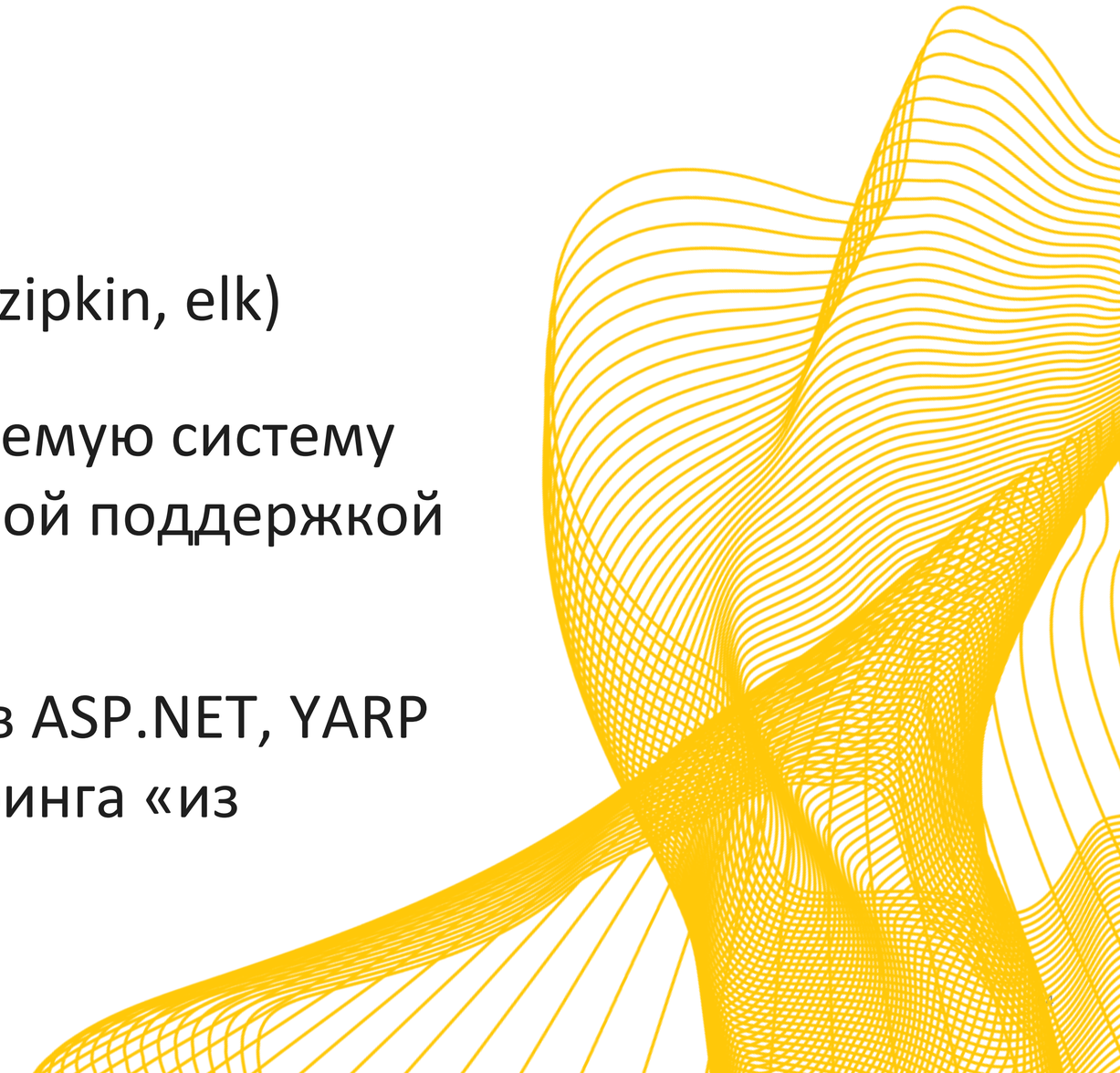


Наблюдаемость

- Журнал (logs)
- Метрики (prometheus, counters)
- Распределённая трассировка (jaeger, zipkin, elk)

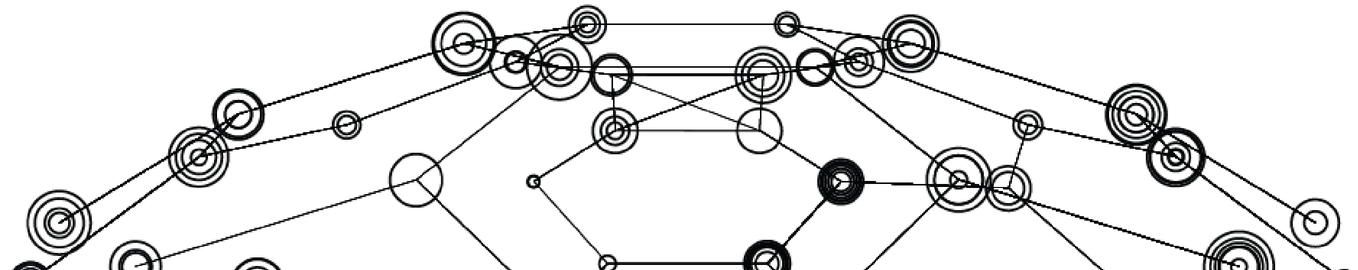
Наблюдаемость

- Журнал (logs)
- Метрики (prometheus, counters)
- Распределённая трассировка (jaeger, zipkin, elk)
- ASP.NET имеет встроенную настраиваемую систему трассировки и наблюдаемости с полной поддержкой Open Telemetry
- Будучи встраиваемым компонентом в ASP.NET, YARP поддерживает все сценарии мониторинга «из коробки»



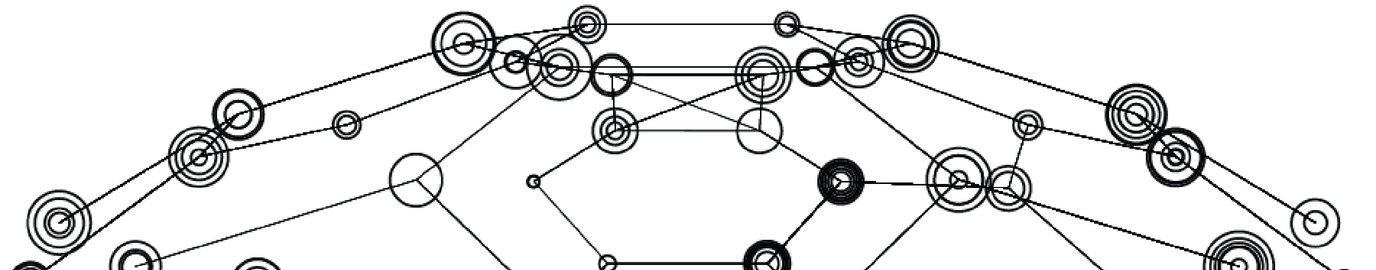
Журналирование

- Используем сбор логов из **stdout** в формате JSON
- Логи размещаются в ELK
- Ранее мы уже разработали компоненты ведения журнала на базе встроенной системы **IloggerFactory** и **Serilog**
- Это подошло для всех сценариев использования YARP
- Если вы используете для приложений ASP.NET уже настроенное журналирование, это подойдёт и для YARP
- Документация: https://learn.microsoft.com/en-us/aspnet/core/fundamentals/logging



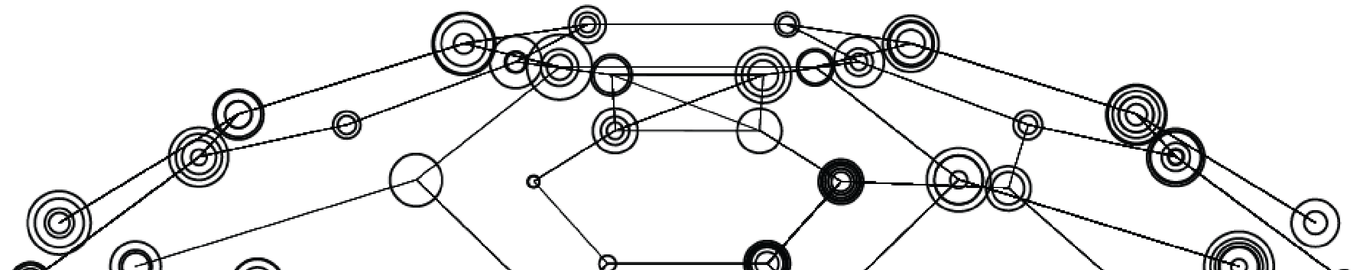
Метрики

- В dotnet на уровне платформы реализовано несколько механизмов для сбора счётчиков производительности
- Нас интересуют счётчики, относящиеся к работе веб-сервера
- Для сбора метрик и экспорта, например, в Prometheus необходимо подключить инструментарий Open Telemetry
- Документация ASP.NET
  https://learn.microsoft.com/en-us/dotnet/core/diagnostics/metrics-collection
- Open Telemetry
  https://opentelemetry.io/docs/languages/net/getting-started/

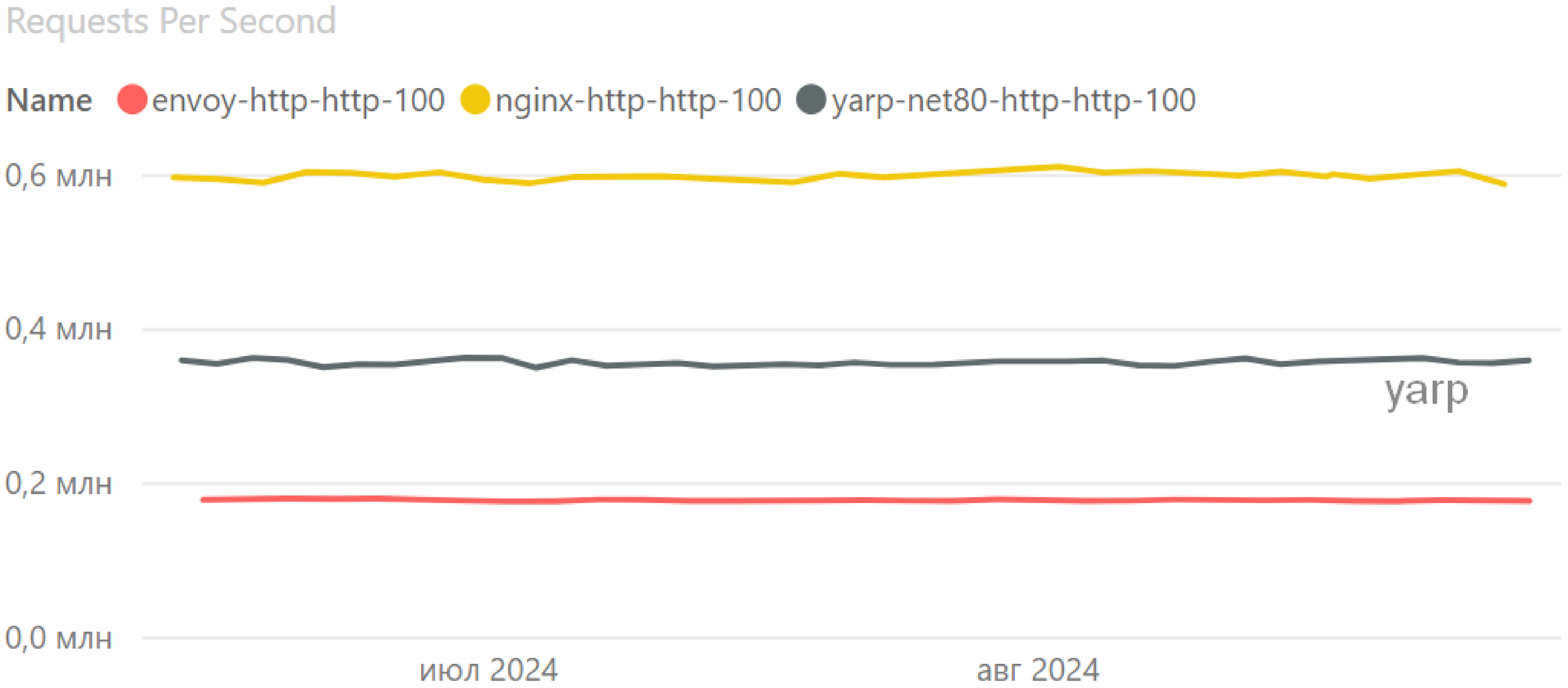


Распределенная трассировка

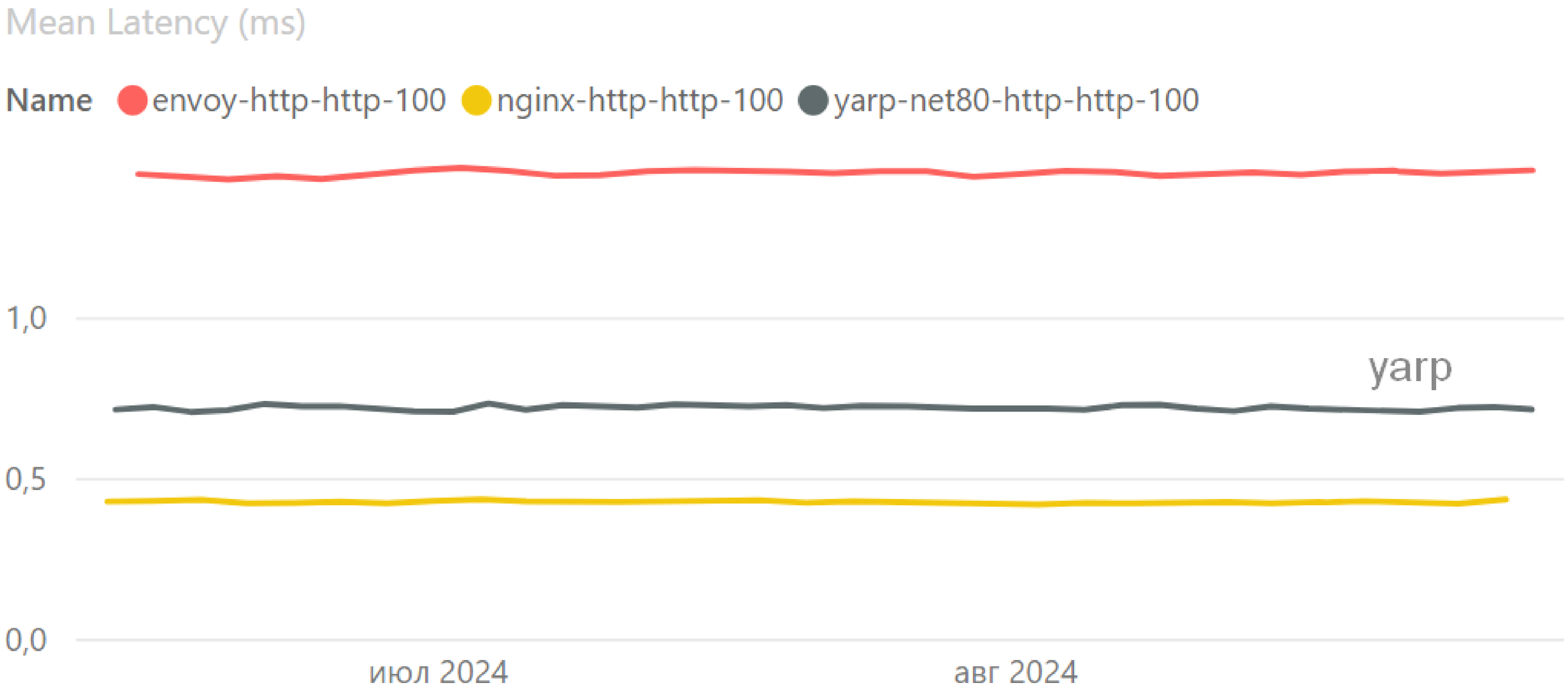
- полностью поддерживает трассировку Open Telemetry
- создаёт отдельный **activity** на каждый переадресованный запрос
- создаёт отдельный **activity** на каждый вызов health check
- распространяет идентификатор Trace Context или создаст его, если необходимо, и создаёт новые **span**/**activity**
- возможно подключение других форматов распространения трассировки (например, **B3**)
- экспорт настраивается стандартно с помощью инструментария Open Telemetry: https://opentelemetry.io/docs/languages/net/getting-started/



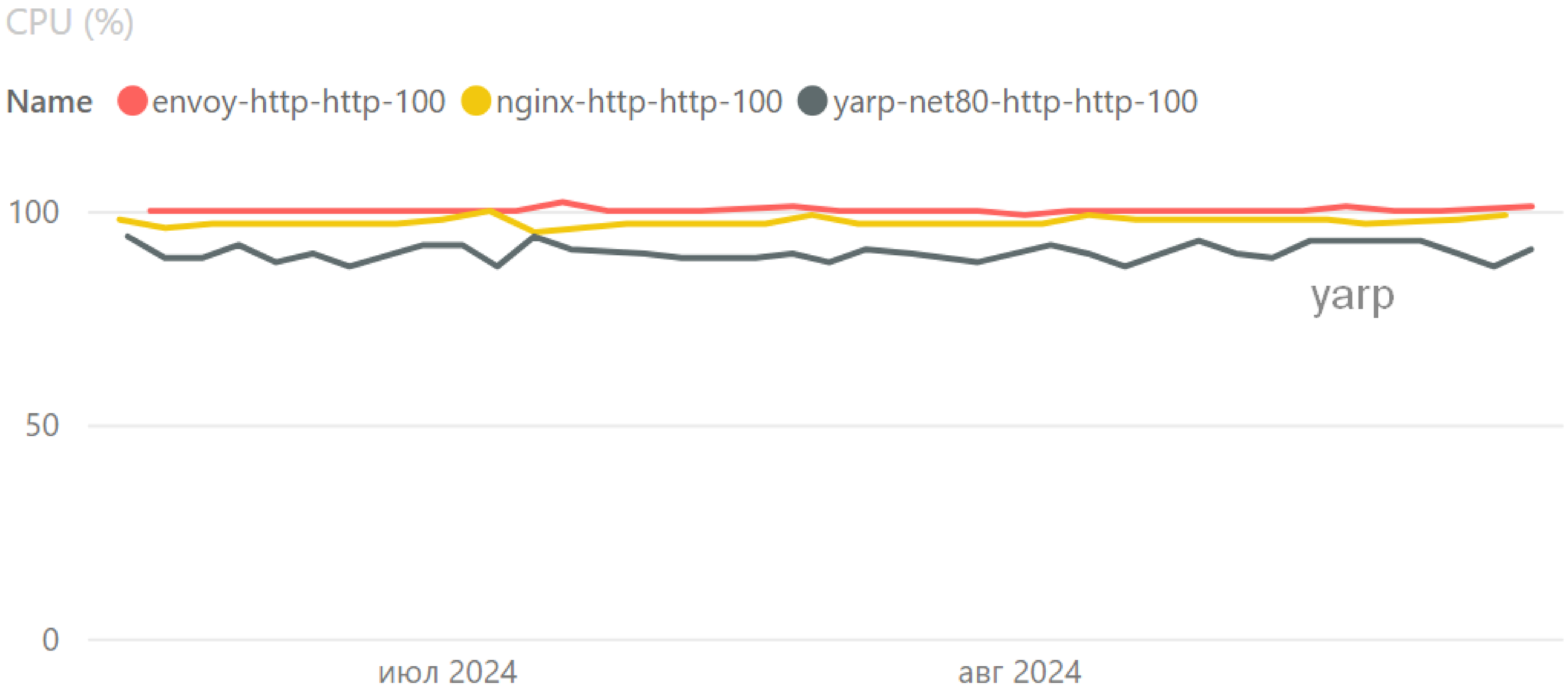
Benchmarks


Requests Per Second
Name
envoy-http-http-100
nginx-http-http-100
yarp-net80-http-http-100
0,6 млн
0,4 млн
0,2 млн
0,0 млн
yarp
июл 2024
авг 2024

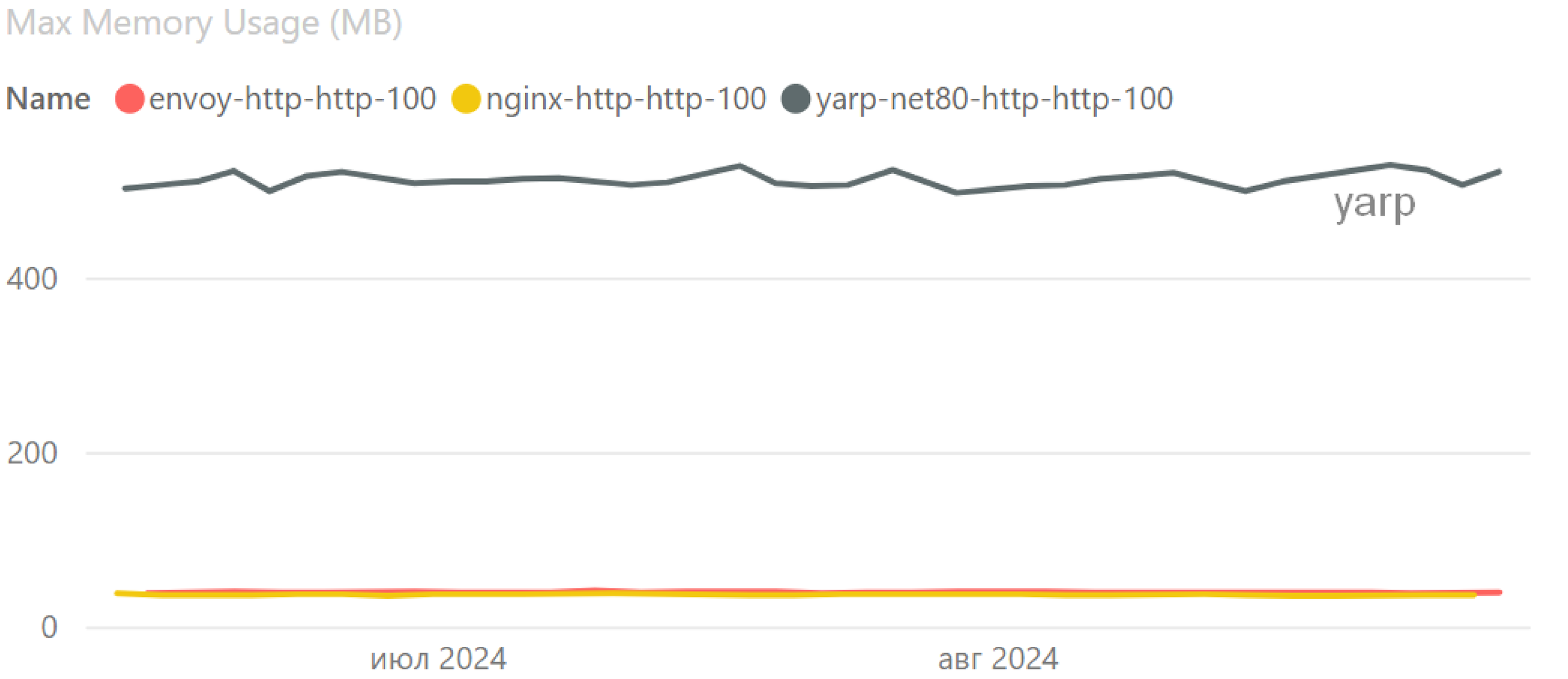
Benchmarks


Mean Latency (ms)
Name
envoy-http-http-100
nginx-http-http-100
yarp-net80-http-http-100
1,0
0,5
0,0
yarp
июл 2024
авг 2024

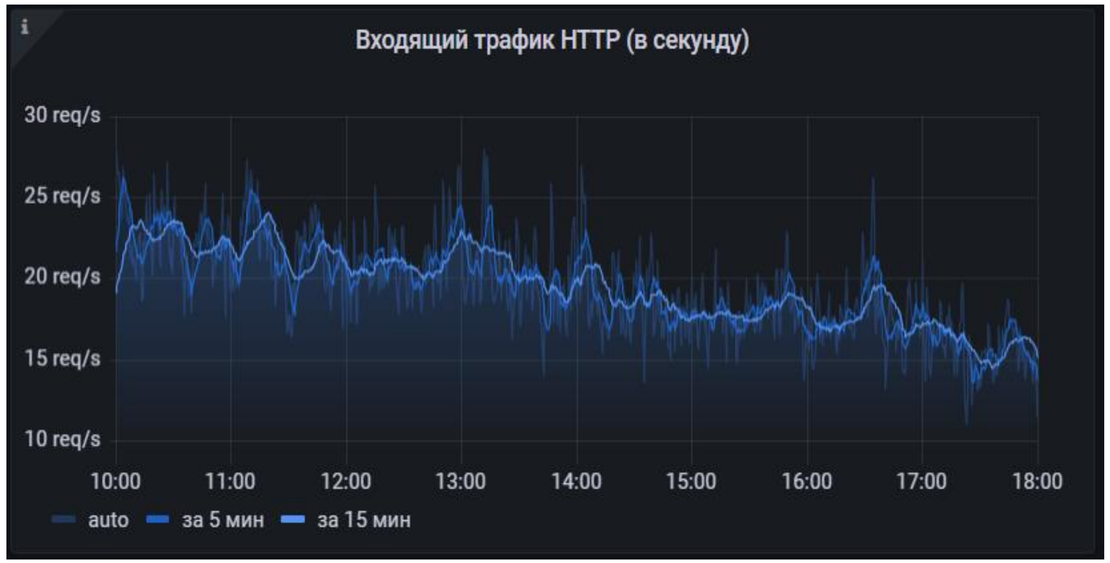
Benchmarks


CPU (%)
Name
envoy-http-http-100
nginx-http-http-100
yarp-net80-http-http-100
100
50
0
yarp
июл 2024
авг 2024

Benchmarks


Max Memory Usage (MB)
Name
envoy-http-http-100
nginx-http-http-100
yarp-net80-http-http-100
yarp
400
200
0
июл 2024
авг 2024

Боевые графики производительности



Наш опыт

- Как API Gateway + BFF
- Как Reverse Proxy + Static Web Server
  - Создали внутренний самостоятельный продукт OmniProxy
  - Доступен как образ в корпоративном docker-реестре
- Как Reverse Proxy + API Gateway для продуктов на других технологиях (проекты на PHP, Python)
- Как Static Proxy для публичного доступа к файлам
- Как Reverse Proxy для внутренних интеграций между продуктами, для наблюдения и мониторинга
- Как балансировщик для legacy-продуктов (.NET Framework 4 on Windows Server)

Будущее YARP

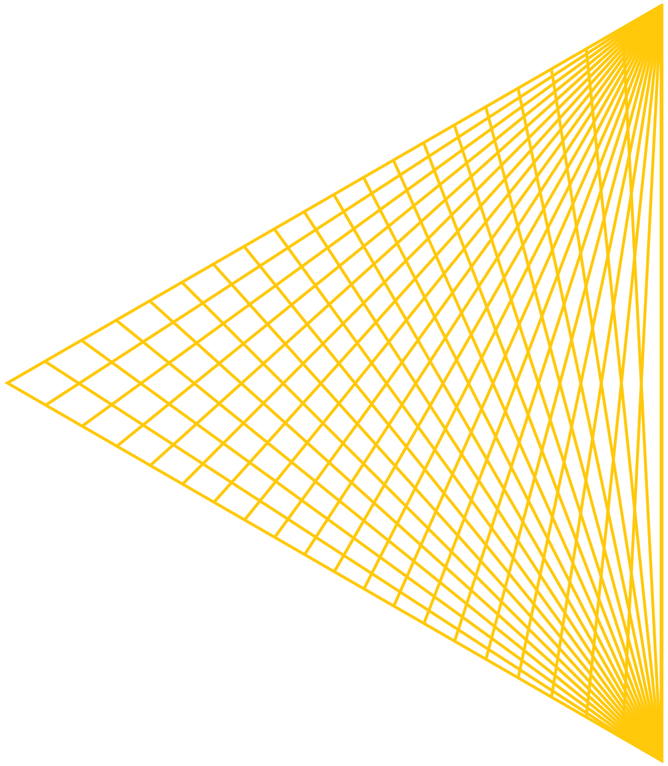
- На текущий момент активно развивается
- Коллаборация с командой core-разработки ASP.NET
  - приводит к повышению производительности и эффективности
- Наблюдается активная работа по интеграции с kubernetes
  - в частности, использование как ingress-controller
- Ожидается поддержка Service Discovery в интеграции с новым продуктом Aspire
- Всё больше команд и продуктов используют YARP для самых различных задач

Выводы

- Не самодостаточный продукт, а расширение
- Нет агрегации запросов и, скорее всего, не предвидится
- В первую очередь программируемый и только затем, конфигурируемый
- Позволяет реализовать полноценный API Gateway и BFF
- Легко сопровождается администраторами и DevOps
- Полностью закрыл все наши изначальные потребности
- Открыл новые возможности по улучшению наблюдаемости существующих продуктов
- Не выглядит, как подходящее решение в качестве **reverse proxy** для экстремальных нагрузок (пока что)

Полезные ссылки

- https://microsoft.github.io/reverse-proxy/ — основной сайт YARP и документация
- https://github.com/ThreeMammals/Ocelot — Ocelot GitHub
- https://github.com/ProxyKit/ProxyKit — ProxyKit GitHub
- https://learn.microsoft.com/ru-ru/dotnet/architecture/microservices/ — электронная книга Microsoft по архитектуре микросервисов
- https://learn.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-8.0 — фундаментальные знания по ASP.NET
- https://aka.ms/aspnet/benchmarks — много бенчмарков в реальном времени



Вопросы?

Спасибо
за внимание!