@neekgreen

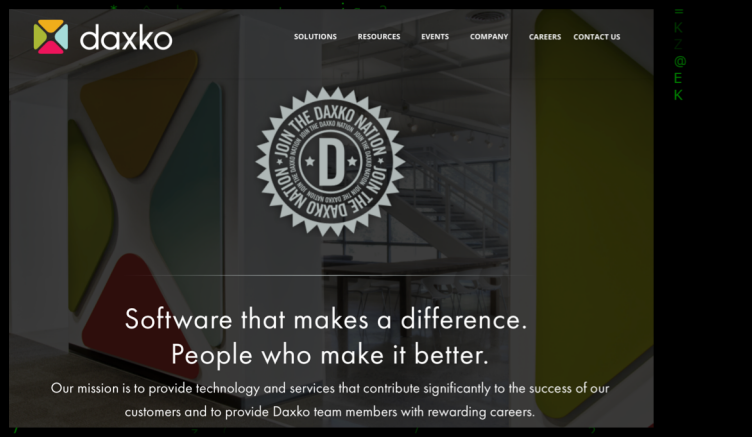
Building Resilient Web APIs with .NET Core 2.1

@neekgreen

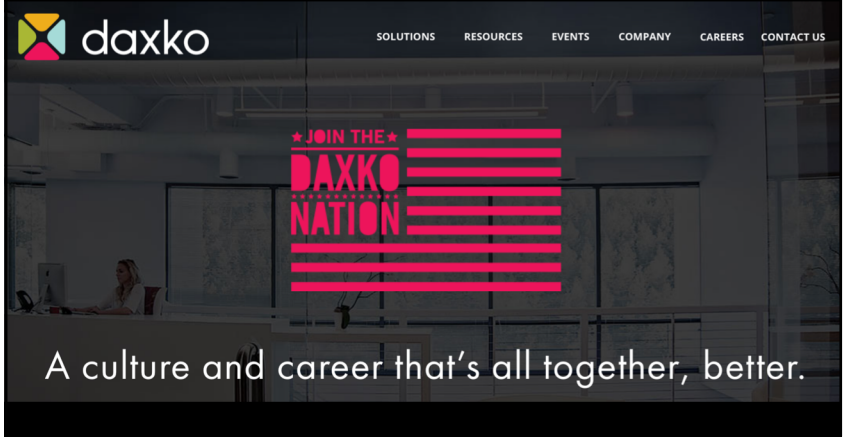
Console | Win Forms | WPF | MVC | Razor Pages | Web API

Building Resilient Applications in .NET Core

@neekgreen



@neekgreen



@neekgreen

@neekgreen

Resiliency 101

Strategies to Build Resilience

.NET Core Resiliency Features

@neekgreen

Resiliency 101

Strategies to Build Resilience

.NET Core Resiliency Features

@neekgreen

https://www.videoblocks.com/video/beautiful-epic-scale-mountain-range-man-standing-on-top-of-cliff-edge-snow-winter-landscape-nature-success-accomplishment-opportunity-concept-aerial-flight-npdilbjieiksmaufq

@neekgreen

*How do I **run** my applications in a **scalable** and **fault tolerant** manner?*

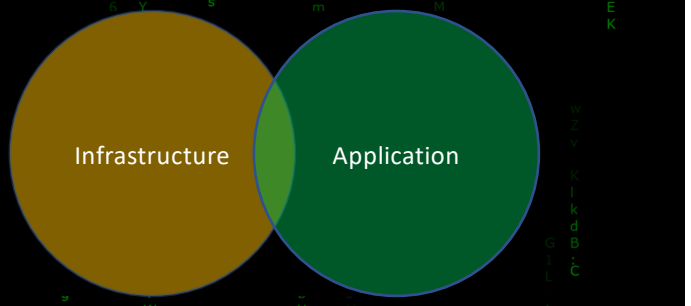
https://medium.com/microhq/building-resilient-and-fault-tolerant-applications-with-micro-53b454a8e8eb

@neekgreen

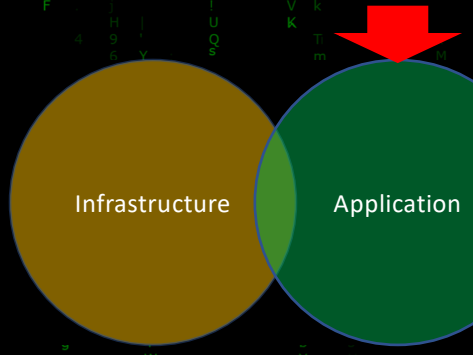
*How do I actually **write** applications in a **scalable** and **fault tolerant** manner?*

https://medium.com/microhq/building-resilient-and-fault-tolerant-applications-with-micro-53b454a8e8eb

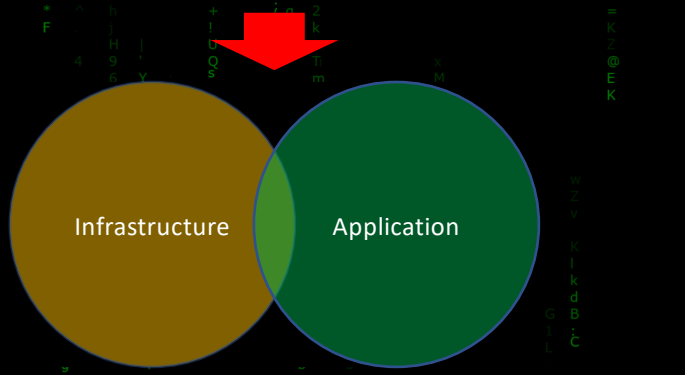
@neekgreen



@neekgreen



@neekgreen



@neekgreen

re·li·a·bil·i·ty

/rəˌlīəˈbilədē/

noun

the quality of being trustworthy or of performing consistently well.
"the car's background gives me every confidence in its reliability"

- the degree to which the result of a measurement, calculation, or specification can be depended on to be accurate.
  plural noun: **reliabilities**
  "these data throw doubt on the reliability of national statistics"

@neekgreen

*Frustrated **customers** can impact our **livelihood**.*

@neekgreen

***Reliability** is our applications operating **perfectly**, as designed, **for a period of time**.*

@neekgreen

Reliability *is our applications operating* ***perfectly****, as designed,* ***for a period of time****.*

@neekgreen

Reliability *is the* ***goal****.*

@neekgreen

re·sil·ience

/rəˈzilyəns/

noun

noun: **resiliency**

1. the capacity to recover quickly from difficulties; toughness.
   "the often remarkable resilience of so many British institutions"
2. the ability of a substance or object to spring back into shape; elasticity.
   "nylon is excellent in wearability and resilience"
   synonyms: flexibility, pliability, suppleness, plasticity, elasticity, springiness, spring, give; durability, ability to last, strength, sturdiness, toughness
   "he uses different types of vertical and cross strings in his rackets for added resilience"
   - strength of character, strength, toughness, hardiness; adaptability, buoyancy, flexibility, ability to bounce back; *informal* bouncebackability
     "she displayed an indomitable resilience in the face of misfortune"

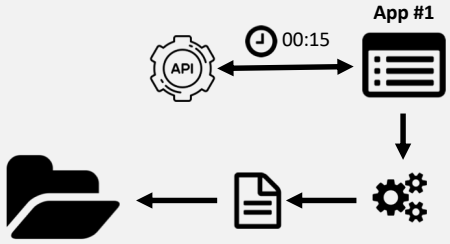
@neekgreen

It is not about ***avoiding*** *failures but* ***accepting*** *the fact that* ***failures will happen****.*

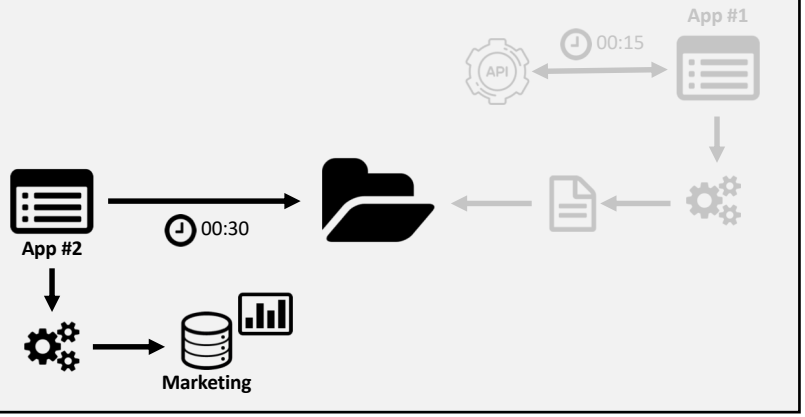
@neekgreen

Reliability *is the* ***goal****.*
Resiliency *is how we* ***reach*** *the* ***goal****.*

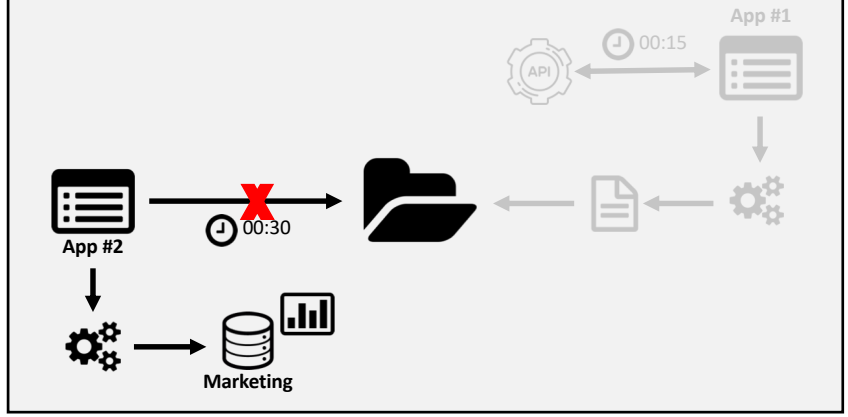
@neekgreen



@neekgreen



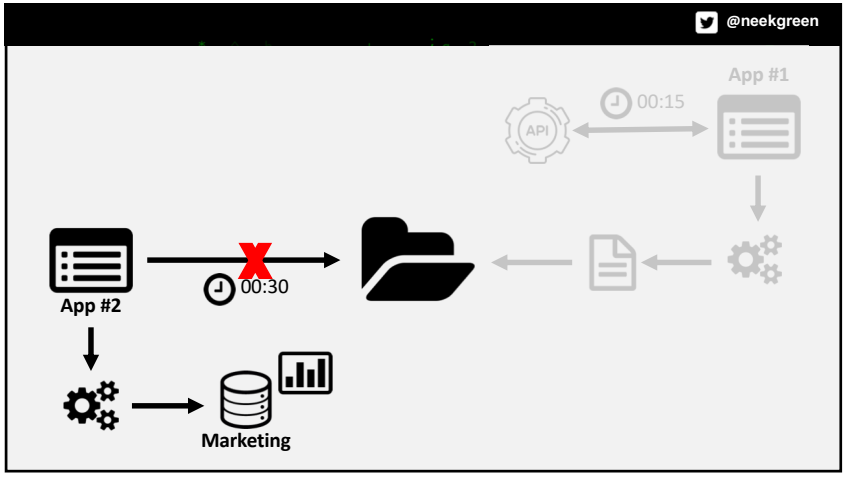
@neekgreen



@neekgreen

*To be **resilient**, our application must **do more** than **log errors**.*

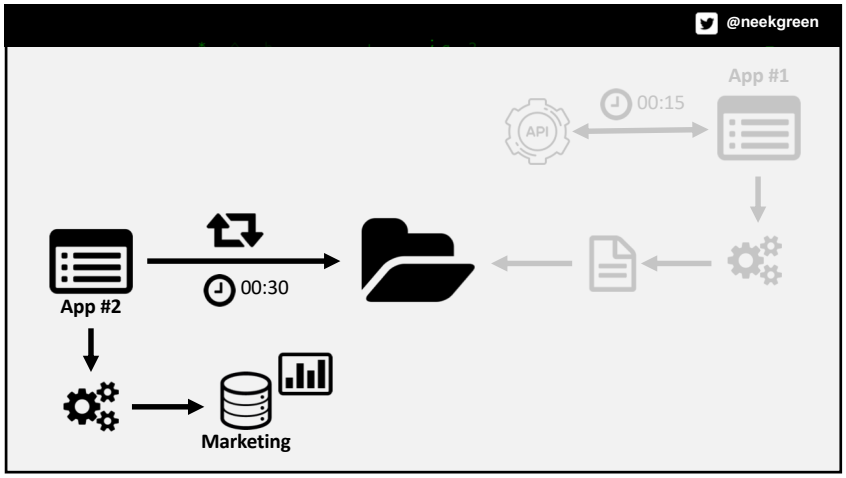
@neekgreen



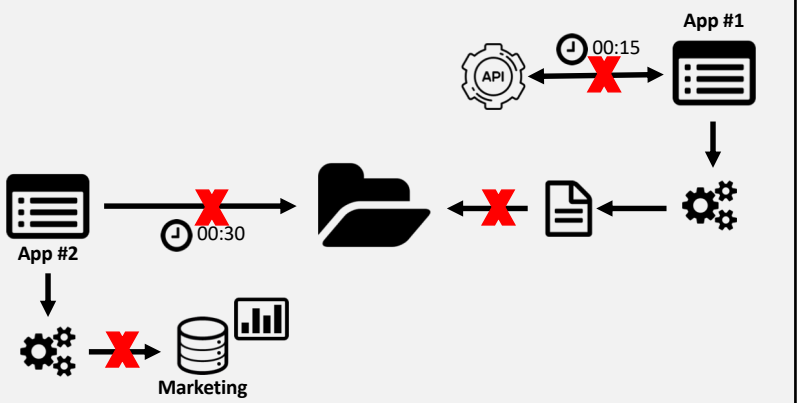
@neekgreen

***Plan** for **failure**.*

@neekgreen



@neekgreen



@neekgreen

Transient faults *are* ***errors*** *whose cause is expected to be a* ***temporary*** *condition such as* ***temporary service unavailability*** *or* ***network connectivity issues****.*

@neekgreen

A ***transient error*** *is the most common form of network* ***failure****.*

@neekgreen

The network is ***reliable****.*

@neekgreen

*A **fallacy** is a mistaken **belief**, especially one based on **unsound argument**.*

@neekgreen

Fallacies of distributed computing

Peter Deutsch and James Gosling

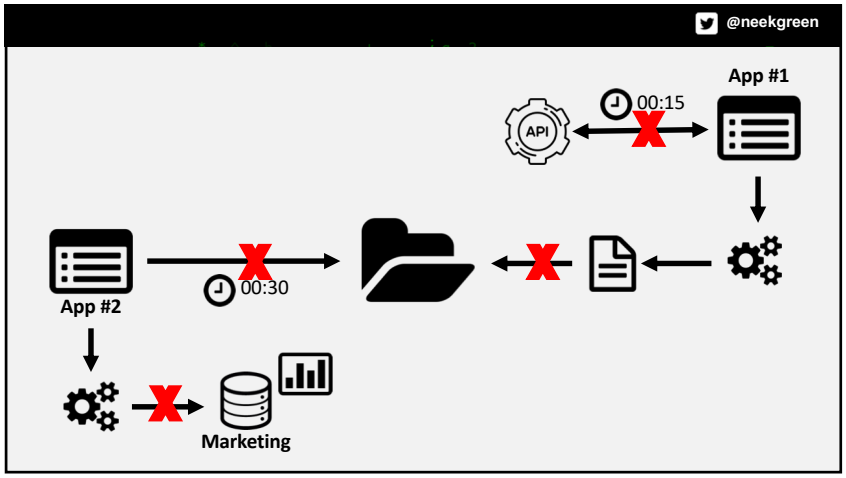
The network is reliable	Topology doesn't change
Latency is zero	There is one administrator
Bandwidth is infinite	Transport cost is zero
The network is secure	The network is homogeneous

@neekgreen

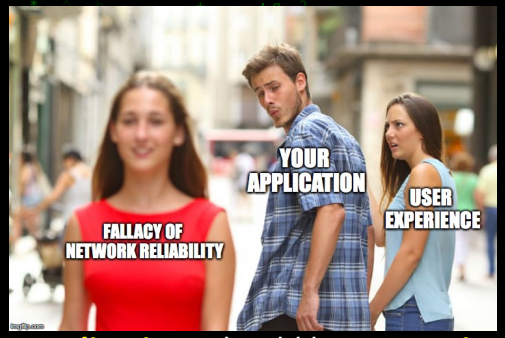
SYSTEM FAILURE

*The network is **not reliable**.*

@neekgreen

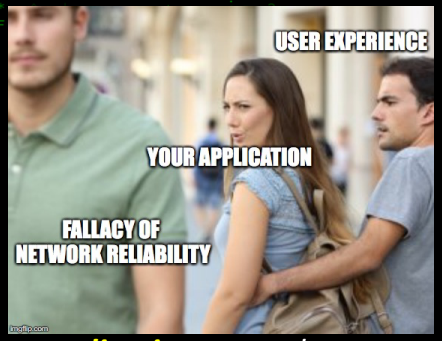


@neekgreen



*Our **applications** should have **trust issues** with the **network**.*

@neekgreen



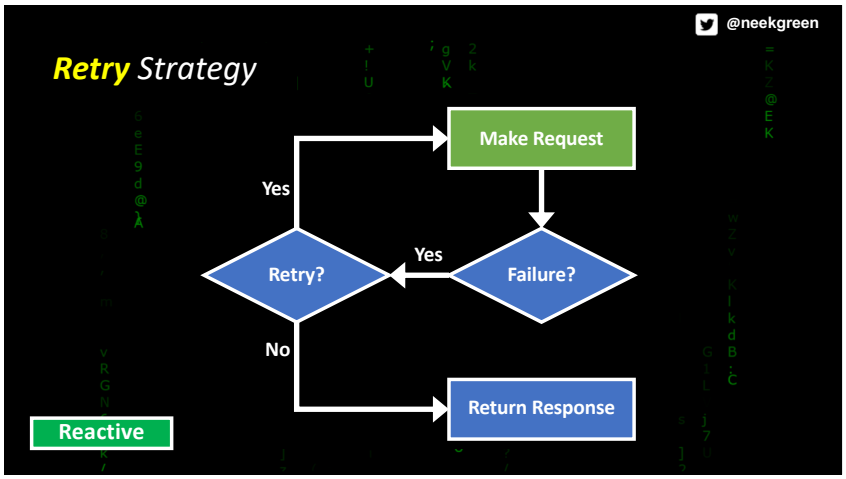
*Resilient **applications must** have **trust issues** with the **network**.*

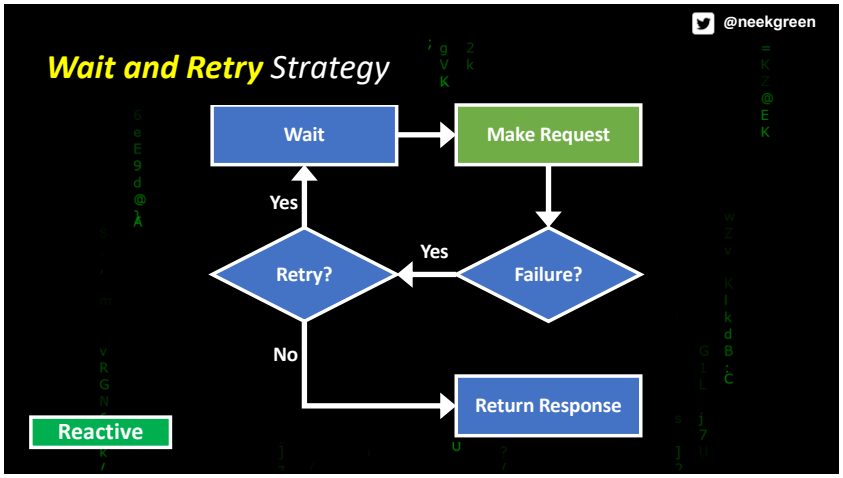
@neekgreen

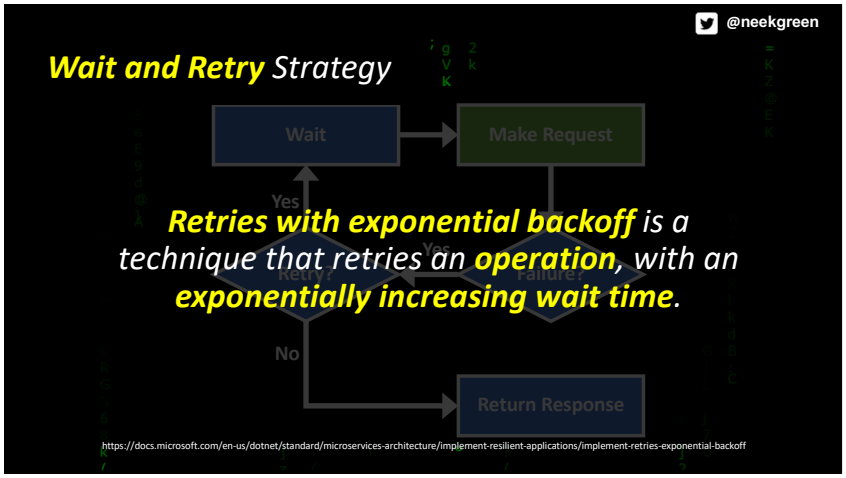
***Reliability** is the **goal**.*
***Resiliency** is how we **reach** the **goal**.*
***Transient errors** are the **barriers**.*

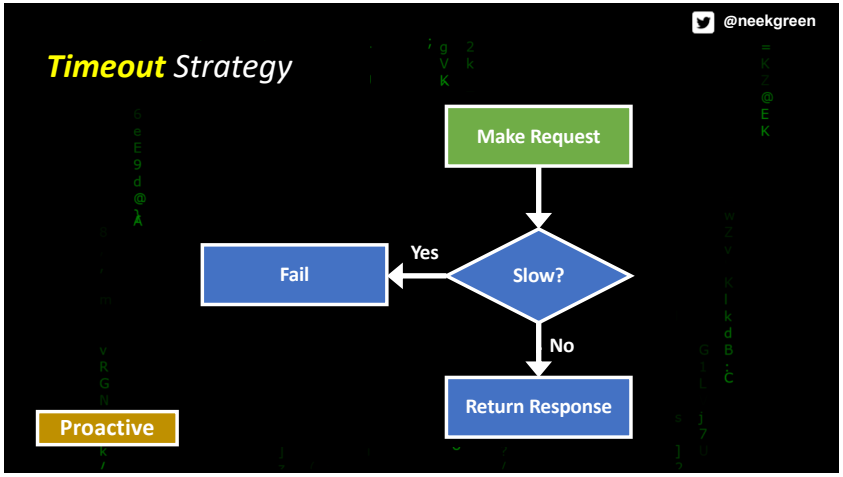
@neekgreen

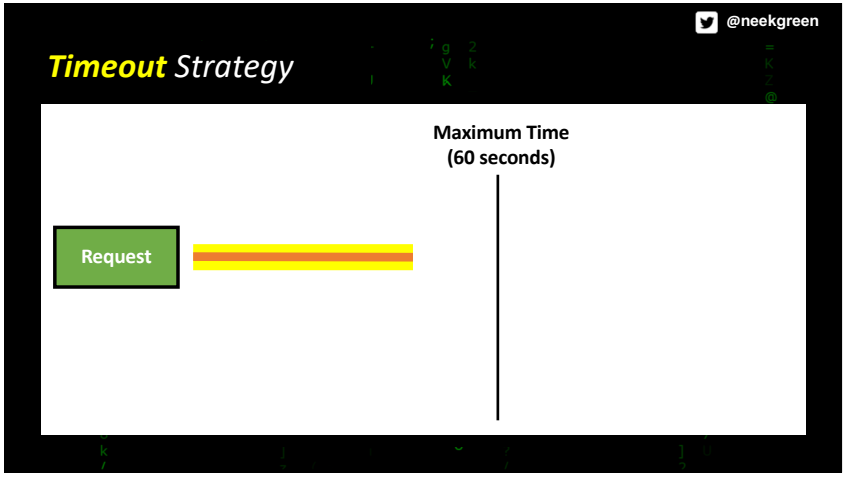

@neekgreen
Strategies to Build Resilience
Reactive
Proactive

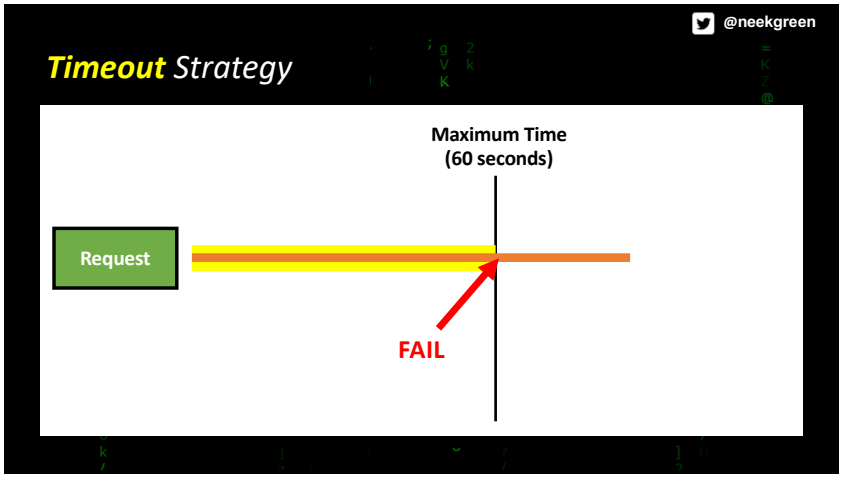

@neekgreen
Retry Strategy
Make Request
Yes
Retry?
Yes
Failure?
No
Return Response
Reactive

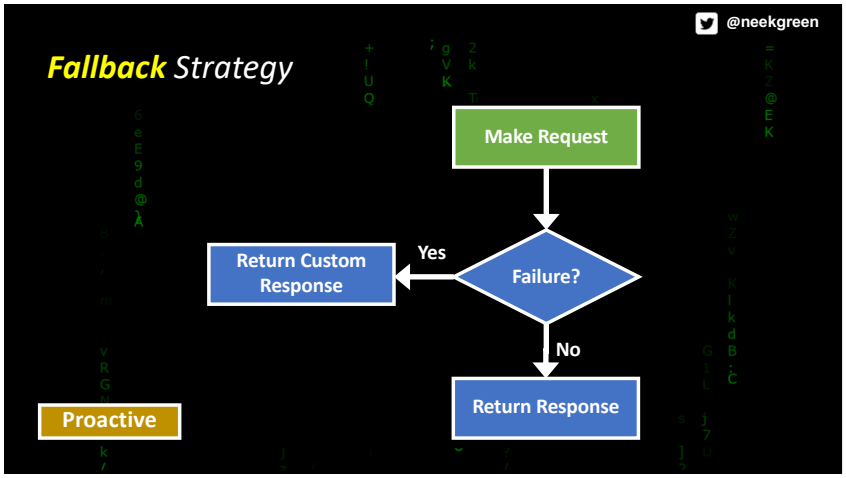

@neekgreen
Wait and Retry Strategy
Wait
Make Request
Yes
Retry?
Yes
Failure?
No
Return Response
Reactive

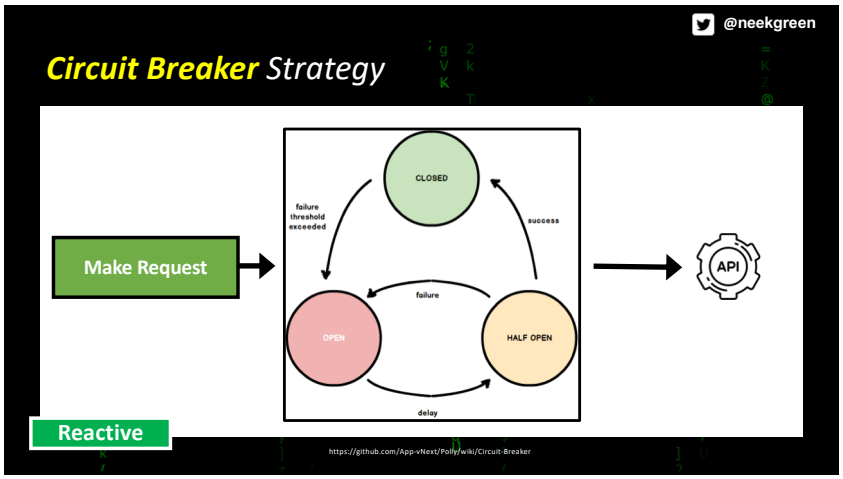

@neekgreen
Wait and Retry Strategy
Wait
Make Request
Retries with exponential backoff is a technique that retries an operation, with an exponentially increasing wait time.
No
Return Response
https://docs.microsoft.com/en-us/dotnet/standard/microservices-architecture/implement-resilient-applications/implement-retries-exponential-backoff

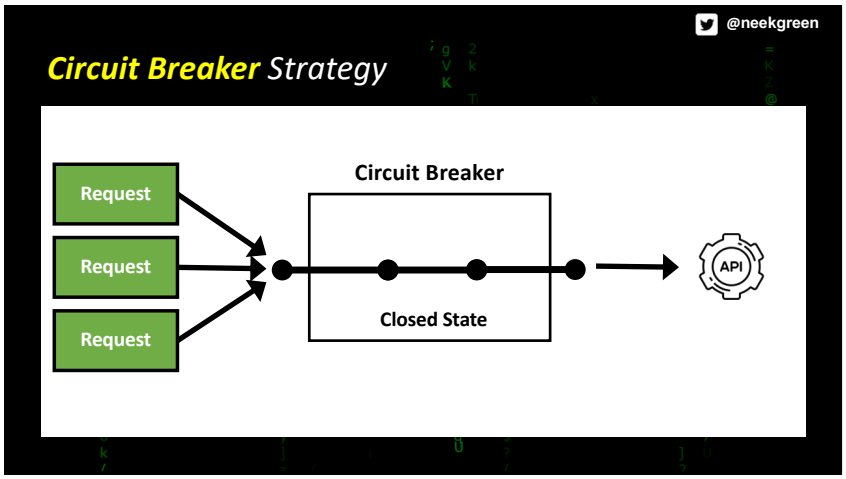

@neekgreen
Timeout Strategy
Make Request
Slow?
Yes
Fail
No
Return Response
Proactive

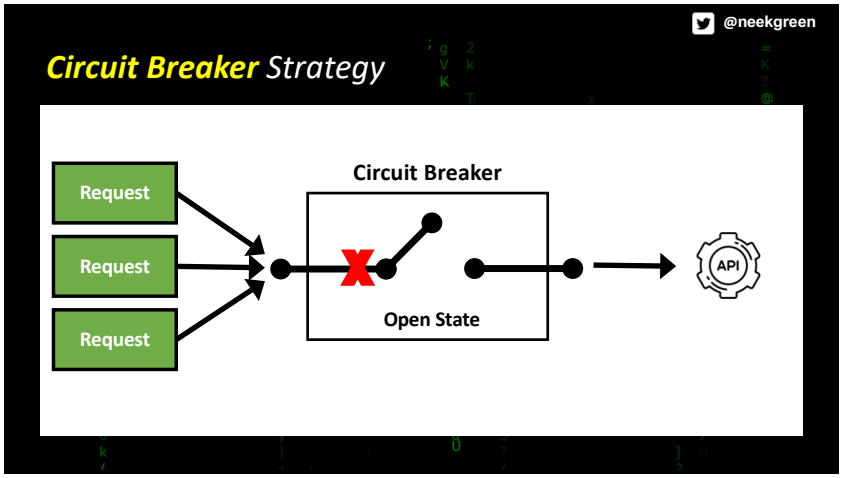

@neekgreen
Timeout Strategy
Maximum Time
(60 seconds)
Request

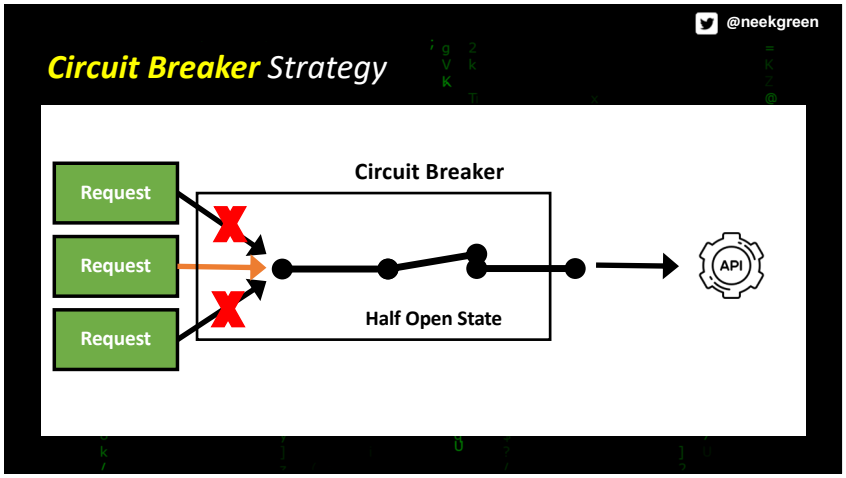

@neekgreen
Timeout Strategy
Maximum Time
(60 seconds)
Request
FAIL


@neekgreen
Fallback Strategy
Make Request
Failure?
Yes
Return Custom
Response
No
Return Response
Proactive

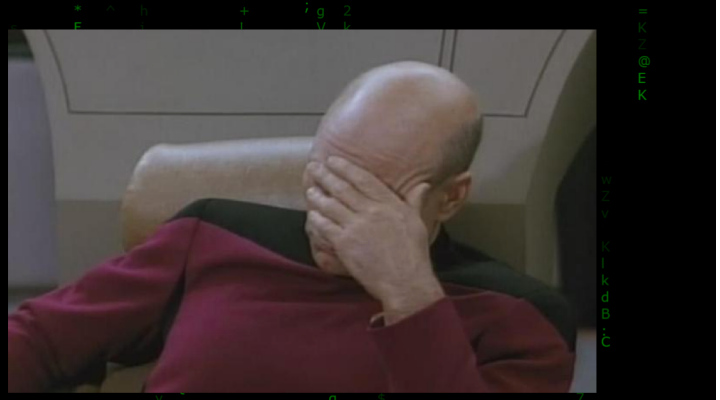

@neekgreen
Circuit Breaker Strategy
Make Request
CLOSED
failure threshold exceeded
success
failure
OPEN
HALF OPEN
delay
API
Reactive
https://github.com/App-vNext/Polly/wiki/Circuit-Breaker


@neekgreen
Circuit Breaker Strategy
Request
Request
Request
Circuit Breaker
Closed State
API


@neekgreen
Circuit Breaker Strategy
Request
Request
Request
Circuit Breaker
Open State
API


@neekgreen
Circuit Breaker Strategy
Request
Request
Request
Circuit Breaker
Half Open State
API

@neekgreen



@neekgreen

Resiliency 101

Strategies for Building Resilience

.NET Core Resiliency Features

@neekgreen

```
public class Program
{
    public static async Task Main(string[] args)
    {
        Console.WriteLine("Starting connections");

        for(int i = 0; i<10; i++)
        {
            using(var client = new HttpClient())
            {
                var result =
                    await client.GetAsync("http://neekgreen.com");

                Console.WriteLine(result.StatusCode);
            }
        }

        Console.WriteLine("Connections done");
    }
}
```

@neekgreen

```
C:\code\testapp>dotnet run
Compiling testapp for .NETCoreApp,Version=v2.1

...

Time elapsed 00:00:01.2501667

Starting connections
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
Connections done
```

@neekgreen

```
public class Program
{
    public static async Task Main(string[] args)
    {
        Console.WriteLine("Starting connections");

        for(int i = 0; i<10; i++)
        {
            using(var client = new HttpClient())
            {
                var result =
                    await client.GetAsync("http://neekgreen.com");

                Console.WriteLine(result.StatusCode);
            }
        }

        Console.WriteLine("Connections done");
    }
}
```

@neekgreen

```
public class Program
{
    public static async Task Main(string[] args)
    {
        Console.WriteLine("Starting connections");

        for(int i = 0; i<10; i++)
        {
            using(var client = new HttpClient())
            {
                var result =
                    await client.GetAsync("http://neekgreen.com");

                Console.WriteLine(result.StatusCode);
            }
        }

        Console.WriteLine("Connections done");
    }
}
```

@neekgreen

```
C:\code\socket>NETSTAT.EXE
...

Proto  Local Address          Foreign Address        State
TCP    10.211.55.6:12050      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12051      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12053      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12054      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12055      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12056      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12057      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12058      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12059      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12060      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12061      waws-prod-bay-017:http  TIME_WAIT
TCP    10.211.55.6:12062      waws-prod-bay-017:http  TIME_WAIT
...
```

https://aspnetmonsters.com/2016/08/2016-08-27-httpclientwrong/

@neekgreen

Unable to connect to the remote server

System.Net.Sockets.SocketException: Only one usage of each socket address (protocol/network address/port) is normally permitted.

@neekgreen

```
public class Program
{
    private static HttpClient client = new HttpClient();

    public static async Task Main(string[] args)
    {
        Console.WriteLine("Starting connections");

        for(int i = 0; i<10; i++)
        {
            var result =
                await client.GetAsync("http://neekgreen.com");
            Console.WriteLine(result.StatusCode);
        }

        Console.WriteLine("Connections done");
    }
}
```

@neekgreen

```
C:\code\testapp>dotnet run
Compiling testapp for .NETCoreApp,Version=v2.1

...

Time elapsed 00:00:01.2501667

Starting connections
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
Connections done
```

```
C:\code\socket>NETSTAT.EXE
...

Proto  Local Address          Foreign Address         State
TCP    10.211.55.6:12050      waws-prod-bay-017:http  TIME_WAIT
...
```

https://aspnetmonsters.com/2016/08/2016-08-27-httpclientwrong/

@neekgreen

```
public class Program
{
    private static HttpClient client = new HttpClient();

    public static async Task Main(string[] args)
    {
        Console.WriteLine("Starting connections");

        for(int i = 0; i<10; i++)
        {
            var result =
                await client.GetAsync("http://neekgreen.com");
            Console.WriteLine(result.StatusCode);
        }

        Console.WriteLine("Connections done");
    }
}
```

@neekgreen

HttpClient *has a lot of* ***baggage****.*

@neekgreen

HttpClient *is very important in building* ***resilient*** *applications in* ***.NET Core****.*

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient();

        // register other services...
        services
            .AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }
}
```

```
[Route("api/values")]
public class ValuesController : Controller
{
    private readonly IHttpClientFactory _httpClientFactory;

    public ValuesController(IHttpClientFactory httpClientFactory)
    {
        _httpClientFactory = httpClientFactory;
    }

    [HttpGet]
    public async Task<ActionResult> Get()
    {
        var client = _httpClientFactory.CreateClient();
        var result = await client.GetStringAsync("https://api.github.com");

        return Ok(result);
    }
}
```

```
public class Startup
{
    //# startup stuff

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient("github", t =>
        {
            t.BaseAddress = new Uri("https://api.github.com/");
            t.DefaultRequestHeaders.Add("Accept", "application/json");
            t.DefaultRequestHeaders.Add("User-Agent", "blah");
        });

        //# register other stuff...
        services
            .AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }
}
```

```
[Route("api/values")]
public class ValuesController : Controller
{
    private readonly IHttpClientFactory _httpClientFactory;

    public ValuesController(IHttpClientFactory httpClientFactory)
    {
        _httpClientFactory = httpClientFactory;
    }

    [HttpGet]
    public async Task<ActionResult> Get()
    {
        var client = _httpClientFactory.CreateClient("github");
        var result = await client.GetStringAsync("/");

        return Ok(result);
    }
```

```
public interface IGitHubClient
{
    Task<GitHubUser> GetUser(string username);
}


public class GitHubClient : IGitHubClient
{
    private readonly HttpClient _httpClient;

    public GitHubClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    async Task<GitHubUser> IGitHubClient.GetUser(string username)
    {
        throw new NotImplementedException();
    }
```

```
public class Startup
{
    //# startup stuff

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient<IGitHubClient, GitHubClient>(t =>
        {
            t.BaseAddress = new Uri("https://api.github.com/");
            t.DefaultRequestHeaders.Add("Accept", "application/json");
            t.DefaultRequestHeaders.Add("User-Agent", "blah");
        });

        //# register other stuff...
        services
            .AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }
}
```

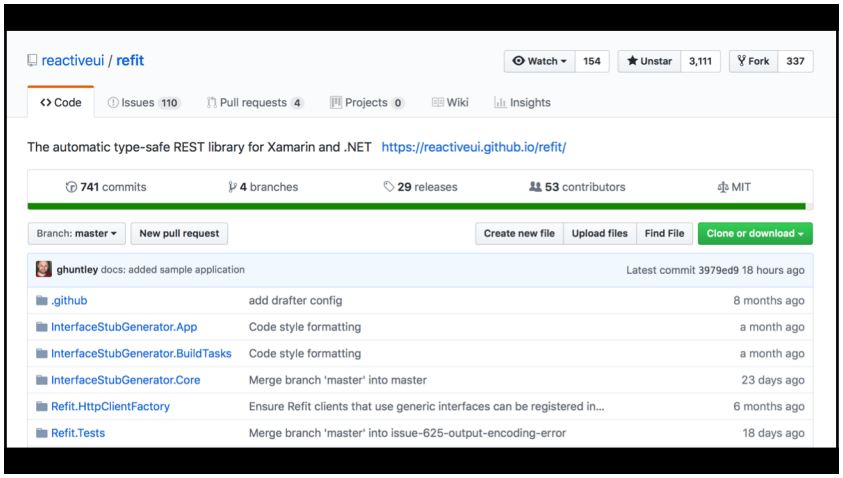
```
[Route("api/values")]
public class ValuesController : Controller
{
    private readonly IGitHubClient _gitHubClient;

    public ValuesController(IGitHubClient gitHubClient)
    {
        _gitHubClient = gitHubClient;
    }

    [HttpGet]
    public async Task<ActionResult> Get(string username)
    {
        var result = await _gitHubClient.GetUser(username);
        return Ok(result);
    }
}
```

```
public class GitHubClient
{
    private readonly HttpClient _httpClient;

    public GitHubClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    public async Task<GitHubUser> GetUser(string username)
    {
        throw new NotImplementedException();
    }
}
```

```
public class Startup
{
    //# startup stuff

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHttpClient<GitHubClient>(t =>
        {
            t.BaseAddress = new Uri("https://api.github.com/");
            t.DefaultRequestHeaders.Add("Accept", "application/json");
            t.DefaultRequestHeaders.Add("User-Agent", "blah");
        });

        //# register other stuff...
        services
            .AddMvc()
            .SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
    }
}
```

```
[Route("api/values")]
public class ValuesController : Controller
{
    private readonly GitHubClient _gitHubClient;

    public ValuesController(GitHubClient gitHubClient)
    {
        _gitHubClient = gitHubClient;
    }

    [HttpGet]
    public async Task<ActionResult> Get(string username)
    {
        var result = await _gitHubClient.GetUser(username);
        return Ok(result);
    }
}
```


```
public interface IGitHubClient
{
    Task<GitHubUser> GetUser(string username);
}


public class GitHubClient : IGitHubClient
{
    private readonly HttpClient _httpClient;

    public GitHubClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    async Task<GitHubUser> IGitHubClient.GetUser(string username)
    {
        throw new NotImplementedException();
    }
}
```




```
public interface IGitHubClient
{
    [Get("/users/{username}")]
    Task<GitHubUser> GetUser(string username);
}


public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services
            .AddHttpClient("github", t =>
            {
                t.BaseAddress = new Uri("https://api.github.com/");
                t.DefaultRequestHeaders.Add("Accept", "application/json");
                t.DefaultRequestHeaders.Add("User-Agent", "blah");
            })
            .AddTypedClient<IGitHubClient>(e => RestService.For<IGitHubClient>(e));

        //# register other stuff...
    }
```

```
public interface IGitHubClient
{
    Task<GitHubUser> GetUser(string username);
}


public class GitHubClient : IGitHubClient
{
    private readonly HttpClient _httpClient;

    public GitHubClient(HttpClient httpClient)
    {
        _httpClient = httpClient;
    }

    async Task<GitHubUser> IGitHubClient.GetUser(string username)
    {
        throw new NotImplementedException();
    }
}
```

*New **HttpClient** support is **awesome**, but what about building **resiliency** in our applications?*

patterns & practices

App-vNext / Polly

Watch 328 | Unstar 4,466 | Fork 439

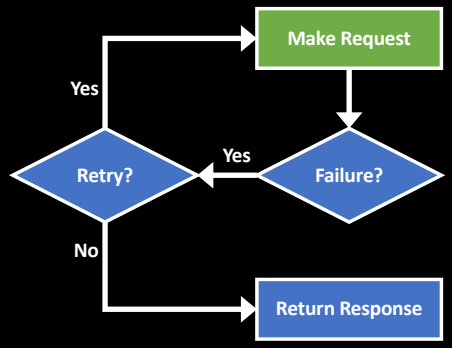
Code | Issues 22 | Pull requests 0 | Projects 0 | Wiki | Insights

Polly is a .NET resilience and transient-fault-handling library that allows developers to express policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback in a fluent and thread-safe manner. From version 6.0.1, Polly targets .NET Standard 1.1 and 2.0+. http://www.thepollyproject.org

circuit-breaker | circuit-breaker-pattern | resilience | resiliency-patterns | retry-strategies | dotnet | fault-handler | transient-fault-handling

@neekgreen

Retry Strategy



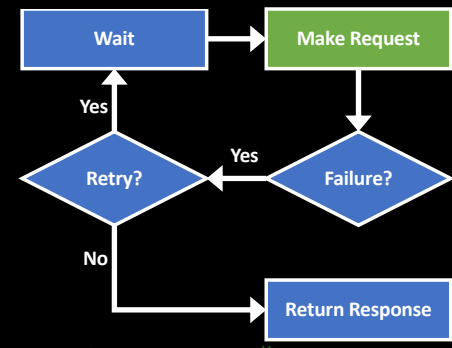
@neekgreen

Retry Strategy

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services
            .AddHttpClient("github", t =>
            {
                t.BaseAddress = new Uri("https://api.github.com/");
                t.DefaultRequestHeaders.Add("Accept", "application/json");
                t.DefaultRequestHeaders.Add("User-Agent", "blah");
            })
            .AddTransientHttpErrorPolicy(p => p.RetryAsync(3));

        //# register other stuff...
    }
```

@neekgreen

Wait and Retry Strategy



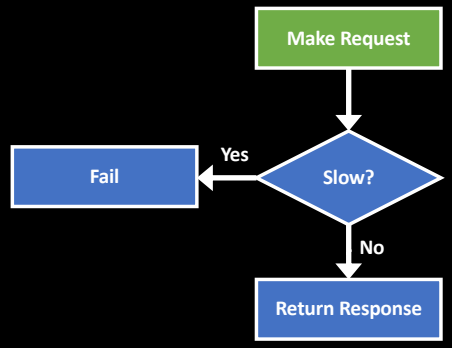
@neekgreen

Wait and Retry Strategy

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services
            .AddHttpClient("github", t =>
            {
                t.BaseAddress = new Uri("https://api.github.com/");
                t.DefaultRequestHeaders.Add("Accept", "application/json");
                t.DefaultRequestHeaders.Add("User-Agent", "blah");
            })
            .AddTransientHttpErrorPolicy(p =>
                p.WaitAndRetryAsync(3, attempt => TimeSpan.FromSeconds(Math.Pow(2, attempt))));

        //# register other stuff...
    }
}
```

@neekgreen

Timeout Strategy



@neekgreen

Timeout Strategy

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services
            .AddHttpClient("github", t =>
            {
                t.BaseAddress = new Uri("https://api.github.com/");
                t.DefaultRequestHeaders.Add("Accept", "application/json");
                t.DefaultRequestHeaders.Add("User-Agent", "blah");
            })
            .AddPolicyHandler(Policy.TimeoutAsync<HttpResponseMessage>(30));

        //# register other stuff...
    }
}
```

@neekgreen

Timeout Strategy

HttpClient.Timeout Property

Namespace: System.Net.Http

Assemblies: System.Net.Http.dll, netstandard.dll

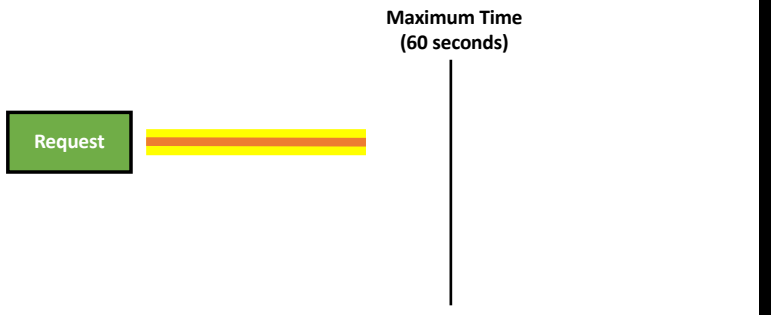
Gets or sets the timespan to wait before the request times out.

C# Copy

```
public TimeSpan Timeout { get; set; }
```

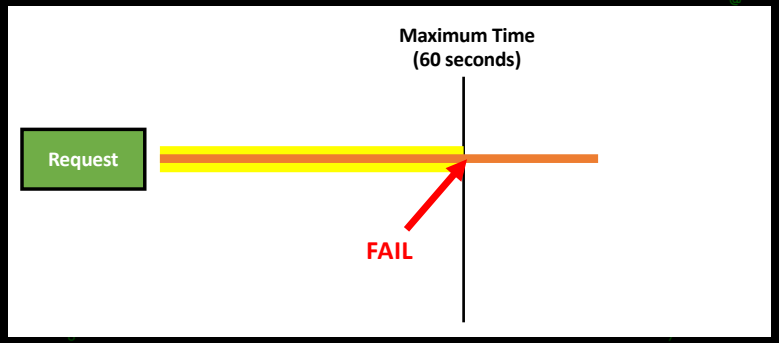
@neekgreen

Timeout Strategy



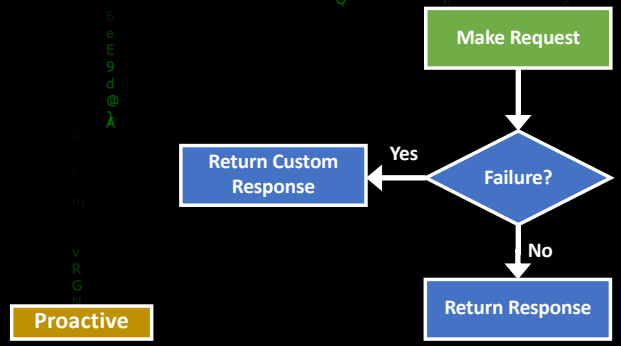
@neekgreen

Timeout Strategy



@neekgreen

Fallback Strategy



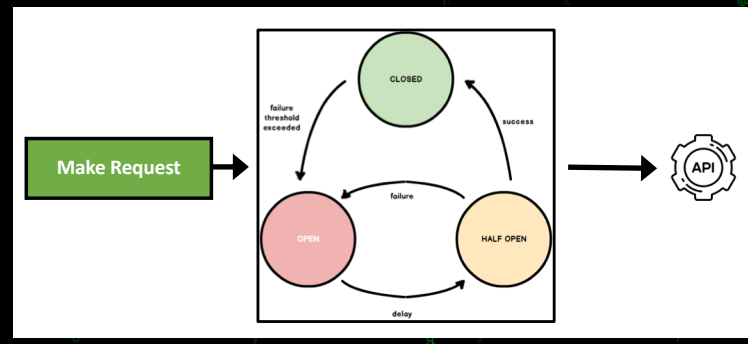
@neekgreen

Fallback Strategy

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services
            .AddHttpClient("github", t =>
            {
                t.BaseAddress = new Uri("https://api.github.com/");
                t.DefaultRequestHeaders.Add("Accept", "application/json");
                t.DefaultRequestHeaders.Add("User-Agent", "blah");
            })
            .AddPolicyHandler(
                Policy.HandleResult<HttpResponseMessage>(
                    r => r.StatusCode == HttpStatusCode.InternalServerError)
                    .FallbackAsync(
                        new HttpResponseMessage(HttpStatusCode.OK) { }));
```

@neekgreen

Circuit Breaker Strategy



@neekgreen

Circuit Breaker *Strategy*

```
{
    public void ConfigureServices(IServiceCollection services)
    {
        services
            .AddHttpClient("github", t =>
            {
                t.BaseAddress = new Uri("https://api.github.com/");
                t.DefaultRequestHeaders.Add("Accept", "application/json");
                t.DefaultRequestHeaders.Add("User-Agent", "blah");
            })
            .AddPolicyHandler(
                Policy.Handle<HttpRequestException>()
                    .CircuitBreaker(
                        exceptionsAllowedBeforeBreaking: 5,
                        durationOfBreak: TimeSpan.FromMinutes(1)
                    ).AsAsyncPolicy<HttpResponseMessage>());
    }
```

@neekgreen

*There is more to **Polly** than just providing **HttpClient** support.*

@neekgreen

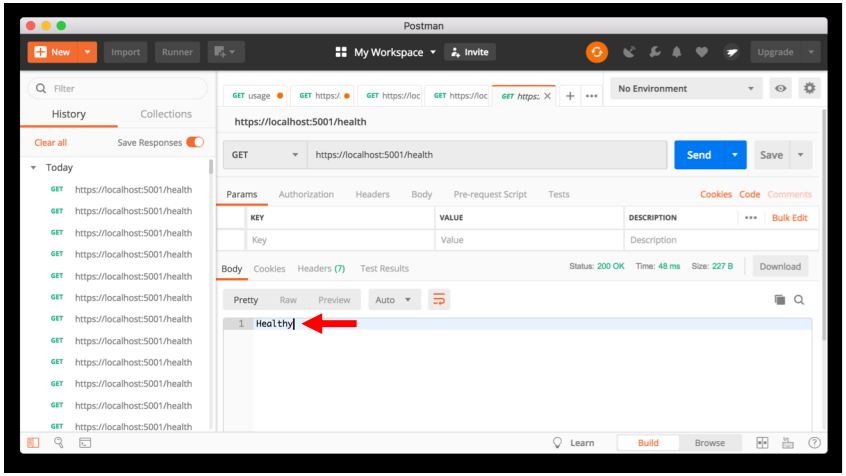
***Health Checks** are used to report the **health** of our application components.*

@neekgreen

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthCheck();
    }
}
```

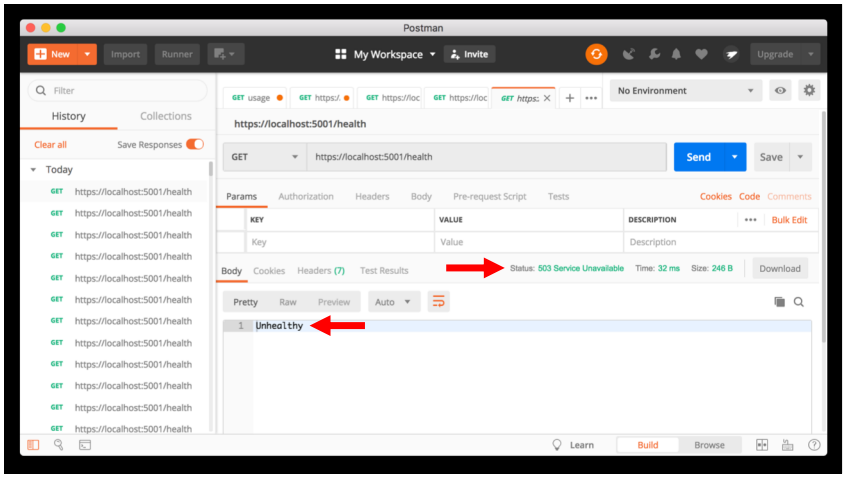
@neekgreen

```
public class Startup
{
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseHealthChecks("/health");
    }
}
```



@neekgreen

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthChecks()
            .AddCheck("Example 1", () =>
                HealthCheckResult.Healthy("Example 1 is OK!"),
                    tags: new[] { "tag1", "tag2" })
            .AddCheck("Example 2", () =>
                HealthCheckResult.Degraded("Example 2 is Meh!"),
                    tags: new[] { "tag2", "tag3" })
            .AddCheck("Example 3", () =>
                HealthCheckResult.Unhealthy("Example 3 is Dead!"),
                    tags: new[] { "tag3", "tag4" });
    }
}
```



@neekgreen

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthChecks()
            .AddCheck("Example 1", () =>
                HealthCheckResult.Healthy("Example 1 is OK!"),
                      tags: new[] { "tag1", "tag2" })
            .AddCheck("Example 2", () =>
                HealthCheckResult.Degraded("Example 2 is Meh!"),
                      tags: new[] { "tag2", "tag3" })
            .AddCheck("Example 3", () =>
                HealthCheckResult.Unhealthy("Example 3 is Dead!"),
                      tags: new[] { "tag3", "tag4" });
    }
}
```

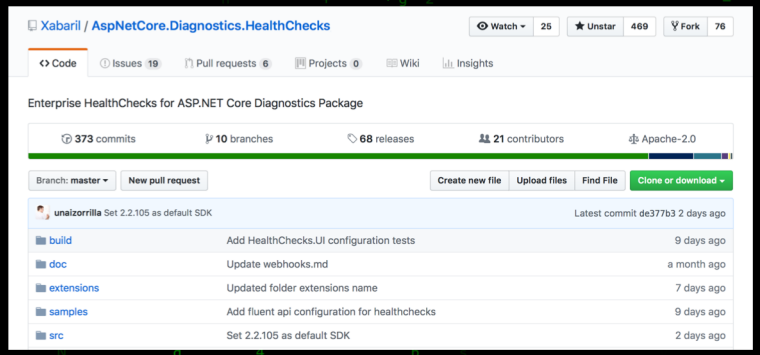
@neekgreen

```
public class RandomHealthCheck : IHealthCheck
{
    private readonly Random _random = new Random();

    public Task<HealthCheckResult> CheckHealthAsync(
        HealthCheckContext context,
        CancellationToken cancellationToken = default)
    {
        return Task.FromResult(
            new HealthCheckResult(
                (HealthStatus)_random.Next(0, 3),
                description: "i dunno what i am doing..."));
    }
}
```

@neekgreen

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthCheck()
            .AddCheck<RandomHealthCheck>(
                "Random Health Check", () =>
                tags: new[] { "tag1", "tag2 })
    }
}
```

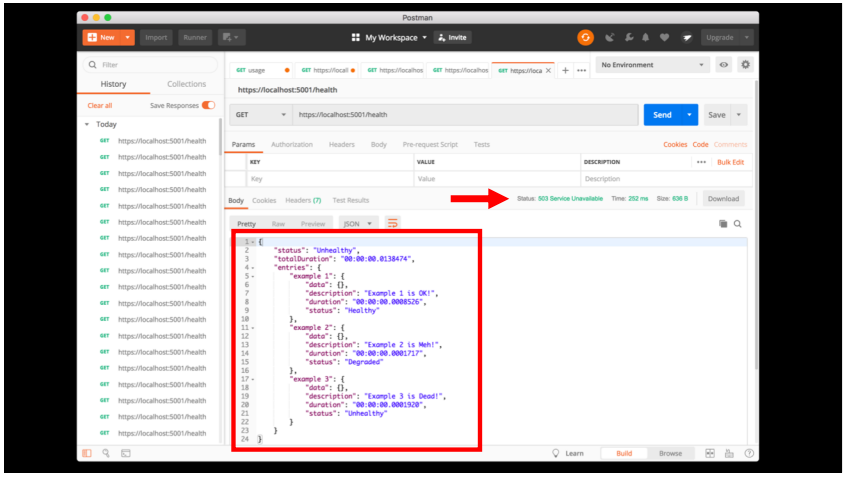
@neekgreen



*This is a rebranding of **BeatPulse**.*

@neekgreen

```
public class Startup
{
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseHealthChecks("/health", new HealthCheckOptions
        {
            Predicate = _ => true,
            ResponseWriter = UIResponseWriter.WriteHealthCheckUIResponse
        });
    }
}
```



@neekgreen

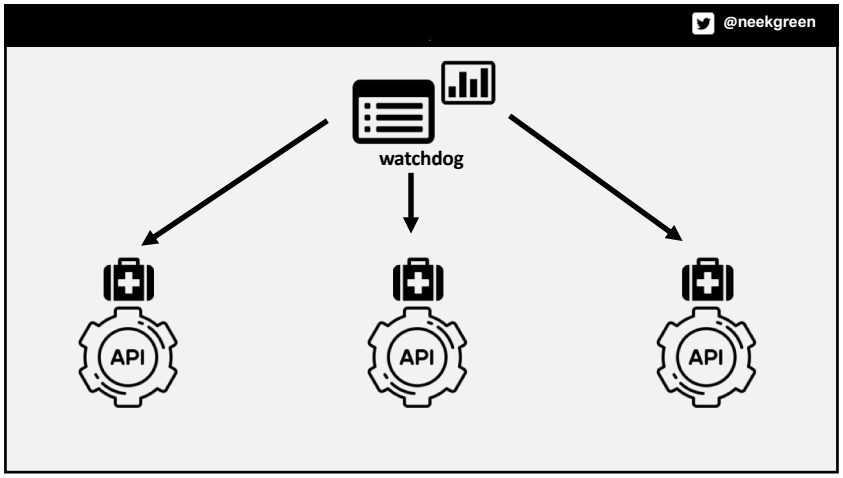
elastic
SQLite
MySQL
RabbitMQ
OpenID
amazon web services
APACHE kafka
RAVENDB
mongoDB.
redis
Microsoft SQL Server

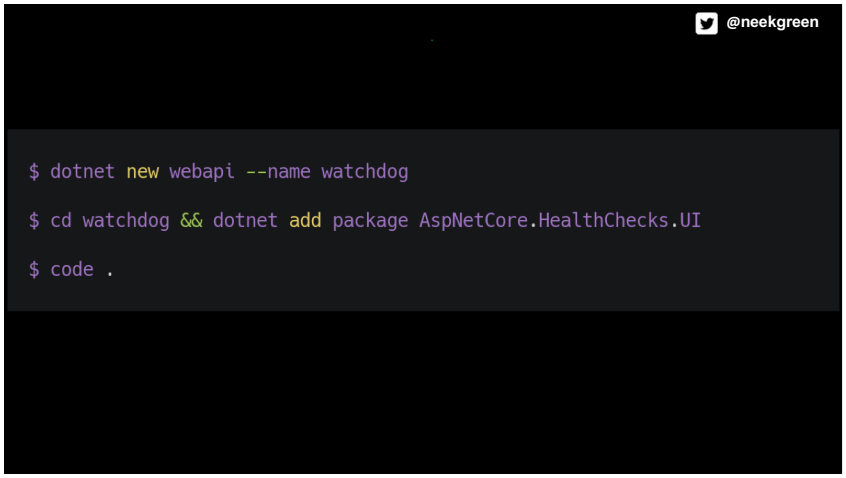
@neekgreen

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthChecks()
            .AddSqlServer(
                connectionString: Configuration["Data:ConnectionStrings:Sql"],
                healthQuery: "SELECT 1;",
                name: "sql",
                failureStatus: HealthStatus.Degraded,
                tags: new string[] { "db", "sql", "sqlserver" });
    }
}
```


@neekgreen
API
API
API


@neekgreen
A watchdog is a separate service that can watch and report health and load across services.


@neekgreen
watchdog
API
API
API


@neekgreen
$ dotnet new webapi --name watchdog
$ cd watchdog && dotnet add package AspNetCore.HealthChecks.UI
$ code .

@neekgreen

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddHealthChecksUI();
    }
}
```

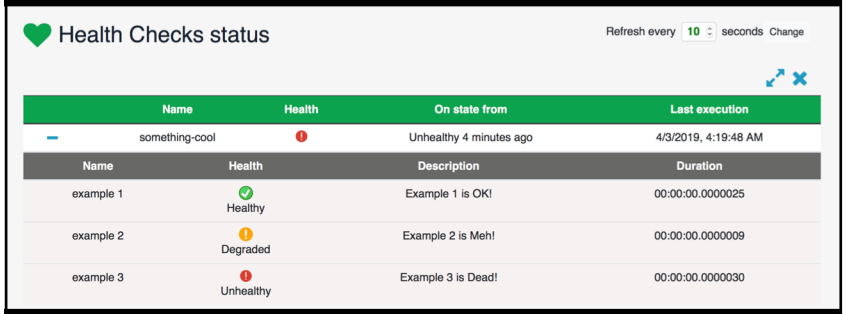
@neekgreen

```
public class Startup
{
    public void Configure(IApplicationBuilder app, IHostingEnvironment env)
    {
        app.UseHealthChecksUI(config => config.UIPath = "/hc-ui");
    }
}
```

@neekgreen

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "AllowedHosts": "*",

  "HealthChecks-UI": {
      "HealthChecks": [
      {
        "Name": "something-cool",
        "Uri": "http://localhost:5001/health"
      }
    ]
  }
}
```

@neekgreen



@neekgreen

Resiliency 101

Strategies to Build Resilience

.NET Core Resiliency Features

@neekgreen

Reliability *is the* ***goal****.*
Resiliency *is how we* ***reach*** *the* ***goal****.*
Transient errors *are the* ***barriers****.*

@neekgreen

Fin

@neekgreen