Tratamento de falhas com Polly - .NET 6

*"Se na primeira tentativa você não conseguir, tente de novo, e de novo, e de novo..."*

Tratamento de falhas transitórias : tratamento de falhas de transição

O que vamos cobrir?

* O que são falhas transitórias?
* As políticas que podem ser usadas para manipular falhas transitórias
* Como podemos usar Polly para implementar essas políticas (em .NET)

### Ingredientes

* .NET 6 SDK (free)
* VSCode (free)
* API Client, e.g.: Insomnia ou Postman (free)

**Transient:** transitório

*adjetivo*

Durar apenas por um curto período de tempo

* Impermanente

### O que são falhas transitórias?

Transients faults relate to fault occurrences that exist for short periods of time, example are:

As falhas transitórias referem-se a ocorrências de falhas que existem por curtos períodos de tempo, exemplos são:

* Uma conexão de rede não está disponível durante a reinicialização de um roteador
* Microservice starting up
* Inicialização do microsserviço
* Servidor recusando conexões devido ao esgotamento do pool de conexões

### Por que nos importamos?

* Em vez de obter uma resposta de erro e aceitar a falha
* Poderíamos eventualmente ter uma resposta de sucesso

Isso é particularmente vantajoso em arquiteturas de aplicativos distribuídos

## Tratamento de falhas transitórias

Vamos nos concentrar nas variações da política de repetição:

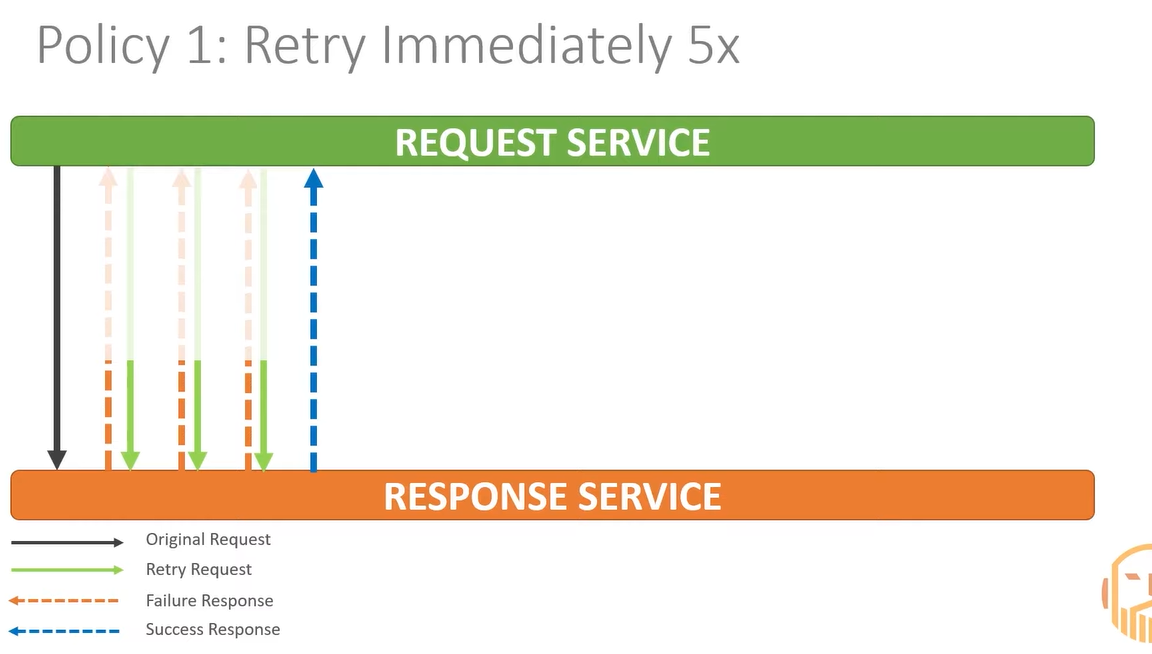
* Tentando a requisição de novo, (e de novo?), pra ver se dá certo dessa vez...

Nós podemos configurar:

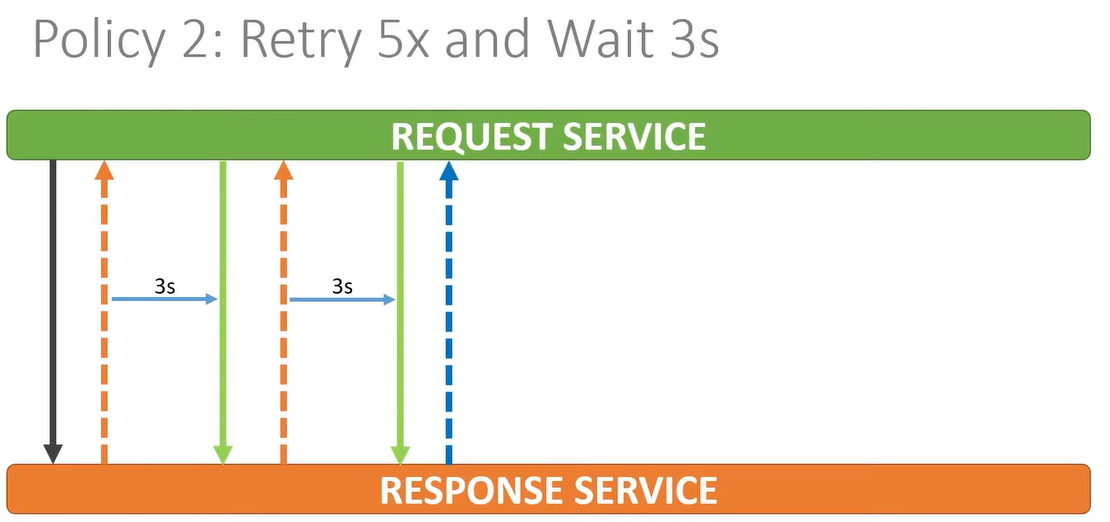
* Número de repetição (provavelmente não queremos tentar para sempre)
* Intervalo de tempo entre as retentativas (constante ou variável)

## Política

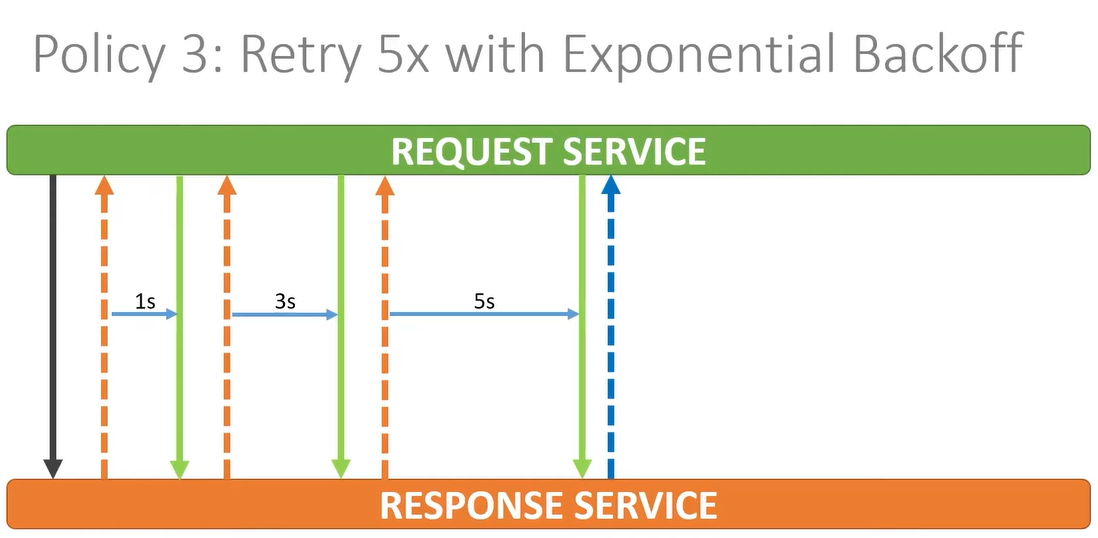
### Policy 1: Retry Immediately



### Policy 2: Retry 5x and Wait 3s



### Policy 3: Retry 5x with Exponential Backoff



## O que é Polly?

* A biblioteca "de fato" de resiliência e tratamento de falhas transitórias para .NET
* Podemos usá-lo para criar Políticas em nossos aplicativos .NET
* Este vídeo é realmente uma rampa de acesso para você usar

- Disjuntor

- Tempo esgotado

Isolamento da Antepara

Mais Informações:<https://github.com/App-VNext/Polly>

# Code

## Response

using Microsoft.AspNetCore.Mvc;

namespace ResponseService.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class ResponseController : ControllerBase

{

// GET /api/response/3

[Route("{id:int}")]

[HttpGet]

public ActionResult GetResponse(int id)

{

Random rnd = new Random();

var rndInteger = rnd.Next(1, 101);

if (rndInteger >= id)

{

System.Console.WriteLine("--> Failure - Generate a HTTP 500");

return StatusCode(StatusCodes.Status500InternalServerError);

}

System.Console.WriteLine("--> Sucess - Generate a HTTP 200");

return Ok();

}

}

}

## Request

using Microsoft.AspNetCore.Mvc;

namespace RequestService.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class RequestController : ControllerBase

{

// GET: api/request

[HttpGet]

public async Task<ActionResult> MakeRequest()

{

var client = new HttpClient();

var response = await client.GetAsync("http://localhost:5002/api/response/25");

if (response.IsSuccessStatusCode)

{

Console.WriteLine("--> ResponseService returned SUCESS");

return Ok();

}

Console.WriteLine("--> ResponseService returned FAILURE");

return StatusCode(StatusCodes.Status500InternalServerError);

}

}

}

## Polly

### Class - Definindo as políticas do ClientPolly em classe

#### Tente 5x imediatamente

using Polly;

using Polly.Retry;

namespace RequestService.Policies

{

public class ClientPolicy

{

public AsyncRetryPolicy<HttpResponseMessage> ImmediateHttpRetry { get; }

public ClientPolicy()

{

ImmediateHttpRetry = Policy.HandleResult<HttpResponseMessage>(

res => !res.IsSuccessStatusCode)

.RetryAsync(5);

}

}

}

#### Tente 5x com intervalo de 3 segundos

using Polly;

using Polly.Retry;

namespace RequestService.Policies

{

public class ClientPolicy

{

public AsyncRetryPolicy<HttpResponseMessage> LinearHttpRetry { get; }

public ClientPolicy()

{

LinearHttpRetry = Policy.HandleResult<HttpResponseMessage>(

res => !res.IsSuccessStatusCode)

.WaitAndRetryAsync(5, retryAttempt => TimeSpan.FromSeconds(3));

}

}

}

#### Tente 5x com intervalo exponencial

public class ClientPolicy

{

public AsyncRetryPolicy<HttpResponseMessage> ExponentialHttpRetry { get; }

public ClientPolicy()

{

ExponentialHttpRetry = Policy.HandleResult<HttpResponseMessage>(

res => !res.IsSuccessStatusCode)

.WaitAndRetryAsync(5, retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt)));

}

}

### Program - Adicionando no Container de Serviço

Registra a classe ClientPolicy para que fique disponível para uso através de Injeção de Dependência

using RequestService.Policies;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddSingleton<ClientPolicy>(new ClientPolicy());

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

### Controller - Uso de Polly através de Injeção de Dependência

#### Injeção via Construtor

namespace RequestService.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class RequestController : ControllerBase

{

private readonly ClientPolicy \_clientPolicy;

public RequestController(ClientPolicy clientPolicy)

{

\_clientPolicy = clientPolicy;

}

// GET: api/request

[HttpGet]

public async Task<ActionResult> MakeRequest()

{

return Ok();

}

}

}

#### Policy Immediate

[HttpGet]

public async Task<ActionResult> MakeRequest()

{

var client = new HttpClient();

var response = await \_clientPolicy.ImmediateHttpRetry.ExecuteAsync(

() => client.GetAsync("http://localhost:5002/api/response/25"));

if (response.IsSuccessStatusCode)

{

Console.WriteLine("--> ResponseService returned SUCCESS");

return Ok();

}

Console.WriteLine("--> ResponseService returned FAILURE");

return StatusCode(StatusCodes.Status500InternalServerError);

}

#### Policy Linear

[HttpGet]

public async Task<ActionResult> MakeRequest()

{

var client = new HttpClient();

var response = await \_clientPolicy.ImmediateHttpRetry.ExecuteAsync(

() => client.GetAsync("http://localhost:5002/api/response/25"));

if (response.IsSuccessStatusCode)

{

Console.WriteLine("--> ResponseService returned SUCCESS");

return Ok();

}

Console.WriteLine("--> ResponseService returned FAILURE");

return StatusCode(StatusCodes.Status500InternalServerError);

}

#### Policy Exponential

public async Task<ActionResult> MakeRequest()

{

var client = new HttpClient();

var response = await \_clientPolicy.ExponentialHttpRetry.ExecuteAsync(

() => client.GetAsync("http://localhost:5002/api/response/25"));

if (response.IsSuccessStatusCode)

{

Console.WriteLine("--> ResponseService returned SUCCESS");

return Ok();

}

Console.WriteLine("--> ResponseService returned FAILURE");

return StatusCode(StatusCodes.Status500InternalServerError);

}

## HttpClient Factory

Program

using RequestService.Policies;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddHttpClient();

builder.Services.AddSingleton<ClientPolicy>(new ClientPolicy());

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

### Controller

using Microsoft.AspNetCore.Mvc;

using RequestService.Policies;

namespace RequestService.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class RequestController : ControllerBase

{

private readonly ClientPolicy \_clientPolicy;

private readonly IHttpClientFactory \_clientFactory;

public RequestController(ClientPolicy clientPolicy, IHttpClientFactory clientFactory)

{

\_clientPolicy = clientPolicy;

\_clientFactory = clientFactory;

}

// GET: api/request

[HttpGet]

public async Task<ActionResult> MakeRequest()

{

var client = \_clientFactory.CreateClient();

var response = await \_clientPolicy.ImmediateHttpRetry.ExecuteAsync(

() => client.GetAsync("http://localhost:5002/api/response/25"));

if (response.IsSuccessStatusCode)

{

Console.WriteLine("--> ResponseService returned SUCCESS");

return Ok();

}

Console.WriteLine("--> ResponseService returned FAILURE");

return StatusCode(StatusCodes.Status500InternalServerError);

}

}

}

### Refatorando HttpFactory

#### Controller

using Microsoft.AspNetCore.Mvc;

namespace RequestService.Controllers

{

[Route("api/[controller]")]

[ApiController]

public class RequestController : ControllerBase

{

private readonly IHttpClientFactory \_clientFactory;

public RequestController(IHttpClientFactory clientFactory)

{

\_clientFactory = clientFactory;

}

// GET: api/request

[HttpGet]

public async Task<ActionResult> MakeRequest()

{

var client = \_clientFactory.CreateClient("Test");

var response = await client.GetAsync("http://localhost:5002/api/response/25");

if (response.IsSuccessStatusCode)

{

Console.WriteLine("--> ResponseService returned SUCCESS");

return Ok();

}

Console.WriteLine("--> ResponseService returned FAILURE");

return StatusCode(StatusCodes.Status500InternalServerError);

}

}

}

#### Program

using RequestService.Policies;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddHttpClient("Test").AddPolicyHandler(

request => request.Method == HttpMethod.Get ? new ClientPolicy().ImmediateHttpRetry : new ClientPolicy().ImmediateHttpRetry);

builder.Services.AddHttpClient();

builder.Services.AddSingleton<ClientPolicy>(new ClientPolicy());

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();