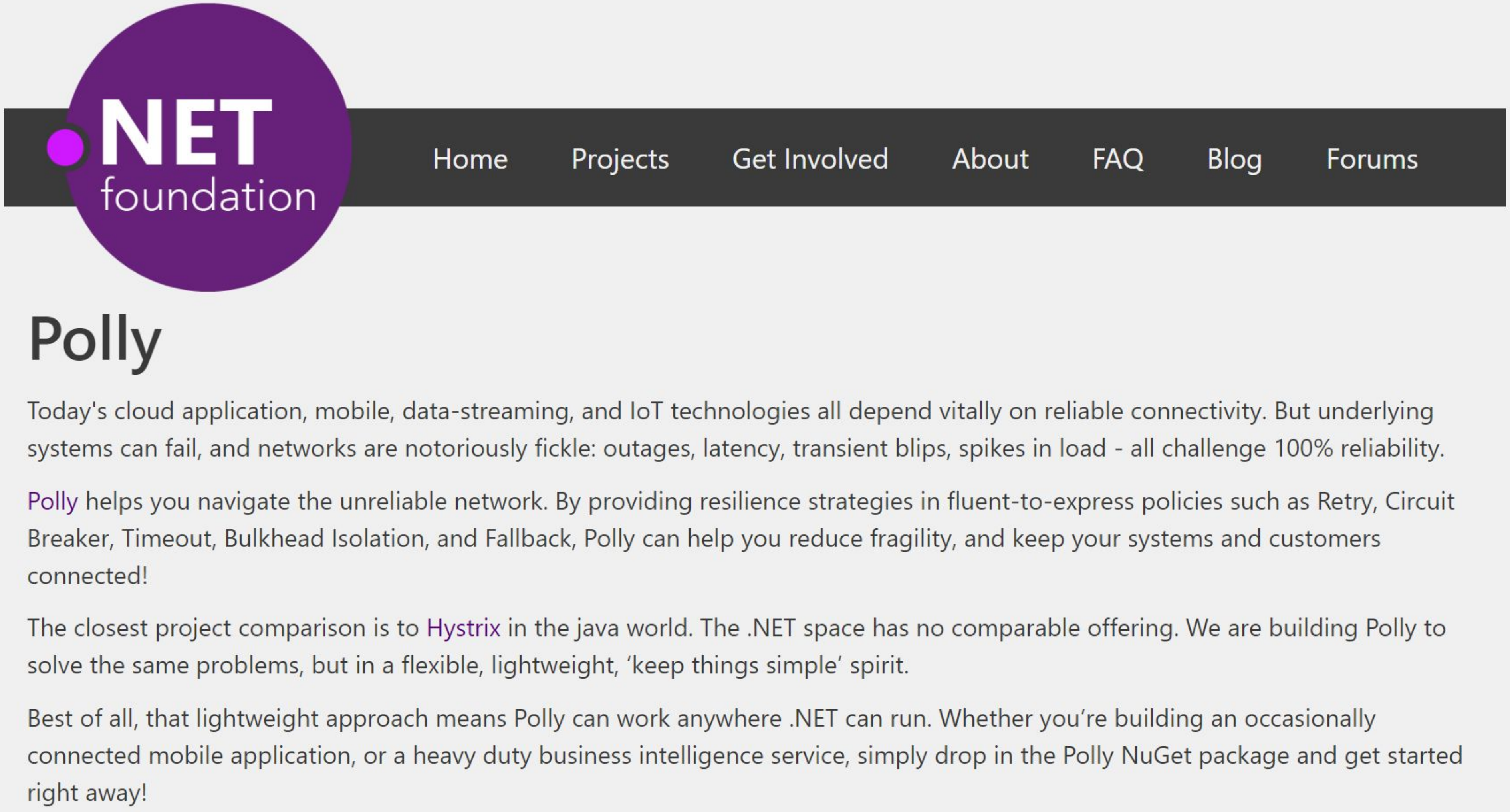
Bulletproof Transient Error Handling with Polly

AppvNext
create. combine. connect.



Polly

Today's cloud application, mobile, data-streaming, and IoT technologies all depend vitally on reliable connectivity. But underlying systems can fail, and networks are notoriously fickle: outages, latency, transient blips, spikes in load - all challenge 100% reliability.

Polly helps you navigate the unreliable network. By providing resilience strategies in fluent-to-express policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback, Polly can help you reduce fragility, and keep your systems and customers connected!

The closest project comparison is to Hystrix in the java world. The .NET space has no comparable offering. We are building Polly to solve the same problems, but in a flexible, lightweight, ‘keep things simple’ spirit.

Best of all, that lightweight approach means Polly can work anywhere .NET can run. Whether you’re building an occasionally connected mobile application, or a heavy duty business intelligence service, simply drop in the Polly NuGet package and get started right away!

Transient Errors

Network outages

Service outages

Denial of Service attacks

IO locks

Connected device failures

Polly to the rescue!

.NET 4.0 / 4.5+ / PCL / .NET Standard / .NET Core

Fluently express transient exception handling policies:

Retry, Circuit Breaker, Timeout,

Bulkhead Isolation, Fallback

https://github.com/App-vNext/Polly

Nuget: `Install-Package Polly`

Polly offers multiple resilience strategies …

Retry … ‘Maybe it’s just a blip’

Circuit Breaker … ‘That system is down / struggling’

Timeout … ‘Don’t wait forever!’

Bulkhead isolation … ‘One fault shouldn’t sink the whole ship’

Cache … ‘You’ve asked that one before!’

Fallback … ‘If all else fails … degrade gracefully’

All policies can be combined, for multiple protection!

History of Polly

2013 Michael Wolfenden invents Polly.
2014 Targets multi .NET versions including PCL.
2015 Scott Hanselman (Microsoft) recommends Polly!
Thoughtworks (Martin Fowler et al) recommend Polly!

Nov 2015 App-vNext take stewardship
Dec 2015 Full async support
April 2016 Advanced circuit breaker. Circuit health reporting.
June 2016 Handle return values as faults.
July 2016 .NET Core / .NET Standard support.
October 2016 Timeout, Bulkhead Isolation, Fallback, PolicyWrap

Step 1: Define Policy

```
var retryPolicy = Policy
    .Handle<EndpointNotFoundException>()
    .RetryForeverAsync();
```

Step 2: Execute with Policy

```
var response =
    await retryPolicy.ExecuteAsync(() => DoSomething());
```

Retry Patterns

Retry immediately on failure. Specify number of retries.

Wait and Retry Retry with a timeout in between each try. Change the timeout between each retry, eg exponential back-off.

Retry Forever Keep retrying until succeeds.

Retry addresses ... ‘It’s probably a blip. Give it another go - it might succeed.’

Circuit Breaker

Circuit Breaker Breaks the circuit for a configured period if too many errors occur

+ Blocks calls while circuit is broken.
+ Protects downstream system - chance to recover.
+ Fail fast to the caller.

Circuit Breaker addresses … ‘Whoa, that system is struggling / down. Give it a break. And don’t hang around waiting for an answer that’s unlikely, right now!’

Timeout

Timeout Stop waiting once you think an answer will not come

Optimistic mode Co-operative timeout via `CancellationToken`

Pessimistic mode Enforces timeout (returns to caller) even when governed delegate doesn’t support timeouts/cancellation.

Timeout ensures … calls can ‘walk away’ from a faulting downstream system, release blocked threads/connections etc.

Bulkhead Isolation

Bulkhead Prevents one operation from consuming more than its fair share of resources.

+ Imagine one stream of calls starts faulting slowly...
+ All threads in a caller could end up waiting on that system … until it starves the caller doing anything else.

Bulkhead prevents this, by limiting the resources (threads) used by separate call streams.

Bulkhead … ‘One fault shouldn’t sink the whole ship!’

Read-through Cache (Oct 16: forthcoming)

Cache A certain proportion of calls will be duplicates.
Serve from cache if you can - reduce latency and save calls.

+ Pluggable interface - use any cache provider you like.
+ Supplied providers for Redis, Azure, Amazon Cache, etc.

Cache addresses … ‘You’ve asked that one before!’

Fallback

Fallback Specifies a substitute value to provide (or action to run) when an operation still fails.

Fallback addresses … ‘Failures will occur … prepare how you will respond when that happens’

Policy Wrap

PolicyWrap Combine any of the previous strategies into one concise policy.

```
PolicyWrap myResilience =
    Policy.Wrap(fallback, retry, breaker, timeout);

myResilience.Execute(() => DoSomething());
```

Policy Basics

Define how transient exceptions should be handled

Fluent and concise

Thread-safe

Reusable across call sites

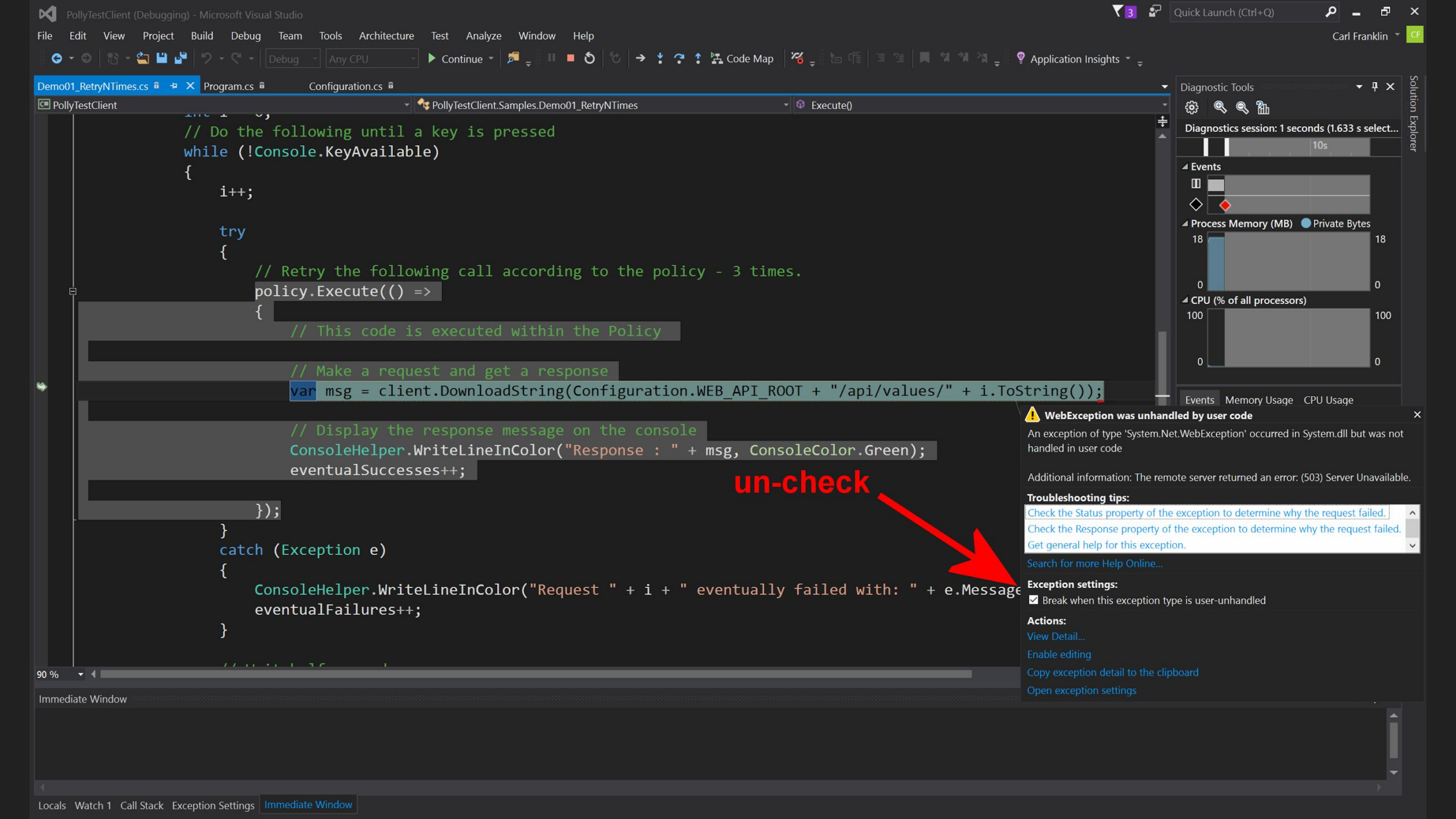
Sync and async

Chain policies together

Apply to any Action or Func (service calls, data stores, web requests, mobile connectivity)

Demos

https://github.com/App-vNext/Polly-Samples


PollyTestClient (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Quick Launch (Ctrl+Q)
Carl Franklin
Debug
Any CPU
Continue
Code Map
Application Insights
Demo01_RetryNTimes.cs
Program.cs
Configuration.cs
PollyTestClient
PollyTestClient.Samples.Demo01_RetryNTimes
Execute()
Diagnostic Tools
Diagnostics session: 1 seconds (1.633 s select...
10s
Events
Process Memory (MB)
Private Bytes
CPU (% of all processors)
Events Memory Usage CPU Usage
Solution Explorer
90 %
Immediate Window
Locals Watch 1 Call Stack Exception Settings Immediate Window
un-check

```
            // Do the following until a key is pressed
            while (!Console.KeyAvailable)
            {
                i++;

                try
                {
                    // Retry the following call according to the policy - 3 times.
                    policy.Execute(() =>
                    {
                        // This code is executed within the Policy

                        // Make a request and get a response
                        var msg = client.DownloadString(Configuration.WEB_API_ROOT + "/api/values/" + i.ToString());

                        // Display the response message on the console
                        ConsoleHelper.WriteLineInColor("Response : " + msg, ConsoleColor.Green);
                        eventualSuccesses++;

                    });
                }
                catch (Exception e)
                {
                    ConsoleHelper.WriteLineInColor("Request " + i + " eventually failed with: " + e.Message
                    eventualFailures++;
                }
```

WebException was unhandled by user code

An exception of type 'System.Net.WebException' occurred in System.dll but was not handled in user code

Additional information: The remote server returned an error: (503) Server Unavailable.

Troubleshooting tips:

Check the Status property of the exception to determine why the request failed.
Check the Response property of the exception to determine why the request failed.
Get general help for this exception.

Search for more Help Online...

Exception settings:

[✓] Break when this exception type is user-unhandled

Actions:

View Detail...
Enable editing
Copy exception detail to the clipboard
Open exception settings

Further features

Handle multiple exception types in one policy; filter exceptions handled:

```
var policy = Policy.Handle<SqlException>(ex => ex.Number == 1205)
    .Or<TimeoutException>();
```

Register delegates (onRetry, onBreak etc) to capture policy events, eg for logging.

Future roadmap?

Cache policy

Configure from config

Dynamic reconfiguration Tweak timeouts, circuit-breaker sensitivity etc in production

Telemetry Emit eg circuit health, latency to dashboards, for real-time monitoring

App vNext Polly team

Carl Franklin .NET Rocks, Music to Code By

Joel Hulen Enterprise software and cloud architect

Dylan Reisenberger .NET coder and enterprise architect, special interest in microservices, messaging, resilience.

… and all you folks who want to make open-source contributions …

https://github.com/App-vNext/Polly/

Polly Wiki

Extended documentation

Configuration recommendations

Patterns

Future Roadmap

https://github.com/App-vNext/Polly/wiki

Polly Slack channel and Blog

http://www.pollytalk.org/ (slack)

Ask questions

Discuss the roadmap

http://www.thepollyproject.org/ (blog)

Project updates, the inside track

THANK YOU!