criteo

Christophe Nasarre @chnasarre

.NET Monitoring pipelines

AGENDA

ETW and CLR Events

- Better performance counters
- Inside the CLR
- Main events to monitor

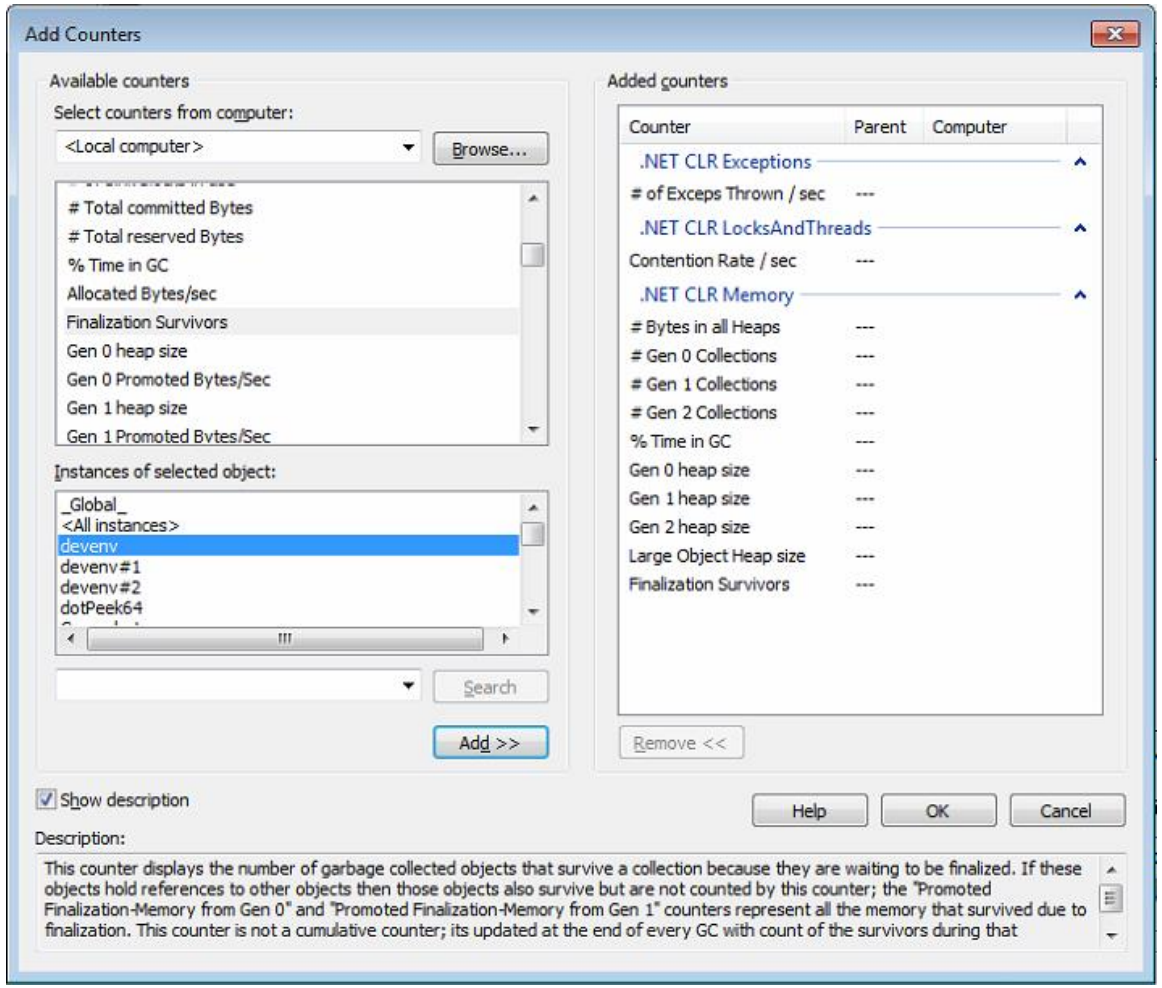
.NET Core 3.0 and EventPipes

- EventPipe architecture
- New tooling
- Listen to events
- Build your own counters

criteo

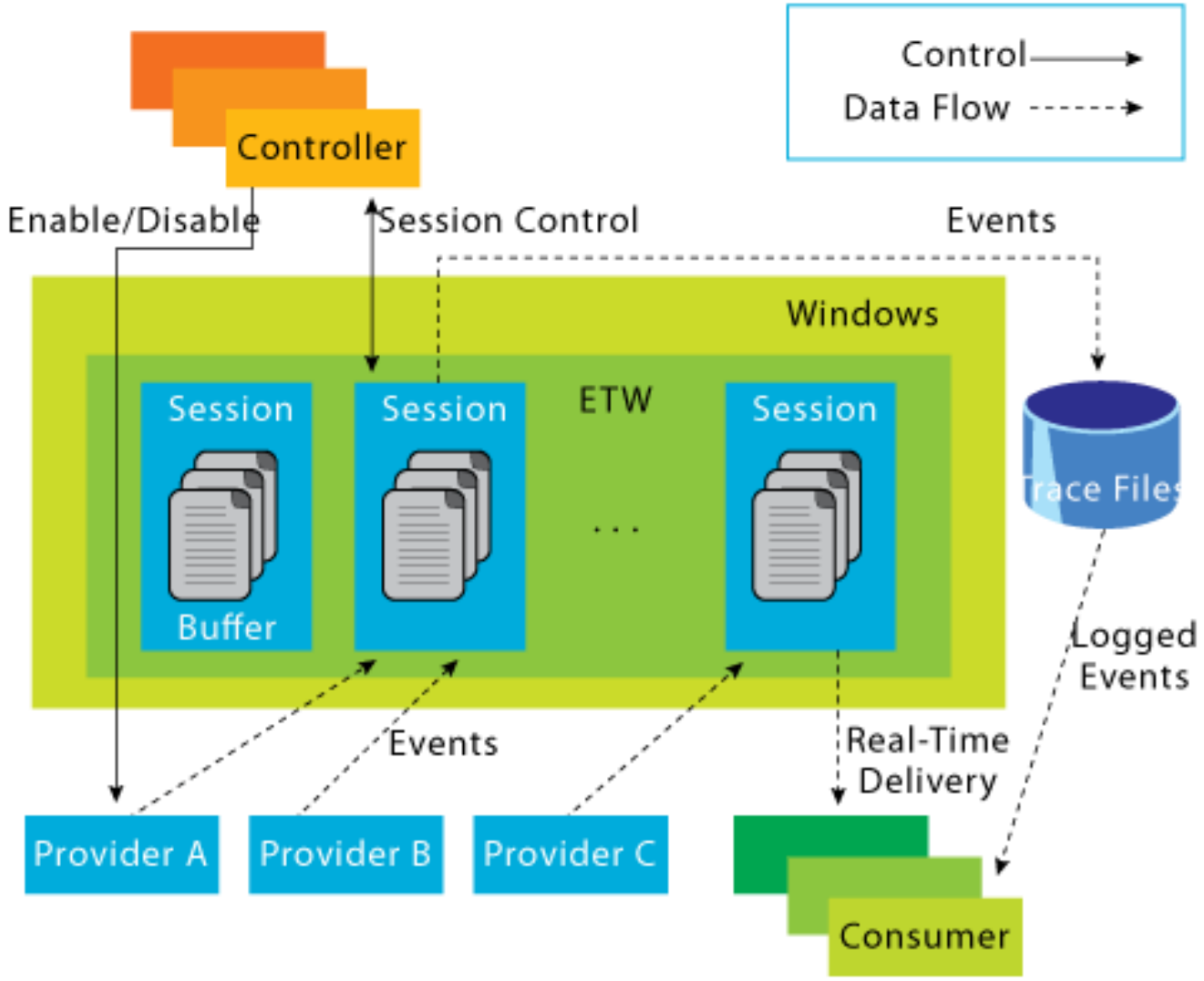
How to measure your application performances

- Performance counters
  - A lot of interesting details for .NET
  - … but also wrong ones
    (*Gen 0 Size, Gen 0/1 counts, Thread count, …*)
  - Windows only
  - Only a few are usable to start an investigations



Event Tracing for Windows (a.k.a. **ETW**) architecture

- Kernel logging system
  - Very low impact on production…
- Many providers
  - Including .NET (with documentation)
- Create a session and listen
  - … to **all processes** traces



criteo

CLR events internals

- All events are XML-described
  - https://github.com/dotnet/coreclr/blob/master/src/vm/ClrEtwAll.man
  - … but some embedded data might be missing with event pipes
    (ex *GCPerHeapHistory issue*)

- The CLR is supporting keyword (=category) and level filtering
  - … but might not be perfect and have impact on performance
  - Additional filtering for GC events

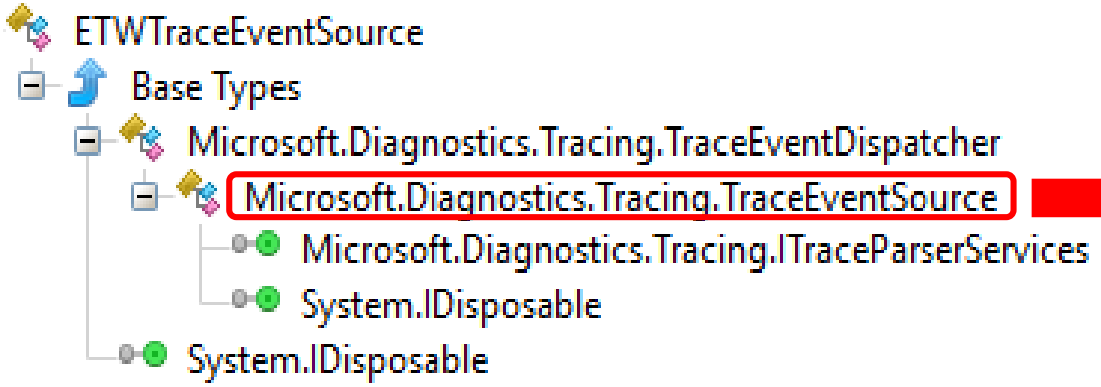
- Great way to better understand how the CLR is working!
  - Look for **FireEtwXXX** helper functions

- DEMO: collecting and viewing events with **PerfView**

criteo

How to listen to CLR events in C#: **TraceEvent** is your friend!

- TraceEvent is available as nuget but also from GitHub (Perfview repo)
  - https://www.nuget.org/packages/Microsoft.Diagnostics.Tracing.TraceEvent/
  - https://github.com/microsoft/perfview/tree/master/src/TraceEvent




```
public RegisteredTraceEventParser Registered { get; }
public DynamicTraceEventParser Dynamic { get; }
public ClrTraceEventParser Clr { get; }
public KernelTraceEventParser Kernel { get; }
```

- DEMO: collecting and viewing events in C# code

criteo

Interesting CLR events (1/2)

- Exceptions thrown and caught
  - ExceptionThrown
- Thread contention duration
  - ContentionStart and ContentionStop
- ThreadPool starvation
  - ThreadPoolWorkerThreadAdjustmentAdjustment

criteo

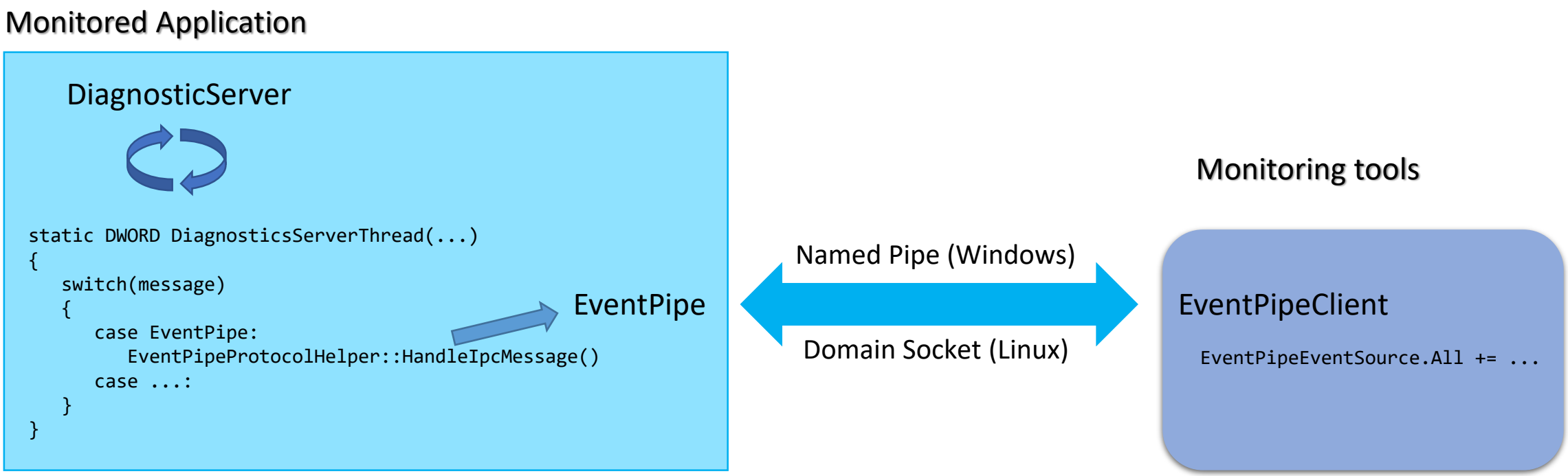
Interesting CLR events (2/2)

- Called finalizers
  - TypeBulkType and GCFinalizeObject
- Every 100 KB allocations *(could be expensive)*
  - GCAllocationTick
- GC (Suspension + Pause) duration
  - GCSuspendEEBegin and GCRestartEEEnd
- GC type, condemned generation and gens size
  - GCStart, GCHeapStats, and GCGlobalHeapHistory

criteo

.NET Core and **EventPipe** architecture

- Dedicated listener thread spawn by CLR
  - Listen to session creation message from listener
  - Create **EventPipe** to allow 2-way communication (*No need to know the IPC protocol*)



DEMO: namepipe on Windows with **WinObjEx**

.NET Core and tooling – *dotnet-trace*, *dotnet-counters*, and… *dotnet-dump*!

- Installed "easily"… when .NET SDK 3.0 is already there
  - `dotnet tool list -g`
  - `dotnet tool update <dotnet-XXX> -g`
- Or recompiled from https://github.com/dotnet/diagnostics
  - Could be easier if you need to deploy in containers
  - Beware the changes between Previews (and probably next versions)
- Use different syntaxes: dotnet-XXX or dotnet XXX
  - Tools are installed under C:\Users\<account>\.dotnet\tools
  - Versioning under C:\Users\<account>\.dotnet\tools\.store\dotnet-trace\3.0.47001

DEMO: using the tools

criteo

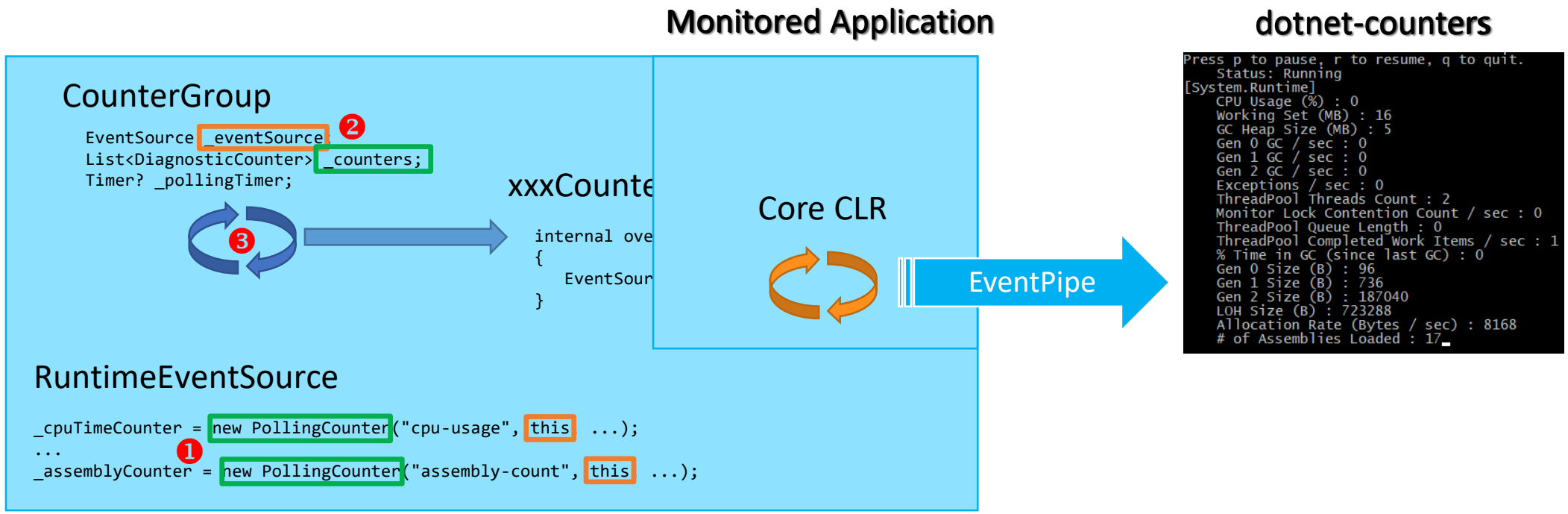
.NET Core and tooling – getting the traces in C#

- TraceEvent support for event pipes
  - via `EventPipeEventSource` and a stream return by `EventPipeClient.CollectTracing`
    - same parsing code than ETW-based event tracing
  - It is also possible to receive events in-proc with `EventListener`
- Work for both Windows and Linux
- Work both in-process and out-of-process
- DEMO: receiving events in-proc with **EventListener**

criteo

Under the hood of .NET Core “counters”

- CLR and ASP.NET Core counters inherit from `DiagnosticCounter`
  - `EventCounter`: min/max/mean based on a value
  - `IncrementingEventCounter`: increment of a value
  - `PollingCounter`: min=max=mean based on a value computed in a callback
  - `IncrementingPollingCounter`: increment of a value computed in a callback



Listening to .NET Core “counters” in C#

- Because you need to feed your monitoring pipeline
- Because dotnet-counters in not really “usable”…
- Because it is easy :^)

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

- DEMO: look at the code!

criteo

Writing your own .NET Core “counters” in C#

1. Derive a type from `EventSource` and give it a name
2. Create counters in its `OnEventCommand`
   - In the `EventCommand.Enable` message processing
   - Pick between `EventCounter` and `PollingCounter`
3. Update `EventCounter` with `WriteMetric()`
4. Update numbers used in `PollingCounter` callbacks
   - Be thread safe!
5. Use the event source name as provider in **dotnet-counters**

- DEMO: show “Request with(out) GC” counters sample

criteo

Resources

Documentation & source code

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

  (source code https://github.com/chrisnas/ClrEvents)
- Core CLR source code https://github.com/dotnet/coreclr

Tools

PerfView https://github.com/microsoft/perfview