Why You Should Upgrade Your Java In Containers Right Now

Ben Evans, New Relic (He / Him)

Safe Harbor

This presentation and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. (“New Relic”) to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic’s express written permission.

Such information may contain forward-looking statements within the meaning of federal securities laws. Any statement that is not a historical fact or refers to expectations, projections, future plans, objectives, estimates, goals, or other characterizations of future events is a forward-looking statement. These forward-looking statements can often be identified as such because the context of the statement will include words such as “believes,” “anticipates,” “expects” or words of similar import.

Actual results may differ materially from those expressed in these forward-looking statements, which speak only as of the date hereof, and are subject to change at any time without notice. Existing and prospective investors, customers and other third parties transacting business with New Relic are cautioned not to place undue reliance on this forward-looking information. The achievement or success of the matters covered by such forward-looking statements are based on New Relic’s current assumptions, expectations, and beliefs and are subject to substantial risks, uncertainties, assumptions, and changes in circumstances that may cause the actual results, performance, or achievements to differ materially from those expressed or implied in any forward-looking statement. Further information on factors that could affect such forward-looking statements is included in the filings New Relic makes with the SEC from time to time. Copies of these documents may be obtained by visiting New Relic’s Investor Relations website at ir.newrelic.com or the SEC’s website at www.sec.gov.

New Relic assumes no obligation and does not intend to update these forward-looking statements, except as required by law. New Relic makes no warranties, expressed or implied, in this presentation or otherwise, with respect to the information provided.

About Me – Career

- New Relic, Lead Architect
- jClarity, Co-founder
  - Sold to Microsoft
- Deutsche Bank
  - Chief Architect (Listed Derivatives)
- Morgan Stanley
  - Google IPO
- Sporting Bet
  - Chief Architect

About Me – Community

- Java Champion
- JavaOne Rock Star Speaker
- Java Community Process Executive Committee
- London Java Community
  - Organising Team
  - Co-founder, AdoptOpenJDK

Today's Talk

- How We Got Here
- Introduction to New Relic
- Current State of Java
- Why is 11 better in containers?
- JFR
- Conclusions

How We Got Here

- Java & OpenJDK History
- New Release & Support Model
- Mainline dev
- OpenJDK 8 & 11

A Brief History of Java

- Sun release Java in beta to much hype (1995)
- Sun fully open-source Java (2006)

A Brief History of Java

- Sun release Java in beta to much hype (1995)
- Sun fully open-source Java (2006)
- Oracle acquire Sun (2010)
- Java 7: First release based on OSS codebase (2011)

A Brief History of Java

- Sun release Java in beta to much hype (1995)
- Sun fully open-source Java (2006)
- Oracle acquire Sun (2010)

- Java 7: First release based on OSS codebase (2011)
- Java 8: "Classic" Long-Term Support Release (2014)
- Java 9: New release model (2017)
- Java 11: Current Long-Term Support Release (2018)

New Release Model

- Feature Releases
  - Every 6 months
  - Only supported for 6 months by Oracle
    - Other vendors may offer other options
- Long-Term Support releases (LTS)
  - Every 3 years
  - Java 8 & 11 are LTS (& 17 will be)
  - **Java 9, 10, 12, 13, 14, 15 & 16 are NOT LTS**

What has Changed in Java?

- Paid support options
  - Oracle (LTS only)
  - Azul, various other OpenJDK vendors

What has Changed in Java?

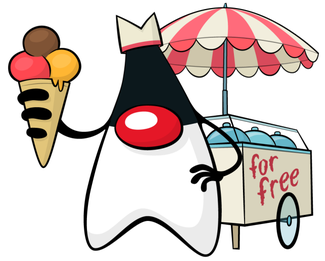
- Paid support options
  - Oracle (LTS only)
  - Azul, various other OpenJDK vendors
- Free updates are still available from:
  - Oracle (must upgrade every 6 months)
  - OpenJDK vendors (for LTS versions only)

What has Changed in Java?

- Paid support options
  - Oracle (LTS only)
  - Azul, various other OpenJDK vendors
- Free updates are still available from:
  - Oracle (must upgrade every 6 months)
  - OpenJDK vendors (for LTS versions only)
- Oracle’s Java market share is diminishing
  - OpenJDK is gaining greater prominence

Who are the New Players?

- Eclipse Adoptium (AdoptOpenJDK)
- Amazon (Corretto)
- Microsoft
- Red Hat (IcedTea)
- Azul Systems (Zulu)
- AliBaba (Dragonwell)
- IBM (OpenJ9)



Mainline Dev

- OpenJDK now uses a mainline dev model
- Features are merged only when code complete
- Releases occur on a strict time cadence
- Late features are held over for the next release
- Trunk / mainline is always releasable
  - Emergency fixes can be pushed out immediately
- Longer-term projects explore / research future directions

OpenJDK 8 & 11

- OpenJDK 8 & 11 now run by the community
  - Oracle engineers no longer contribute directly
- Oracle are still producing security patches for $$$
  - Same patches must also appear in OpenJDK
- Adoptium have committed to support 8 until 2023
  - At least...

Ongoing Maintenance

- “Housekeeping updates”
  - Japanese Era
  - Xcode 10+ (Mac)
  - Timezone database
  - TLS 1.3
- Selected bug fixes backported (e.g. security)
- Some potential for (very small) features
  - Features may not change semantics
  - JFR

Introduction to New Relic

- New Relic is a performance monitoring company
- Billions of events handled per minute

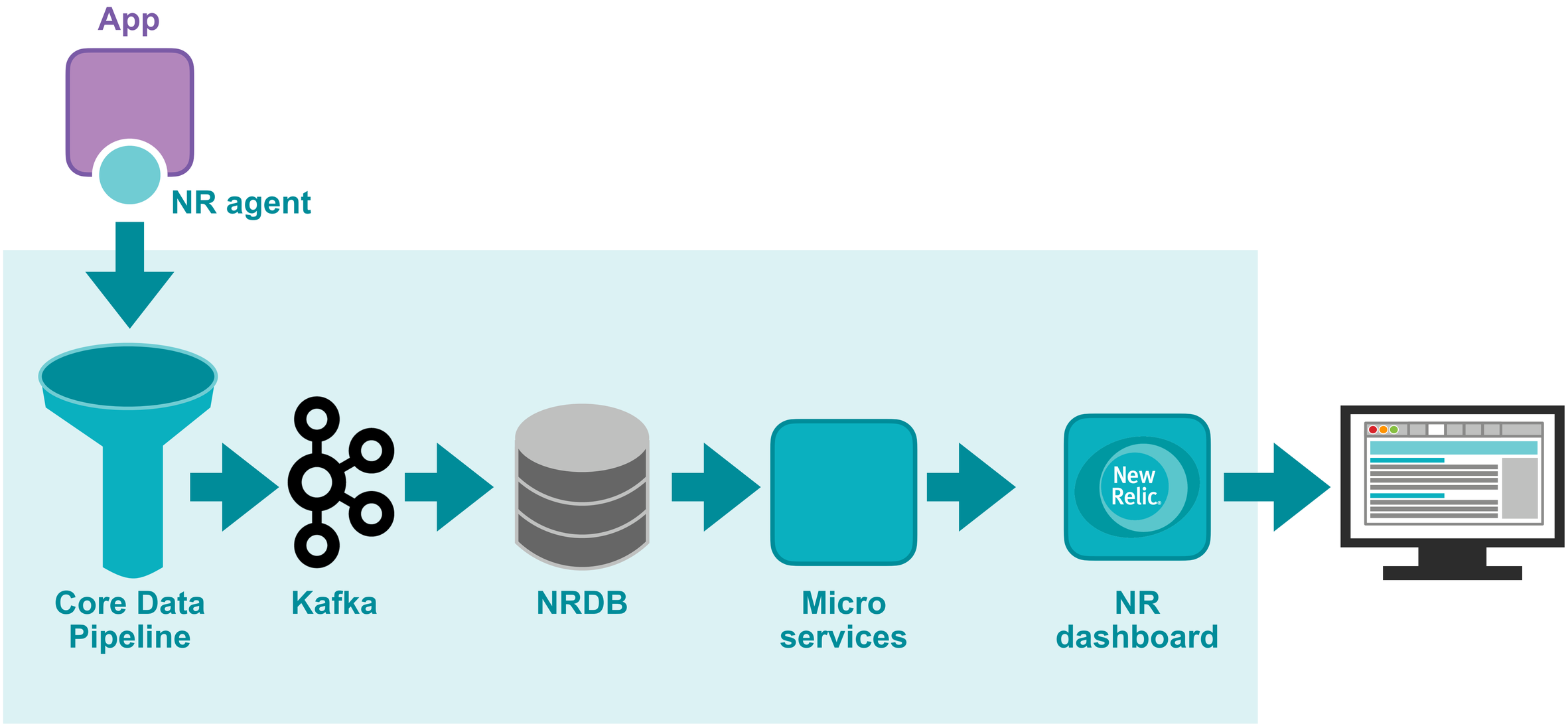
Introduction to New Relic

- New Relic is a performance monitoring company
- Billions of events handled per minute
- New Relic One
  - Market’s first Observability Platform
- Recently open-sourced $700M of our code

Introduction to New Relic

- New Relic is a performance monitoring company
- Billions of events handled per day
- New Relic One
  - Market's first Observability Platform
- Recently open-sourced $700M of our code
- Java is the majority of our services
  - One of the biggest Kafka installs in the world!
  - We also use the Kotlin language extensively

High-Level Product Architecture



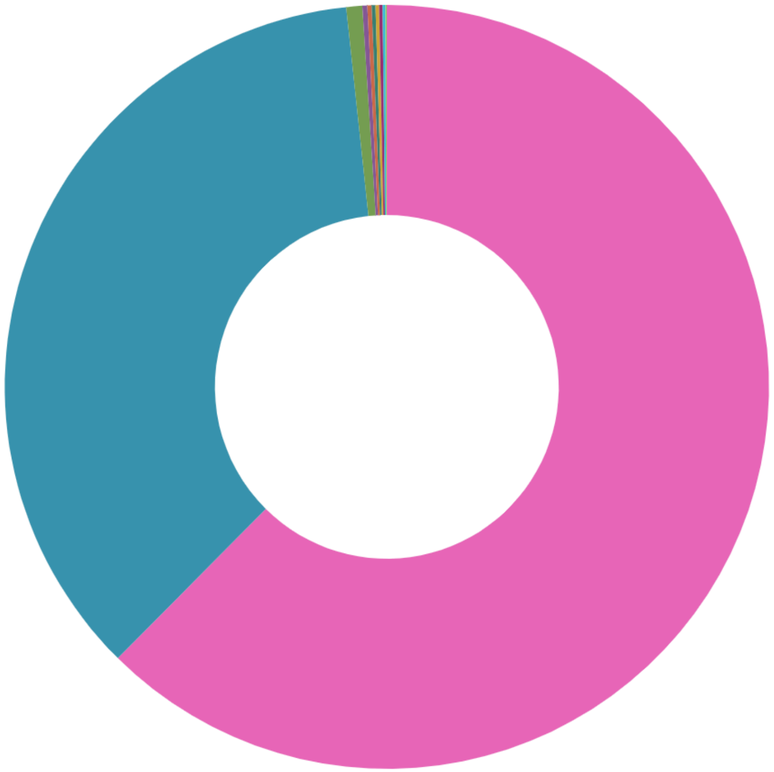
Current State of Java

- New Relic aggregates data from our customers
- Reveals trends about the shape of the market
  - Which versions, which vendors etc people use
- Live data, accurately reported from customers VMs
- Analyst estimates: ~1% of Java SE VMs worldwide

Java Versions

Since 1 week ago

Share



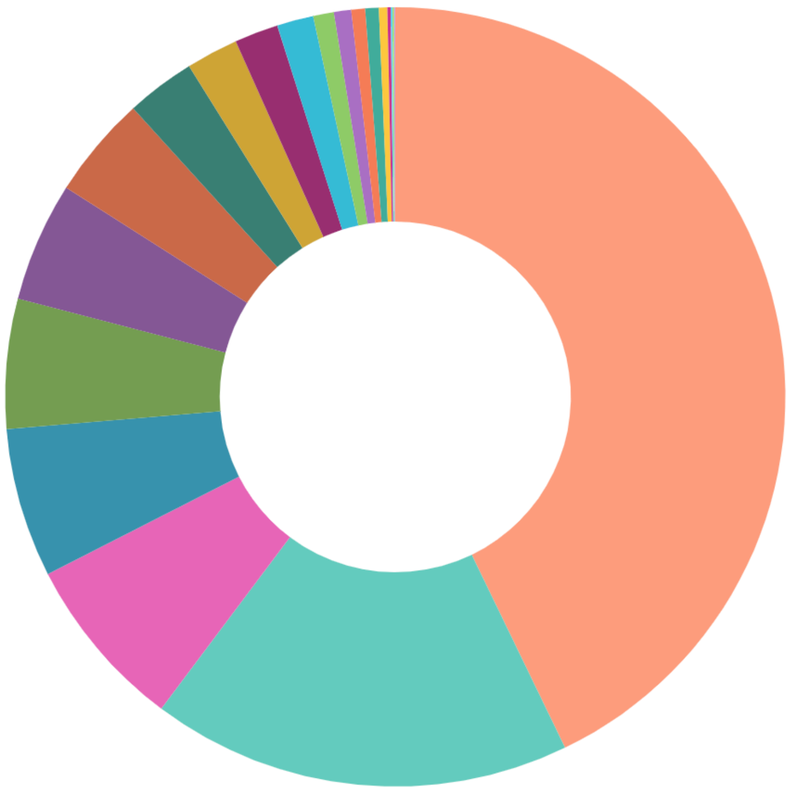
JVMMETADATASUMMARIES

8	62.43 %
11	35.87 %
13	0.68 %
14	0.21 %
9	0.18 %
15	0.16 %
10	0.15 %
7	0.15 %
6	0.12 %
12	0.056 %

Java Vendors

Since 1 week ago

Share



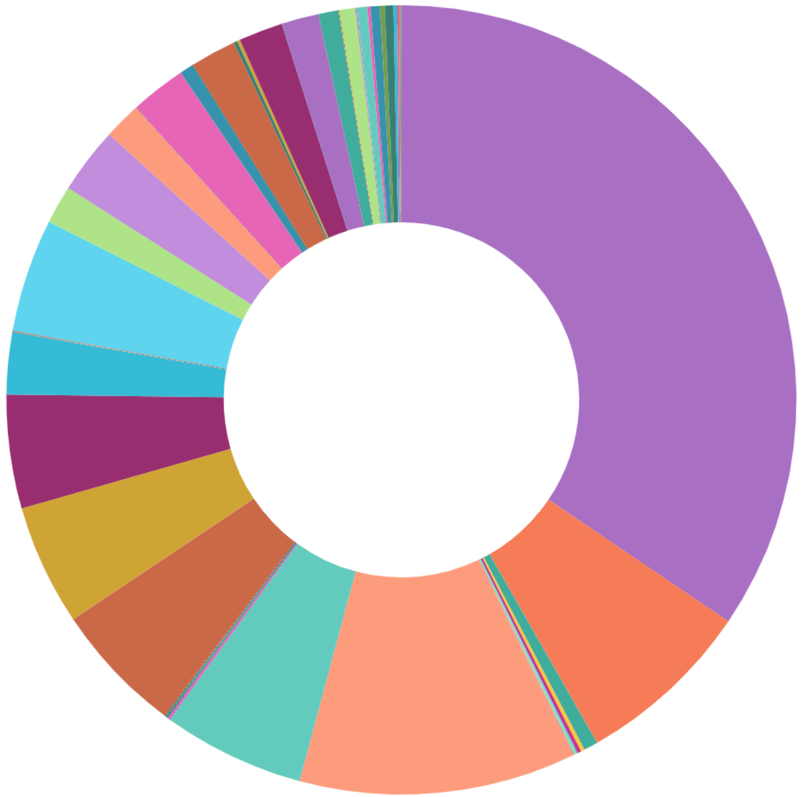
JVMMETADATASUMMARIES

Oracle Corporation	42.83 %
AdoptOpenJDK	17.4 %
Azul Systems, Inc.	7.25 %
Red Hat, Inc.	6.2 %
Ubuntu	5.38 %
IcedTea	4.95 %
Amazon.com Inc.	4.26 %
BellSoft	2.87 %
IBM Corporation	2.15 %
Private Build	1.82 %
Tableau	1.51 %
N/A	0.85 %
Pivotal Software Inc	0.72 %
Eclipse OpenJ9	0.57 %
GraalVM Community	0.56 %
Debian	0.35 %
SAP SE	0.14 %

Vendors and Versions

Since 1 week ago

Share



JVMMETADATASUMMARIES

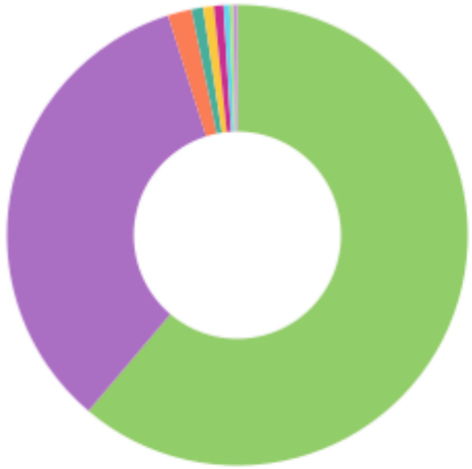
Oracle Corporation, 8	34.49 %
AdoptOpenJDK, 11	11.32 %
Oracle Corporation, 11	7.27 %
AdoptOpenJDK, 8	5.83 %
Ubuntu, 11	5.38 %
IcedTea, 8	4.95 %
Azul Systems, Inc., 11	4.66 %
Red Hat, Inc., 8	4.64 %
Amazon.com Inc., 8	2.73 %
Azul Systems, Inc., 8	2.53 %
BellSoft, 8	2.29 %
IBM Corporation, 8	1.89 %
Private Build, 8	1.81 %
Red Hat, Inc., 11	1.56 %
Amazon.com Inc., 11	1.52 %
Tableau, 11	1.51 %

Containers

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

62.92 %
Percentage

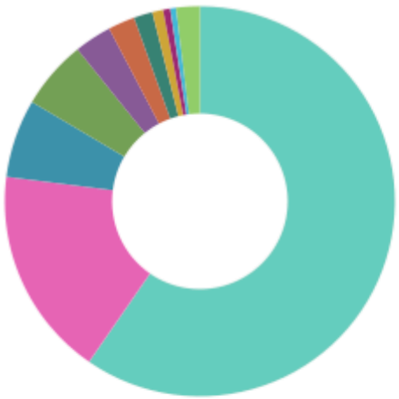
Java Versions in Containers
Since Feb 10, 06:29 am until Feb 11, 07:22 am



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

CPUs & Memory In Use

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 %

-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 %

Other GC Parameters

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

72.36 %
-Xmx

8.47 %
-XX:MaxRAMPercentage

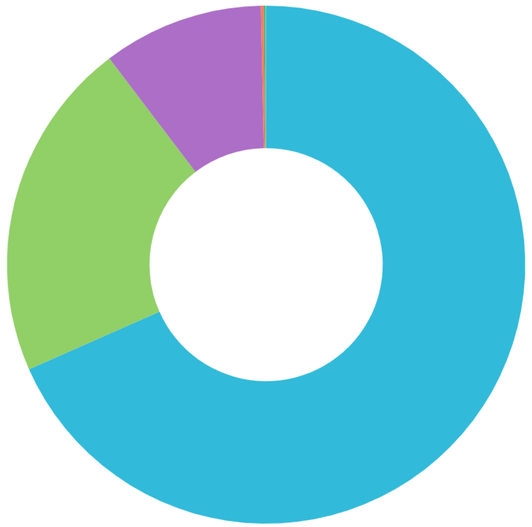
Explicitly Configured GC Threads
Since Feb 10, 06:29 am until Feb 11...

6.13 %
Percentage

Who Actively Selects A GC?

Since 1 week ago

Share

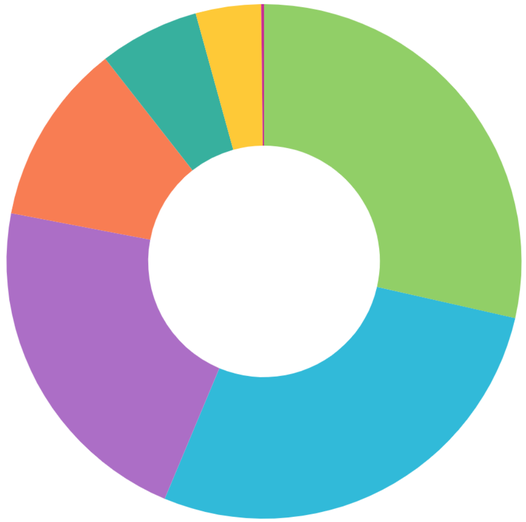


JVMMETADATASUMMARIES

Unconfigured	68.38 %
G1	21.28 %
CMS	10.01 %
Parallel	0.18 %
ZGC	0.11 %
Shenandoah	0.032 %

Since 1 week ago

Share



JVMMETADATASUMMARIES

Unknown	28.54 %
G1	27.76 %
Serial	21.7 %
CMS	11.42 %
Parallel	6.29 %
gencon	4.11 %
Other	0.18 %

Why is 11 better in containers?

Why is 11 better in containers?

- Main reasons:
  - `var`
  - Modules
  - HTTP/2

Why is 11 better in containers?

- Main reasons:
  - `var`
  - Modules
  - HTTP/2

- Just Kidding...

Real Reasons for Using 11 in Containers?

- "Container-Aware"
- Decent version of G1GC
- Compact Strings & Heap Reduction
- JDK Flight Recorder

"Container-Aware"

- Containers requires thought about:
  - GC algorithms and selections
  - Memory usage
  - CPU Usage
- What does `Runtime.getAvailableProcessors()` return?

How Is A GC selected?

- "GC Ergonomics"
- Depends upon
  - Java version
  - "Server" or "client" class determination
  - CPU count

Selecting a GC

```
GCArguments* GCConfig::select_gc() {
  // Fail immediately if an unsupported GC is selected
  fail_if_non_included_gc_is_selected();

  if (is_no_gc_selected()) {
    // Try select GC ergonomically
    select_gc_ergonomically();

    if (is_no_gc_selected()) {
      // Failed to select GC ergonomically
      vm_exit_during_initialization("Garbage collector not selected "
                                    "(default collector explicitly disabled)", NULL);
    }

    // Succeeded to select GC ergonomically
    _gc_selected_ergonomically = true;
  }
```

Max Heap Size

- By default, on bare metal, 1/4 physical memory

```
$ java -XX:+PrintFlagsFinal -version | grep -iE 'MaxHeapSize'
    size_t MaxHeapSize   = 4294967296  {product} {ergonomic}
```

- But what about in a container?

Max Heap Size

- By default, on bare metal, 1/4 physical memory

```
$ java -XX:+PrintFlagsFinal -version | grep -iE 'MaxHeapSize'
    size_t MaxHeapSize   = 4294967296  {product} {ergonomic}
```

- But what about in a container?

- It depends...
  - Early versions of 8 can't see the container
  - 8u191 improves the situation somewhat

Memory in Containers

- Container memory consists of:
  - Java Heap memory
  - Offheap
    - Metaspace
    - JFR data
    - General book-keeping
  - Memory for auxiliary processes
- Not setting heap memory size means potential OOM
  - ~20% of containers are in this situation

Java 8 CPU Limits

- Java 8 is not well-suited for deploying in containers
  - Prior to 8u131 cgroups settings are not respected at all
  - Post-8u131 a fixed approx, based on cpu_shares, is used
  - Post-8u191 more support is backported
- Need to be careful of
  - # of GC threads used for parallel (& concurrent) GC phases
  - # of threads in auto-sized, VM-managed thread pools
- Consider explicitly setting flags to size these exactly

New Garbage Collector - G1

New version of G1GC

- “Garbage First” collector
  - experimental in 7
  - supported in 8
  - production-quality in 8u40
  - default in 9
  - very improved in 11
- Originally intended to be low-pause
  - replacement for CMS
- Ended up as a general-purpose collector
  - replacement for Parallel collectors

Tradeoffs Between Collectors

- No “one size fits all” for GC
- Different metrics are important for different apps
  - Pause time
  - Throughput (%age)
  - Pause frequency
  - Reclamation efficiency
  - Pause consistency

G1

- Design aims of G1
  - scalable to larger heaps
  - better control of pause times
  - easy to tune (`-XX:MaxGCPauseMillis`)
  - Predictable

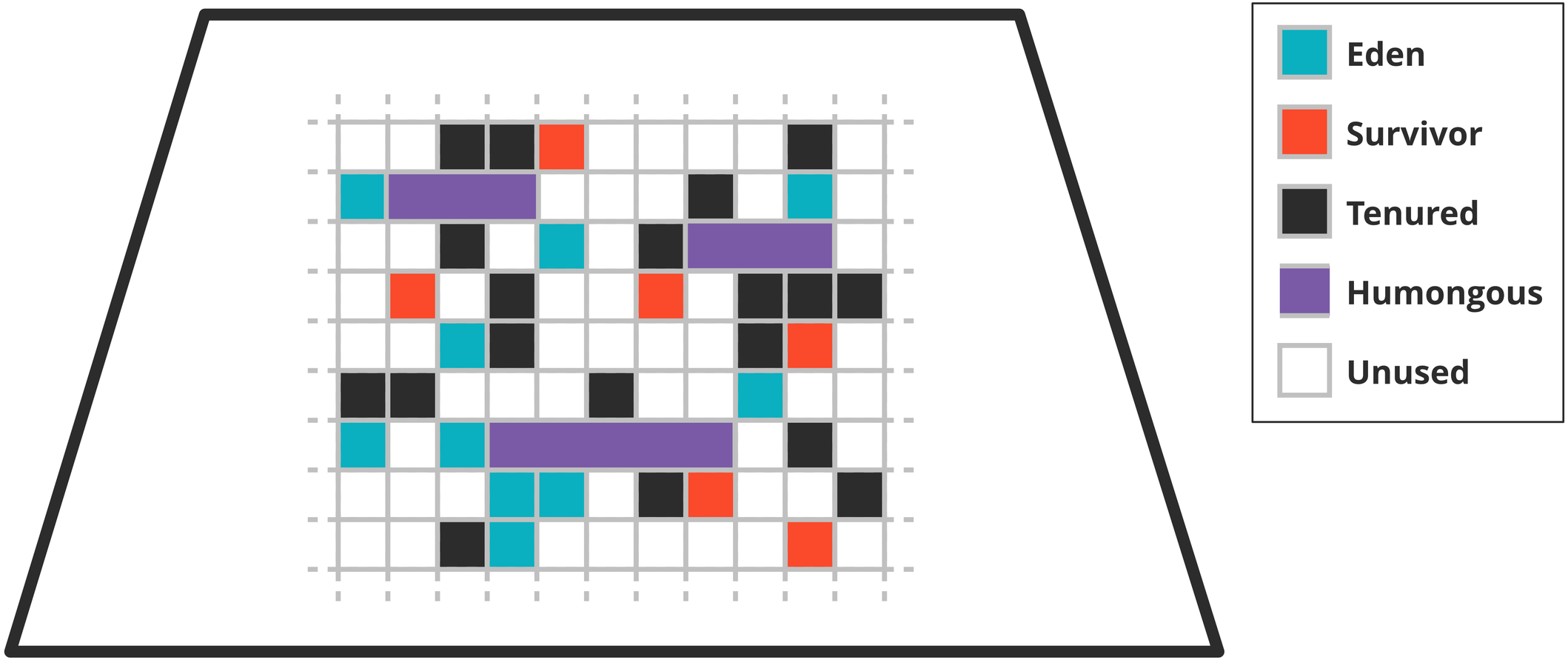
G1

- Design aims of G1
  - scalable to larger heaps
  - better control of pause times
  - easy to tune (`-XX:MaxGCPauseMillis`)
  - Predictable
- As a collector, G1 is...
  - Parallel
  - Concurrent (for marking)
  - Exact
  - Evacuating
  - "Statistically Compacting"

G1 – Regional Collection

- G1 uses regions for collection
  - not hemispherical heap (like Parallel & CMS)
- Regions
  - allow GC cycles to "partially clean" & then restart app threads
  - can be 1 - 64M in size (1M default for small heaps)
- Generational Collection
  - regions still belong to generations
  - generations are not contiguous
  - heap is still contiguous

The G1 Heap



G1 Heap

Remembered Sets (RSets)

- Similar idea to GC "card tables"
  - track pointers between regions
- If app thread mutates
  - change is put on a “refinement queue”
  - reduce work done on app thread
  - separate threads drain refinement queues
- Example of “balancing between allocator & collector”

G1 – The Bad News

- Can interfere with application throughput
  - write barriers, RSet update threads and back pressure
- Concurrent GC - uses cores while GC is running
  - Full STW Fallback can still occur
  - e.g. if allocation greatly exceeds reclamation
- Full predictability of G1 pauses is still lacking
  - 200ms goal is easy to achieve
  - Guaranteed <50ms not at all easy
- G1 not a true compacting collector

Java 11 & G1

- Java 11's G1 is significantly better
  - Has a Parallel fallback STW collector
  - Better able to meet pause time guarantees
- Algorithm is significantly different between versions
  - Ensure that tuning advice relates to the correct version
- Most apps see benefit from G1 on Java 11
  - But overall CPU utilization may increase slightly
- Other changes in 11 may also help GC performance

New Default GC

- Java 9 switched the default GC from Parallel to G1
  - This refers to the GC used to collect “old” objects
  - Both GCs use STW collection to collect “young” objects
- G1 is a concurrent GC
  - Parallel is STW
- G1 will use more CPU than Parallel
  - In exchange for shorter pause times
  - Default G1 pause is 200ms

Compact Strings

Practical Impacts

- Before Java 9 Strings are represented as char[]
  - 2 bytes per char (UTF-16)
  - In Western European langs, this wastes 50% storage
  - First byte is always zero

Practical Impacts

- Before Java 9 Strings are represented as char[]
  - 2 bytes per char (UTF-16)
  - In Western European langs, this wastes 50% storage
  - First byte is always zero
- Java 9 introduces a per-string choice
  - Latin-1
  - UTF-16
- Internal representation moves to bytes
  - Saves space in common case

Compact Strings

```
private final byte[] value;

/**
 * The identifier of the encoding used to encode the bytes in
 * {@code value}. The supported values in this implementation are
 *
 * LATIN1
 * UTF16
 *
 * @implNote This field is trusted by the VM, and is a subject to
 * constant folding if String instance is constant. Overwriting this
 * field after construction will cause problems.
 */
private final byte coder;

static final byte LATIN1 = 0;
static final byte UTF16  = 1;
```

JDK Flight Recorder (JFR)

- A profiling tool to gather diagnostics & profiling data
  - From an in-flight Java application
- Proprietary tool in Java 8, OSS in Java 11
  - Now backported to OpenJDK 8u262+
- Low overhead
  - Oracle claim ~1% impact to steady state performance
  - We observe ~3% on a useful data profile
- GUI console available - Mission Control (JMC)

Using Flight Recorder

- JFR is started with a command line flag
- Generates an output file

```
java -XX:+FlightRecorder
 -XX:StartFlightRecording=duration=200s,filename=flight.jfr Klass
```

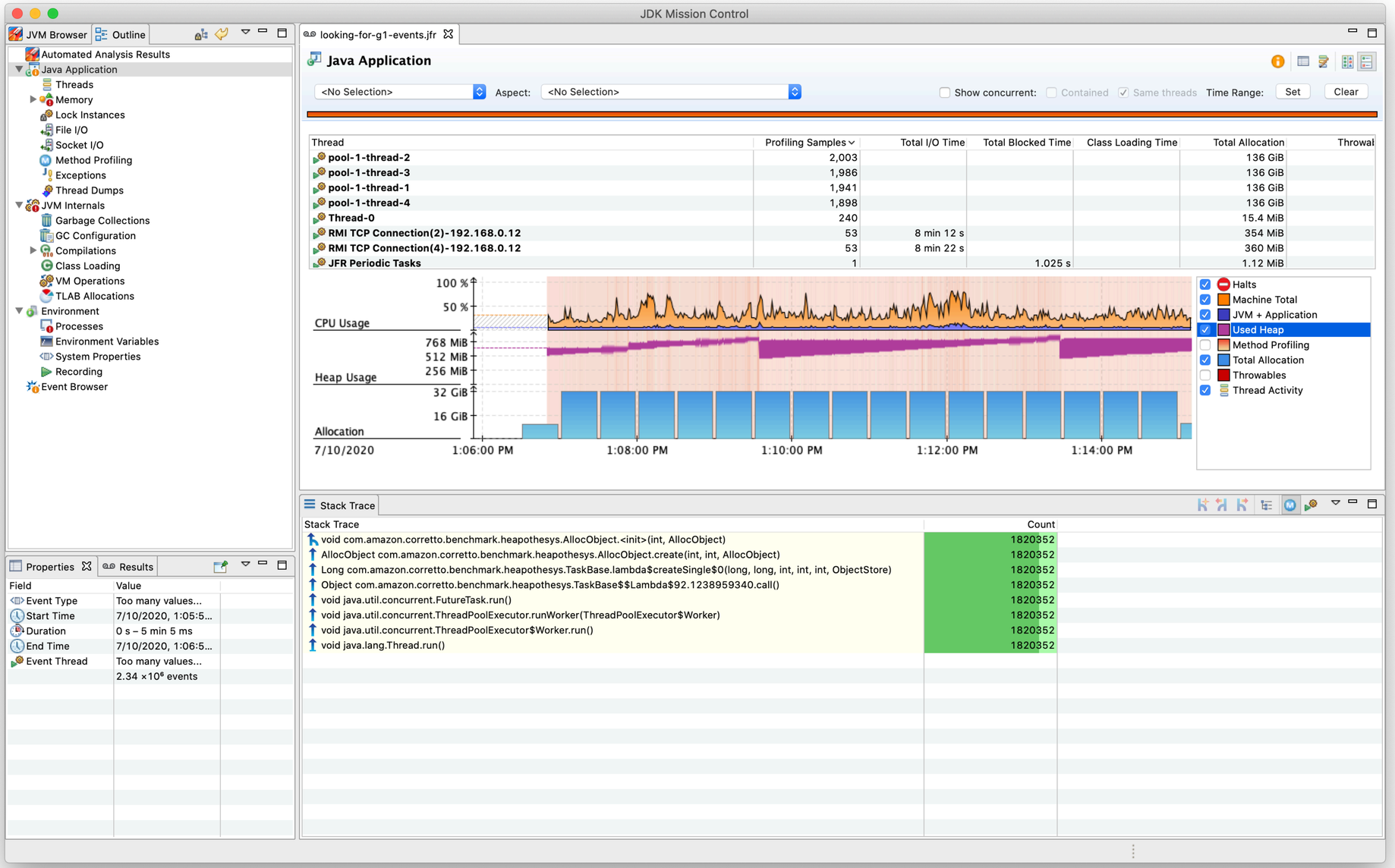
- Can be challenging to work with in containers
- Streaming solution exists (in Java 14, but not LTS)

Using jcmd

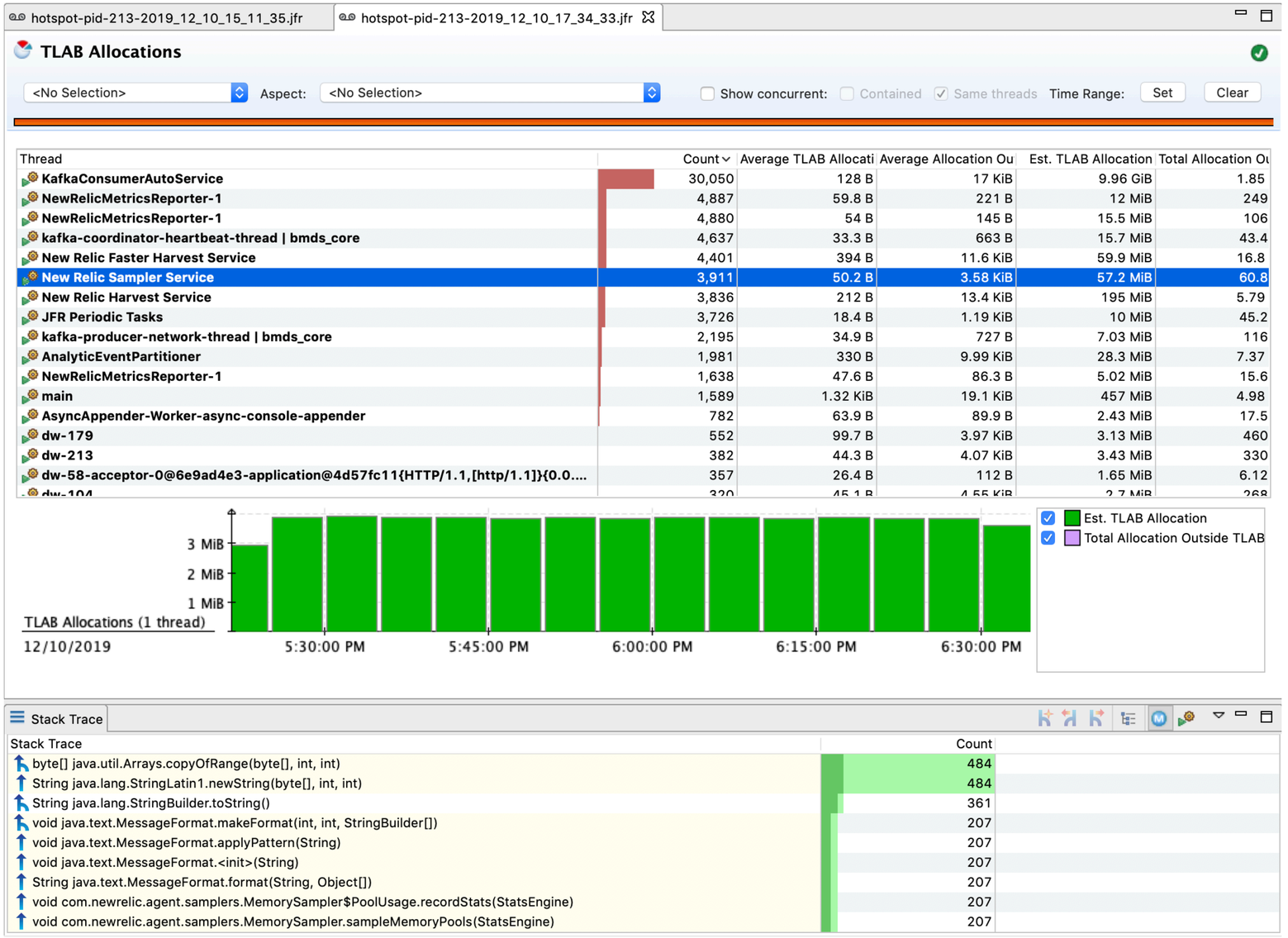
- The Java command - jcmd can be used to control JFR
- Can start and stop
- Dump a current snapshot

```
$ jcmd <pid> JFR.start name=Recording1 settings=default
$ jcmd <pid> JFR.dump filename=recording.jfr
$ jcmd <pid> JFR.stop
```

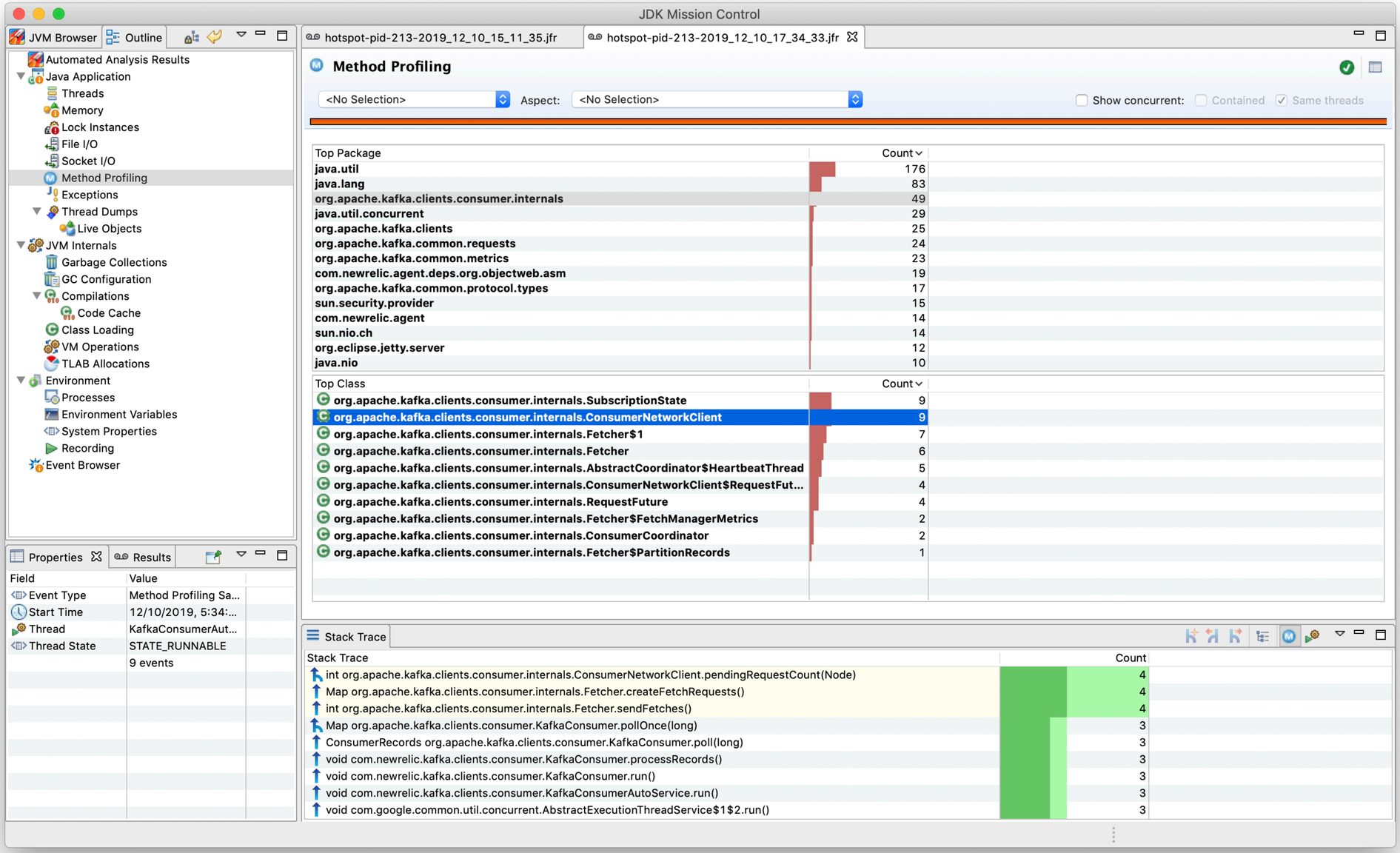
Using Mission Control



Allocation Detail (TLAB)



Method Profiling



New Relic.

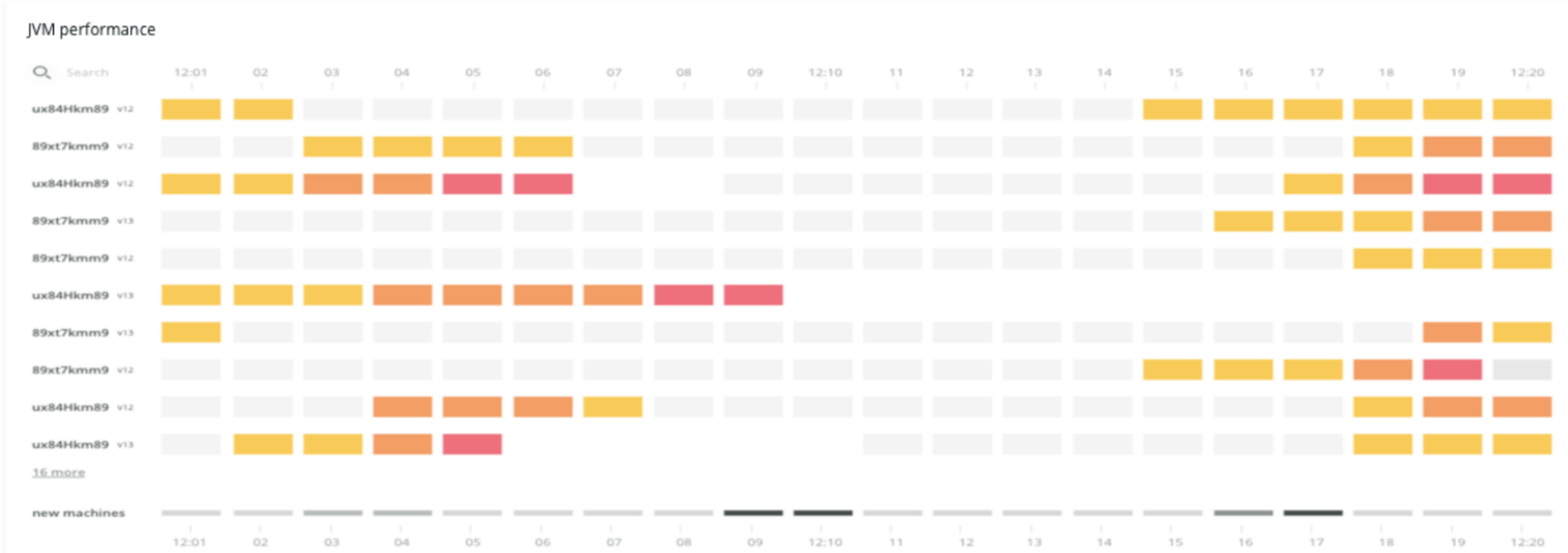
Best Practices

- Use JFR as a “ring buffer”
- Use jcmd to dump the file as required
- Allows you to ssh in & dump the buffer
  - Allows you to “go back in time”
- Not ideal
  - Need sshd running
  - Not very "DevOps Pro"

New Relic: Real-Time Profiling for Java

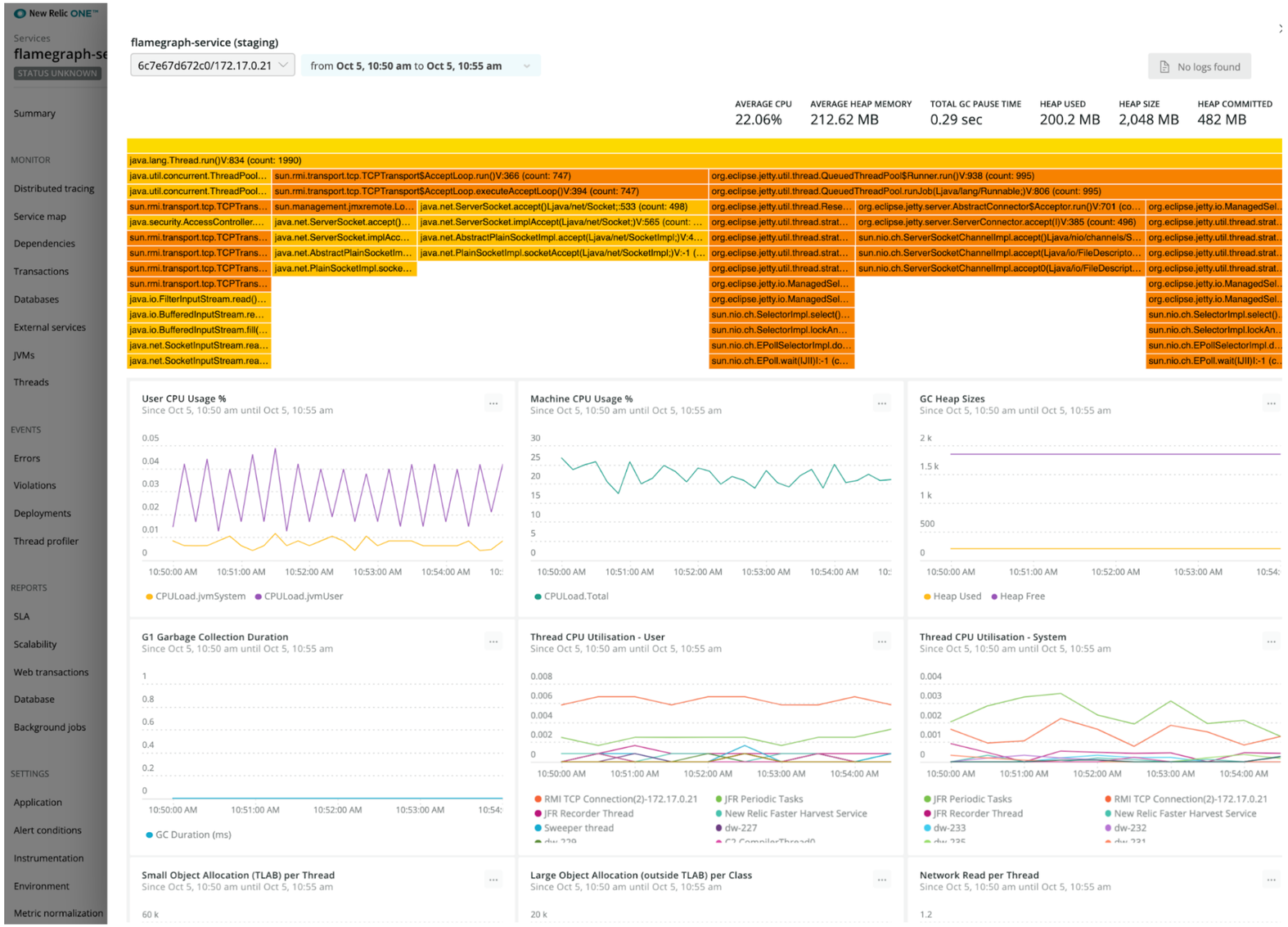
- New Relic released GA support for JFR
  - Called "Real-Time Profiling For Java"
- Open-source codebase
  - https://github.com/newrelic/newrelic-jfr-core
  - Version 1.1.0 out now
- Support for jlink'd deployments is coming
- https://newrelic.com/signup
  - 100GB / month free forever

Cluster Explorer Timeline

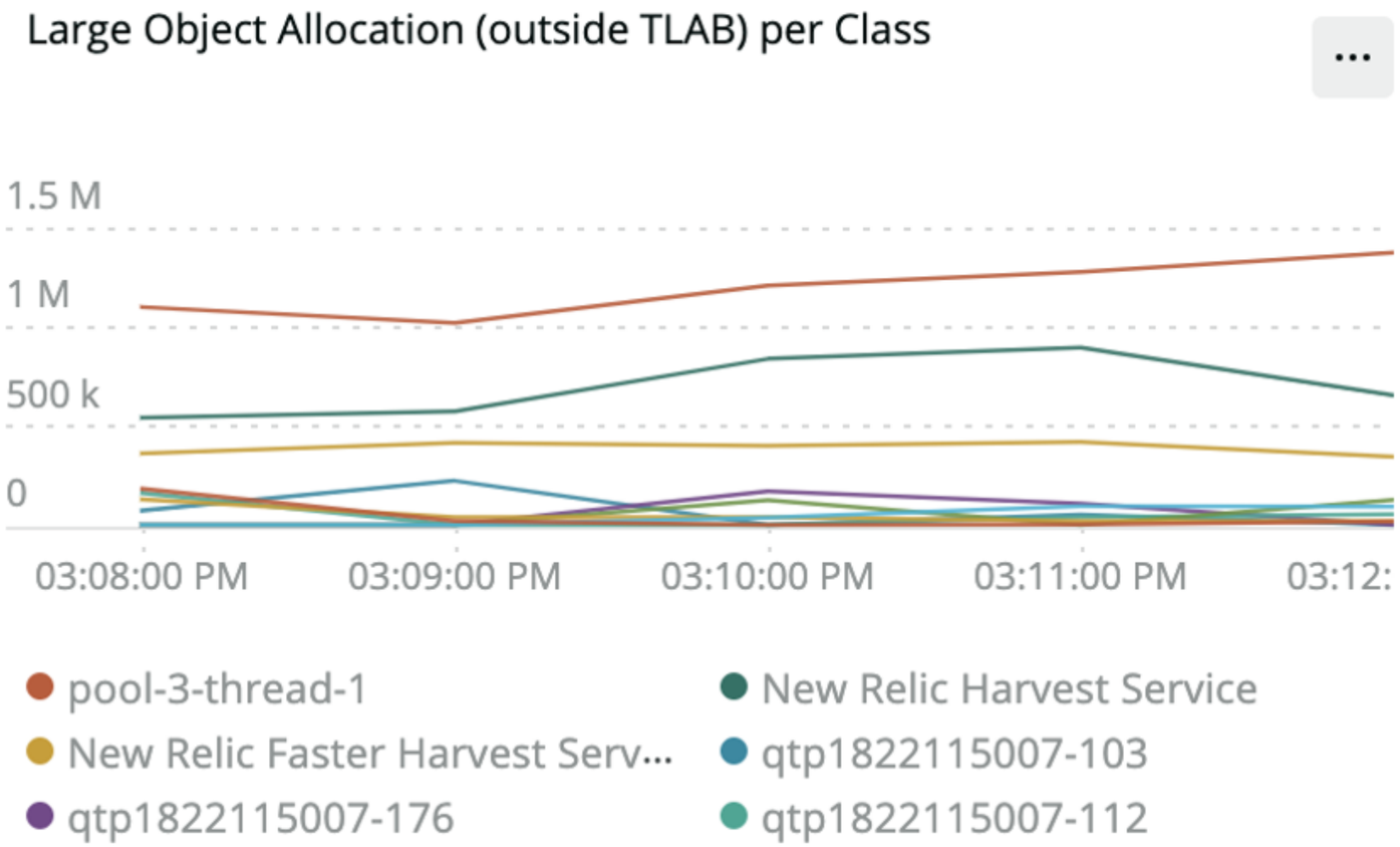


JVM	APDEX	RESPONSE TIME	THROUGHPUT	CPU	MEMORY (MB)
005dd73ff9e3 10.18.36.143:30652 005dd73ff9e3:11350	1.00	0.525 ms	11 rpm	19.3%	24,956
005dd73ff9e3 10.18.36.143:30652 005dd73ff9e3:11350	1.00	0.525 ms	11 rpm	19.3%	24,956
005dd73ff9e3 10.18.36.143:30652 005dd73ff9e3:11350	1.00	0.525 ms	11 rpm	19.3%	24,956
005dd73ff9e3 10.18.36.143:30652 005dd73ff9e3:11350	1.00	0.525 ms	11 rpm	19.3%	24,956

Execution Flamegraph



Deep Dive Graphs



JFR & Open Instrumentation

- Java Flight Recorder
  - Oracle technology (open-sourced as of Java 11)
  - Backport of the tech to OpenJDK 8
- JFR is key piece of the ecosystem - not all of it
  - Part of the pivot towards Open Instrumentation
  - JFR can be bridged to OpenTracing and other OSS tools

`jlink` & GraalVM

- Further frontiers for fast startup
  - GraalVM Native mode
  - Quarkus
  - jlink'd binaries
- Challenges
  - Full modularization
  - Closed world assumption

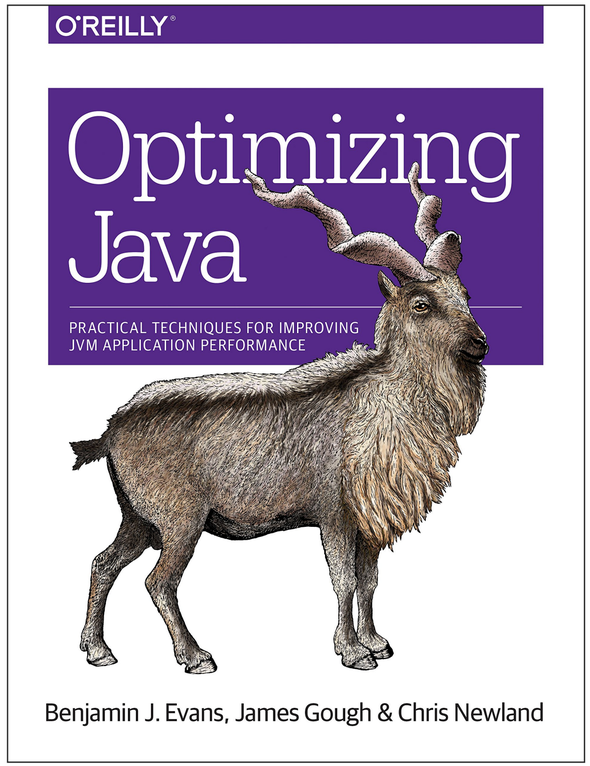
Conclusions

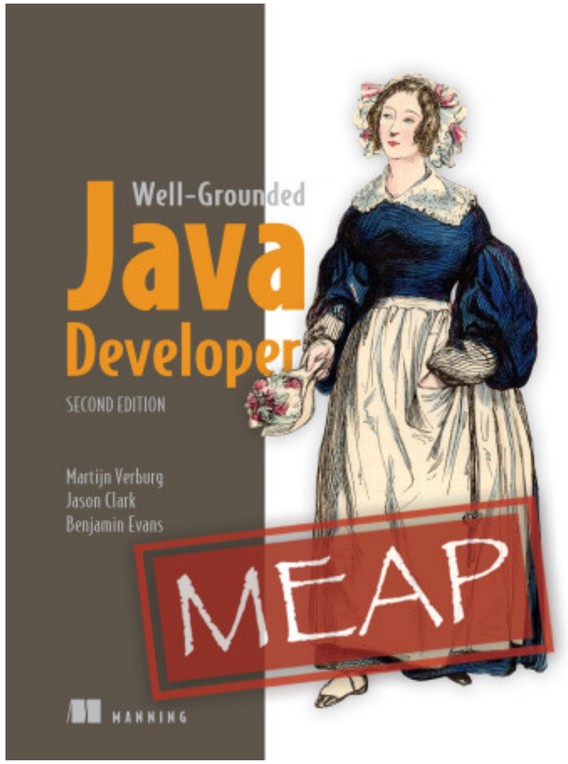
- Upgrade to 11
- Size your container correctly
- Don't use single-core containers
- Explicitly choose your memory & GC flags
  - Use a concurrent GC
- Enable JFR

Why Are We Going to 11?

- It’s the long-term support release (through 2023)
- Move from 8 … then don’t have to upgrade again
- Smaller footprint, cloud friendly, cool new tech
- Teams are using:
  - Version 11 for new apps
  - Version 8 for sustaining / BAU apps
- Upgrades are occurring at teams own pace
  - Almost all major New Relic systems have started migration

Questions & Thank You





bevans@newrelic.com

New Relic ONE™

The first observability platform.

OPEN

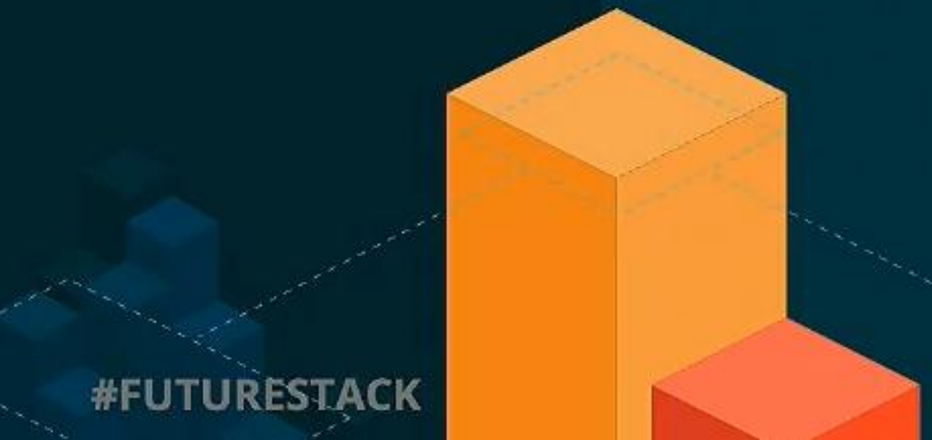
Instrument everything so you have no blind spots

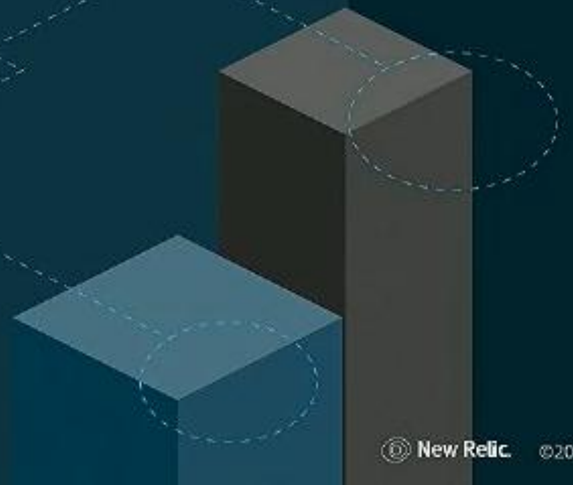
CONNECTED

Understand quickly and act more effectively

PROGRAMMABLE

Build unique applications that drive your business





#FUTURESTACK