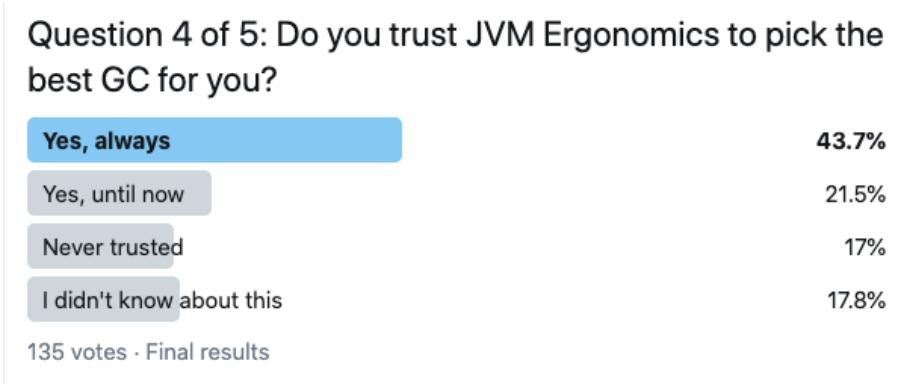
Enhanced Default Ergonomics

JVM “tuning” for beginners

Microsoft

Problem



- JVM tuning is hard, so developers trust the defaults
- Developers are deploying small JVMs, often in containers
  - Small memory footprint (often 0.5 to 2 GB)
  - Small CPU amount (often, “1 core / 1000 milli-cores”)
- Not setting max heap size
  - Often get default ergonomics (50% or 25%)
- Not selecting GC
  - Often, SerialGC (not bad)
  - But they expected the *“default GC”* - G1GC
  - Or worse, select a GC ignoring consequences of constrained resources

Some data

New Relic 2021

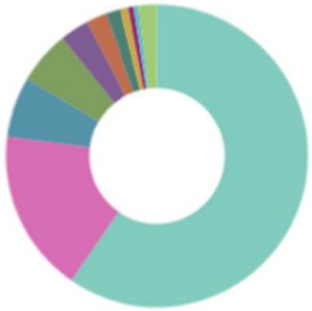
Containerized JVMs
Since Feb 10, 06:29 ...

62.92 %
Percentage

Java Versions in Containers
Since Feb 10, 06:29 am until Feb 11, 07:22 am

8	61.19 %
11	33.95 %
12	1.67 %
14	0.82 %
13	0.75 %
7	0.66 %
15	0.42 %

CPUs in Containers
Since Feb 10, 06:29 am until Feb 11, 07:22 am



1	59.62 %
2	17.38 %
8	6.51 %
4	5.7 %
3	3.09 %
16	2.22 %
Other	1.94 %

JVM Heap Sizing in Containers
Since Feb 10, 06:29 am until Feb 11, 07:22 am

72.36 %
-Xmx

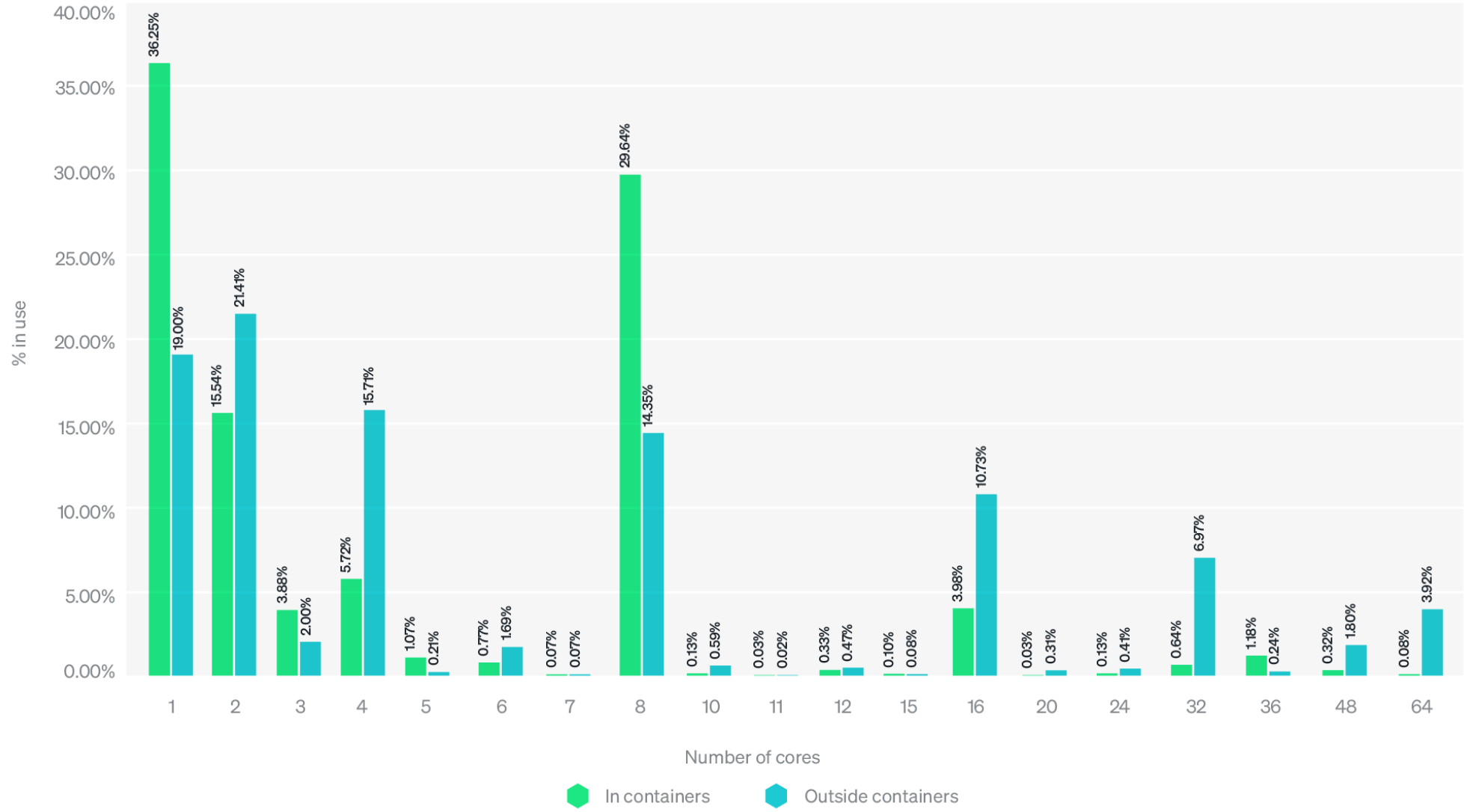
8.47 %
-XX:MaxRAMPercentage

-Xmx Ranges in Containers
Since Feb 10, 06:29 am until Feb 11, 07:22 am

<= 1GB	41.5 %
Not Set	27.64 %
<=2GB	10.71 %
<=3GB	8.38 %
>4GB	8.03 %
<=4GB	3.74 %

Some data

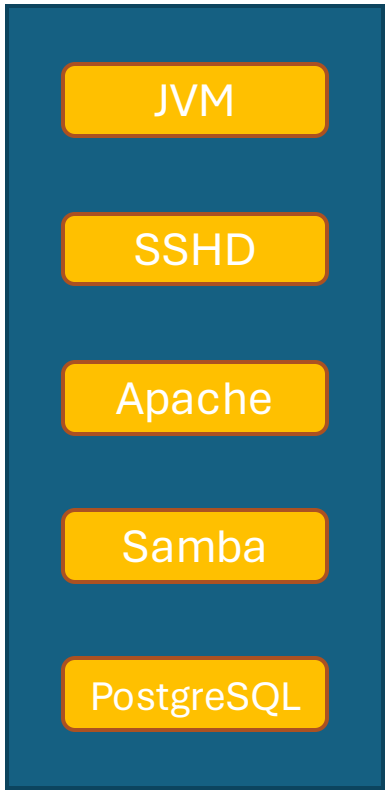
New Relic 2023



Some data

New Relic 2023

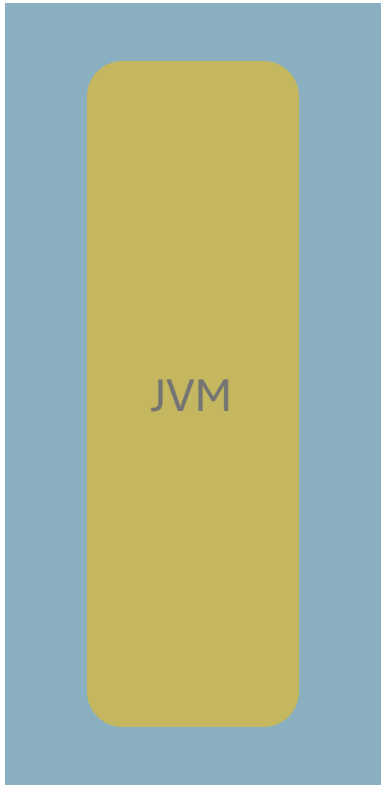
Assumption in 2000



Bare metal
"shared"

- 2 Cores
- 2 GB of RAM
- Shared resources

Assumption in 2023



Container/VM
"dedicated"

- 2 Cores
- 2 GB of RAM
- Isolated resources

Static JVM Ergonomics – Current state (as of 2000)

- Default heap size (of available memory):
  - Maximum

<256 MB	50%
256-512 MB	~126 MB
>512 MB	25%

  - Initial

<= 512 MB	8 MB
>=512 MB	1.5625%-1.7%

  - Minimum

64 MB – 8192 MB	8 MB

- **GC selection**

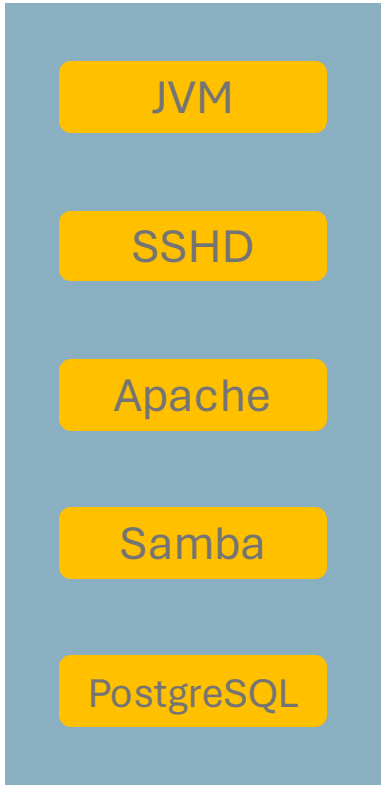
Memory	Processors	GC selected
>=1792 MB	>1	G1
Otherwise		Serial

- **GC Threads**

GC	# of CPUs	ConcGCThreads	ParallelGCThreads
SerialGC	any	0	0
G1 GC	1-8	max((ParallelGCThreads+2)/4, 1)	ceil(#CPUS)
G1 GC	>8	max((ParallelGCThreads+2)/4, 1)	8 + (#CPUS-8) * (5/8)

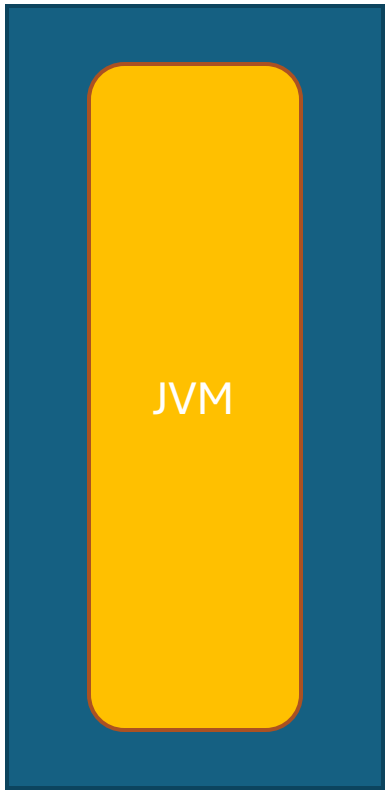
- **Available processors**
  - Server: 1 "active processor" for each CPU
  - Container: cgroups-based counting on cpu_quota/cpu_period

Assumption in 2000



Bare metal
"shared"
- 2 Cores
- 2 GB of RAM
- Shared resources

Assumption in 2023



Container/VM
"dedicated"
- 2 Cores
- 2 GB of RAM
- Isolated resources

Static JVM Ergonomics – Container-aware (2023)

- Default heap size (of available memory):
  - Maximum
    - TBD
  - Initial
    - TBD
  - Minimum
    - TBD
- **GC selection**
  - TBD
- **GC Threads**
  - TBD
- **Available processors**
  - TBD

Ideas

- Produce generic Java runtimes for containers with tweaked defaults
- Add more if/elses into select_gc_ergonomically() set_heap_size()
- JVM launcher configuration files
- Stuff we haven’t thought of ☺
- Ergonomics profiles

Naïve and simplistic proposal

- Ergonomics Profile
  - shared: traditional ergonomics (unchanged)
  - dedicated: when the JVM believes it will be likely alone (e.g. containers)

Memory	Processors	GC selected
Any	1	Serial
<=2048 MB	>1	ParallelGC
>2048 MB	>1	G1
>=16 GB	>1	ZGC

Memory	Heap size
< 0.5 GB	50%
>= 0.5 GB	75%
>= 4 GB	80%
>= 6 GB	85%
>= 16 GB	90%

Should we disable `-XX:-UseDynamicNumberOfCompilerThreads` for single proc and <=512MB envs ??

Works on VMs too (image templates)

```
JAVA_TOOL_OPTIONS="-XX:ErgonomicsProfile=dedicated"
```

Challenges

- Sooner
  - JDK-8261242: is_containerized() returns true when run outside a container
  - JDK-8134507: Improve the tiered compilation compiler thread ergonomics

  - Conscious of non-heap consumption: code cache, direct memory, metaspace, GC, etc.

- Later
  - JDK-8254091: Need a mechanism (and API) to reliably determine if a JVM is executing in a container context

  - What to do with Client/Server Class Machines feature?

Diving into code

- https://github.com/microsoft/openjdk-jdk/pull/9

Next steps

- Confirm settings for “dedicated” profile are fair. Any feedback?
- What other flags should be enabled, and under what conditions?
- What other profiles may come in the future? Hardware-based profiles?
- How does this ties to Leyden profiles?
- Is this better than status quo?

- Is this a good path forward to improve new Java app onboarding in the Cloud?