Les “Counters” en .NET (Core)

Christophe Nasarre
@chnasarre

DATADOG

Les counters : comment ça marche ?

Introduction

Meilleurs (performance) counters

Ecouter

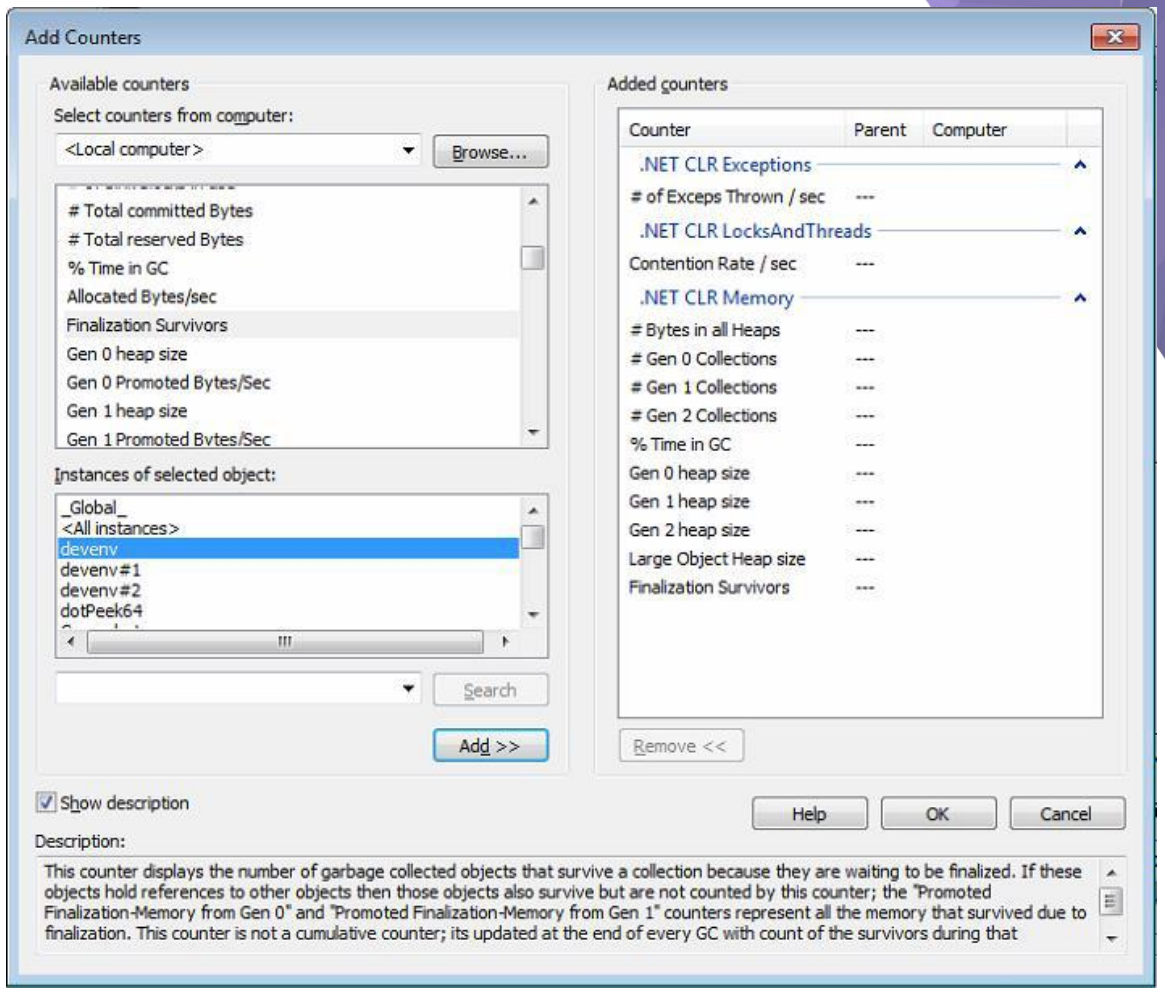
EventPipe et son orchestre

Participer

Construire ses propres counters

Comment mesurer les perfs de nos applications ?

- Compteurs de Performance
  - Beaucoup de détails pour le .NET Framework
  - … mais aussi des mauvais
    (*Gen 0 Size, Gen 0/1 counts, Thread count, …*)
  - Seulement pour Windows
  - Un petit nombre pour démarrer une investigation…



.NET Core CLI - *dotnet-counters/trace/dump/gcdump*

- Installé “facilement” … si le.NET SDK (3.0+) est déjà là
  - `dotnet tool list -g`
  - `dotnet tool update <dotnet-XXX> -g`
  - Pas simple d’installer le SDK en production/container…

- …ou téléchargé depuis les pages https://learn.microsoft.com/en-us/dotnet/core/diagnostics/
  - Plus simple à déployer dans des containeurs

DEMO: dotnet counters monitor –p <pid>

Au delà de dotnet counters

- Parce que **`dotnet counters monitor`** est peu “utilisable”…
  …pour nourrir votre pipeline de surveillance

- **DEMO**: **`collect`** → .csv

- **DEMO**: **en C# !**

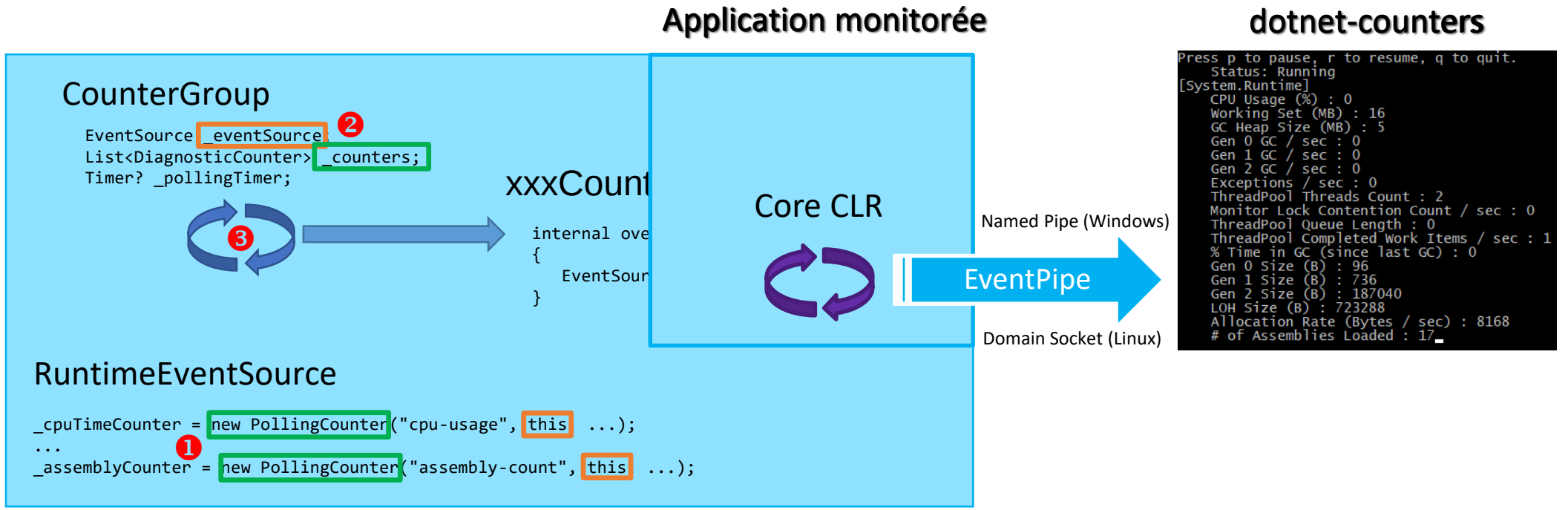
- Pensez aux events CLR
  - Lire mes billets pour les details

Counter	API	Type
cpu-usage	RuntimeEventSourceHelper.GetCpuUsage()	Mean
working-set	Environment.WorkingSet / 1000000	Mean
gc-heap-size	GC.GetTotalMemory(false) / 1000000	Mean
gen-0-gc-count	GC.CollectionCount(0)	Sum
gen-1-gc-count	GC.CollectionCount(1)	Sum
gen-2-gc-count	GC.CollectionCount(2)	Sum
exception-count	Exception.GetExceptionCount()	Sum
threadpool-thread-count	ThreadPool.ThreadCount	Mean
monitor-lock-contention-count	Monitor.LockContentionCount	Sum
threadpool-queue-length	ThreadPool.PendingWorkItemCount	Mean
threadpool-completed-items-count	ThreadPool.CompletedWorkItemCount	Sum
time-in-gc	GC.GetLastGCPercentTimeInGC()	Mean
gen-0-size	GC.GetGenerationSize(0)	Mean
gen-1-size	GC.GetGenerationSize(1)	Mean
gen-2-size	GC.GetGenerationSize(2)	Mean
loh-size	GC.GetGenerationSize(3)	Mean
alloc-rate	GC.GetTotalAllocatedBytes()	Sum
assembly-count	System.Reflection.Assembly.GetAssemblyCount()	Mean

Sous le capot des “counters” .NET Core

- Les counters CLR et ASP.NET Core dérivent de `DiagnosticCounter`
  - `EventCounter`: min/max/moy basés sur une valeur
  - `IncrementingEventCounter`: incrémente une valeur

  - `PollingCounter`: min=max=moy basé sur une valeur calculée par une callback
  - `IncrementingPollingCounter`: incrémente une valeur calculée par une callback



Ecrire vos propres “counters” .NET Core en C#

1. Dériver un type de `EventSource` et lui donner un nom
2. Créer ses counters dans le constructeur
   - Choisir entre `EventCounter` et `PollingCounter`
3. Mettre à jour `EventCounter` avec `WriteMetric()`
4. Mettre à jour les chiffres retournés par `PollingCounter` callbacks
   - Soyez “thread safe” !
5. Utiliser le nom de l’event source comme provider dans *dotnet-counters*

- **DEMO: compter les requêtes subissant (ou pas) une garbage collection**

```
dotnet counters monitor -p <pid> Sample.RequestCounters
```

Ressources

Documentation & code source

- https://github.com/microsoft/dotnet-samples/tree/master/Microsoft.Diagnostics.Tracing/TraceEvent
- Blog series https://chnasarre.medium.com

  (code source https://github.com/chrisnas/ClrEvents)
- Code source de la CLR https://github.com/dotnet/runtime

Outils

- PerfView https://github.com/microsoft/perfview
- gcmon https://github.com/Maoni0/realmon

  → Fabriquez vos propres outils CLI :^)

DATADOG