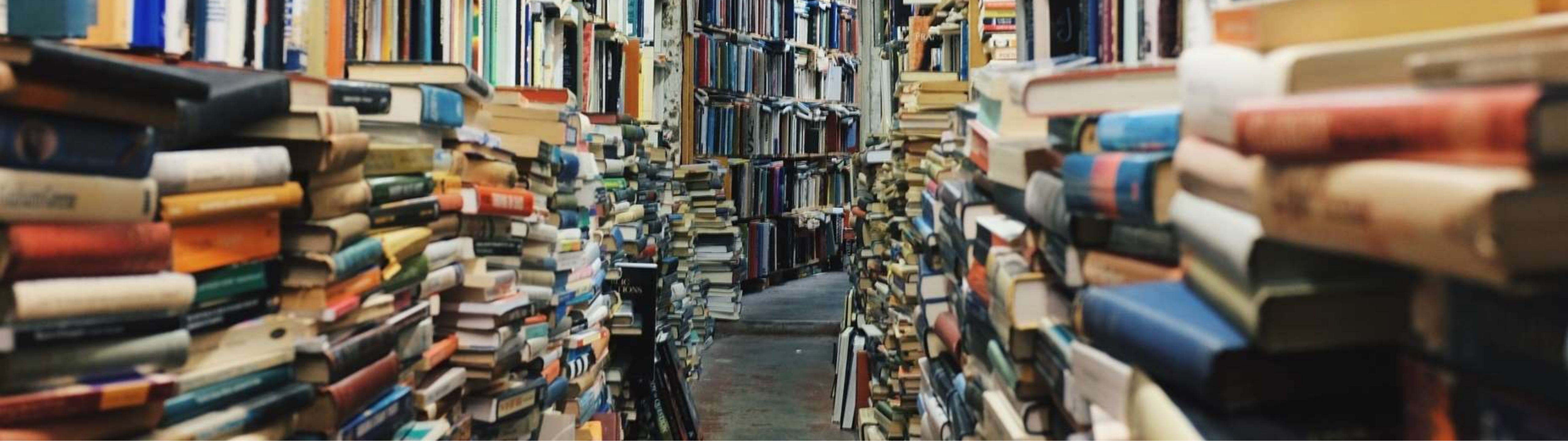


Clean up this mess!

API Gateway and Service Discovery in .NET

Marcin Tyborowski

- .NET Developer at Billennium
- Speaker
- Co-organizer of Programistok

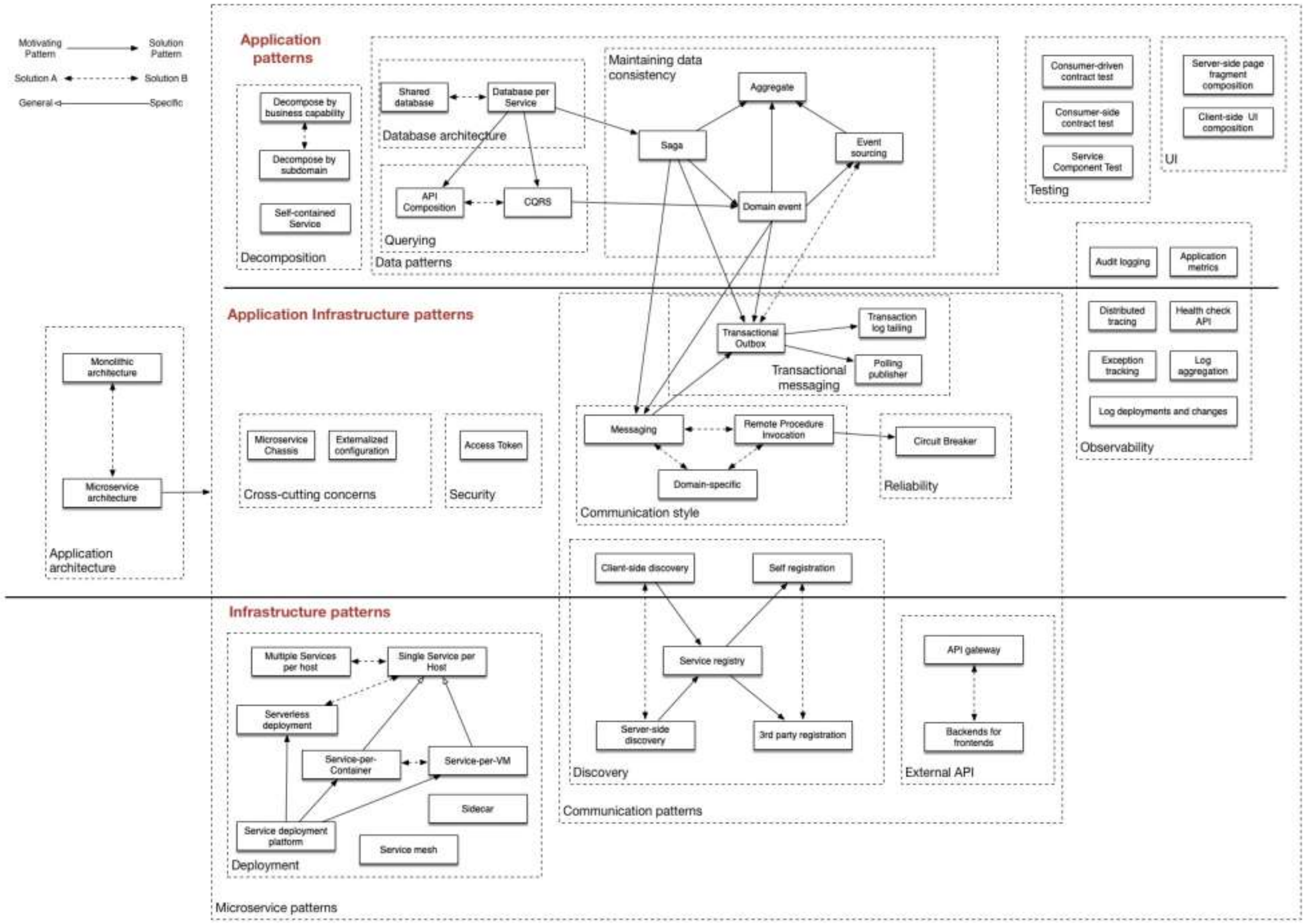
Billennium Inspiration Team

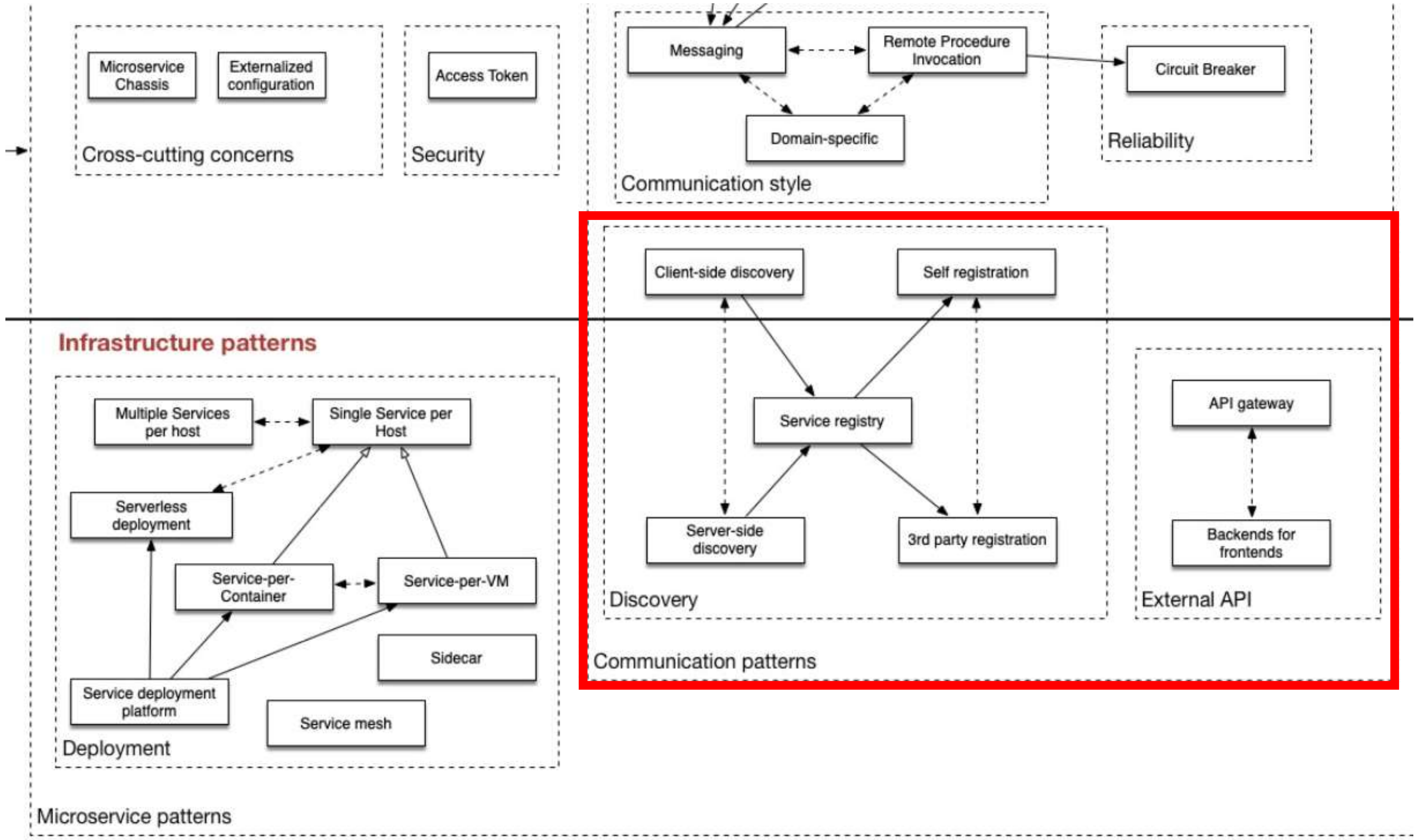
Agenda

- About the project
- API Gateway
- Detect services! – Service Discovery
- Summary

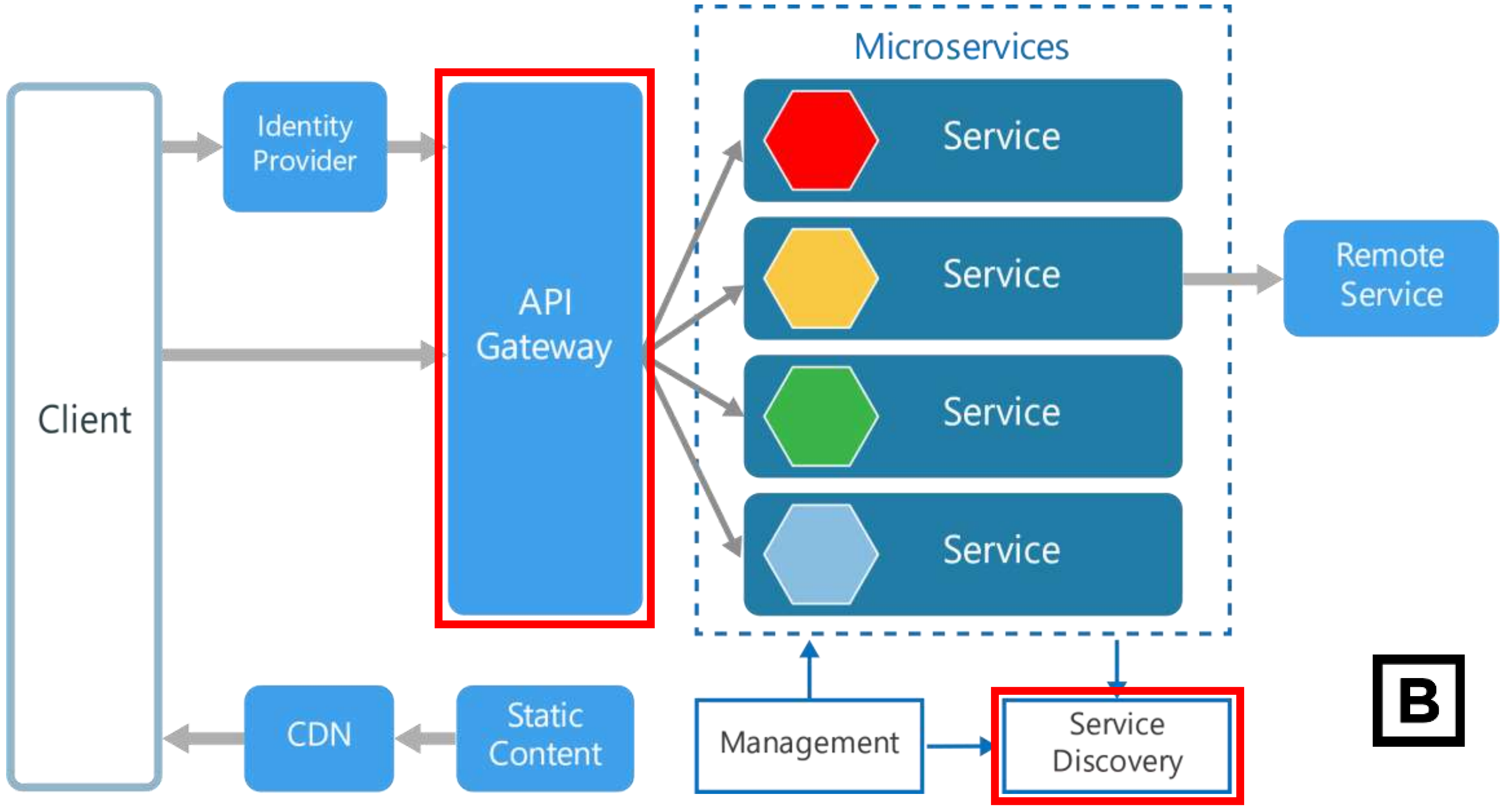
Microservices

- Application as collection of services
- Way of designing software applications
- Is not a golden mean


Motivating Pattern
Solution Pattern
Solution A
Solution B
General
Specific
Application patterns
Decompose by business capability
Decompose by subdomain
Self-contained Service
Decomposition
Shared database
Database per Service
Database architecture
API Composition
CQRS
Querying
Data patterns
Maintaining data consistency
Aggregate
Saga
Event sourcing
Domain event
Consumer-driven contract test
Consumer-side contract test
Service Component Test
Testing
Server-side page fragment composition
Client-side UI composition
UI
Audit logging
Application metrics
Distributed tracing
Health check API
Exception tracking
Log aggregation
Log deployments and changes
Observability
Application Infrastructure patterns
Monolithic architecture
Microservice architecture
Application architecture
Microservice Chassis
Externalized configuration
Cross-cutting concerns
Access Token
Security
Transactional Outbox
Transaction log tailing
Polling publisher
Transactional messaging
Messaging
Remote Procedure Invocation
Domain-specific
Communication style
Circuit Breaker
Reliability
Client-side discovery
Self registration
Service registry
Server-side discovery
3rd party registration
Discovery
API gateway
Backends for frontends
External API
Communication patterns
Infrastructure patterns
Multiple Services per host
Single Service per Host
Serverless deployment
Service-per-Container
Service-per-VM
Sidecar
Service deployment platform
Service mesh
Deployment
Microservice patterns



Microservice Chassis
Externalized configuration
Cross-cutting concerns
Access Token
Security
Messaging
Remote Procedure Invocation
Domain-specific
Communication style
Circuit Breaker
Reliability
Client-side discovery
Self registration
Service registry
Server-side discovery
3rd party registration
Discovery
API gateway
Backends for frontends
External API
Communication patterns
Infrastructure patterns
Multiple Services per host
Single Service per Host
Serverless deployment
Service-per-Container
Service-per-VM
Sidecar
Service deployment platform
Service mesh
Deployment
Microservice patterns


Client
Identity Provider
API Gateway
Microservices
Service
Service
Service
Service
Remote Service
CDN
Static Content
Management
Service Discovery

B

About the project

Divide city into zones

Calculate optimal routes

Synchronization data

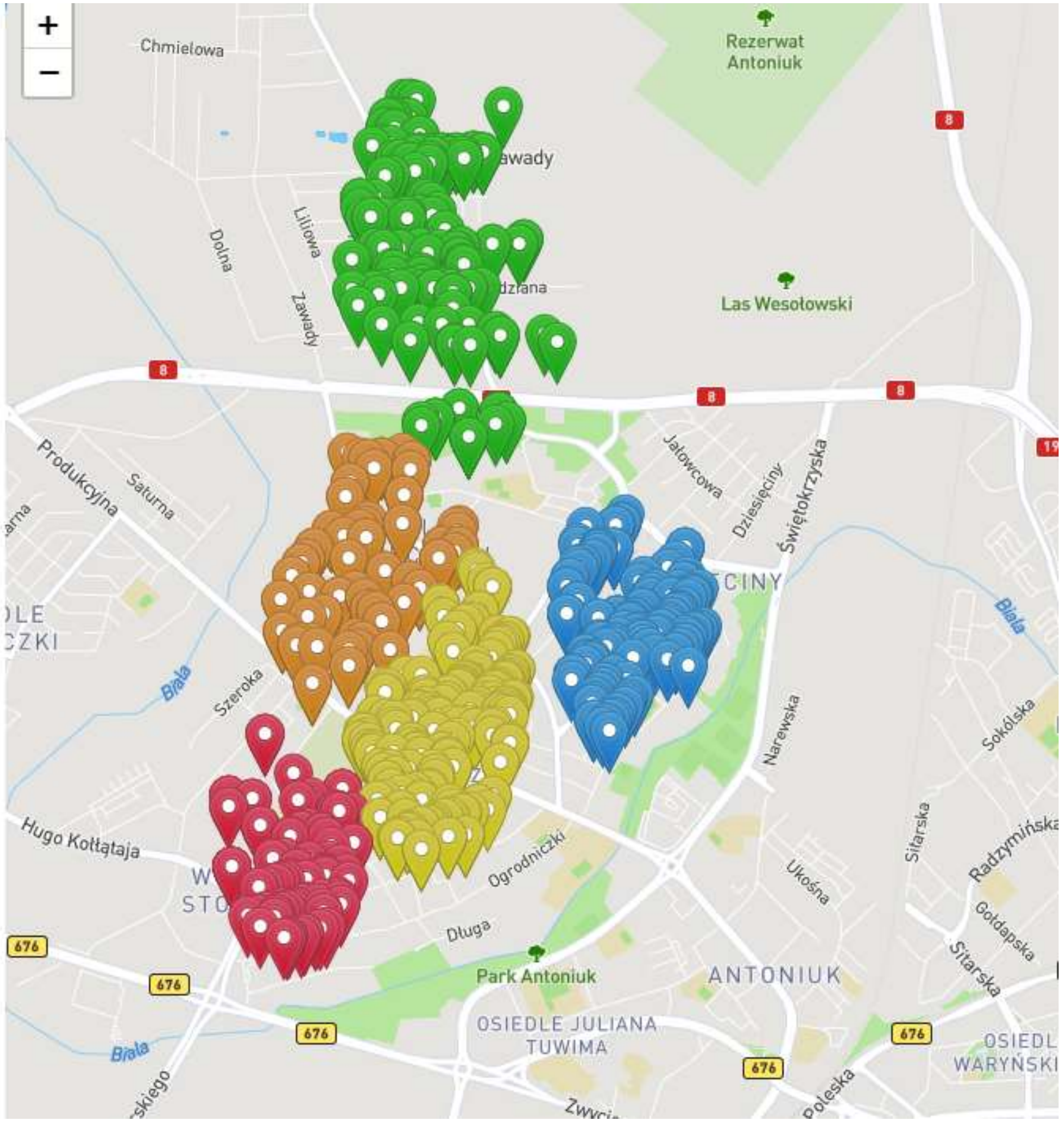
11.03.2019

Podział na klastry

ZAŁOŻENIA ✎ Edytuj

Miasto	Białystok
Liczba firm	5
Liczba klastrów	5

- 🔵 Firma1 - Klaster 1 ▾
- 🟢 Firma2 - Klaster 2 ▾
- 🔴 Firma3 - Klaster 3 ▾
- 🟠 Firma4 - Klaster 4 ▾
- 🟡 Firma5 - Klaster 5 ▾



Wprowadź dane do wyszukania archiwalnej trasy

Data

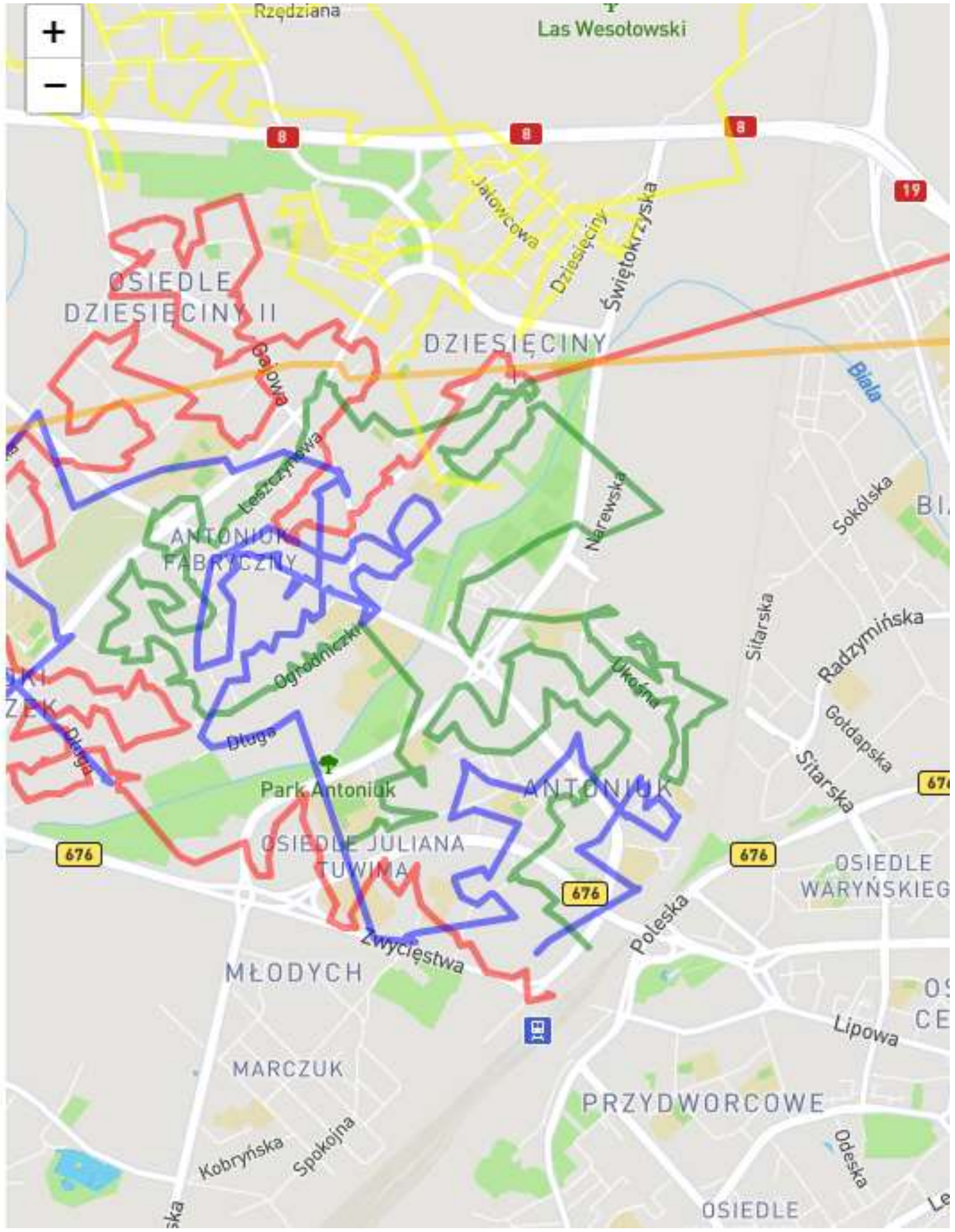
dd.mm.rrrr

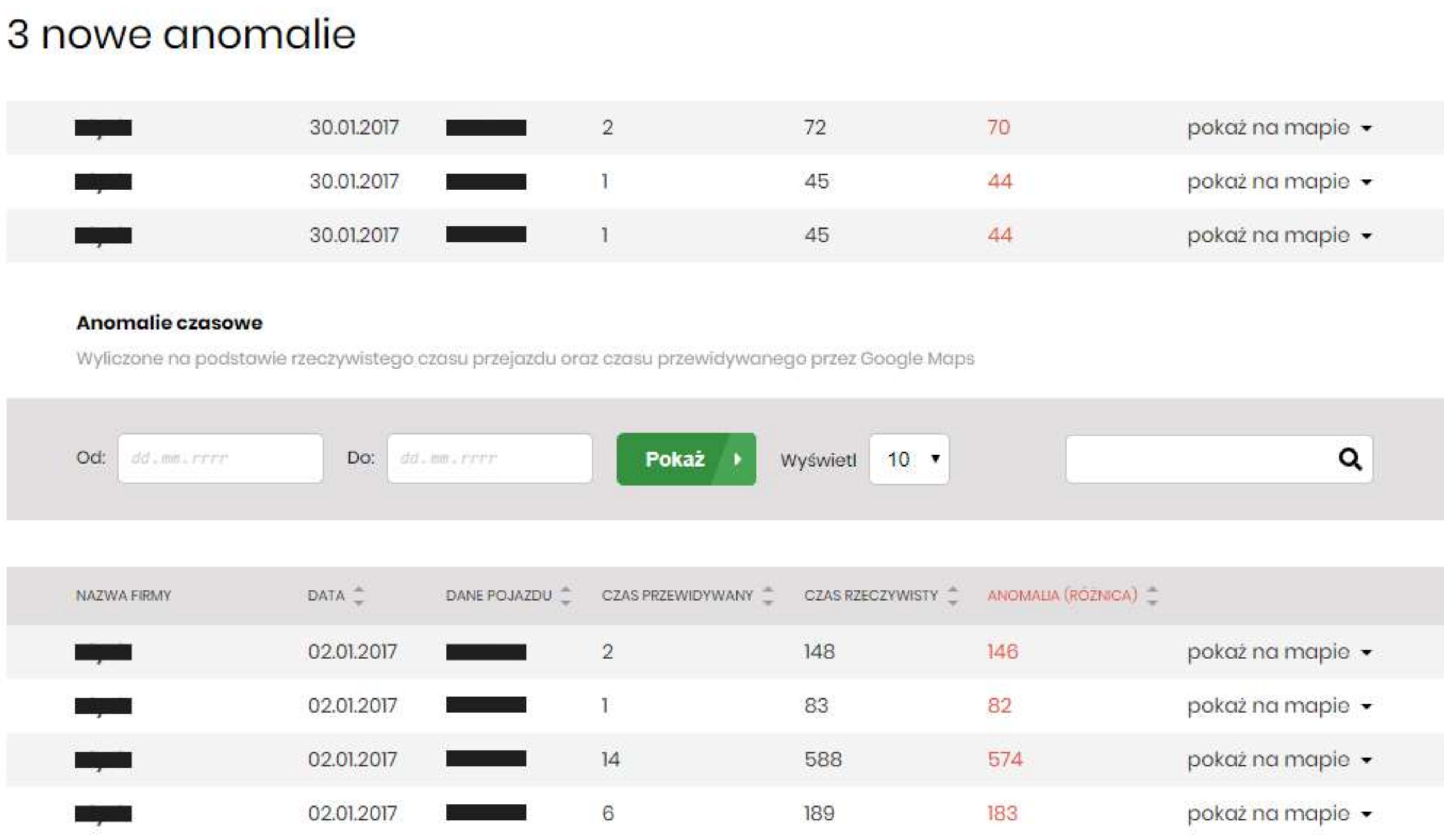
Numer rejestracyjny pojazdy

wyszukaj

optymalizuj

Przebyty dystans po optymalizacji:
1 ciężarówka - 39km
2 ciężarówka - 41km
3 ciężarówka - 53km
4 ciezarowka - 64km
Suma: 197km





3 nowe anomalie

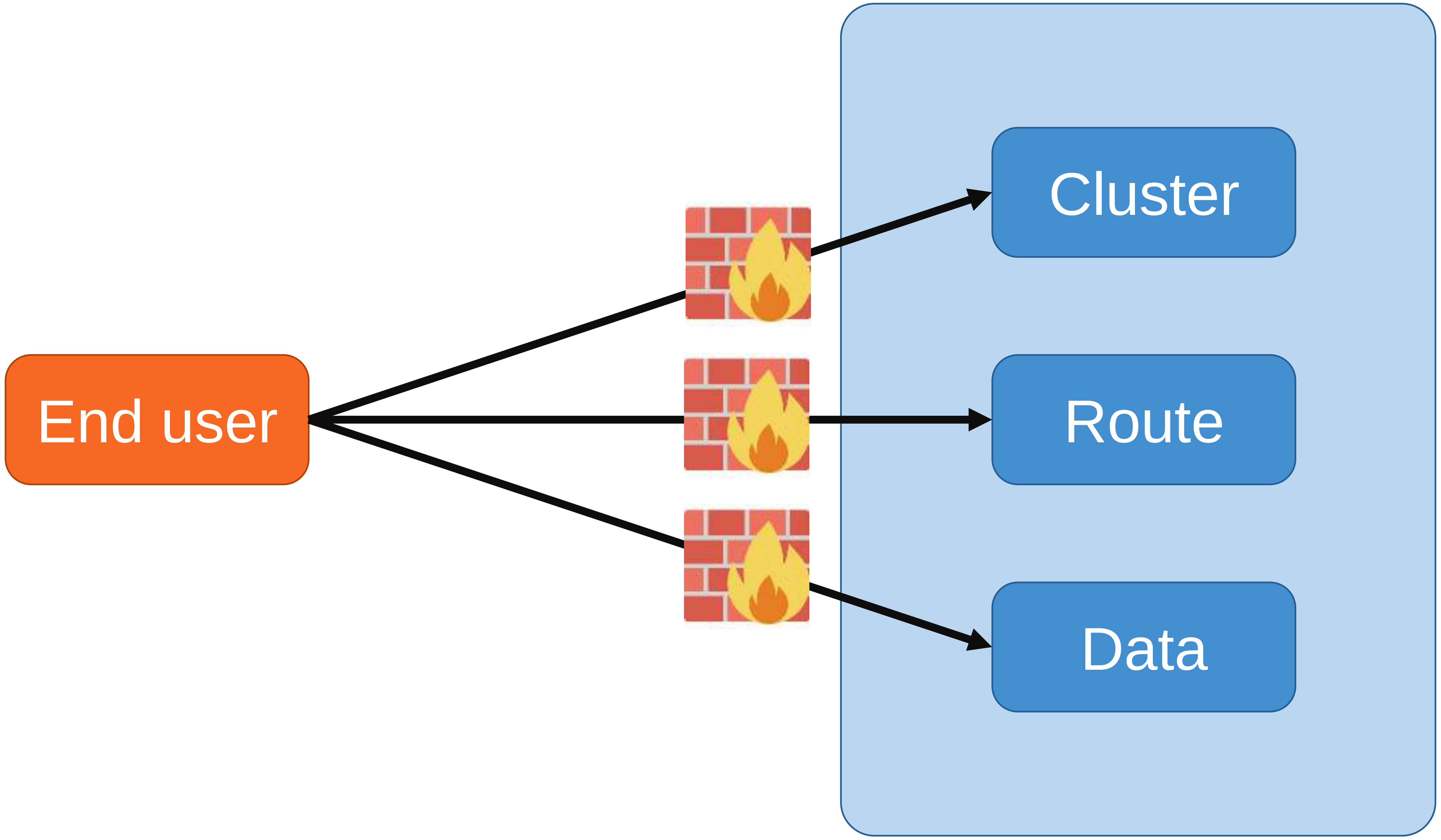
[redacted]	30.01.2017	[redacted]	2	72	70	pokaż na mapie ▾
[redacted]	30.01.2017	[redacted]	1	45	44	pokaż na mapie ▾
[redacted]	30.01.2017	[redacted]	1	45	44	pokaż na mapie ▾

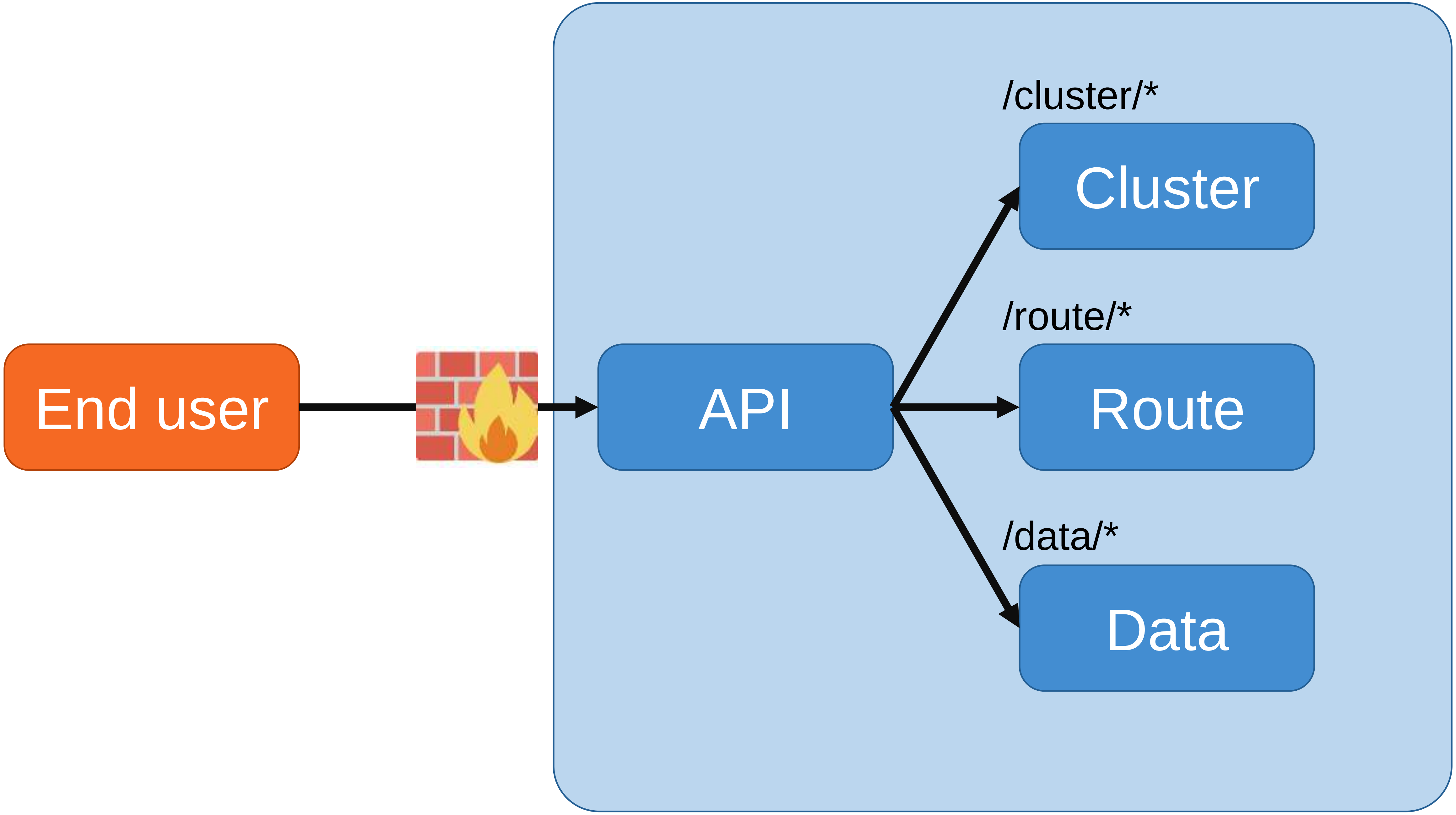
Anomalie czasowe

Wyliczone na podstawie rzeczywistego czasu przejazdu oraz czasu przewidywanego przez Google Maps

Od: dd.mm.rrrr Do: dd.mm.rrrr Pokaż Wyświetl 10

NAZWA FIRMY	DATA	DANE POJAZDU	CZAS PRZEWIDYWANY	CZAS RZECZYWISTY	ANOMALIA (RÓŻNICA)	
[redacted]	02.01.2017	[redacted]	2	148	146	pokaż na mapie ▾
[redacted]	02.01.2017	[redacted]	1	83	82	pokaż na mapie ▾
[redacted]	02.01.2017	[redacted]	14	588	574	pokaż na mapie ▾
[redacted]	02.01.2017	[redacted]	6	189	183	pokaż na mapie ▾

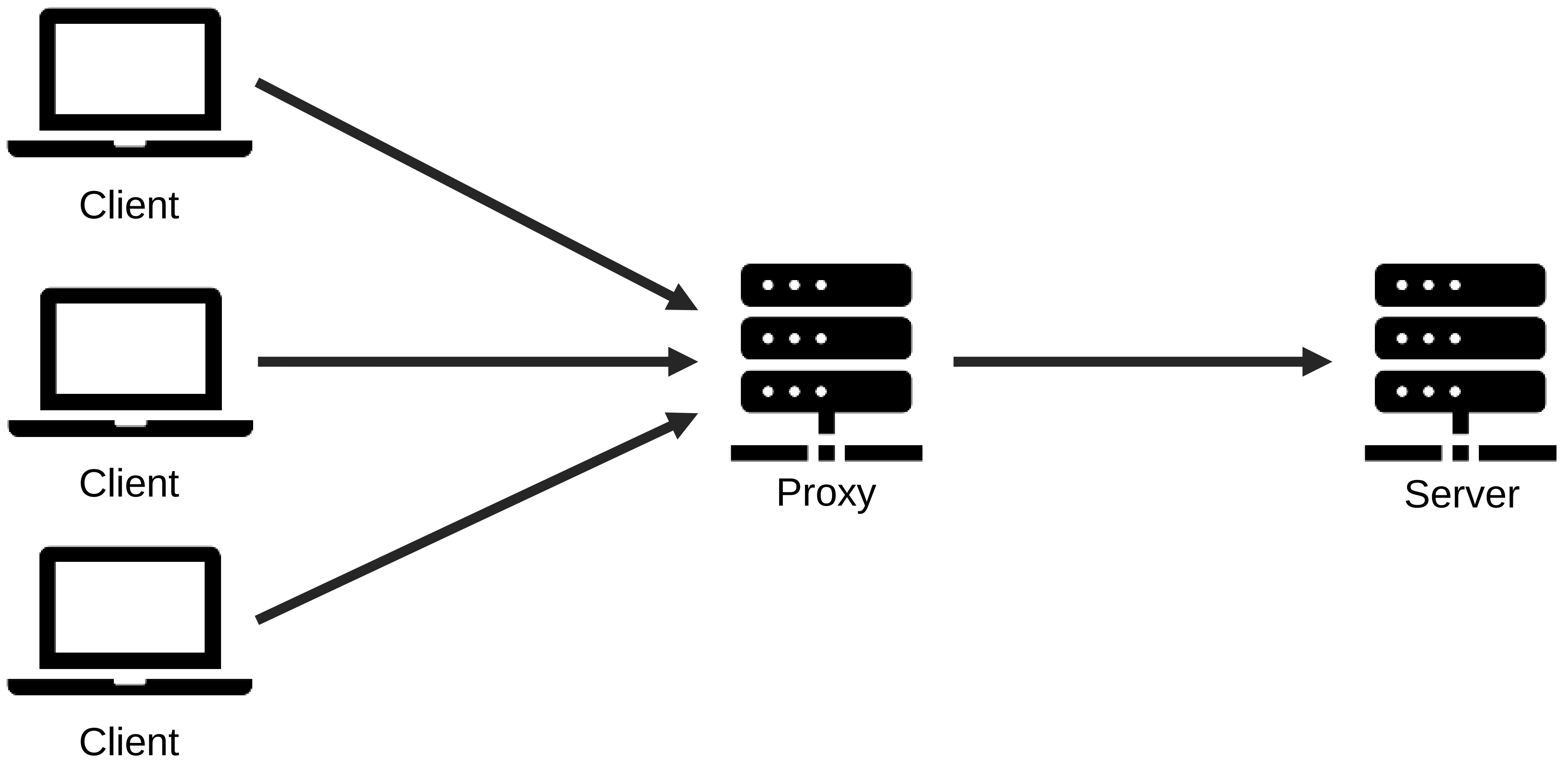

End user
Cluster
Route
Data


End user
API
/cluster/*
Cluster
/route/*
Route
/data/*
Data

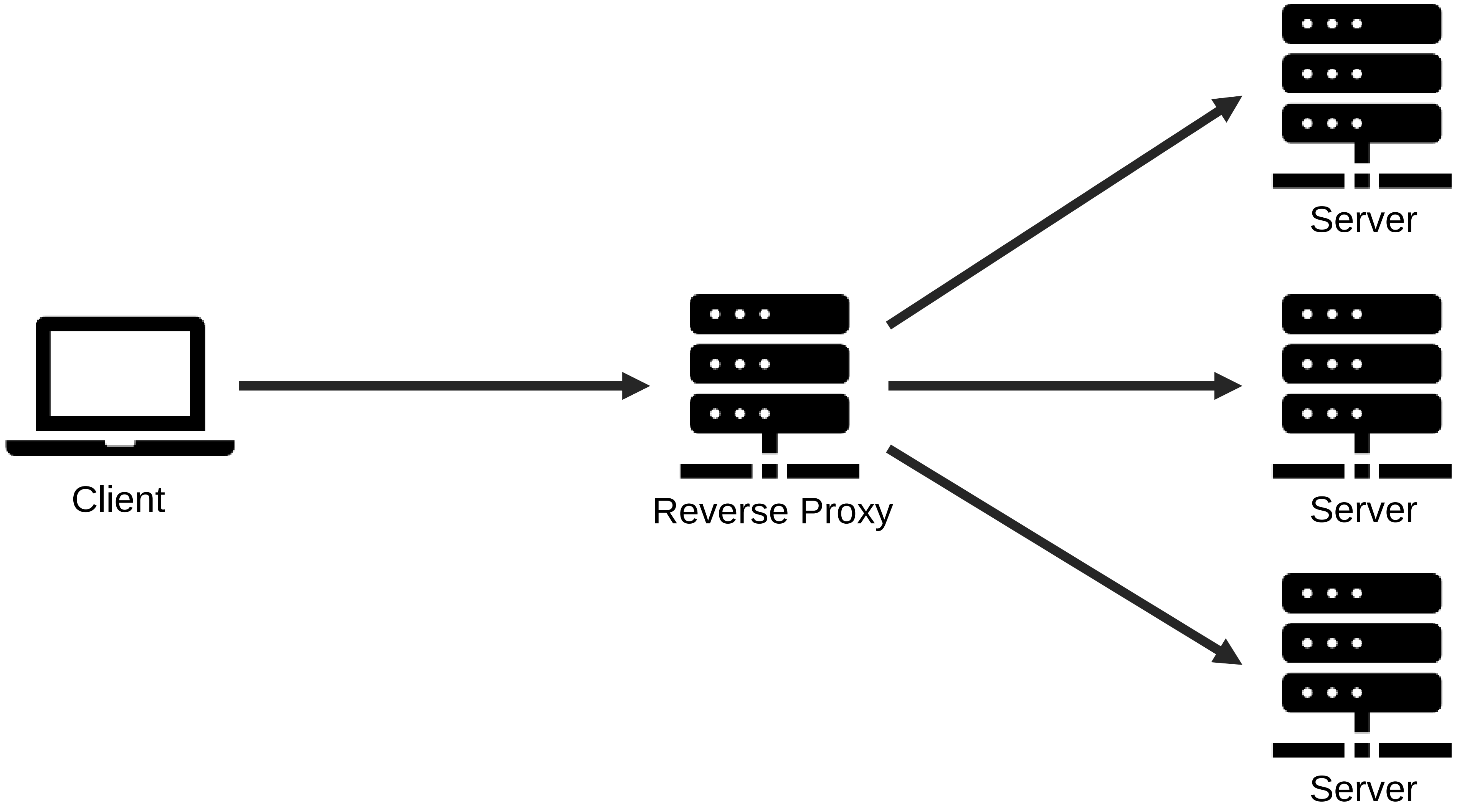
API Gateway

- One public IP
- Separate common mechanism from services
- Canary release
- Reverse proxy

Proxy



Reverse Proxy



OCELOT

RestEase

Service Contract in RestEase

API

```
public interface IRouteService
{
   [Get("route")]
   Task<IEnumerable<Route>> GetRoutes();

   [Get("route/{id}")]
   Task<Route> GetRouteById([Path] int id);

   [Post("route")]
   Task<Response<HttpResponseMessage>>
   Post(AddRouteCommand route);
}
```

Ocelot config file

API

```
{
  "DownstreamPathTemplate": "/route/{everything}",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5001
    }
  ],
  "UpstreamPathTemplate": "/route/{everything}",
  "UpstreamHttpMethod": [ "POST", "PUT", "GET" ]
}
```

Service Initialization

API

```
[Route("[controller]")]
public class RouteController : ControllerBase
{
   private readonly IRouteService routeService;

   public RouteController()
   {
      routeService =
   RestClient.For<IRouteService>("http://localhost:5020");
   }

   [HttpGet("{id}")]
   public async Task<Route> GetRouteById(int id)
      => await routeService.GetRouteById(id);
}
```

Service Implementation

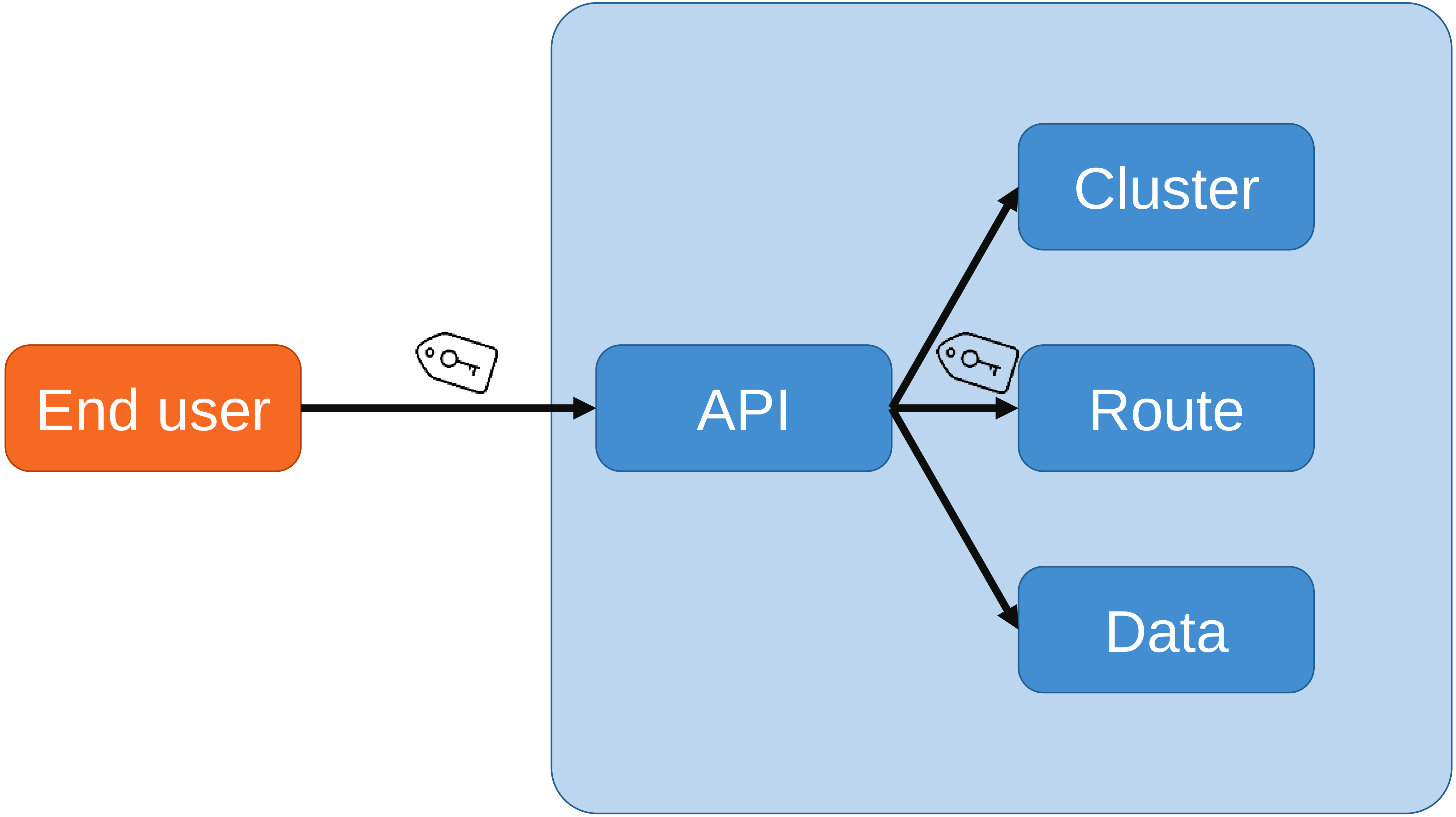
```
[Route("[controller]")]
public class RouteController : ControllerBase
{
  [HttpGet("{id}")]
  public async Task<Route> GetRouteById(int id)
  {
    return ExampleRouteData();
  }
  [HttpGet]
  public async Task<IEnumerable<Route>> GetRoutes()
  {
    return ExampleRoutesData();
  }
}
```

Route Service

B

Secure our application

- **Add JWT Authentication**
- Hide internal services in Virtual Network
- Enable CORS


End user
API
Cluster
Route
Data

JWT Authentication

API

```
public interface IRouteService
{
    [Header("Authorization")]
    AuthenticationHeaderValue Authorization { get; }

    [AllowAnyStatusCode]
    [Get("route")]
    Task<IEnumerable<Route>> GetRoutes();

    [AllowAnyStatusCode]
    [Get("route/{id}")]
    Task<Route> GetRouteById([Path] int id);

    [AllowAnyStatusCode]
    [Post("route")]
    Task<Response<HttpResponseMessage> Post(AddRouteCommand route);
}
```

JWT Authentication

API

```
public async Task<IEnumerable<Route>> GetRoutes()
{
    routeService.Authorization =
        new AuthenticationHeaderValue(
            JwtBearerDefaults.AuthenticationScheme,
            Request.Headers["Authorization"].ToString().Substring(7));

    return await routeService.GetRoutes();
}
```

JWT Authentication

Route Service

```
[Route("[controller]")]
[Authorize(AuthenticationSchemes = "Bearer")]
public class RouteController : ControllerBase
{
    [HttpGet("{id}")]
    public async Task<Route> GetRouteById(int id)
    {
        return ExampleRouteData();
    }
    [HttpGet]
    public async Task<IEnumerable<Route>> GetRoutes()
    {
        return ExampleRoutesData();
    }
}
```

Best practices

- Offload cross-cutting concerns
  - Auth
  - SSL offloading
  - Security
  - Monitoring & logging
- Bottleneck!!
- Keep domain knowledge/logic out of GW

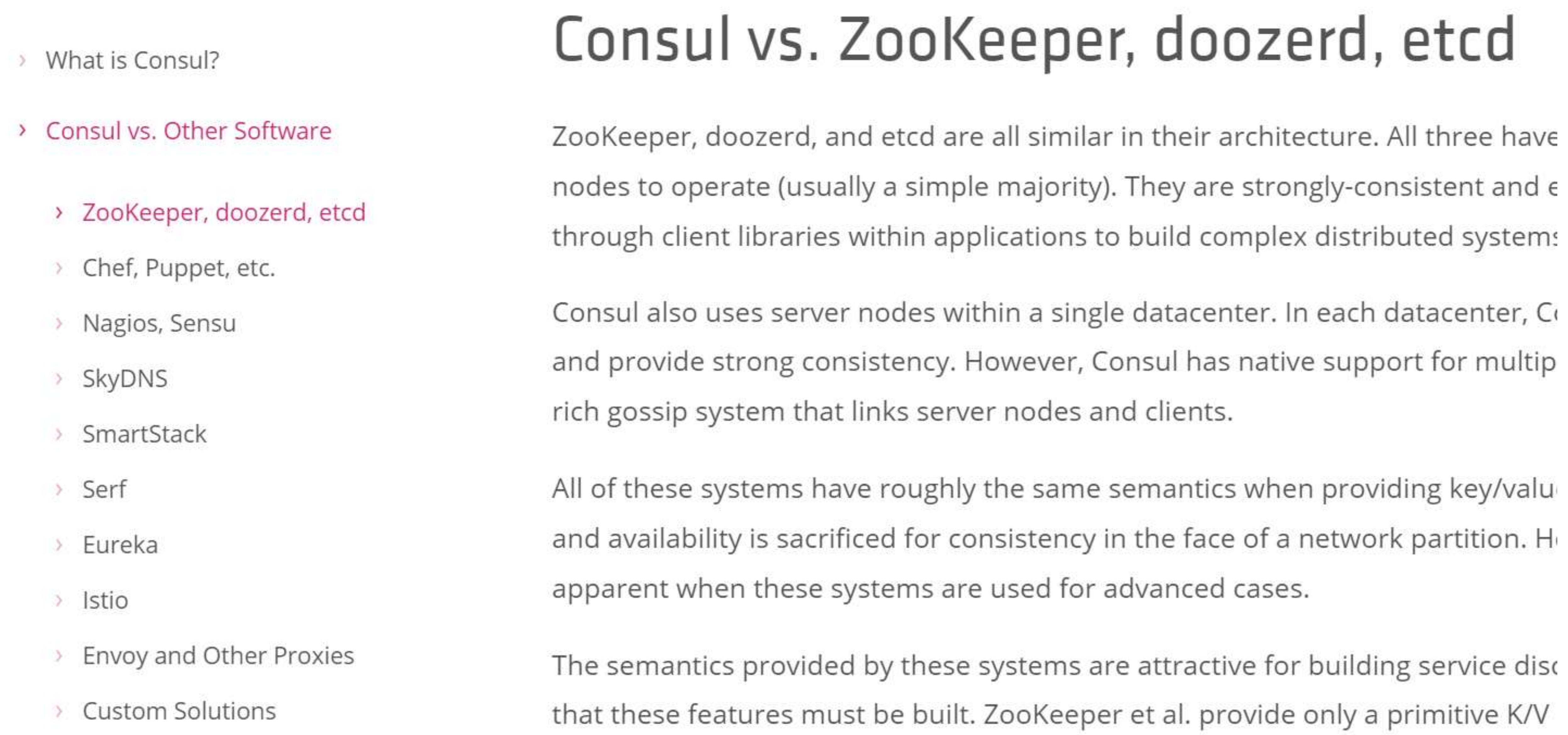
Is there something at the end of the road?

Service Discovery

- Services actual state
- Health probes
- Key-value store
- Client-side (CSSD)
  - API asks register
- Server-side (SSSD)
  - API send request to LB which uses register

HashiCorp
Consul

NETFLIX
EUREK
A



Consul vs. ZooKeeper, doozerd, etcd

ZooKeeper, doozerd, and etcd are all similar in their architecture. All three have nodes to operate (usually a simple majority). They are strongly-consistent and e through client libraries within applications to build complex distributed system

Consul also uses server nodes within a single datacenter. In each datacenter, C and provide strong consistency. However, Consul has native support for multip rich gossip system that links server nodes and clients.

All of these systems have roughly the same semantics when providing key/valu and availability is sacrificed for consistency in the face of a network partition. H apparent when these systems are used for advanced cases.

The semantics provided by these systems are attractive for building service disc that these features must be built. ZooKeeper et al. provide only a primitive K/V

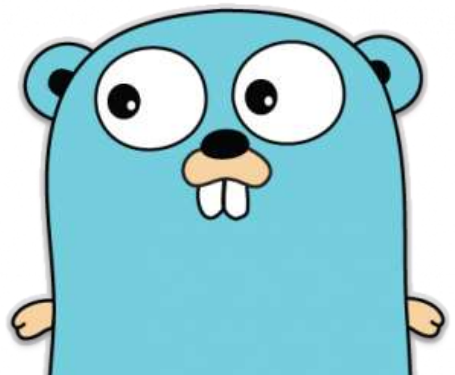
Consul

Service Discovery

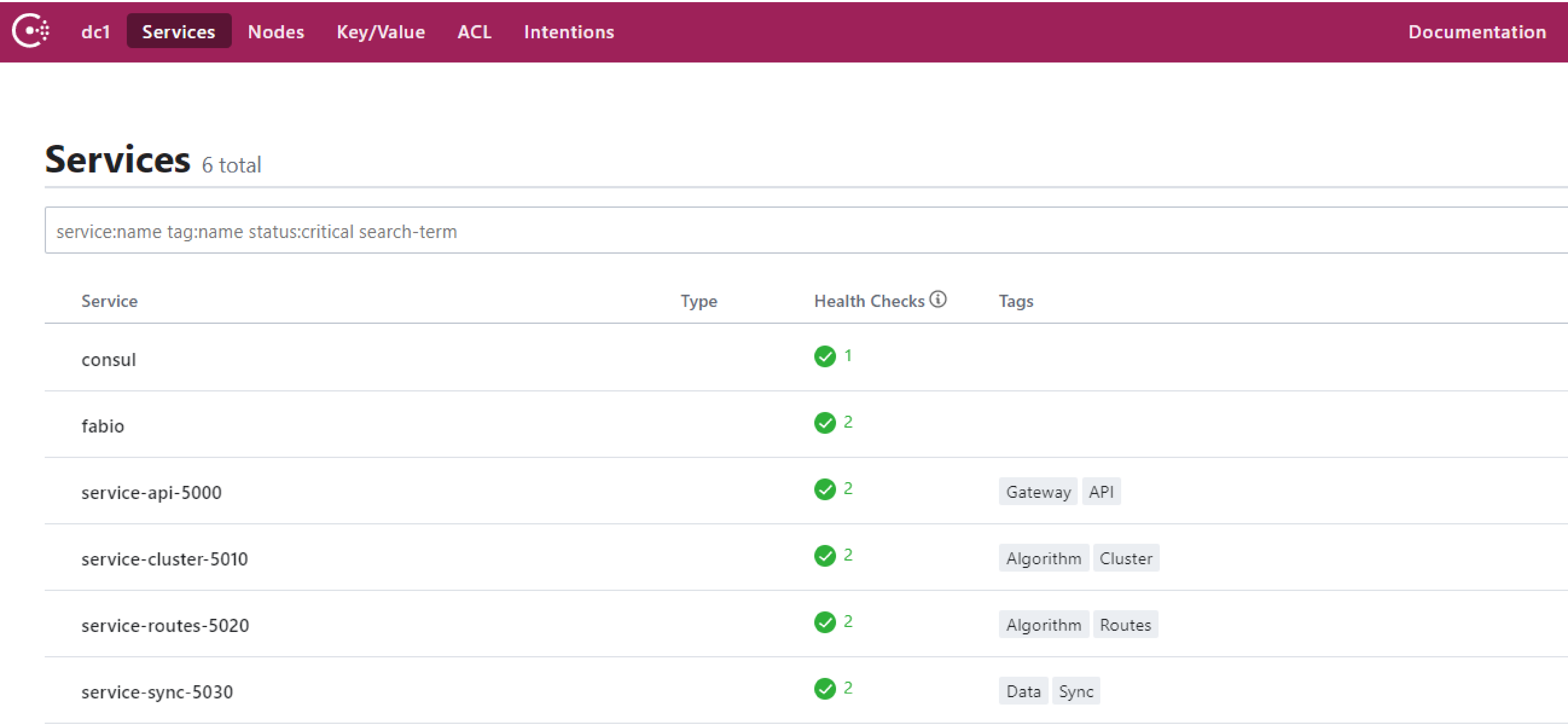
- Registry with service state

Service Mesh

- Service-to-service communication
- Traffic management
- Observability



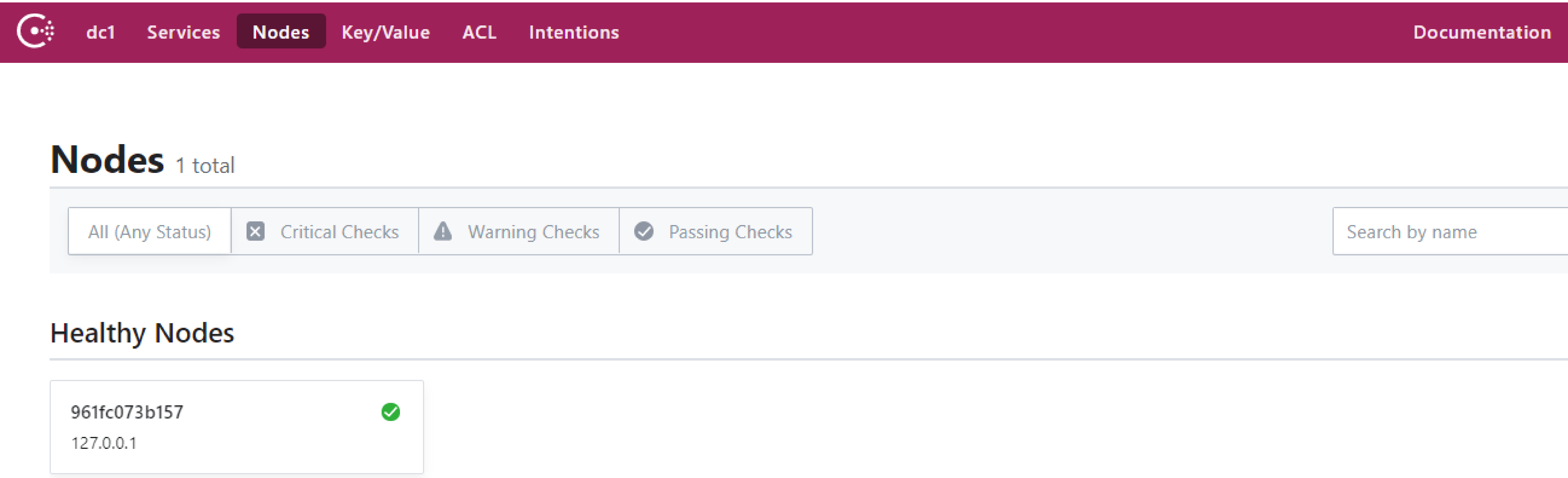
HashiCorp
Consul



Services 6 total

service:name tag:name status:critical search-term

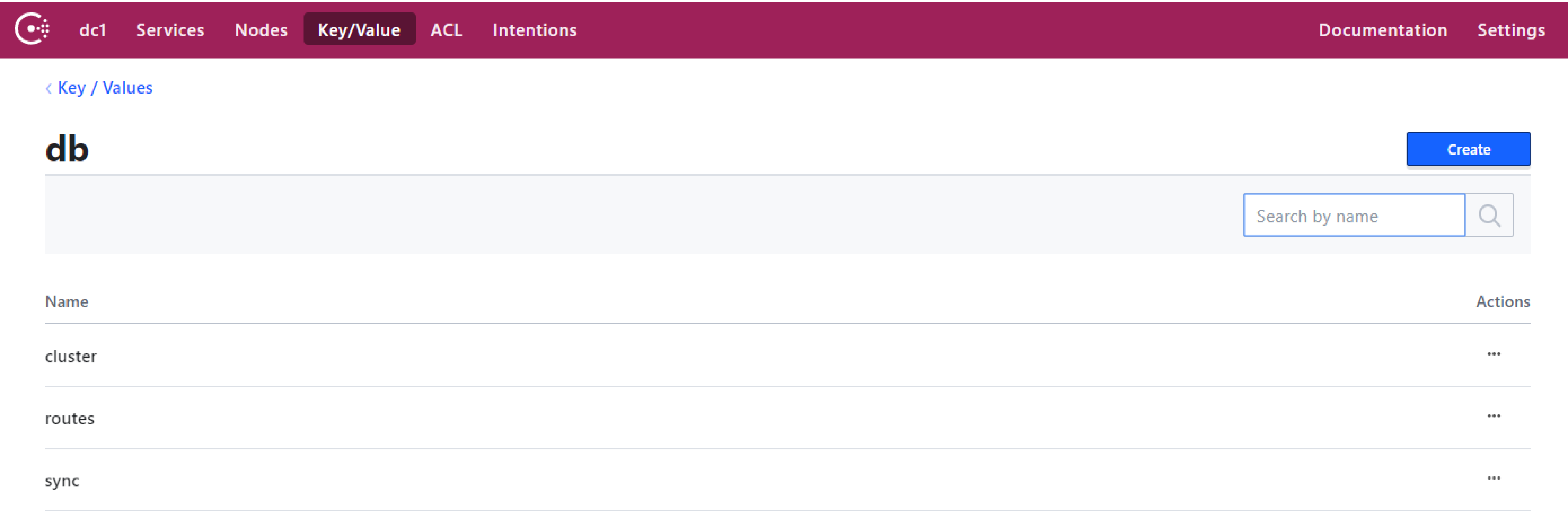
Service	Type	Health Checks	Tags
consul		✓ 1	
fabio		✓ 2	
service-api-5000		✓ 2	Gateway API
service-cluster-5010		✓ 2	Algorithm Cluster
service-routes-5020		✓ 2	Algorithm Routes
service-sync-5030		✓ 2	Data Sync



Nodes 1 total

Healthy Nodes

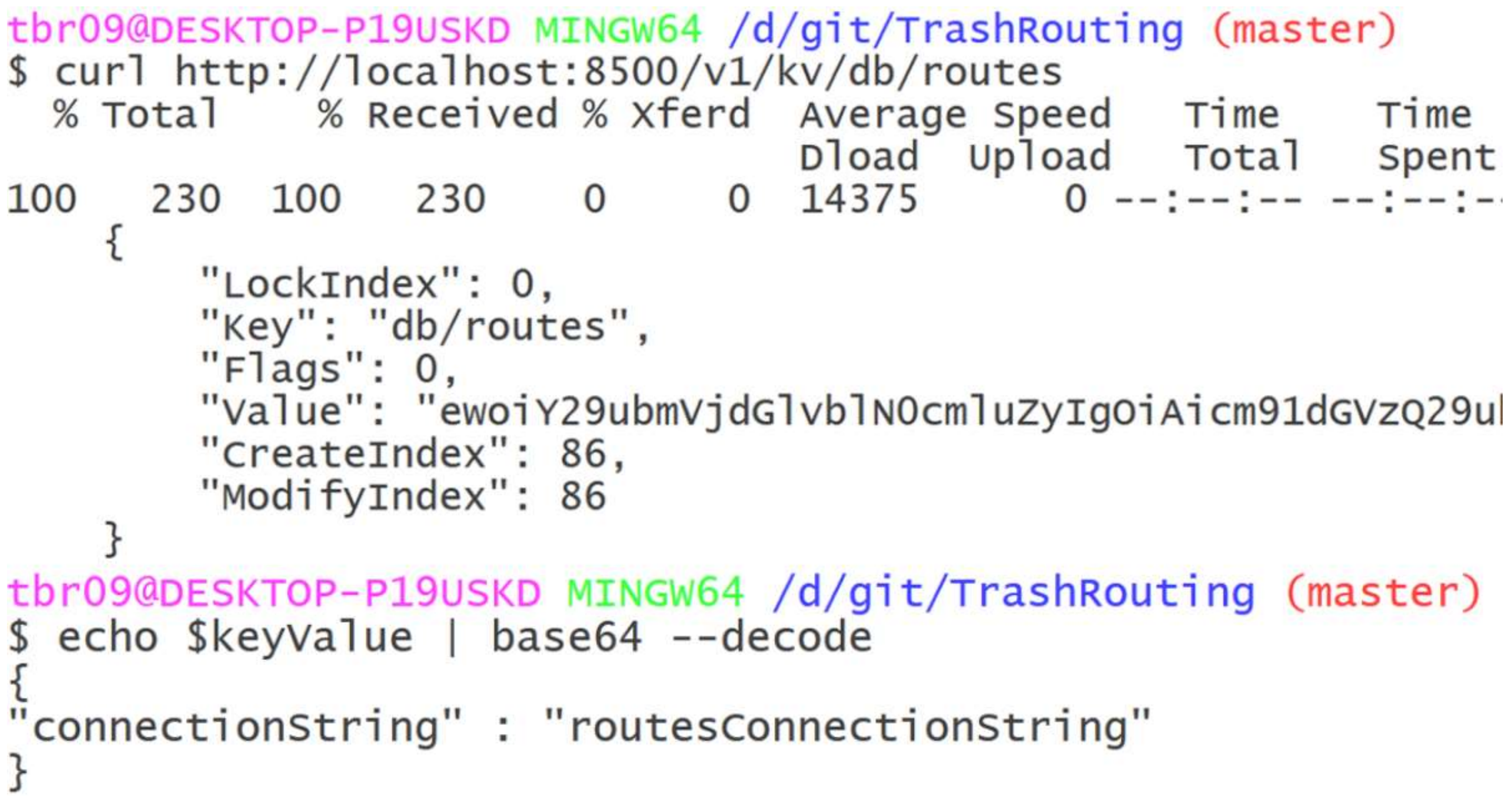
961fc073b157
127.0.0.1



db

Name	Actions
cluster	···
routes	···
sync	···

db/cluster
db/routes
db/sync

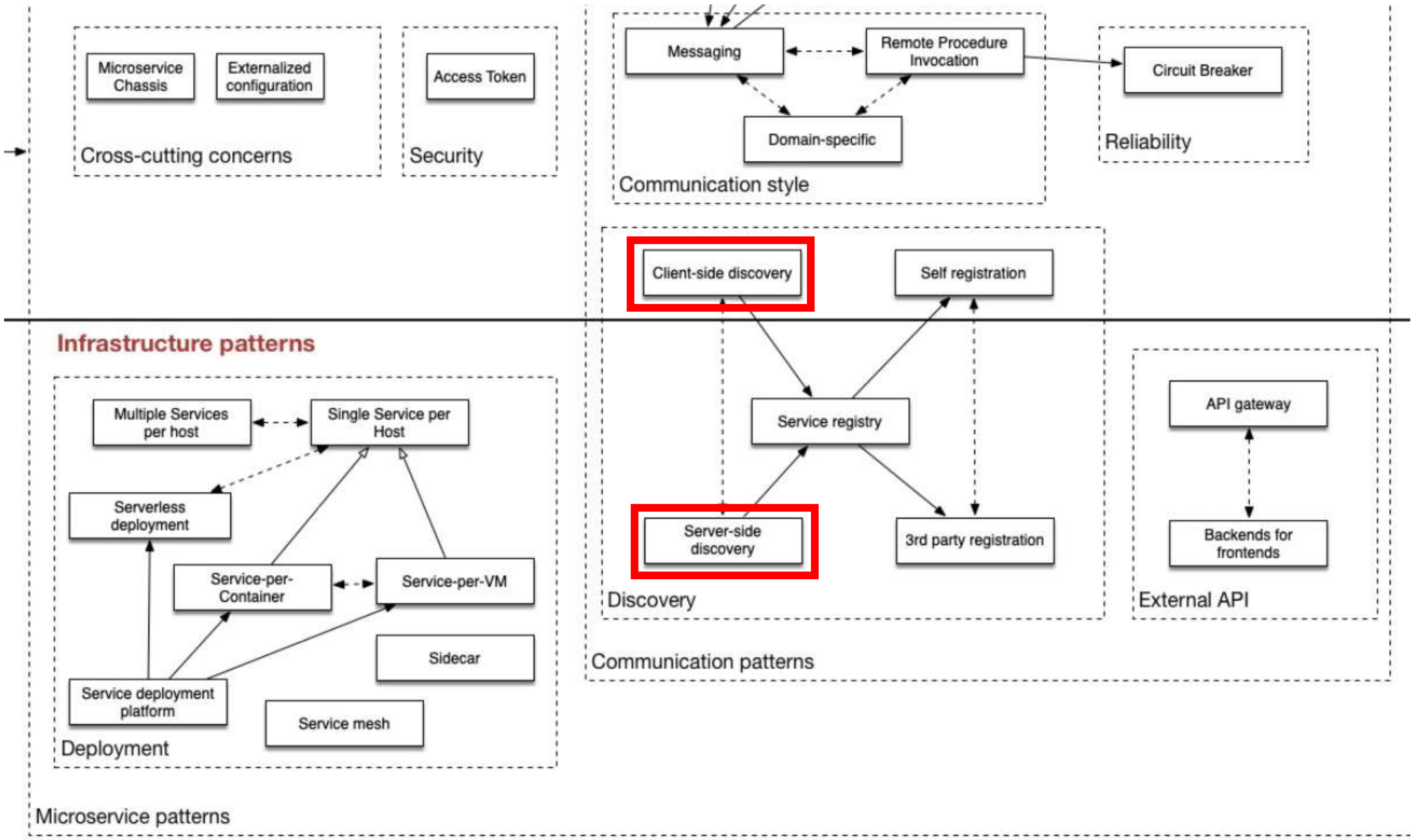


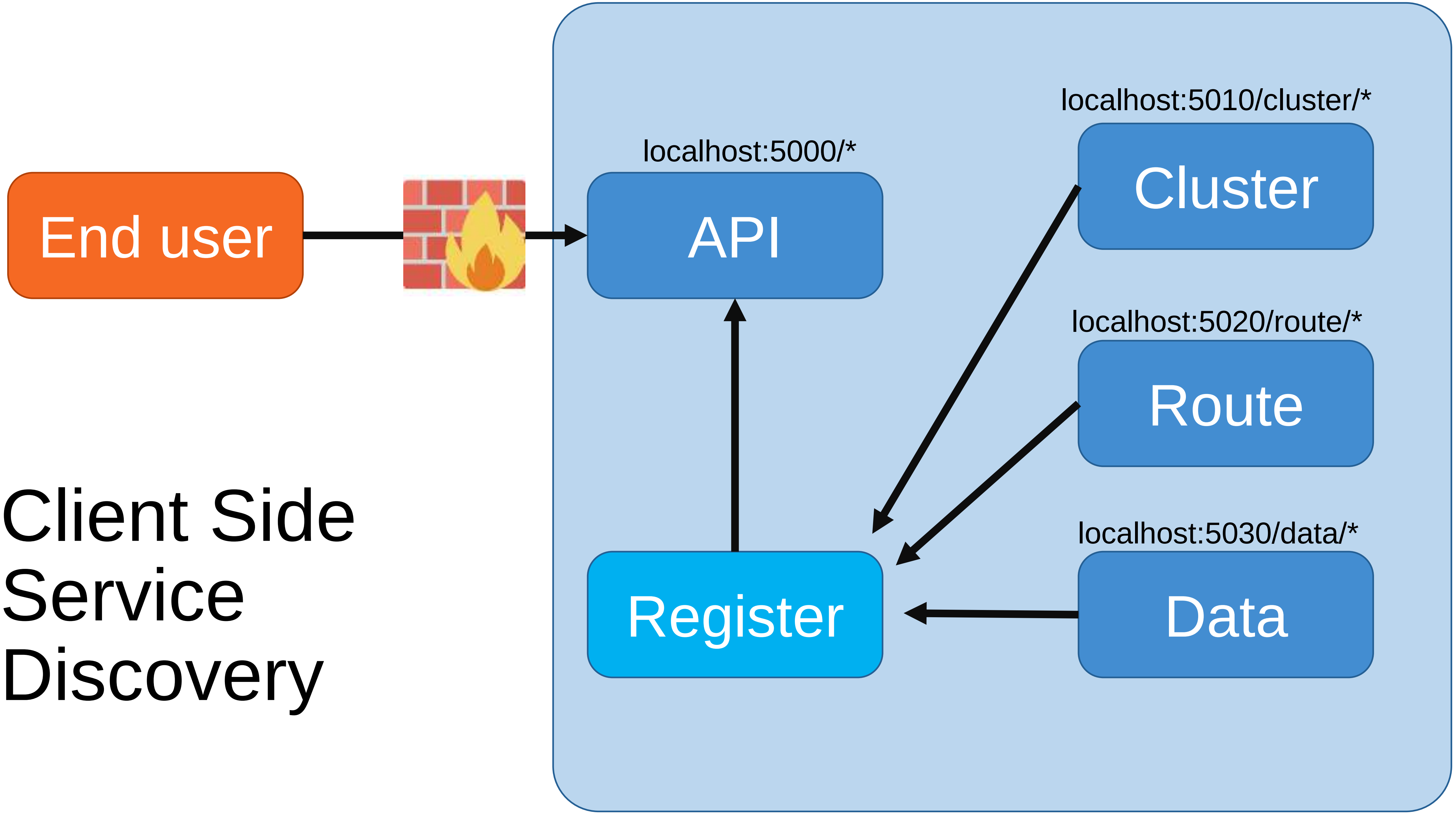
```
tbr09@DESKTOP-P19USKD MINGW64 /d/git/TrashRouting (master)
$ curl http://localhost:8500/v1/kv/db/routes
  % Total    % Received % Xferd  Average Speed   Time    Time
                                 Dload  Upload   Total   Spent
100   230  100   230    0     0  14375      0 --:--:-- --:--:-
    {
        "LockIndex": 0,
        "Key": "db/routes",
        "Flags": 0,
        "Value": "ewoiY29ubmVjdGlvblN0cmluZyIgOiAicm91dGVzQ29u
        "CreateIndex": 86,
        "ModifyIndex": 86
    }
tbr09@DESKTOP-P19USKD MINGW64 /d/git/TrashRouting (master)
$ echo $keyValue | base64 --decode
{
"connectionString" : "routesConnectionString"
}
```

Access Control List

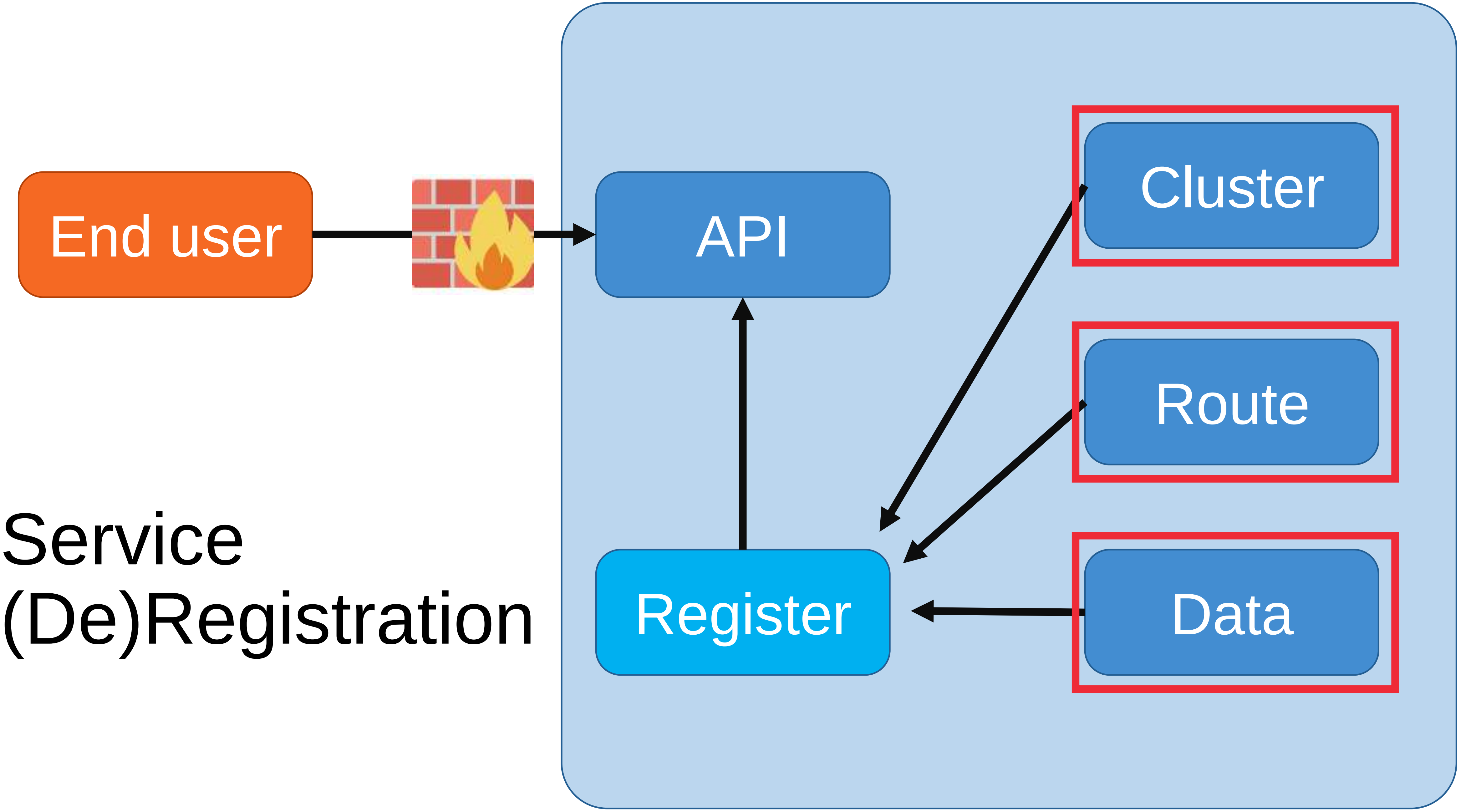
Access tokens for:

- Agents
- Services
- Consul KV
- Consul UI


Microservice Chassis
Externalized configuration
Cross-cutting concerns
Access Token
Security
Messaging
Remote Procedure Invocation
Domain-specific
Communication style
Circuit Breaker
Reliability
Client-side discovery
Self registration
Service registry
Server-side discovery
3rd party registration
Discovery
API gateway
Backends for frontends
External API
Communication patterns
Infrastructure patterns
Multiple Services per host
Single Service per Host
Serverless deployment
Service-per-Container
Service-per-VM
Sidecar
Service deployment platform
Service mesh
Deployment
Microservice patterns



Client Side
Service
Discovery



Service (De)Registration

Registration options

- Self registration
- Third-party registration

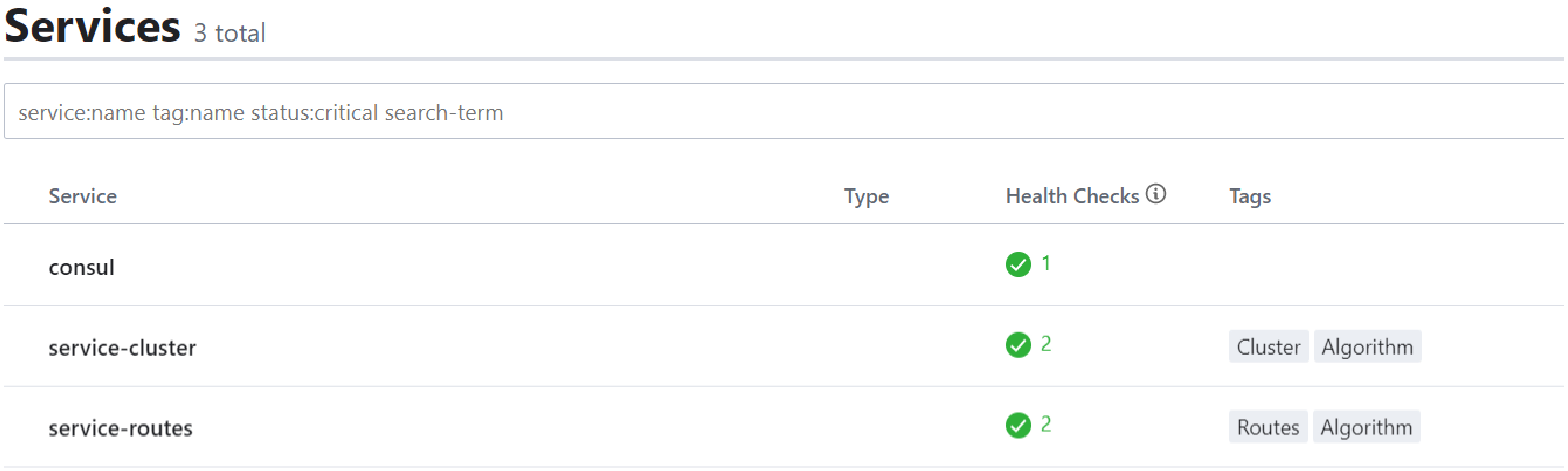
Consul service registration - startup.cs

```csharp
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
IApplicationLifetime lifetime)
{
    var address = Configuration["Consul:ServiceAddress"];
    var servicePort = Configuration["Consul:ServicePort"];
    var serviceName = Configuration["Consul:ServiceName"];

    var registration = new AgentServiceRegistration()
    {
        ID = $"{service}-{servicePort}",
        Name = service,
        Address = address,
        Port = Int32.Parse(servicePort)
    };
    consulClient.Agent.ServiceRegister(registration).Wait();
}
```

Consul service deregistration - startup.cs

```
public void Configure(IApplicationBuilder app, IHostingEnvironment
env, IApplicationLifetime lifetime)
{
    // Registration area

    lifetime.ApplicationStopping.Register(() =>
    {
        consulClient.Agent.ServiceDeregister(registration.ID).Wait();
    });
}
```

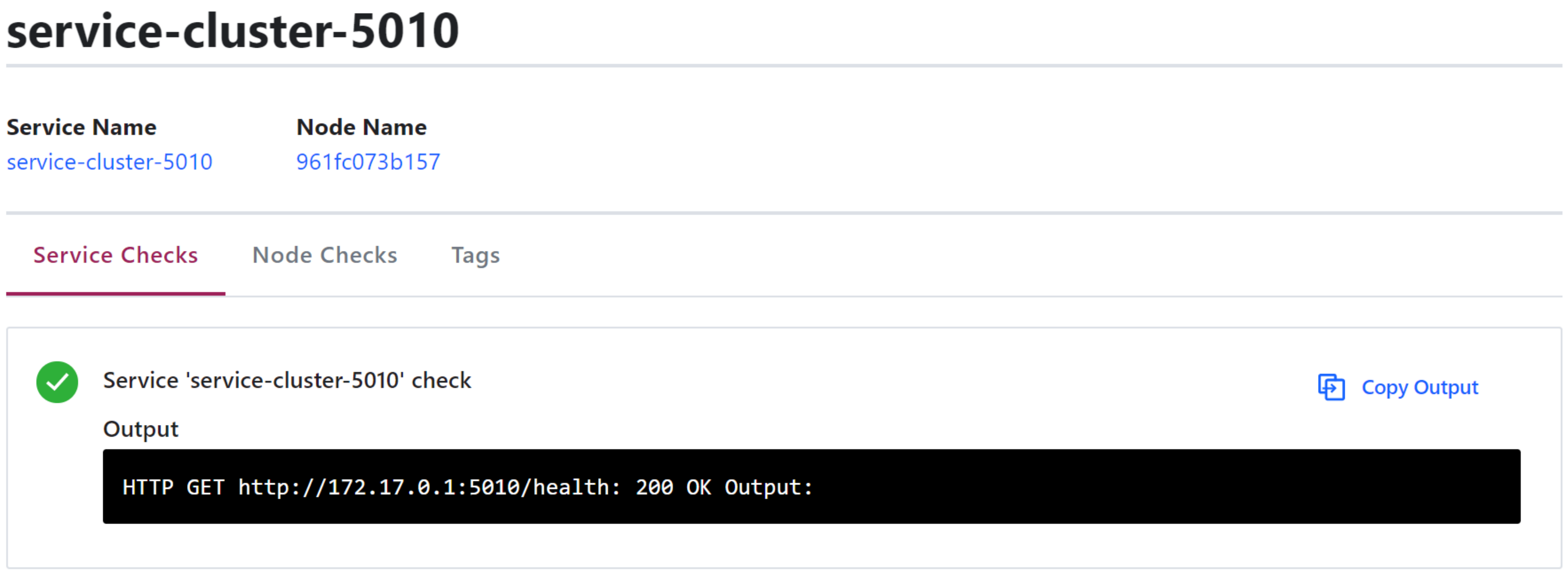
Health checks



Services 3 total

service:name tag:name status:critical search-term

Service	Type	Health Checks	Tags
consul		✓ 1	
service-cluster		✓ 2	Cluster Algorithm
service-routes		✓ 2	Routes Algorithm



service-cluster-5010

Service Name
service-cluster-5010

Node Name
961fc073b157

Service 'service-cluster-5010' check

Output

```
HTTP GET http://172.17.0.1:5010/health: 200 OK Output:
```

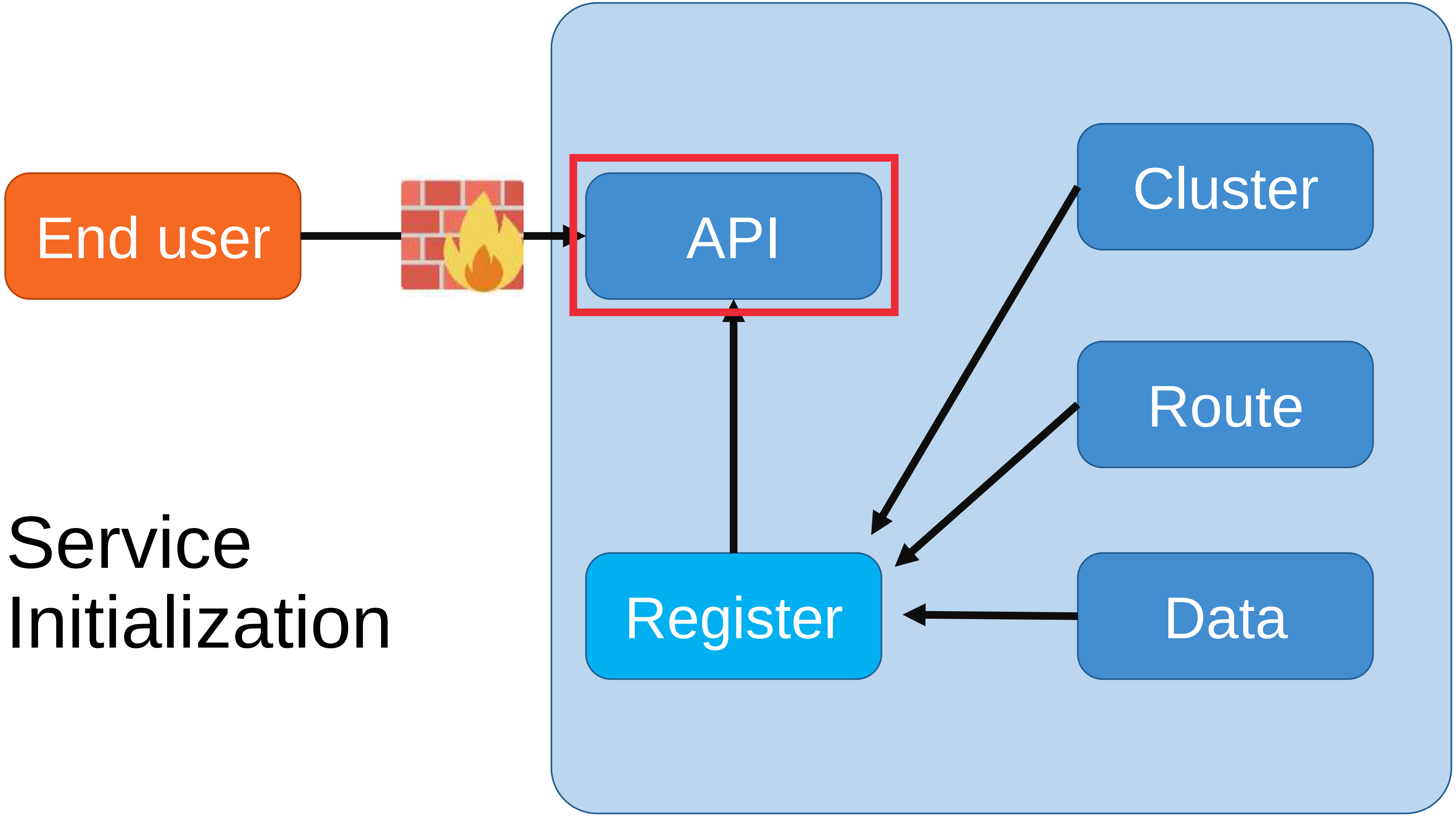
Health checks

```
[Route("[controller]")]
public class HealthController : ControllerBase
{
    [HttpGet]
    public IActionResult Index()
    {
        // Check whatever you want
        return new OkResult();
    }
}
```

Route
Service

Health checks

```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
IApplicationLifetime lifetime)
{
    // Registration area
    var pingEndpoint = Configuration["Consul:PingEndpoint"];

    var healthCheck = new AgentServiceCheck
    {
        Interval = TimeSpan.FromSeconds(10.0),
        DeregisterCriticalServiceAfter = TimeSpan.FromSeconds(30.0),
        HTTP = $"http://{address}:{servicePort}/{pingEndpoint}"
    };
    registration.Checks = new[] {pingEndpoint };
   // Deregistration area
}
```



Service Initialization

Service Initialization – HARDCODED ;_;

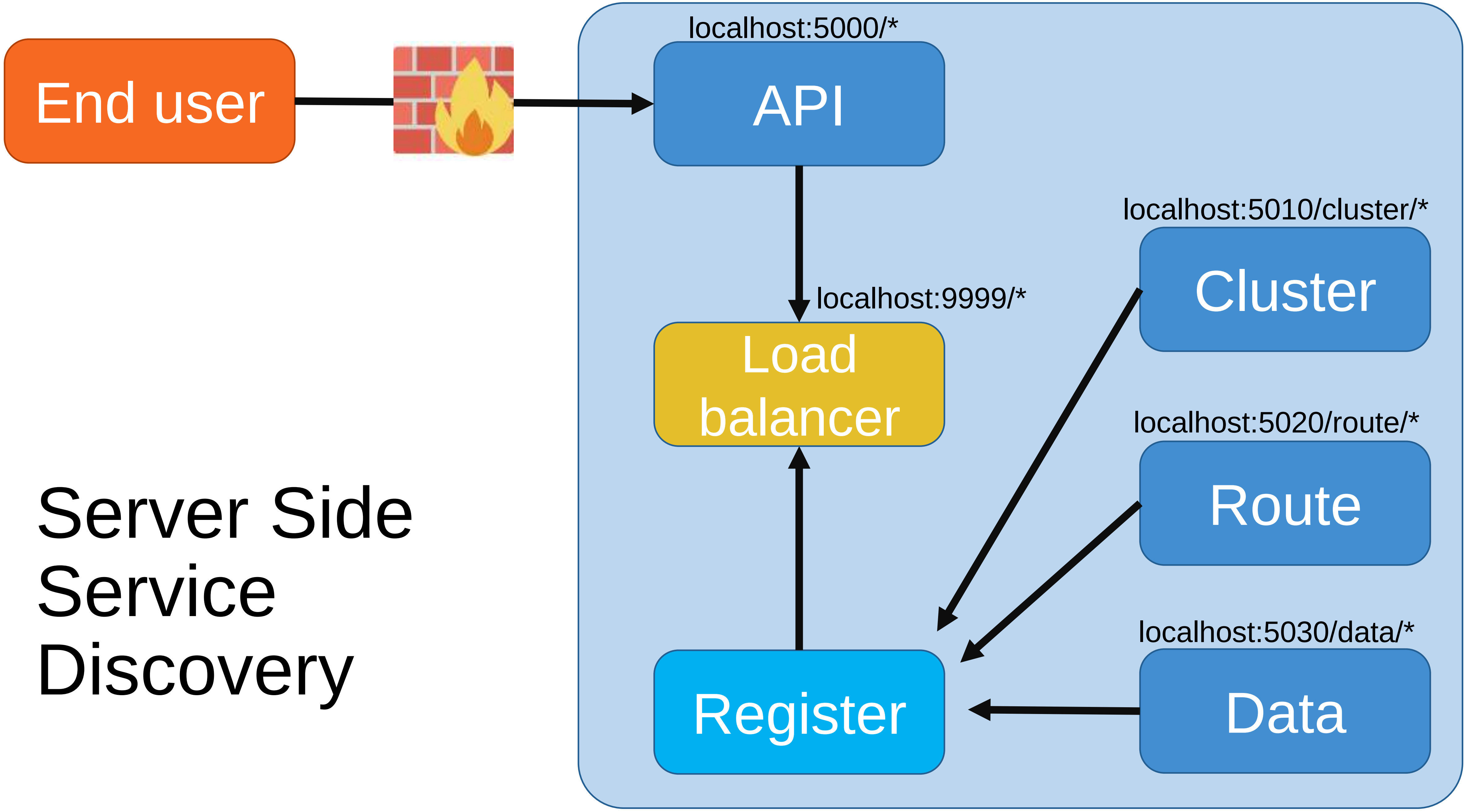
```
[Route("[controller]")]
[ApiController]
public class RouteController : ControllerBase
{
   private readonly IRouteService routeService;

   public RouteController()
   {
      routeService =

      RestClient.For<IRouteService>("http://localhost:5002");
   }
}
```

Service Initialization - appsettings.json

```
[Route("[controller]")]
public class RouteController : ControllerBase
{
   private readonly IRouteService routeService;

   public RouteController()
   {
      routeService = RestClient.For<IRouteService>(
            configuration["Services:Route:Address"]);
   }
}
```

Service Initialization - Consul

```
[Route("[controller]")]
public class RouteController : ControllerBase
{
   private readonly IRouteService routeService;

   public RouteController(IConsulClient consulClient)
   {
      var query = consulClient.Catalog.Service("service-routes")
            .GetAwaiter().GetResult();
      var serviceInstance = query.Response.First();
      routeService = RestClient.For<IRouteService>
      ($"{serviceInstance.ServiceAddress}:{serviceInstance.ServicePort}");
   }
}
```

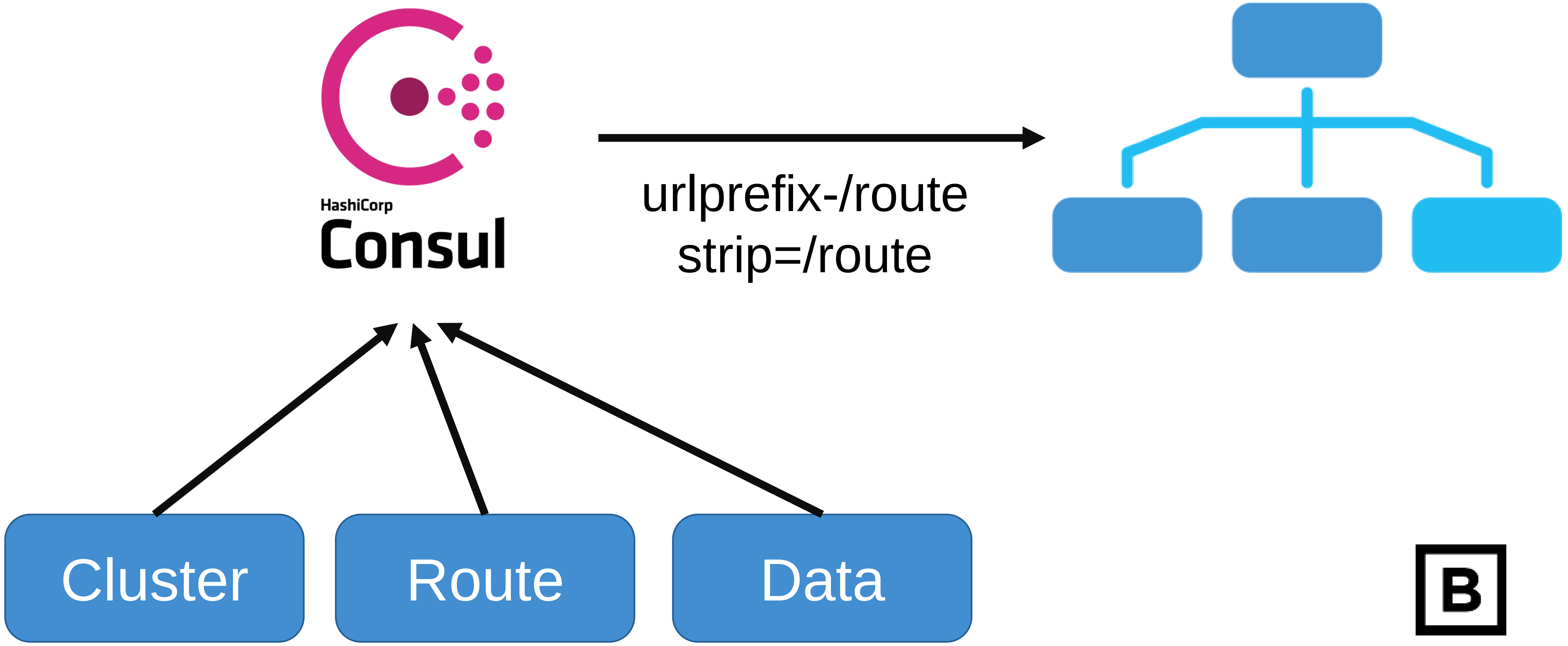


Server Side Service Discovery

Load balancer

- Recommended in Consul docs

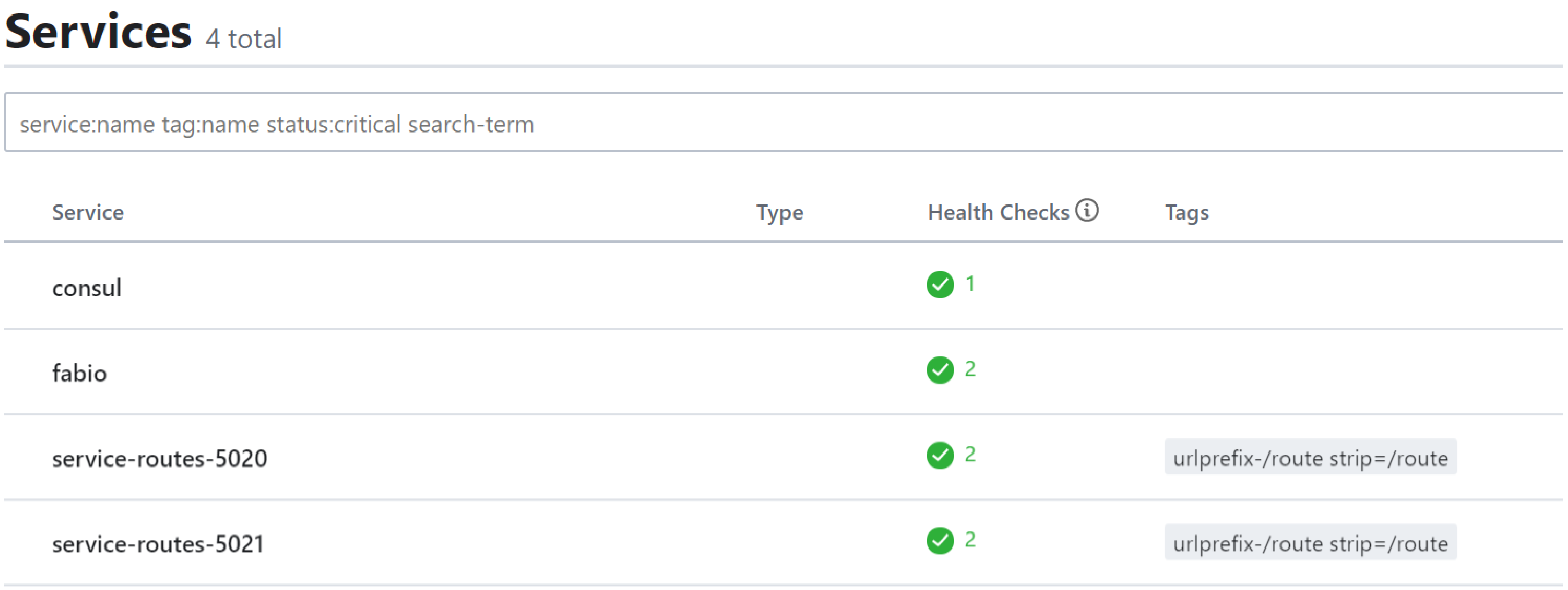
Fabio


HashiCorp
Consul
urlprefix-/route
strip=/route
Cluster
Route
Data

Tag consul service

```
var serviceName = Configuration["Fabio:ServiceName"];

var registration = new AgentServiceRegistration()
{
   ID = $"{Configuration["Consul:ServiceID"]}-{servicePort}",
   Name = serviceName,
   Address = address,
   Port = Int32.Parse(servicePort),
   Tags = $"urlprefix-/{serviceName} strip=/{serviceName}"

};
```

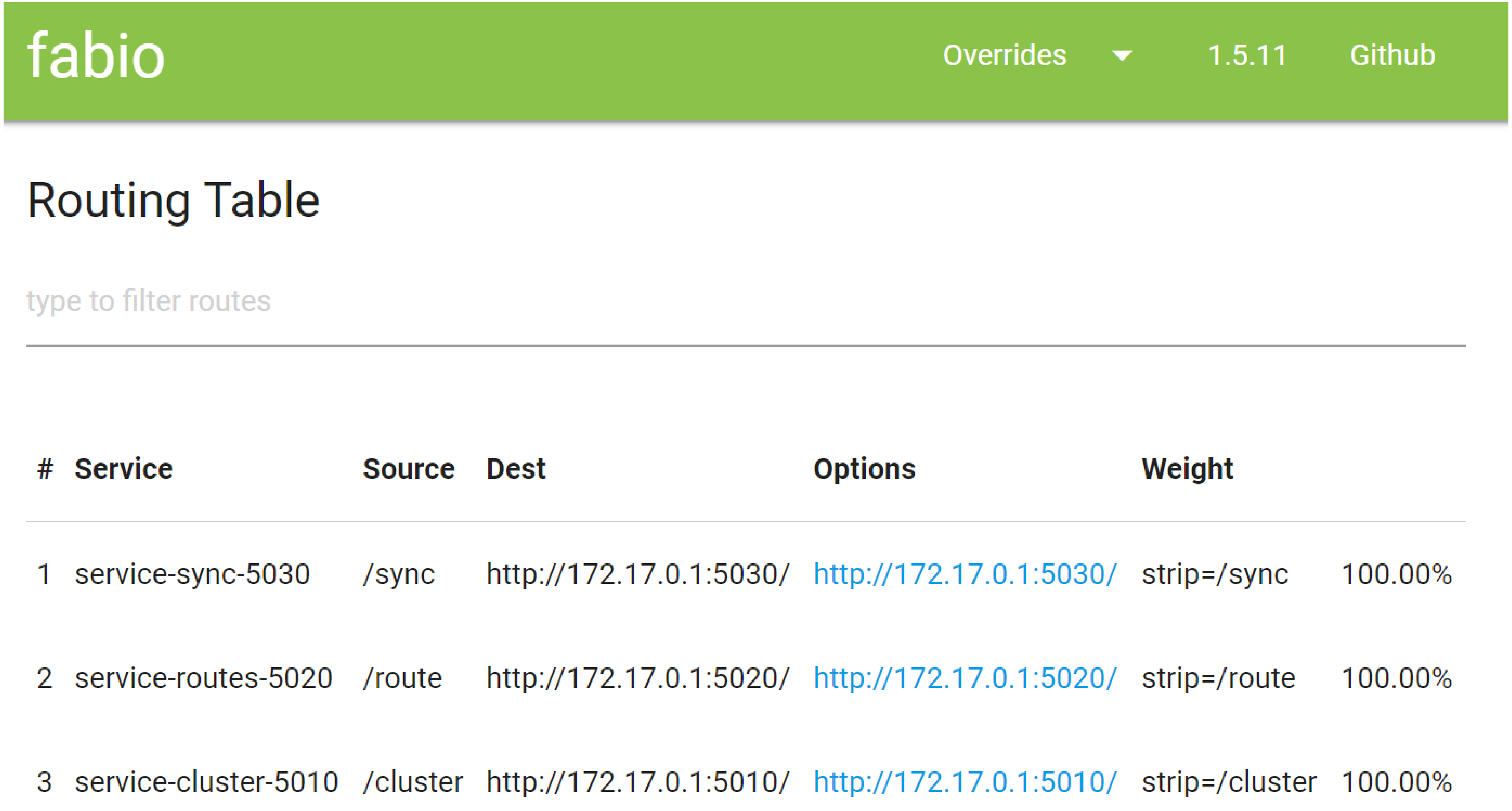
urlprefix-/route strip=/route



Services 4 total

service:name tag:name status:critical search-term

Service	Type	Health Checks	Tags
consul		✓ 1	
fabio		✓ 2	
service-routes-5020		✓ 2	urlprefix-/route strip=/route
service-routes-5021		✓ 2	urlprefix-/route strip=/route

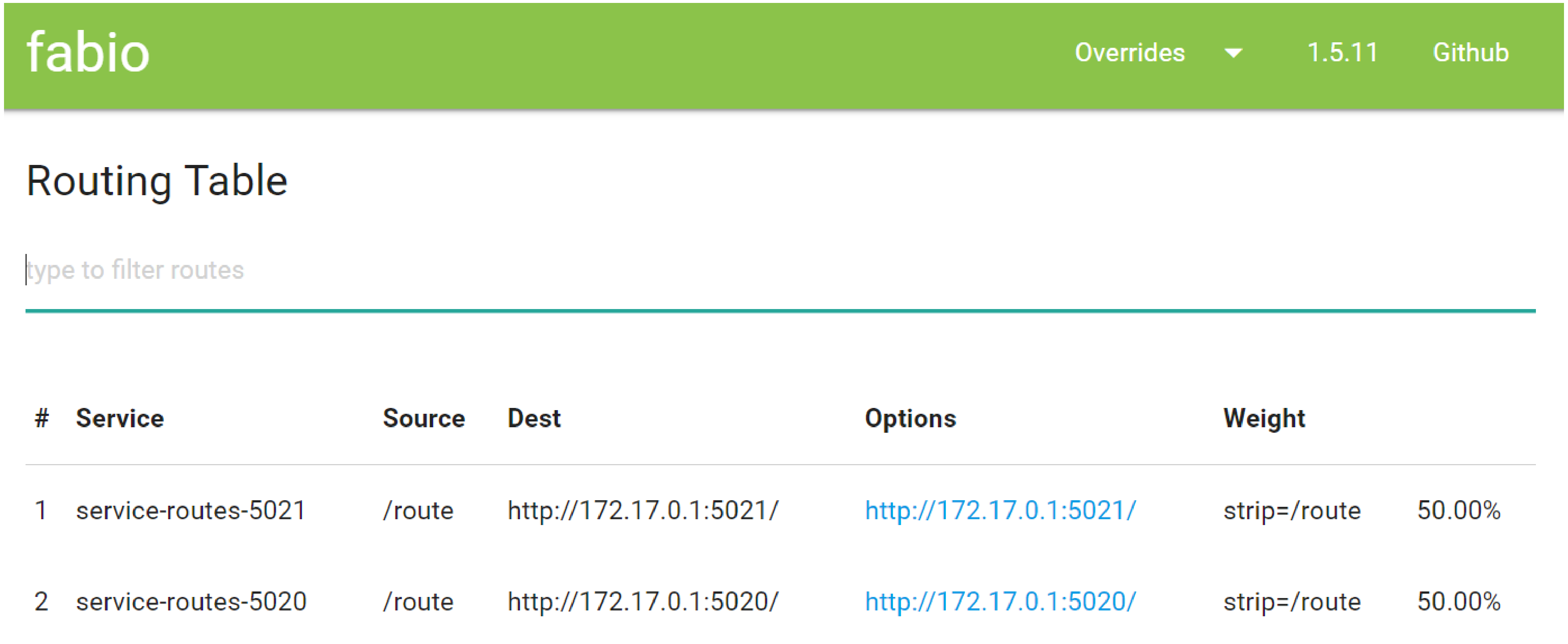


fabio Overrides 1.5.11 Github

Routing Table

type to filter routes

#	Service	Source	Dest	Options	Weight	
1	service-sync-5030	/sync	http://172.17.0.1:5030/	http://172.17.0.1:5030/	strip=/sync	100.00%
2	service-routes-5020	/route	http://172.17.0.1:5020/	http://172.17.0.1:5020/	strip=/route	100.00%
3	service-cluster-5010	/cluster	http://172.17.0.1:5010/	http://172.17.0.1:5010/	strip=/cluster	100.00%



fabio Overrides 1.5.11 Github

Routing Table

type to filter routes

#	Service	Source	Dest	Options	Weight	
1	service-routes-5021	/route	http://172.17.0.1:5021/	http://172.17.0.1:5021/	strip=/route	50.00%
2	service-routes-5020	/route	http://172.17.0.1:5020/	http://172.17.0.1:5020/	strip=/route	50.00%

Fabio

Client Side SD

Fewer network parts

Client must deal with discovery

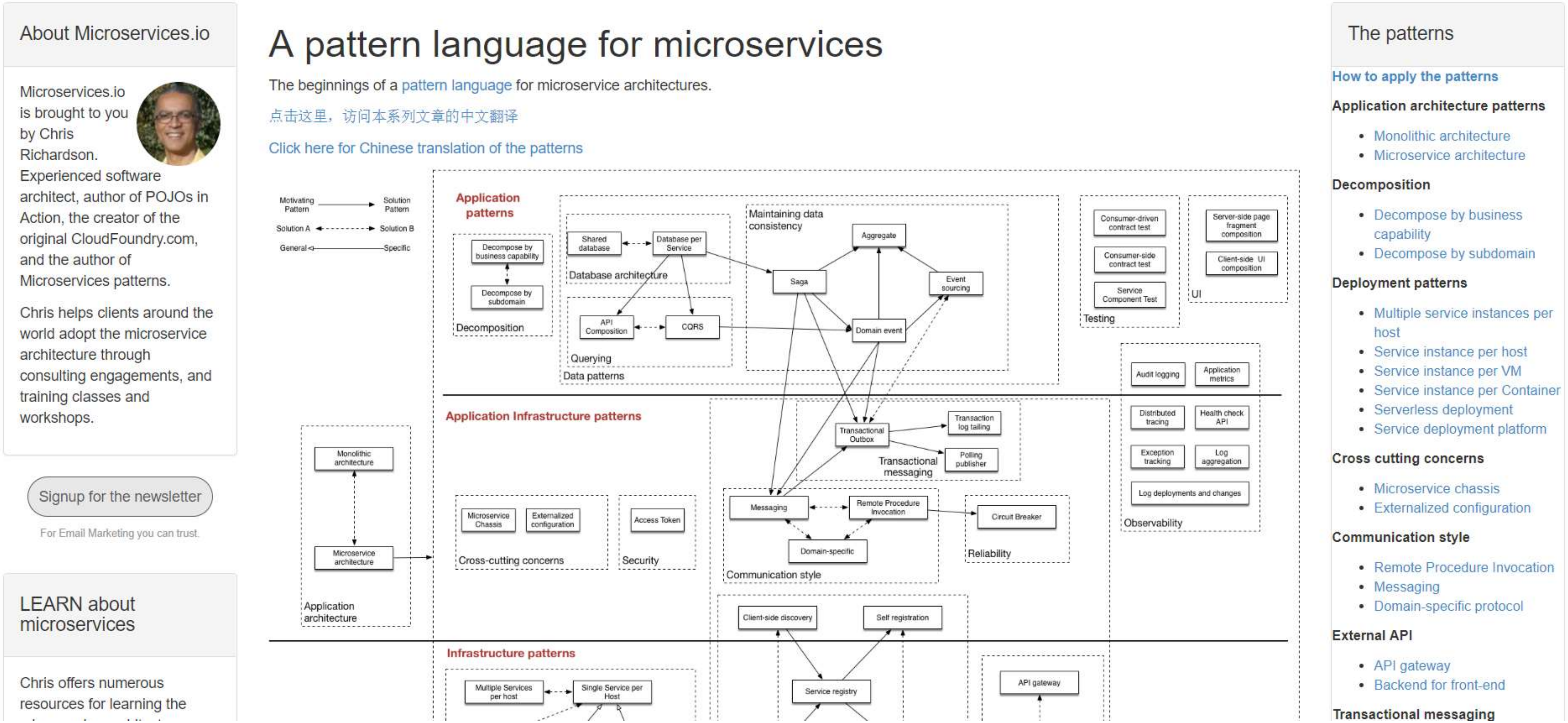
Client uses a load-balancing algorithm

Server Side SD

More traffic control

Client doesn’t have to deal with discovery

Another component to setup and manage



About Microservices.io

Microservices.io is brought to you by Chris Richardson. Experienced software architect, author of POJOs in Action, the creator of the original CloudFoundry.com, and the author of Microservices patterns.

Chris helps clients around the world adopt the microservice architecture through consulting engagements, and training classes and workshops.

Signup for the newsletter

For Email Marketing you can trust.

LEARN about microservices

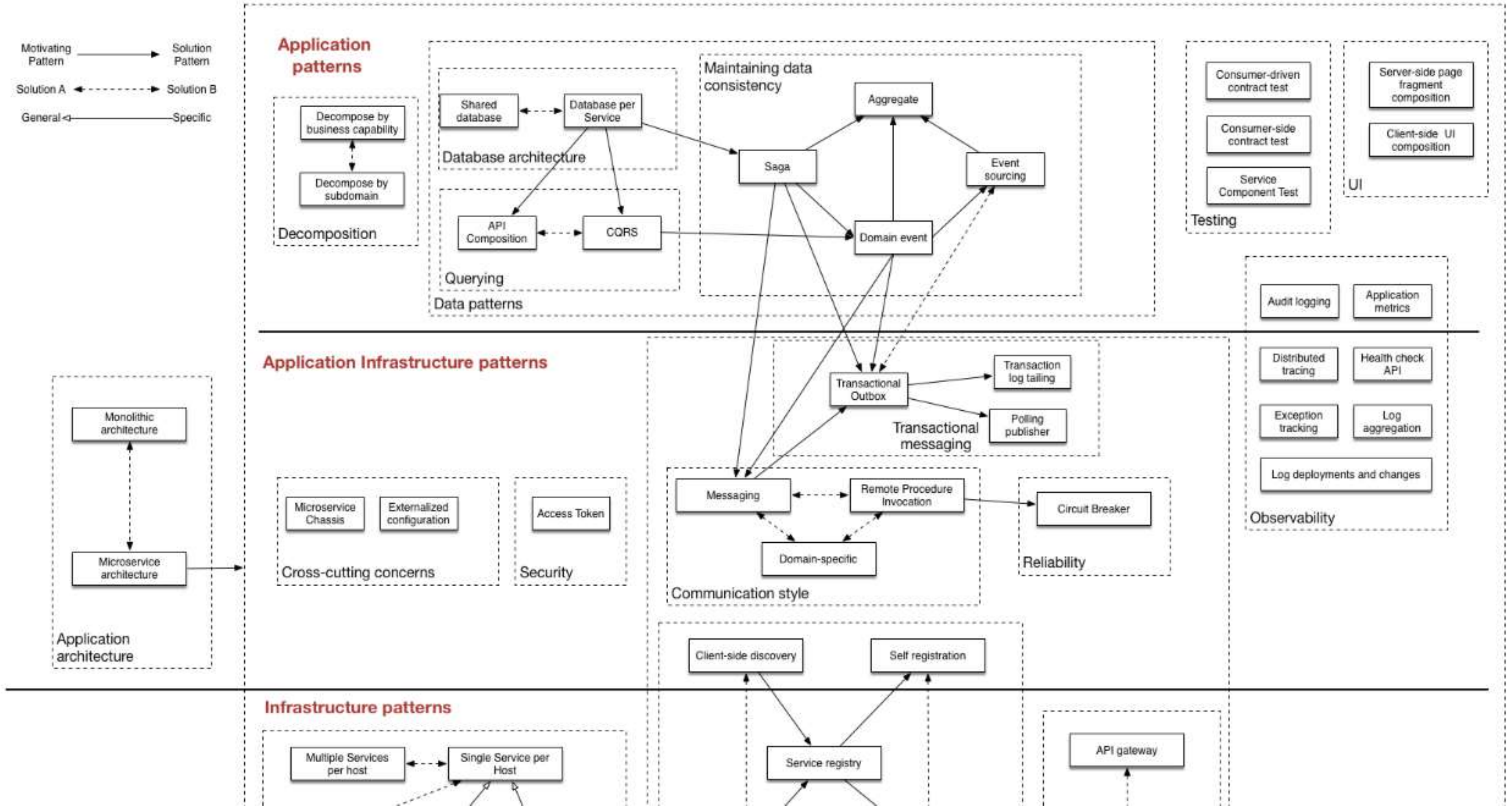
Chris offers numerous resources for learning the

A pattern language for microservices

The beginnings of a pattern language for microservice architectures.

点击这里，访问本系列文章的中文翻译

Click here for Chinese translation of the patterns



The patterns

How to apply the patterns

Application architecture patterns

- Monolithic architecture
- Microservice architecture

Decomposition

- Decompose by business capability
- Decompose by subdomain

Deployment patterns

- Multiple service instances per host
- Service instance per host
- Service instance per VM
- Service instance per Container
- Serverless deployment
- Service deployment platform

Cross cutting concerns

- Microservice chassis
- Externalized configuration

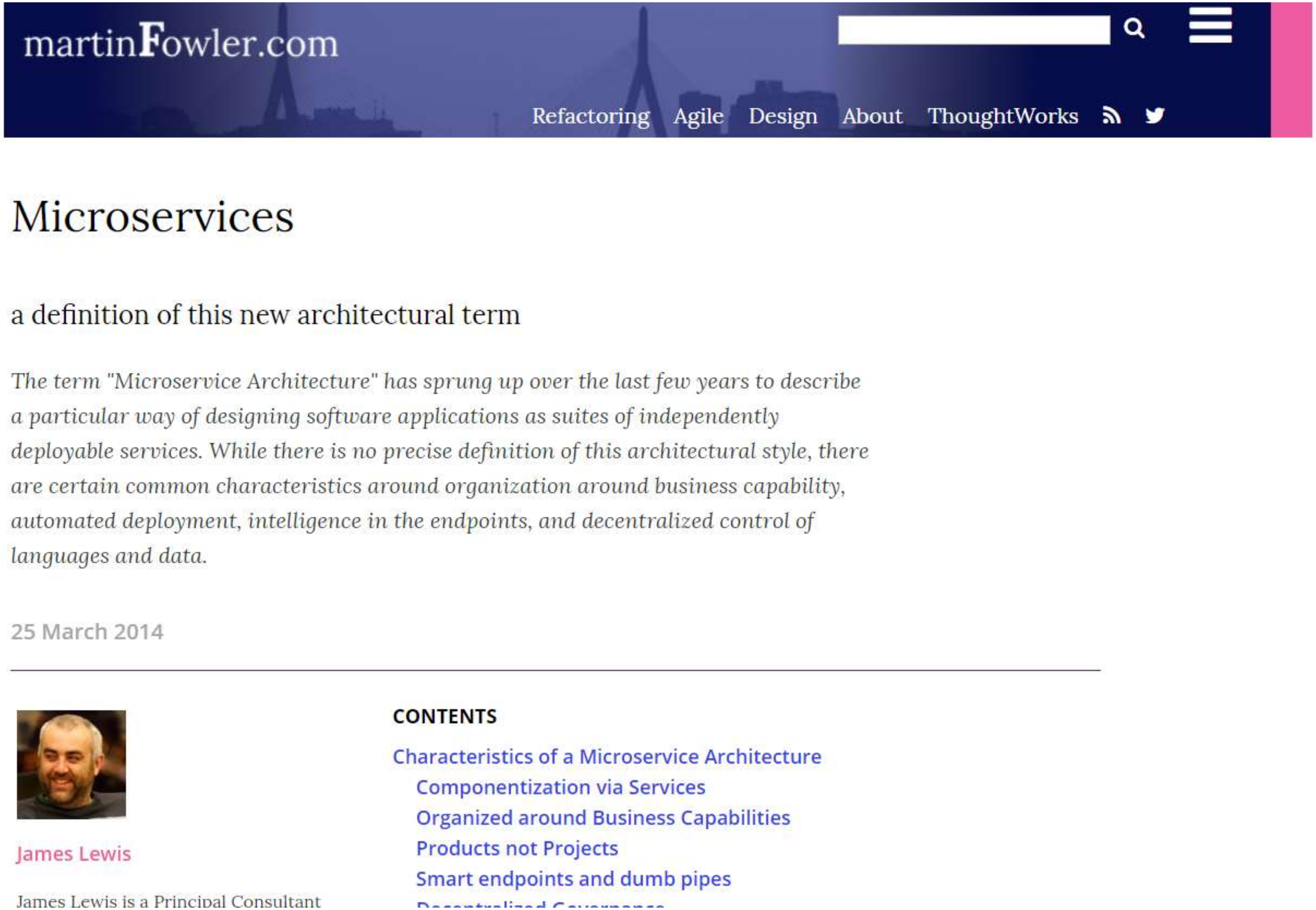
Communication style

- Remote Procedure Invocation
- Messaging
- Domain-specific protocol

External API

- API gateway
- Backend for front-end

Transactional messaging



Microservices

a definition of this new architectural term

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

25 March 2014

James Lewis

James Lewis is a Principal Consultant

CONTENTS

HashiCorp
Consul

Microsoft
Azure

Links

- https://devmentors.io/distributed-net-core/
- https://microservices.io/patterns/index.html
- https://www.consul.io/docs/index.html
- https://github.com/canton7/RestEase
- https://github.com/tbr09/TrashRouting

Questions?

Thanks for your attention ☺

@mtyborowski09 mtyborowski09@gmail.com @tbr09