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)
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

1. App
2. NR agent
3. Core Data Pipeline
4. Kafka
5. NRDB
6. Microservices
7. NR dashboard

New Relic.
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

JVM METADATA SUMMARIES

- 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%
Vendors and Versions

Since 1 week ago

<table>
<thead>
<tr>
<th>Vendor</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle Corporation, 8</td>
<td>34.49 %</td>
</tr>
<tr>
<td>AdoptOpenJDK, 11</td>
<td>11.32 %</td>
</tr>
<tr>
<td>Oracle Corporation, 11</td>
<td>7.27 %</td>
</tr>
<tr>
<td>AdoptOpenJDK, 8</td>
<td>5.83 %</td>
</tr>
<tr>
<td>Ubuntu, 11</td>
<td>5.38 %</td>
</tr>
<tr>
<td>IcedTea, 8</td>
<td>4.95 %</td>
</tr>
<tr>
<td>Azul Systems, Inc., 11</td>
<td>4.66 %</td>
</tr>
<tr>
<td>Red Hat, Inc., 8</td>
<td>4.64 %</td>
</tr>
<tr>
<td>Amazon.com Inc., 8</td>
<td>2.73 %</td>
</tr>
<tr>
<td>Azul Systems, Inc., 8</td>
<td>2.53 %</td>
</tr>
<tr>
<td>BellSoft, 8</td>
<td>2.29 %</td>
</tr>
<tr>
<td>IBM Corporation, 8</td>
<td>1.89 %</td>
</tr>
<tr>
<td>Private Build, 8</td>
<td>1.81 %</td>
</tr>
<tr>
<td>Red Hat, Inc., 11</td>
<td>1.56 %</td>
</tr>
<tr>
<td>Amazon.com Inc., 11</td>
<td>1.52 %</td>
</tr>
<tr>
<td>Tableau, 11</td>
<td>1.51 %</td>
</tr>
</tbody>
</table>
Containers

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

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

62.92 %
Percentage

<table>
<thead>
<tr>
<th>Version</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>61.19 %</td>
</tr>
<tr>
<td>11</td>
<td>33.95 %</td>
</tr>
<tr>
<td>12</td>
<td>1.67 %</td>
</tr>
<tr>
<td>14</td>
<td>0.82 %</td>
</tr>
<tr>
<td>13</td>
<td>0.75 %</td>
</tr>
<tr>
<td>7</td>
<td>0.66 %</td>
</tr>
<tr>
<td>15</td>
<td>0.42 %</td>
</tr>
</tbody>
</table>

New Relic.
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

JVMMETADATASUMMARIES
- Unconfigured: 68.38%
- G1: 21.28%
- CMS: 10.01%
- Parallel: 0.18%
- ZGC: 0.11%
- Shenandoah: 0.002%

Since 1 week ago

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
Containers requires thought about:

- GC algorithms and selections
- Memory usage
- CPU Usage

What does `Runtime.getRuntime().availableProcessors()` return?
How Is A GC selected?

- "GC Ergonomics"

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

```c
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

```bash
$ 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
Design aims of G1

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

Diagram showing the allocation of different regions within the 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’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
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
Before Java 9 Strings are represented as char[]

- 2 bytes per character (UTF-16)
- In Western European languages, 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
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)
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)

#### TLAB Allocations

<table>
<thead>
<tr>
<th>Thread</th>
<th>Count</th>
<th>Average TLAB Alloc.</th>
<th>Average Alloc. Ot</th>
<th>Est. TLAB Allocation</th>
<th>Total Allocation Ot</th>
</tr>
</thead>
<tbody>
<tr>
<td>KafkaConsumerAutoService</td>
<td>30,050</td>
<td>128 B</td>
<td>17 KIB</td>
<td>9.86 GIB</td>
<td>1.85</td>
</tr>
<tr>
<td>NewRelicMetricsReporter-1</td>
<td>4,887</td>
<td>59.8 B</td>
<td>221 B</td>
<td>12 MB</td>
<td>249</td>
</tr>
<tr>
<td>NewRelicMetricsReporter-1</td>
<td>4,880</td>
<td>54.8 B</td>
<td>145 B</td>
<td>15.6 MB</td>
<td>106</td>
</tr>
<tr>
<td>kafka-coordinator-heartbeat-thread</td>
<td>bmds_core</td>
<td>4,637</td>
<td>33.3 B</td>
<td>663 B</td>
<td>15.7 M</td>
</tr>
<tr>
<td>NewRelic Faster Harvest Service</td>
<td>4,401</td>
<td>394 B</td>
<td>11.6 KIB</td>
<td>59.9 MB</td>
<td>16.8</td>
</tr>
<tr>
<td>New Relic Service</td>
<td>592</td>
<td>603.3 B</td>
<td>5.5 MB</td>
<td>77.2 MB</td>
<td>8.9</td>
</tr>
<tr>
<td>New Relic Harvest Service</td>
<td>3,836</td>
<td>212 B</td>
<td>13.4 MB</td>
<td>196 MB</td>
<td>5.79</td>
</tr>
<tr>
<td>JFR Periodic Tasks</td>
<td>3,726</td>
<td>16.8 B</td>
<td>1.19 KIB</td>
<td>10 MB</td>
<td>45.2</td>
</tr>
<tr>
<td>kafka-producer-network-thread</td>
<td>bmds_core</td>
<td>2,195</td>
<td>34.9 B</td>
<td>727 B</td>
<td>7.03 MB</td>
</tr>
<tr>
<td>AnalyticEventPartitioner</td>
<td>1,981</td>
<td>330 B</td>
<td>9.99 MB</td>
<td>28.3 MB</td>
<td>7.37</td>
</tr>
<tr>
<td>NewRelicMetricsReporter-1</td>
<td>1,638</td>
<td>47.6 B</td>
<td>86.3 B</td>
<td>5.02 MB</td>
<td>15.6</td>
</tr>
<tr>
<td>main</td>
<td>1,589</td>
<td>1.32 KIB</td>
<td>19.1 KIB</td>
<td>457 MB</td>
<td>4.98</td>
</tr>
<tr>
<td>AsyncAppender-Worker-async-console-appender</td>
<td>782</td>
<td>63.9 B</td>
<td>89.9 B</td>
<td>2.43 MB</td>
<td>17.5</td>
</tr>
<tr>
<td>dw-179</td>
<td>552</td>
<td>99.7 B</td>
<td>3.97 KIB</td>
<td>3.13 MB</td>
<td>460</td>
</tr>
<tr>
<td>dw-213</td>
<td>382</td>
<td>44.3 B</td>
<td>4.07 KIB</td>
<td>3.43 MB</td>
<td>330</td>
</tr>
<tr>
<td>dm-50-decorator-O@8@9a94de3-applications44d57c11{HTTP/1.1,[http/1.1]}0.0...</td>
<td>367</td>
<td>20.4 B</td>
<td>112 B</td>
<td>1.66 MB</td>
<td>6.12</td>
</tr>
</tbody>
</table>

#### Stack Trace

<table>
<thead>
<tr>
<th>Method</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte() java.util.Arrays.copyOfRange(byte[], int, int)</td>
<td>486</td>
</tr>
<tr>
<td>String java.lang.String.charAt(int)</td>
<td>486</td>
</tr>
<tr>
<td>String java.lang.StringBuilder.toBuilder()</td>
<td>361</td>
</tr>
<tr>
<td>void java.text.MessageFormat.makeFormat(int, int, StringBuilder)</td>
<td>207</td>
</tr>
<tr>
<td>void java.text.MessageFormat.applyPattern(String)</td>
<td>207</td>
</tr>
<tr>
<td>void java.text.MessageFormat.&lt;init&gt;(String)</td>
<td>207</td>
</tr>
<tr>
<td>String java.text.MessageFormat.format(String, Object[])</td>
<td>207</td>
</tr>
<tr>
<td>void com.newrelic.agent.sampler.MemorySamplerSample</td>
<td>207</td>
</tr>
<tr>
<td>void com.newrelic.agent.sampler.MemorySamplerSampleMemoryPoints</td>
<td>207</td>
</tr>
<tr>
<td>void com.newrelic.agent.sampler.MemorySamplerSampleMemoryPoints</td>
<td>207</td>
</tr>
</tbody>
</table>
Method Profiling

Method Profiling results showing the top packages and classes with their respective counts. The screenshot includes a window from New Relic, a performance monitoring tool, displaying details about method profiling for JDK Mission Control.
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 released GA support for JFR
  • Called "Real-Time Profiling For Java"

• Open-source codebase
  • [https://github.com/newrelic/newrelic-jfr-core](https://github.com/newrelic/newrelic-jfr-core)
  • Version 1.1.0 out now

• Support for jlink'd deployments is coming

• [https://newrelic.com/signup](https://newrelic.com/signup)
  • 100GB / month free forever
Execution Flamegraph
Deep Dive Graphs

Large Object Allocation (outside TLAB) per Class

- pool-3-thread-1
- New Relic Faster Harvest Service
- qtp1822115007-103
- qtp1822115007-112
- New Relic Harvest Service
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

Optimizing Java
Benjamin J. Evans, James Gough & Chris Newland

Well-Grounded Java Development
MEAP

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