YARP
Yet Another Reverse Proxy

Christian Nagel

https://www.cninnovation.com

Christian Nagel

- Training
- Coaching
- Consulting
- Development

- Microsoft MVP
- www.cninnovation.com
- csharp.christiannagel.com
- @christiannagel

This hour...

- What's a reverse proxy? Why YARP?
- How to use YARP
- YARP with .NET Aspire

Let’s start with foundations

A forward proxy

- Acts as an intermediate for client machines
- Bypass restrictions
- Content filtering
- Anonymity for the client

A reverse proxy

- Sits in front of web servers
- Intercepts requests from clients
- Gateway between clients and backend servers

Functionality of a reverse proxy

- Security
- Load balancing
- Performance optimization
- Content delivery
- URL rewriting
- Protocol changes
- High availability

Load balancer .vs. reverse proxy

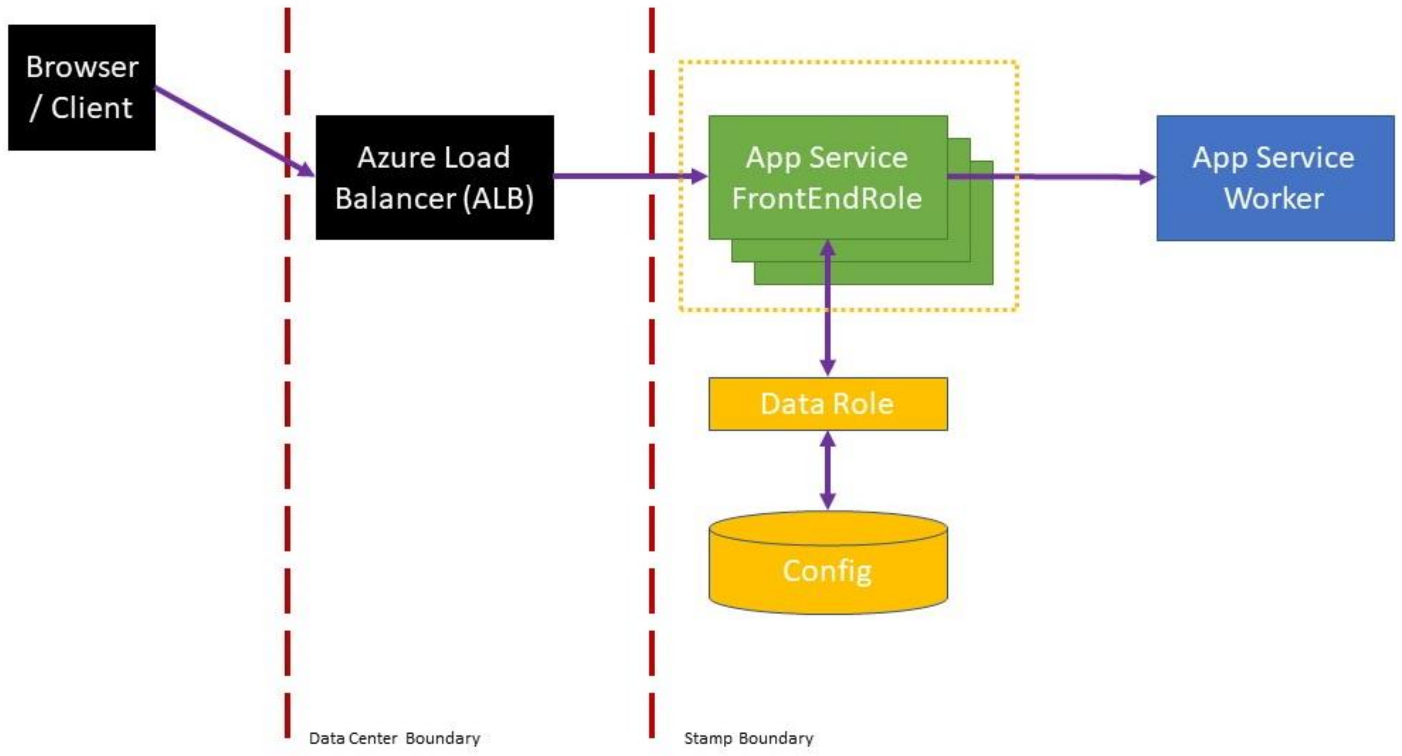
- Load balancer
  - Network layer 4
- Reverse proxy
  - Network layer 7

What's **Kestrel**?

- Web server implemented with .NET Core
- Lightweight
- HTTP/1.1, HTTP/2, HTTP/3, WebSockets
- Integration with ASP.NET Core

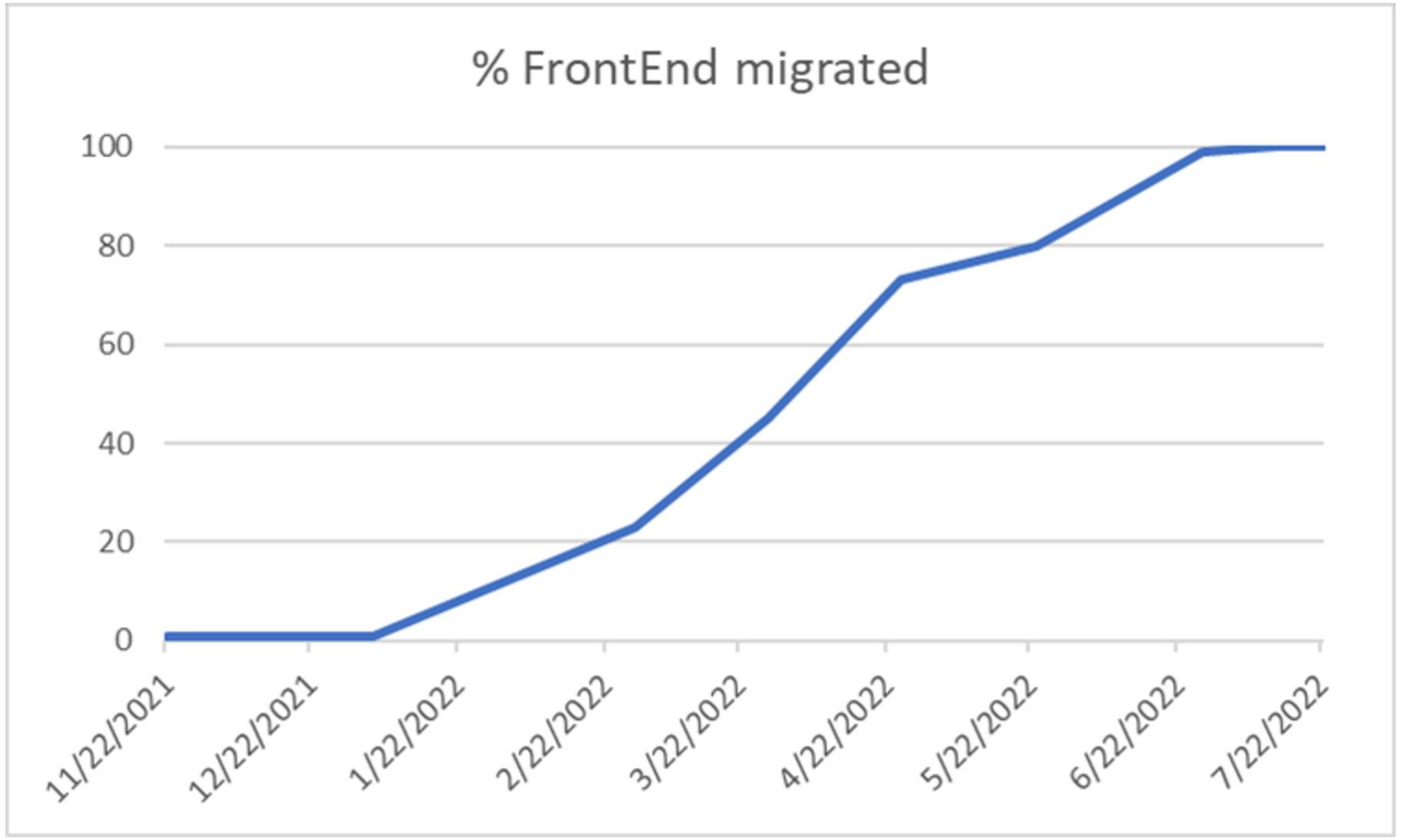
Success story: Azure App Services

- App Service FrontEndRole
    - IIS running on HTTP.sys
    - Application Request Routing (ARR) with WinHTTP



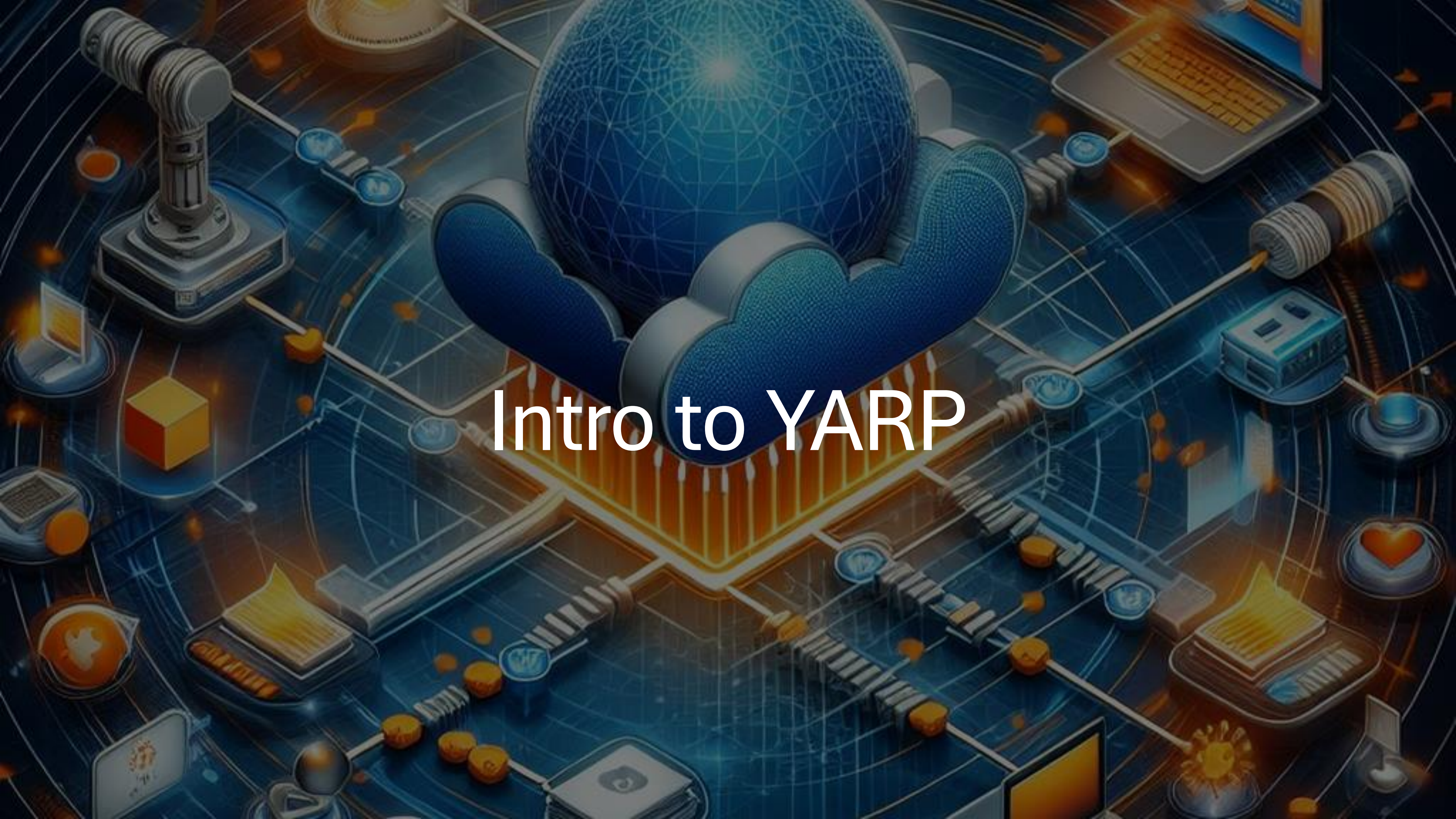
Migration

- 100% FrontEndRoles
    - With Kestrel & YARP



Changes

- Performance
  - 80% throughput improvements
- Linux worker VMs use Kestrel & YARP instead of nginx
- gRPC support



Intro to YARP

Getting started…

- NuGet Package
  - Yarp.ReverseProxy
- DI Container
  - AddReverseProxy
- Middleware
  - MapReverseProxy

Code...

```
IReadOnlyList<RouteConfig> routes =
[
  new()
  {
    RouteId ="route1",
    ClusterId = "cluster1",
    Match = new RouteMatch()
    {
      Path= "{**catch-all}"
    }
  }
];
```

```
IReadOnlyList<ClusterConfig> clusters =
[
  new()
  {
    ClusterId = "cluster1",
    Destinations =
      new Dictionary<string, DestinationConfig>()
      {
        { "first", new DestinationConfig
          {
            Address = http://localhost:5295
          }
        }
      }
    }
  }
];
```

…or configuration

```json
"ReverseProxy": {
  "Routes": {
    "route1": {
      "ClusterId": "cluster1",
      "Match": {
        "Path": "{**catch-all}"
      }
    }
  },
  "Clusters": {
    "cluster1": {
      "Destinations": {
        "destination1": {
          "Address": "http://localhost:5295"
        }
      }
    }
  }
}
```

Routes

- RouteId
  - A unique name
- ClusterId
  - Reference a cluster
- Match
  - Host array
  - Path pattern string
  - Header, authorization, CORS can be configured with every route entry

Transforms

- Modify
  - Request
  - Response
  - Response trailers (request trailers not supported by *HttpClient*)
- Default Headers from the proxy to the backend
  - *Host*, *X-Forwarded-For*, *X-Forwarded-Proto*, *X-Forwarded-Host*

Proxy Middleware

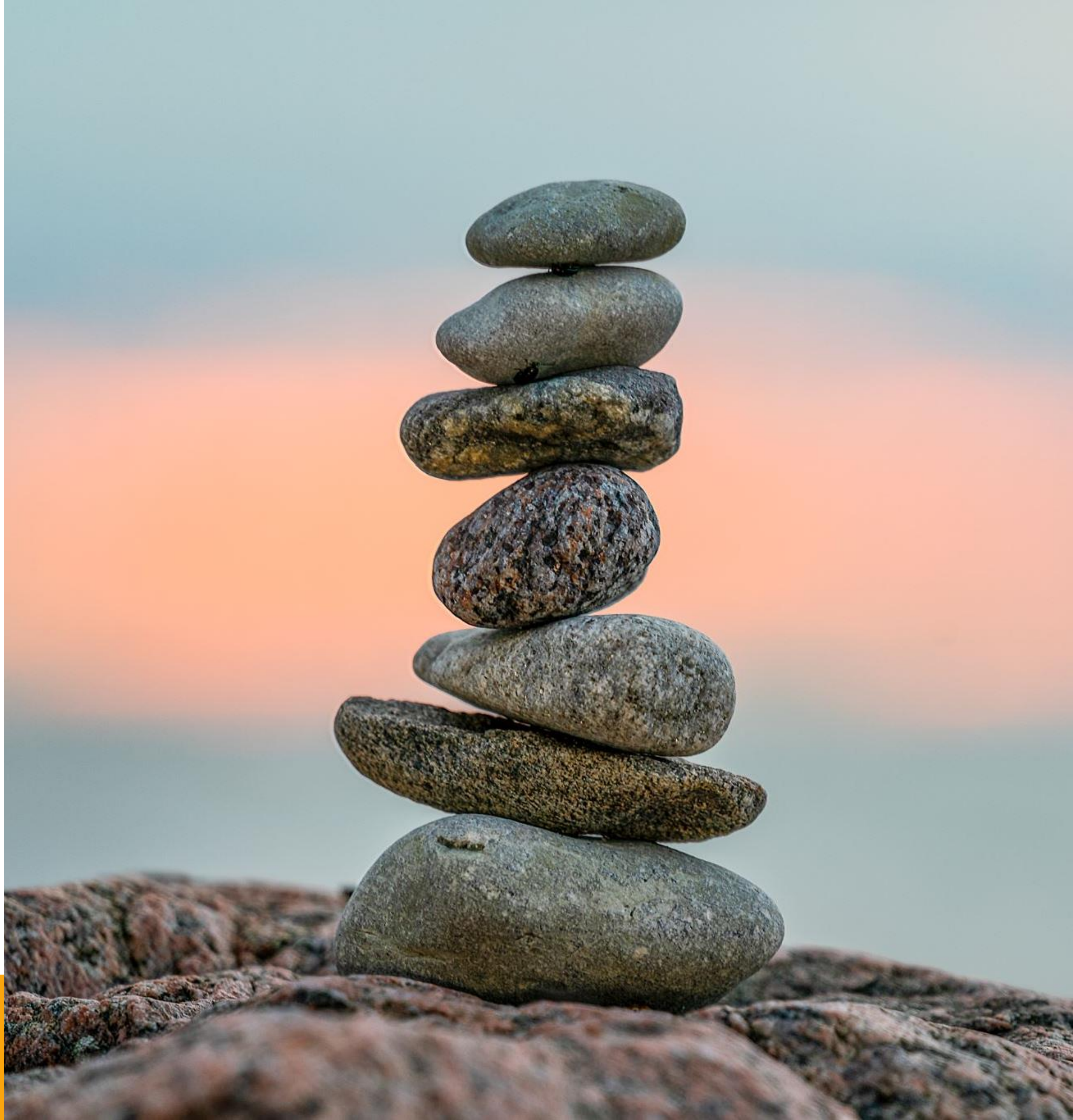
- *MapReverseProxy*
  - Sets up middleware with routing and proxy configured endpoints
  - Add middleware not configured by default, add custom middleware
- Use for
  - Logging
  - Send immediate response
  - Session affinity
  - Load balancing
  - Filter destinations
  - Error handling
- Attention!
  - Middleware must not do multi-threaded work on individual requests
  - *HttpContext* and its methods are not thread-safe!

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

Load balancing

Multiple healthy destinations available? Use load-balancing algorithm.

- FirstAlphabetical
- Random
- PowerOfTwoChoices
- RoundRobin
- LeastRequests (overhead)



Destination health checks

- Active health checks
    - Sending periodic probing requests
    - *IActiveHealthCheckPolicy* – analyzes how destinations responded
    - *IProbingRequestFactory* – creates active health probing requests
- Passive health checks
    - Watch for success and failures in client requests
    - Health policy runs after response is sent
    - Unhealthy: stops receiving all new traffic, reactivation after configured period (unhealthy to unknown)
    - *IPassiveHealthCheckPolicy – analyzes how destinations responded*

Rate limiting

- .NET 7+
- Can be specified per route or globally
- Add rate limiter middleware – *app.UseRateLimiter()*
- Based on System.Threading.RateLimiting
- Algorithms
  - Fixed window
  - Sliding window
  - Token bucket
  - Concurrency

.NET Aspire

What is .NET Aspire?

".NET Aspire is an opinionated, cloud ready stack for building observable, production ready, distributed applications."

https://learn.microsoft.com/en-us/dotnet/aspire/get-started/aspire-overview

.NET Aspire

- YARP is used within .NET Aspire
- Let’s see…

Summary

- .NET Reverse Proxy
- Used with many Microsoft technologies
- Use it in your solutions on-premises or in the cloud

Thank you!

Questions?

- https://github.com/cnilearn/bastaspring2024
- https://csharp.christiannagel.com
- https://www.cninnovation.com