

# SELF-AWARE APPLICATIONS AUTOMATIC PRODUCTION DIAGNOSIS

DINA GOLDSHTEIN



# Agenda

- \* Motivation
- \* Hierarchy of self-monitoring
- \* CPU profiling
- \* GC monitoring
- \* Heap analysis
- \* Deadlock detection

# Agenda

- \* Motivation
- \* Hierarchy of self-monitoring
- \* CPU profiling
- \* GC monitoring
- \* Heap analysis
- \* Deadlock detection
- \* Not today: profiling tools, 3rd party monitoring, dashboards

# Motivation

- \* Why monitor?
  - \* Obviously a must for servers and/or anything at scale

# Motivation

- \* Why monitor?
  - \* Obviously a must for servers and/or anything at scale
  - \* But also a must for “simple” shrink-wrap software

# Motivation

- \* Why monitor?
  - \* Obviously a must for servers and/or anything at scale
  - \* But also a must for “simple” shrink-wrap software
- \* Why our own?

# Motivation

- \* Why monitor?
  - \* Obviously a must for servers and/or anything at scale
  - \* But also a must for “simple” shrink-wrap software
- \* Why our own?
  - \* Customized for specific business needs

# Motivation

- \* Why monitor?
  - \* Obviously a must for servers and/or anything at scale
  - \* But also a must for “simple” shrink-wrap software
- \* Why our own?
  - \* Customized for specific business needs
  - \* Diagnostics flow from ground up

# Motivation

- \* Why monitor?
  - \* Obviously a must for servers and/or anything at scale
  - \* But also a must for “simple” shrink-wrap software
- \* Why our own?
  - \* Customized for specific business needs
  - \* Diagnostics flow from ground up
  - \* That’s what all the cool kids do :-)

# Master Plan - Use Hierarchy!

# Master Plan - Use Hierarchy!

- \* **Lightweight** for continuous and frequent monitoring of all basics
  - \* Numerical resource consumption: CPU, memory, disk
  - \* Performance counters, Win32 APIs

# Master Plan - Use Hierarchy!

- \* **Lightweight** for continuous and frequent monitoring of all basics
  - \* Numerical resource consumption: CPU, memory, disk
  - \* Performance counters, Win32 APIs
- \* **Medium** for less frequency events
  - \* Rare exceptions, deadlocks
  - \* ETW, ClrMD

# Master Plan - Use Hierarchy!

- \* **Lightweight** for continuous and frequent monitoring of all basics
  - \* Numerical resource consumption: CPU, memory, disk
  - \* Performance counters, Win32 APIs
- \* **Medium** for less frequency events
  - \* Rare exceptions, deadlocks
  - \* ETW, ClrMD
- \* **Invasive** for deep-dive and concrete diagnostics
  - \* Memory leaks, bulk call-stack data (e.g. CPU profiling)
  - \* CLR Profiling API, CLR Debugging API, hooks

**LET'S GET TO BUSINESS**

# (Self) CPU-Profiling

- \* Monitor CPU using performance counters
- \* Are we above a certain threshold for a certain amount of time?
  - \* Turn on ETW and collect stacks (live using LiveStacks)
  - \* Find hot paths, produce flame graphs
  - \* Suggest recommendations

# What Can Be Done?

- \* AuthenticationController takes 95% CPU, maybe we're being DDoS'ed
- \* Image processing component takes 100% CPU, need to auto-scale the app
- \* Encoding this 30 second video takes 3 minutes at 100% CPU, tell the user she can send us a bug report

# DEMO

MONITOR FOR CPU SPIKES

# This Is From Real Life

The screenshot shows the Microsoft Visual Studio (Administrator) interface. The main window displays the Start Page. Overlaid on this are several windows:

- Process Explorer**: A window showing a list of running processes. The process `PerfWatson2.exe` is highlighted with a red box. The table below shows the data for this process:

Process	CPU	Private Bytes	Working Set
svchost.exe		1,828 K	7,328 K
svchost.exe		1,751 K	6,592 K
svcs.exe	< 0.01	7,016 K	10,140 K
fontdrvhost.exe		1,750 K	832 K
csrss.exe	0.28	2,676 K	2,108 K
winlogon.exe		2,324 K	2,348 K
fontdrvhost.exe		5,936 K	6,180 K
dwm.exe	0.59	94,952 K	51,420 K
explorer.exe	0.03	62,554 K	85,188 K
MSASCl.exe		1,948 K	2,444 K
TaskHost.exe		1,784 K	7,908 K
PerfWatson2.exe	1.00	25,044 K	35,800 K
PerfWatson2.exe	0.02	59,320 K	36,548 K
conhost.exe		19,330 K	20,064 K
conhost.exe	< 0.01	5,328 K	10,008 K
ServiceHub.IdentityL...		21,752 K	35,272 K
ServiceHub.SettingsH...		16,328 K	51,172 K
ServiceHub.VSDetour	0.03	67,100 K	107,360 K
ServiceHub.Host CLR...		2,056 K	31,228 K
smss.exe		3,070 K	10,200 K
procexp51.exe	0.58	17,850 K	36,056 K
smss.exe	0.01	47,472 K	67,552 K
SnippingTool.exe	0.23	3,910 K	25,316 K
resmon.exe		3,544 K	3,320 K
usmhook.exe		3,204 K	3,440 K
OneDrive.exe		8,872 K	4,856 K

- Performance Monitor**: A window showing a tree view of performance counters. The `Microsoft-VisualStudio-Telemetry-PerfWatson2-2508` counter is selected and highlighted with a red box.
- Microsoft-VisualStudio-Telemetry-PerfWatson2-2508 Properties**: A dialog box showing the properties of the selected counter. The `Providers` tab is active, and the `NET Common Language Runtime` provider is highlighted with a red box. The `Keywords(All)` property is set to `0x0`.

# (Self) GC-Monitoring

- \* Monitor GC performance using performance counters
- \* Register on ETW's GC events such as `GCAllocationTick`
  - \* Types of objects allocated and their stacks(!)
  - \* Number of GCs of each kind and size of reclaimed memory
  - \* Duration of GC pauses
- \* Attach ClrMD to get heap breakdown
  - \* Generations, segments, reserved/committed, number of objects

**DEMO**

**MONITOR FOR ALLOCATION SPIKES**

# (Self) Heap Analysis

- \* Monitor memory usage using performance counters
- \* Has memory increased above a certain threshold for a certain time?
  - \* Attach ClrMD to get heap statistics
  - \* Compare snapshots
  - \* Report which objects are not freed
  - \* Combine with ETW `GCAAllocationTick` to get rate of allocation

**DEMO**

**FIND A LEAK**

# (Self) Deadlock Detection

- \* Monitor for potential deadlock
  - \* Low CPU
  - \* Request timeouts
  - \* Increased thread count
- \* Attach ClrMD to create wait chains and detect deadlocks
- \* Report, try to break, pray for a miracle...

# DEMO

## DETECT A DEADLOCK

# Food for Thought

- \* Many more scenarios are possible

# Food for Thought

- \* Many more scenarios are possible
- \* Monitor heap fragmentation and compact large objects if needed

# Food for Thought

- \* Many more scenarios are possible
- \* Monitor heap fragmentation and compact large objects if needed
- \* Native memory leak analysis using ETW

# Food for Thought

- \* Many more scenarios are possible
- \* Monitor heap fragmentation and compact large objects if needed
- \* Native memory leak analysis using ETW
- \* Side notes:
  - \* CLRMD is also very suitable for automating crash dump analysis
  - \* You can automate opening tickets in bug tracker, consolidate same issue from different users, versions, etc.

# Not Everything Is Perfect

- \* The pros are obvious (visibility, easy scaling...)

# Not Everything Is Perfect

- \* The pros are obvious (visibility, easy scaling...)
- \* But there are some cons as well...

# Not Everything Is Perfect

- \* The pros are obvious (visibility, easy scaling...)
- \* But there are some cons as well...
  - \* Adds complexity (reduce risk by using separate process)

# Not Everything Is Perfect

- \* The pros are obvious (visibility, easy scaling...)
- \* But there are some cons as well...
  - \* Adds complexity (reduce risk by using separate process)
  - \* Adds overhead

# Not Everything Is Perfect

- \* The pros are obvious (visibility, easy scaling...)
- \* But there are some cons as well...
  - \* Adds complexity (reduce risk by using separate process)
  - \* Adds overhead
  - \* Requires additional development

# Summary

- \* Self-monitoring is important for all kinds of software
- \* Best to create a hierarchy of monitoring (and overhead and complexity)
- \* Lots of scenarios: CPU, GC, memory, deadlocks
- \* Demos: <https://github.com/dinazil/self-aware-applications>

**THANK YOU**

**DINA GOLDSHTEIN**  
**@DINAGOZIL**

