Applications infonuagiques résilientes avec Polly

.NET Conf 2019 Quebec City

29 octobre 2019

Tidjani Belmansour

MVP Azure

Architecte de solutions Azure @ Cofomo Québec

Responsable du Centre d'Expertise Microsoft @ Cofomo Québec

http://espacenuagic.com

@Tidjani_B

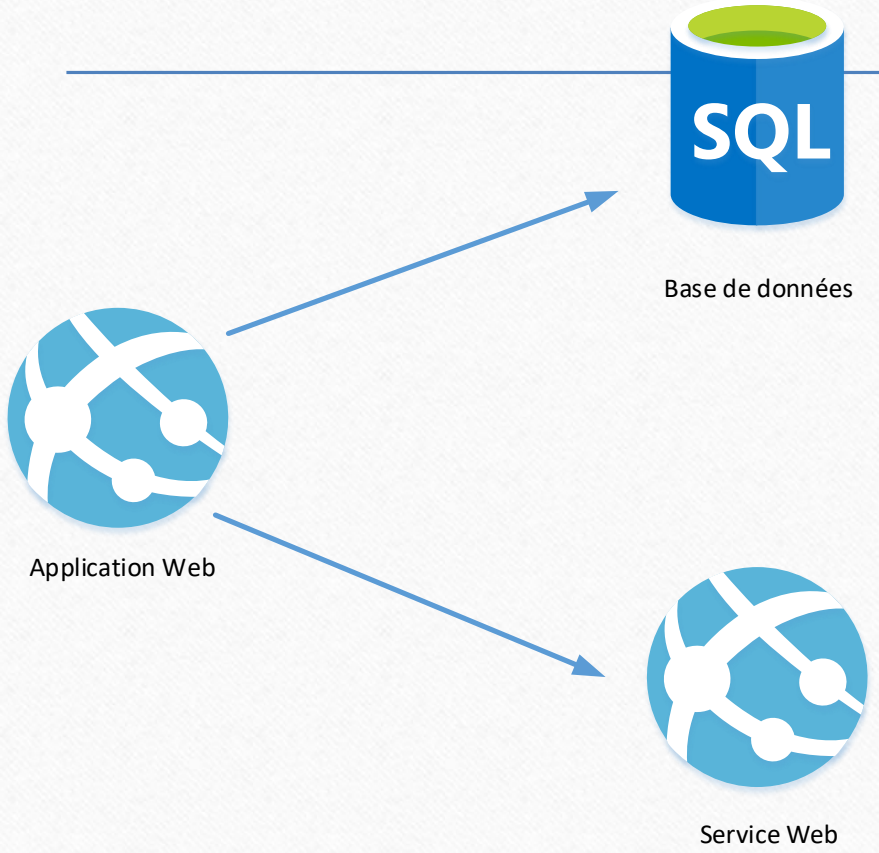
tidjani.belmansour@live.ca

https://ca.linkedin.com/in/tidjani-belmansour-ph-d-98460912

https://github.com/belrarr

Microsoft® MVP Most Valuable Professional

Il était une fois une app…



Il était une fois une app…

http://localhost:21242/Formations/Create

Internal Server Error

localhost:21242/Formations/Create

An unhandled exception occurred while processing the request.

SocketException: Aucune connexion n'a pu être établie car l'ordinateur cible l'a expressément refusée

System.Net.Http.ConnectHelper.ConnectAsync(string host, int port, CancellationToken cancellationToken)

HttpRequestException: Aucune connexion n'a pu être établie car l'ordinateur cible l'a expressément refusée

System.Net.Http.ConnectHelper.ConnectAsync(string host, int port, CancellationToken cancellationToken)

Stack Query Cookies Headers

SocketException: Aucune connexion n'a pu être établie car l'ordinateur cible l'a expressément refusée

System.Net.Http.ConnectHelper.ConnectAsync(string host, int port, CancellationToken cancellationToken)

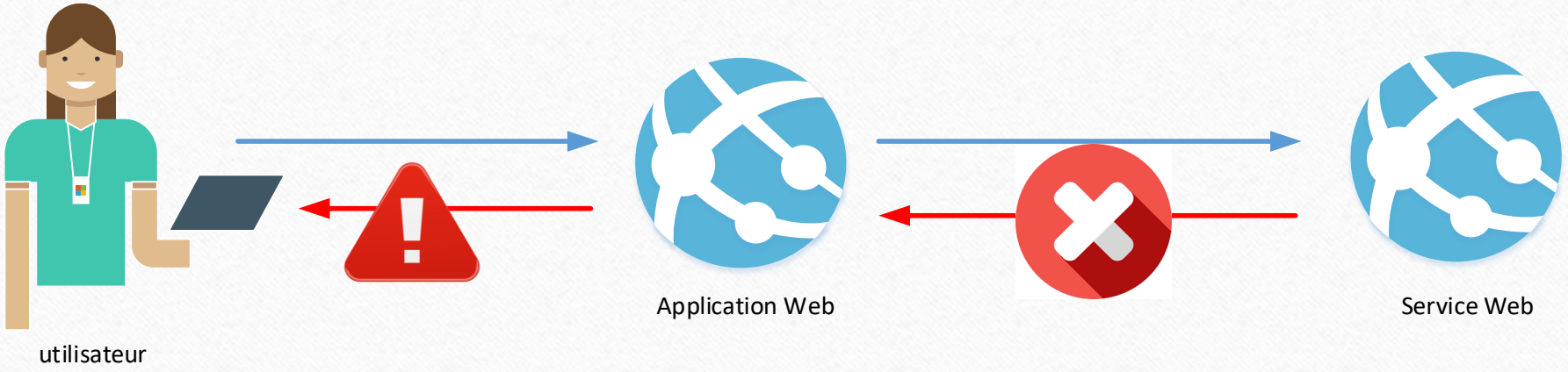
Show raw exception details

HttpRequestException: Aucune connexion n'a pu être établie car l'ordinateur cible l'a expressément refusée

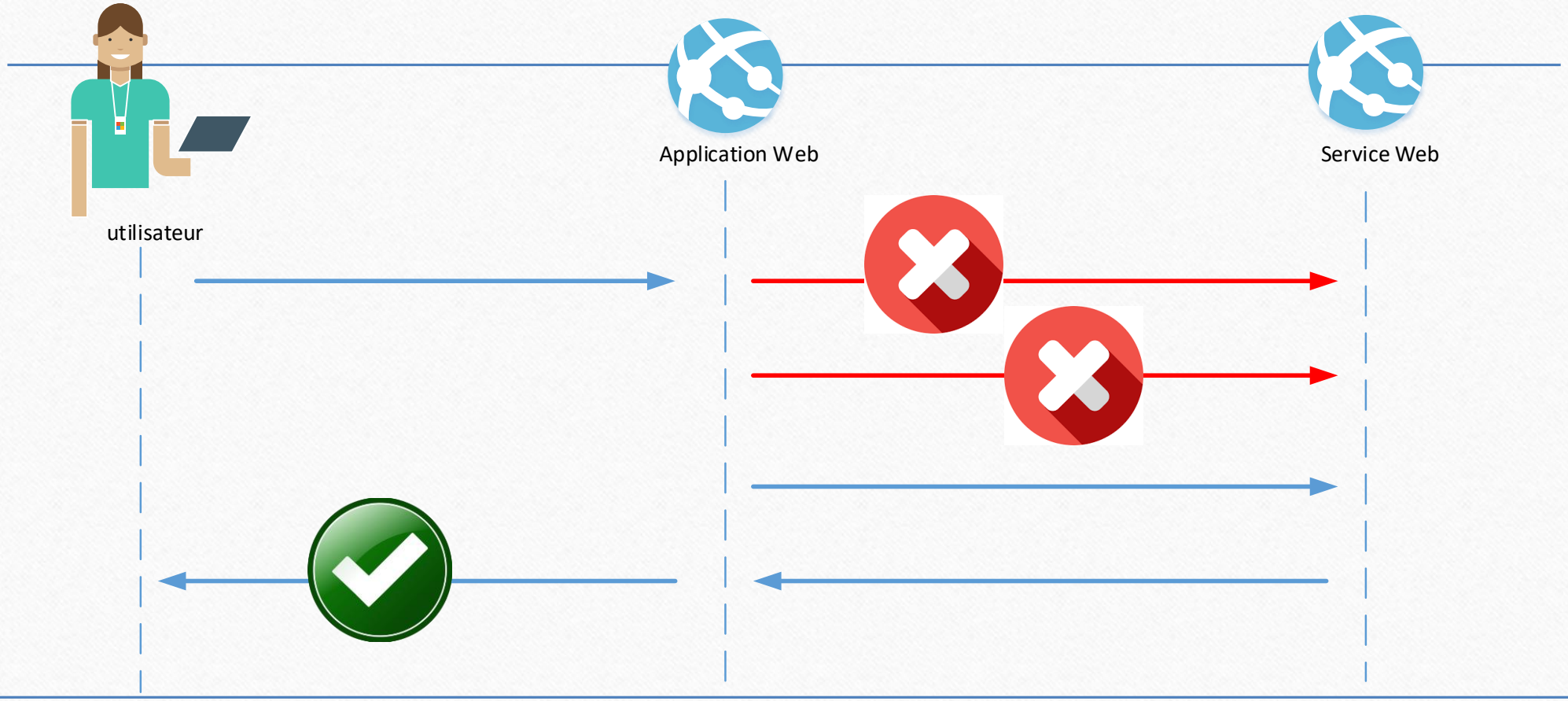
System.Net.Http.ConnectHelper.ConnectAsync(string host, int port, CancellationToken cancellationToken)
System.Threading.Tasks.ValueTask<TResult>.get_Result()
System.Net.Http.HttpConnectionPool.CreateConnectionAsync(HttpRequestMessage request, CancellationToken cancellationToken)
System.Threading.Tasks.ValueTask<TResult>.get_Result()
System.Net.Http.HttpConnectionPool.WaitForCreatedConnectionAsync(ValueTask<ValueTuple<HttpConnection, HttpResponseMessage>> creationTask)
System.Threading.Tasks.ValueTask<TResult>.get_Result()
System.Net.Http.HttpConnectionPool.SendWithRetryAsync(HttpRequestMessage request, bool doRequestAuth, CancellationToken cancellationToken)
System.Net.Http.RedirectHandler.SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)
System.Net.Http.DiagnosticsHandler.SendAsync(HttpRequestMessage request, CancellationToken cancellationToken)
System.Net.Http.HttpClient.FinishSendAsyncBuffered(Task<HttpResponseMessage> sendTask, HttpRequestMessage request, CancellationTokenSource cts, bool disposeCts)
demopolly.Web.Controllers.FormationsController.Create() in FormationsController.cs

```
56.            var response = await _httpClient.GetAsync($"{baseUrl}/api/formateurs");
```

Que s’est-il passé ?



Ce qui aurait dû se passer…



Erreur transitoire: qu’est-ce que c’est ?

Une erreur …

- Éphémère
- Ponctuelle (qui ne dure pas longtemps)
- Souvent imprévisible

Exemples d’erreurs transitoires

- Latence réseau
- Coupure d’internet (ex. wifi)
- Service web en cours de recyclage de pool

Résilience des applications

- Une application est résiliente si elle:
  - Est tolérante aux problèmes
  - Capable de récupérer suite à un problème (se réessayer plutôt que de tout de suite retourner un code d'erreur)
  - Se dégrade de manière gracieuse en cas d'impossibilité de récupérer suite à un problème
  - Désactive certaines fonctionnalités si un problème est détecté

Hello, Polly !

- Est un Framework de résilience
- Mets en place les fonctionnalités de résiliences décrites plus haut

Infonuagique et résilience des applications

- L'infonuagique garanti un SLA
- Pour l'offrir, il faut:
  - Faire du scaling
  - Réapprovisionner des instances en cas d'instabilités
  - Déplacer des instances entre hosts
- Ça peut donc occasionner des pertes ponctuelles de service (généralement, de courte durée)

SOA, microservices et résilience des applications

- Les architectures SOA et (micro/mini)services reposent sur le découplage de services
  - Leurs cycles de vie sont donc différents
  - Leurs disponibilités peuvent donc aussi être différentes
  - D'où la nécessité que ces applications / services soient résilients

Stratégies dans Polly

- **Réactive**
  - Répondre à une solution problématique
- **Proactive**
  - Surveillance de potentiels problèmes et agir en amont

Stratégies réactives

- **Retry**
  - Se réessayer un certain nombre de fois, sans délai entre les tentatives
- **Wait And Retry**
  - Se réessayer un certain nombre de fois, avec un délai entre les tentatives
- **Fallback**
  - « Si toute autre stratégie échoue, faire ceci pour une dégradation gracieuse … »
  - Utilisé conjointement avec d'autres stratégies

Stratégies réactives

- **Circuit Breaker**
  - Empêcher toute requête à une dépendance en problème

Stratégies proactives

- **Timeout**
  - Mettre fin à une requête lorsque souhaité
- **Caching**
  - Stocker la réponse précédente et la servir en cas de problème
- **Bulkhead isolation**
  - Empêcher une situation problématique de faire tomber tout le système

Ajouter Polly à notre projet Web

NuGet : demopolly.Web

Parcourir | Installé | Mises à jour 2

Gestionnaire de package NuGet : demopolly.Web

polly

Inclure la version préliminaire

Source de package : nuget.org

Polly par Michael Wolfenden, App vNext, **22,9M** téléchargements — v7.1.1
Polly is a library that allows developers to express resilience and transient fault handling policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback in a fluent and thread-safe manner.

Polly.Extensions.Http par App vNext, **3,53M** téléchargements — v3.0.0
Polly.Extensions.Http is an extensions package containing opinionated convenience methods for configuring Polly policies to handle transient faults typical of calls through HttpClient.

Microsoft.Extensions.Http.Polly par Microsoft, **3,27M** téléchargements — v3.0.0
The HttpClient factory is a pattern for configuring and retrieving named HttpClients in a composable way. This package integrates IHttpClientFactory with the Polly library, to add transient-fault-handling and resilie...

Polly-Signed par Michael Wolfenden, App vNext, **732K** téléchargements — v5.9.0
Polly is a .NET 4.5 / .NET Standard 1.1 library that allows developers to express resilience and transient fault handling policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation and Fallback in a fluent and th...

Polly.Caching.Memory par App vNext, **163K** téléchargements — v3.0.2
Polly.Caching.Memory is a plug-in for the .NET OSS resilience library Polly, supporting Microsoft.Extensions.Caching.Memory.MemoryCache as a provider for Polly's CachePolicy.

Polly — nuget.org

Version : Dernière version stable 7.1.1 — Installer

Options

Description

Polly is a library that allows developers to express resilience and transient fault handling policies such as Retry, Circuit Breaker, Timeout, Bulkhead Isolation, and Fallback in a fluent and thread-safe manner.

Version : 7.1.1
Auteur(s) : Michael Wolfenden, App vNext
Licence : BSD-3-Clause
Date de publication : lundi 2 septembre 2019 (02/09/2019)

Configurer une Policy

Timeout:

```
var policy = Policy.Timeout(dureeTimeout);
```

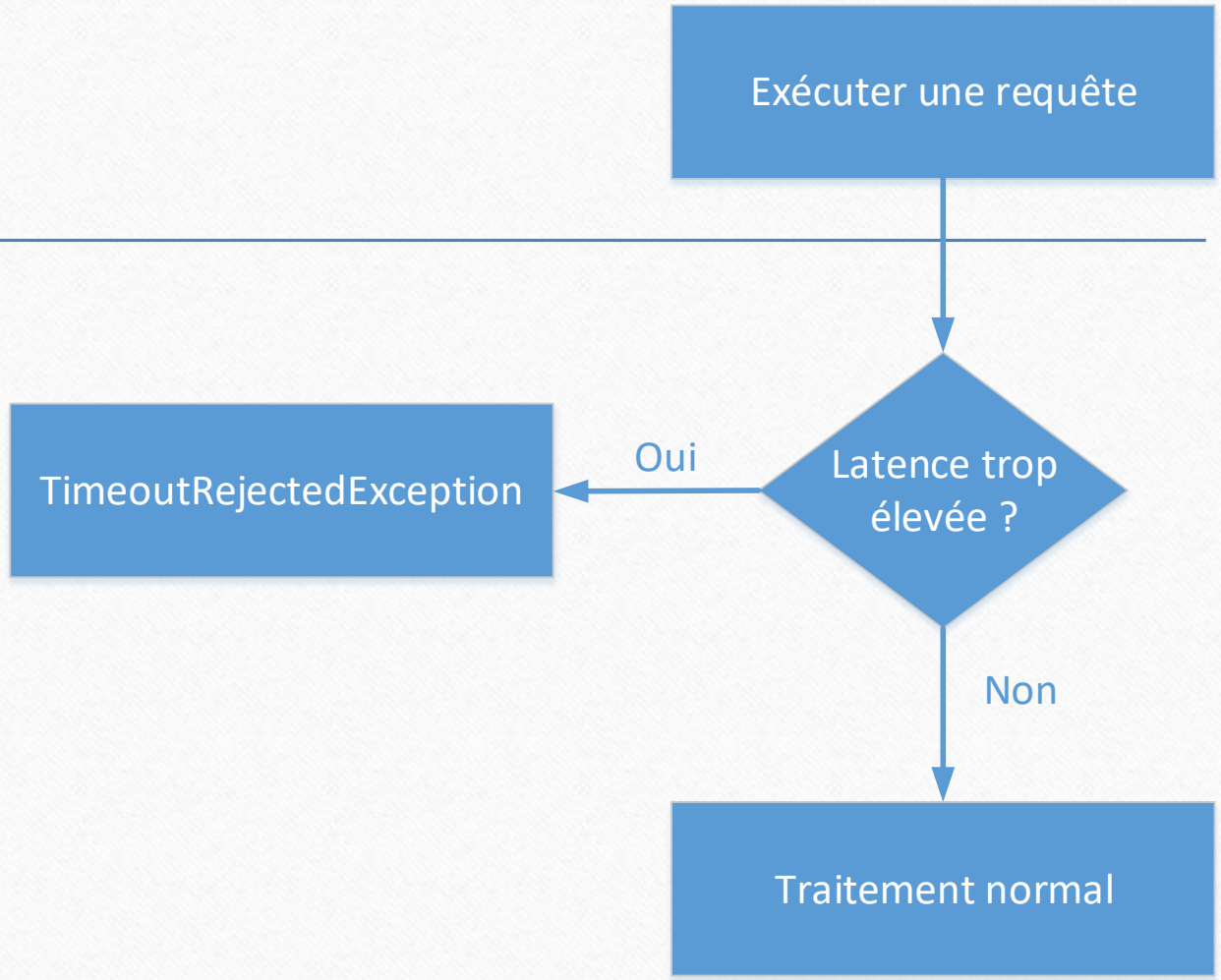
Autres policies:

```
var policy =
    Policy.HandleResult<TypeResultat>().TypePolicy(parametresDeLaPolicy);
```

Utiliser une Policy

```
var retour = policy.Execute(expressionLambda);
```

Timeout Policy



Sans Polly

```
// GET: Formations/TimeoutPolicy
public async Task<IActionResult> TimeoutPolicy()
{
    var formateurs = new List<string>();

    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/123";
    var response = await _httpClient.GetAsync(url);

    if (response.IsSuccessStatusCode)
    {
        var formateur = await response.Content.ReadAsStringAsync();
        formateurs.Add(formateur);
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Avec Polly

```
// GET: Formations/TimeoutPolicy
public async Task<IActionResult> TimeoutPolicy()
{
    // lève un TimeoutRejectedException après 3 secondes.
    var timeoutPolicy = Policy.TimeoutAsync(3);

    var formateurs = new List<string>();
    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/123";

    var response = timeoutPolicy.ExecuteAsync(async token => await _httpClient.GetAsync(url, token),
                                              System.Threading.CancellationToken.None);
    if (response.IsSuccessStatusCode)
    {
        formateurs.Add(await response.Content.ReadAsStringAsync());
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Internal Server Error

localhost:21242/Formations/TimeoutPolicy

An unhandled exception occurred while processing the request.

AggregateException: One or more errors occurred. (The delegate executed asynchronously throug

System.Threading.Tasks.Task<TResult>.GetResultCore(bool waitCompletionNotification)

TimeoutRejectedException: The delegate executed asynchronously through TimeoutPolicy did not

Polly.Timeout.AsyncTimeoutEngine.ImplementationAsync<TResult>(Func<Context, CancellationToken, Task<TResult>> action, Context timeoutStrategy, Func<Context, TimeSpan, Task, Exception, Task> onTimeoutAsync, bool continueOnCapturedContext)

Stack Query Cookies Headers

AggregateException: One or more errors occurred. (The delegate executed asynchronously throug

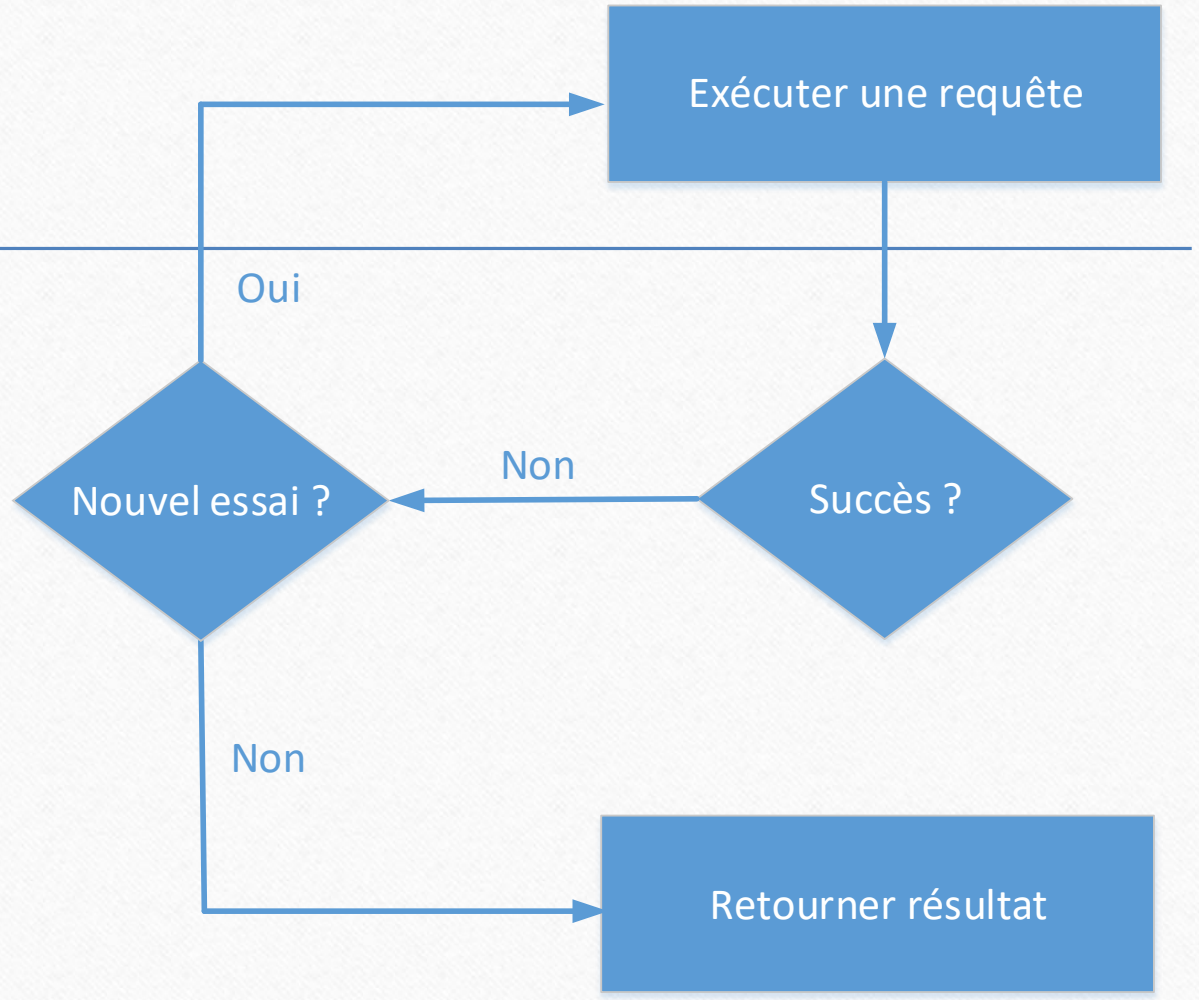
System.Threading.Tasks.Task<TResult>.GetResultCore(bool waitCompletionNotification)

demopolly.Web.Controllers.FormationsController.TimeoutPolicy() in FormationsController.cs

```
+      58.            if (response.Result.IsSuccessStatusCode)
```

Microsoft.AspNetCore.Mvc.Internal.ActionMethodExecutor+TaskOfIActionResultExecutor.Execute(IActionResultTypeMappe

Retry Policy



Sans Polly

```
// GET: Formations/RetryPolicy
public async Task<IActionResult> RetryPolicy()
{
    var formateurs = new List<string>();

    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/404";
    var response = await _httpClient.GetAsync(url);

    if (response.IsSuccessStatusCode)
    {
        var formateur = await response.Content.ReadAsStringAsync();
        formateurs.Add(formateur);
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Avec Polly

```
// GET: Formations/RetryPolicy
public async Task<IActionResult> RetryPolicy()
{
    var retryPolicy = Policy.HandleResult<HttpResponseMessage>(r => !r.IsSuccessStatusCode)
                            .RetryAsync(3);

    var formateurs = new List<string>();
    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/404";

    var response = await retryPolicy.ExecuteAsync(() => _httpClient.GetAsync(url));

    if (response.IsSuccessStatusCode)
    {
        formateurs.Add(await response.Content.ReadAsStringAsync());
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Create - demopolly.Web

localhost:21242/Formations/RetryPolicy

Use this space to summarize your privacy and cookie use policy.

Formation

Titre

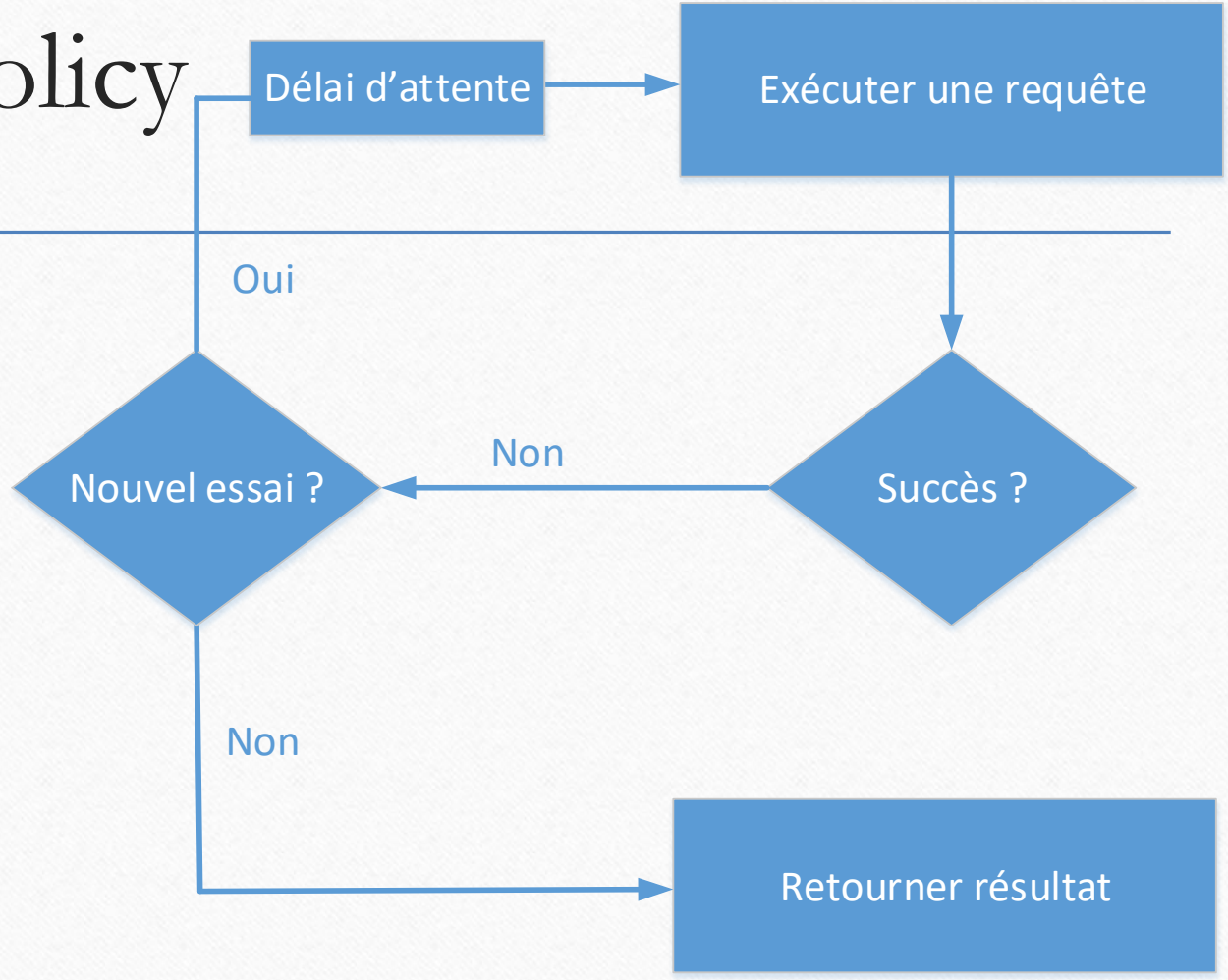
Presentateur -- Sélectionner un formateur --

-- Sélectionner un formateur --

Code

Description

Wait and Retry Policy



Sans Polly

```
// GET: Formations/WaitAndRetryPolicy
public async Task<IActionResult> WaitAndRetryPolicy()
{
    var formateurs = new List<string>();

    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/404";
    var response = await _httpClient.GetAsync(url);

    if (response.IsSuccessStatusCode)
    {
        var formateur = await response.Content.ReadAsStringAsync();
        formateurs.Add(formateur);
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Avec Polly

```
// GET: Formations/WaitAndRetryPolicy
public async Task<IActionResult> WaitAndRetryPolicy()
{
    var waitAndRetryPolicy = Policy.HandleResult<HttpResponseMessage>(r => !r.IsSuccessStatusCode)
                                .WaitAndRetryAsync(3, essai => TimeSpan.FromSeconds(Math.Pow(2, essai)));

    var formateurs = new List<string>();
    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/404";

    var response = await waitAndRetryPolicy.ExecuteAsync(() => _httpClient.GetAsync(url));

    if (response.IsSuccessStatusCode)
    {
        formateurs.Add(await response.Content.ReadAsStringAsync());
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Create - demopolly.Web

localhost:21242/Formations/RetryPolicy

Use this space to summarize your privacy and cookie use policy.

Formation

Titre

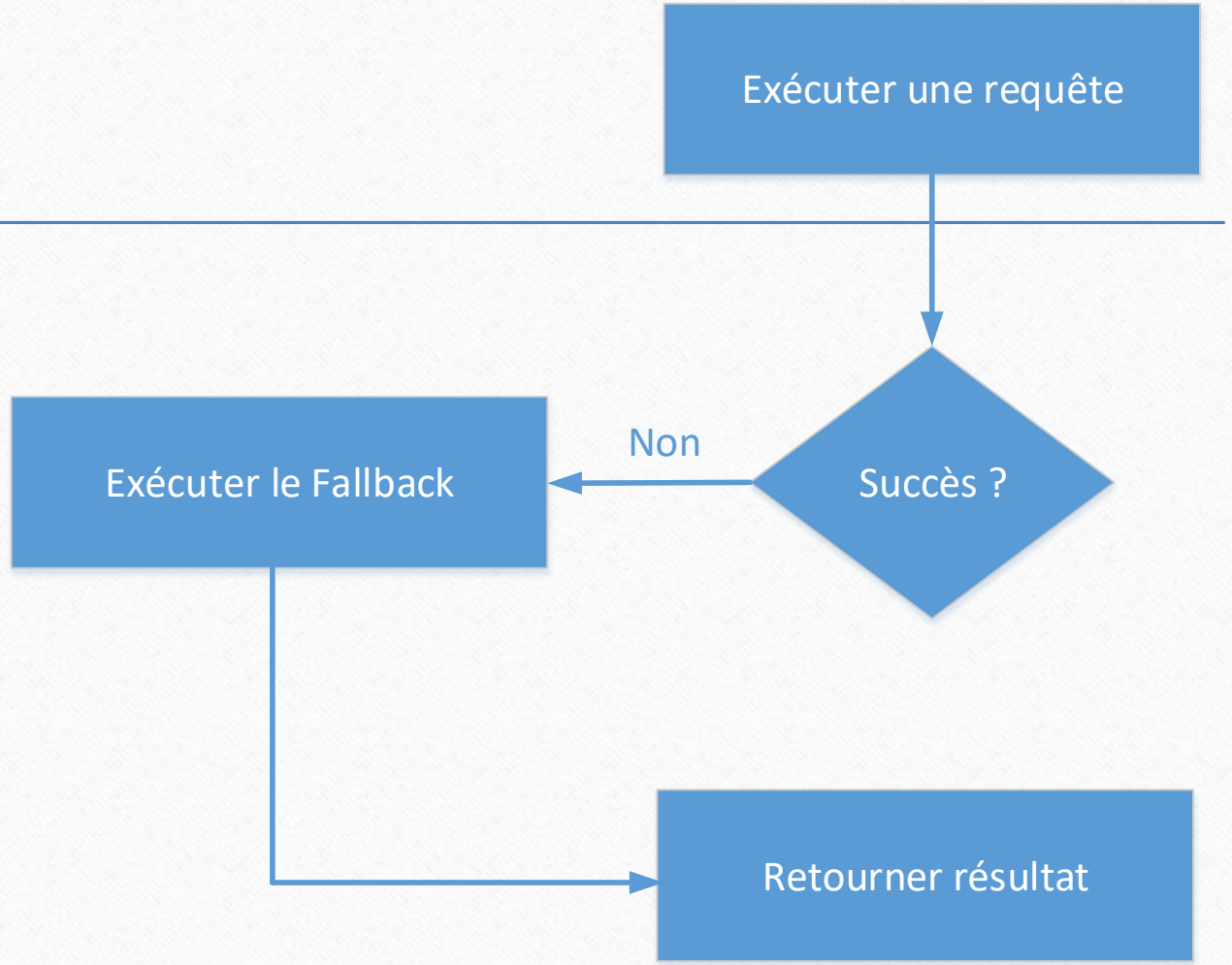
Presentateur -- Sélectionner un formateur --

-- Sélectionner un formateur --

Code

Description

Fallback Policy



Sans Polly

```
// GET: Formations/FallbackPolicy
public async Task<IActionResult> FallbackPolicy()
{
    var formateurs = new List<string>();

    var baseUrl = _configuration["webserviceBaseUrl"];
    var url = $"{baseUrl}/api/formateurs/500";
    var response = await _httpClient.GetAsync(url);

    if (response.IsSuccessStatusCode)
    {
        var formateur = await response.Content.ReadAsStringAsync();
        formateurs.Add(formateur);
    }

    ViewBag.Formateurs = formateurs;
    return View("Create");
}
```

Avec Polly

```
// GET: Formations/FallbackPolicy
public async Task<IActionResult> FallbackPolicy()
{
     var retryPolicy = Policy.HandleResult<HttpResponseMessage>(r => !r.IsSuccessStatusCode)
                                 .RetryAsync(3);

      var fallbackPolicy = Policy
     .HandleResult<HttpResponseMessage>(r => r.StatusCode == System.Net.HttpStatusCode.InternalServerError)
     .FallbackAsync(new HttpResponseMessage(System.Net.HttpStatusCode.OK)
     {
         Content = new StringContent("Scott Hanselman")
     });

      // ...

      var response = await fallbackPolicy
                           .ExecuteAsync(() => retryPolicy.ExecuteAsync(() => _httpClient.GetAsync(url)));

      // même code que précédemment. Retiré pour alléger le contenu de la slide.
}
```

Create - demopolly.Web

localhost:21242/Formations/FallbackPolicy

Use this space to summarize your privacy and cookie use policy.

Create

Formation

Titre

Presentateur -- Sélectionner un formateur --

- -- Sélectionner un formateur --
- Scott Hanselman

Code

Description

Supporter plusieurs erreurs

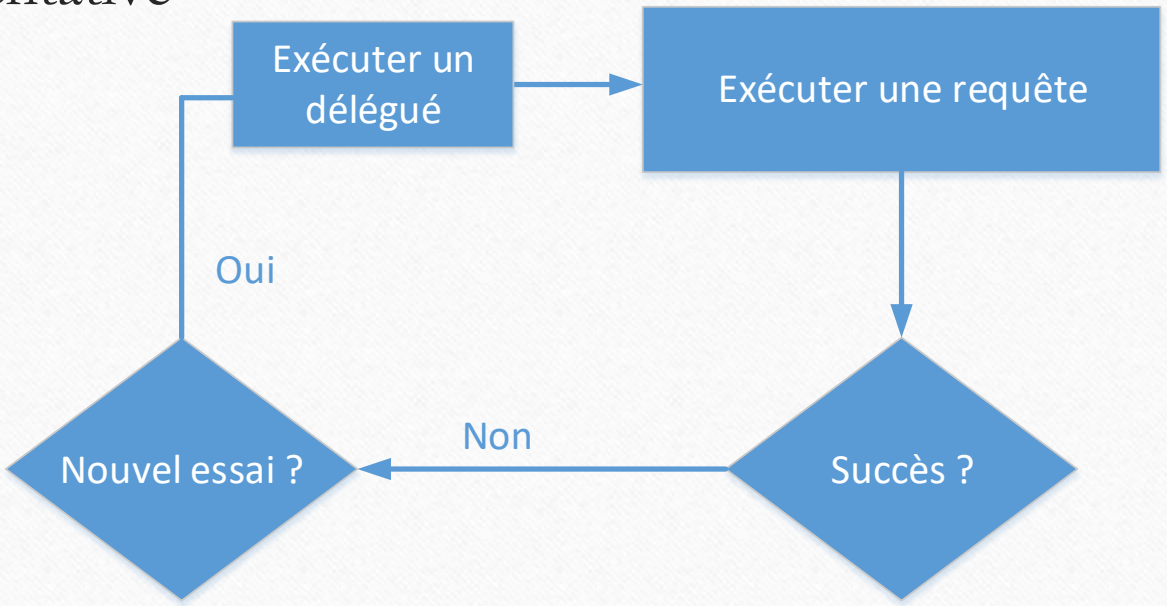
```
public async Task<IActionResult> RetryPolicy()
{
   var waitAndRetryPolicy = Policy
                              .HandleResult<HttpResponseMessage>(r => !r.IsSuccessStatusCode)
                              .Or<System.Net.Sockets.SocketException>()
                              . RetryAsync(3);

        // même code que précédemment. Retiré pour alléger le contenu de la slide.
}
```

Les délégués

- Permettent d'effectuer un traitement juste avant l'exécution de la prochaine tentative de la requête.

- Use cases:
  - Logging
  - Réauthentification si token expiré
  - …



Les délégués

```
public async Task<IActionResult> RetryPolicyWithDelegate()
{
     var retryPolicy = Policy.HandleResult<HttpResponseMessage>(r => !r.IsSuccessStatusCode)
                              .RetryAsync(3, onRetry: (httpResponseMessage, retryCount) =>
                               {
                                   if (httpResponseMessage.Result.StatusCode == HttpStatusCode.NotFound)
                                   {
                                         // logguer l'erreur
                                   }
                                   else if(httpResponseMessage.Result.StatusCode == HttpStatusCode.Forbidden)
                                   {
                                       // rediriger vers la page d'accueil
                                   }
                                });

     // même code que précédemment. Retiré pour alléger le contenu de la slide.
}
```

La résilience ne s'applique qu'aux applications informatiques

FAUX

La résilience s'applique à **toutes** les applications

- L'application communique avec des dépendances externes ? ➔ Résilience
- Les applications qui s'exécutent on-premises ou sur le Cloud sont sujettes à des erreurs transitoires
- Concept trop souvent oublié et non implémenté
  - On considère que c'est difficile à mettre en place
- Polly simplifie la mise en place de la résilience

Applications résilientes avec Polly

.NET Conf 2019 Quebec City

29 octobre 2019

Pour aller plus loin…

- Polly s'utilise aussi avec .NET Framework
  - Parce que les applications .NET Framework ne disparaîtront pas du jour au lendemain…
- Autres concepts à explorer:
  - Circuit Breaker policy
  - Caching policy
  - Bulkhead Isolation policy
  - Combiner des policies
  - Unit Testing Policies
  - PolicyWrap
  - Réutilisation et diffusion de policies
  - Contexte

Polly ou Topaz ?

17

Answering as [disclosure] the lead dev on Polly under the App-vNext umbrella. I'll confine myself to factual answers to your q about support from the internet community and future prospects for the library:

- In December 2015, stewardship of Polly was taken over by http://www.appvnext.com/, a consortium of developers from around the globe led by Carl Franklin of dotnetrocks
- Between December 2015 and April 2016, the App-vNext team made 10 new releases of the Polly library. New features include full async support (including cancellation and continuation context control), and a new AdvancedCircuitBreaker for high-throughput scenarios.
- The future project Roadmap is public at https://github.com/App-vNext/Polly/wiki/Polly-Roadmap. Envisaged features see Polly grow towards a more fully-fledged fault tolerance library, as Hystrix is for Java.
- **EDIT AUTUMN 2016**: Polly v5.0 has added a number of new resilience features: Bulkhead isolation; Fallback; Timeout any delegate; PolicyWrap.
- **EDIT**: Polly is (since end October 2016) a DotNetFoundation-supported project.
- **EDIT May 2017:** Microsoft patterns-and-practices team officially recommend Polly in place of Topaz in certain scenarios.
- **EDIT January 2018:** Microsoft plan to make it easy to use Polly as outgoing middleware for `HttpClient` calls, in the `HttpClientFactory` planned for .NET Core 2.1.
- **EDIT May 2018:** ASPNET Core 2.1 released with Polly as outgoing middleware for calls through `HttpClient` configured with `IHttpClientFactory` : ms doco; Polly doco.

Questions / Commentaires