Red Hat
Runtimes

Tuning Java for Containers

Best Practices & Tools for Optimizing Java

Scott Seighman
Specialist Solutions Architect
sseighma@redhat.com

Agenda

What we’ll discuss today …

- Best Practices & Observations
- Tuning
- Q&A
- Next Steps

Agenda

What we won’t discuss ...

- Anything code-related
  - Optimizing code, best practices, etc

Best Practices & Observations

Collection of best practices and observations for running, tuning and monitoring Java in containers and in the enterprise

What are Your Performance Goals?

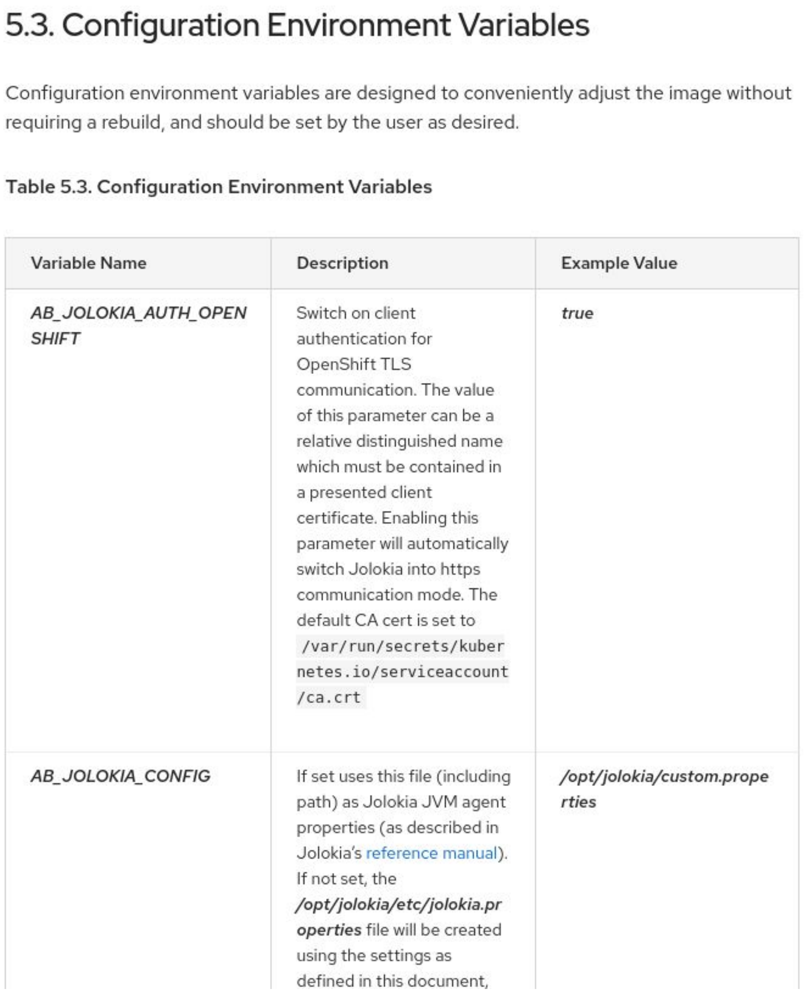
- Startup Time
- Peak Performance
- Time to First Response
- Predictable Performance

What are Your Performance Goals?

- Start by setting performance goals
- Once you define the most important performance characteristics for your system, you can figure out which parameters to change and how to change them
- One important question to answer is whether you want to focus on **minimizing application response times** or **maximizing throughput**
- Max pause and throughput are trade-offs, so you have to decide which is the higher priority
- Application requirements largely determine whether it is preferable to have more frequent collections of a shorter duration or less frequent collections that last longer

There’s no Silver Bullet …

Configuration Environment Variables

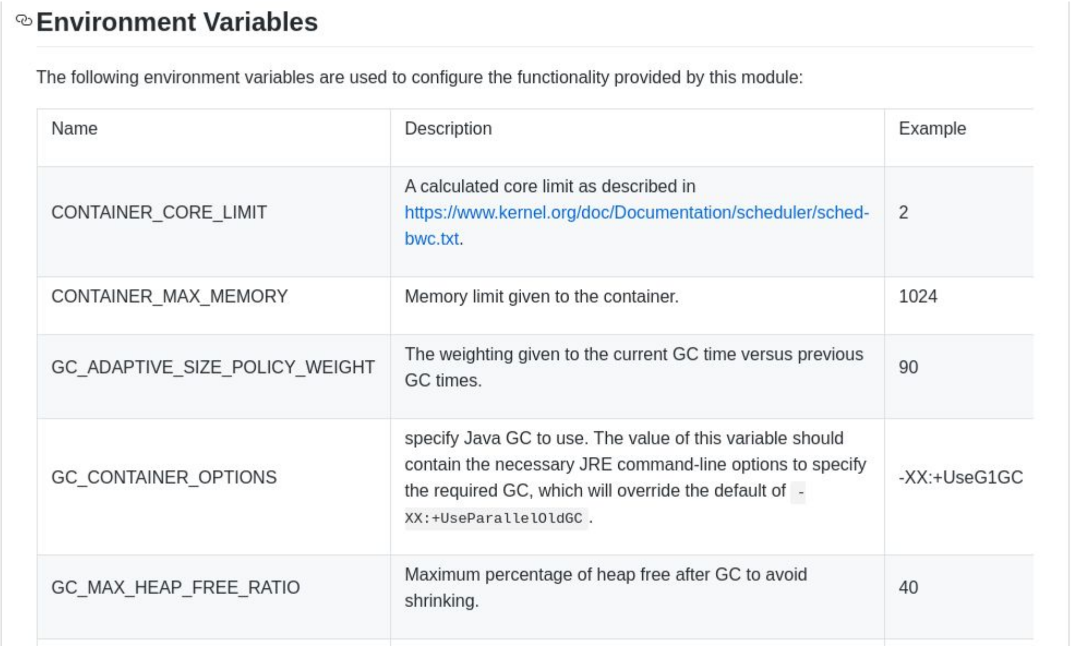


5.3. Configuration Environment Variables

Configuration environment variables are designed to conveniently adjust the image without requiring a rebuild, and should be set by the user as desired.

Table 5.3. Configuration Environment Variables

Variable Name	Description	Example Value
AB_JOLOKIA_AUTH_OPENSHIFT	Switch on client authentication for OpenShift TLS communication. The value of this parameter can be a relative distinguished name which must be contained in a presented client certificate. Enabling this parameter will automatically switch Jolokia into https communication mode. The default CA cert is set to `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`	*true*
AB_JOLOKIA_CONFIG	If set uses this file (including path) as Jolokia JVM agent properties (as described in Jolokia's reference manual). If not set, the ***/opt/jolokia/etc/jolokia.properties*** file will be created using the settings as defined in this document,	*/opt/jolokia/custom.properties*



Environment Variables

The following environment variables are used to configure the functionality provided by this module:

Name	Description	Example
CONTAINER_CORE_LIMIT	A calculated core limit as described in https://www.kernel.org/doc/Documentation/scheduler/sched-bwc.txt.	2
CONTAINER_MAX_MEMORY	Memory limit given to the container.	1024
GC_ADAPTIVE_SIZE_POLICY_WEIGHT	The weighting given to the current GC time versus previous GC times.	90
GC_CONTAINER_OPTIONS	specify Java GC to use. The value of this variable should contain the necessary JRE command-line options to specify the required GC, which will override the default of `-XX:+UseParallelOldGC`.	-XX:+UseG1GC
GC_MAX_HEAP_FREE_RATIO	Maximum percentage of heap free after GC to avoid shrinking.	40

Java Command Line Flags

- In general, the JVM accepts two types of flags:
  - Boolean
    - `-XX:+Flagname` (enables), `-XX:-Flagname` (disables)
  - Parameter
    - `-XX:Flagname=value`
- Default flag values are based on factors of the JVM version, platform
  - https://chriswhocodes.com/vm-options-explorer.html

Java Command Line Flags

- `-XX:+PrintFlagsFinal` prints all options and their values used by the JVM

```
$ java -XX:+UnlockDiagnosticVMOptions -XX:+PrintFlagsFinal -version
```

```
[Global flags]
      int AVX3Threshold                            = 4096          {ARCH diagnostic}    {default}
     bool AbortVMOnCompilationFailure              = false         {diagnostic}         {default}
    ccstr AbortVMOnException                       =               {diagnostic}         {default}
    ccstr AbortVMOnExceptionMessage                =               {diagnostic}         {default}
     bool AbortVMOnSafepointTimeout                = false         {diagnostic}         {default}
     bool AbortVMOnVMOperationTimeout              = false         {diagnostic}         {default}
     intx AbortVMOnVMOperationTimeoutDelay         = 1000          {diagnostic}         {default}
      int ActiveProcessorCount                     = -1            {product}            {default}
...
```

Monitoring/Metrics

Understanding Java Process Memory Allocation

- The following are parts of the memory required by an active Java process:
  - Implementation of the JVM
  - The (C - manual) heap for data structures implementing the JVM
  - Stacks for all of the threads in the system (app + JVM)
  - Cached Java bytecode (for libraries and the application)
  - Static variables of all loaded classes (PermGen)
- These can often reflect the `Xmx` heap exceeded what is set by "`-Xmx`"
- The following blogs explain how the whole heap/stack adds up:
  - https://www.baeldung.com/native-memory-tracking-in-jvm
  - https://shipilev.net/jvm/anatomy-quarks/
  - https://plumbr.io/blog/memory-leaks/why-does-my-java-process-consume-more-memory-than-xmx

Best Practices

- Raise Pod memory limits to reduce the alerts
- Do not raise heap allocation when raising Pod memory limits
- Do not allocate all Pod memory to the JVM Heap because there are memory requests that the JVM makes outside of the heap (code caches, data tables, fast data structures)

Determining & Setting Heap

```
$ docker run openjdk:11 java -XshowSettings:vm -version
  VM settings:
      Max. Heap Size (Estimated): 1.92G
      Using VM: OpenJDK 64-Bit Server VM

  openjdk version "11.0.7" 2020-04-14
  OpenJDK Runtime Environment 18.9 (build 11.0.7+10)
  OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10, mixed mode)
```

Determining & Setting Heap

```
$ podman run openjdk:11 java -XX:MaxRAMPercentage=25 -XshowSettings:vm -version
  VM settings:
      Max. Heap Size (Estimated): 1.92G
      Using VM: OpenJDK 64-Bit Server VM

  openjdk version "11.0.7" 2020-04-14
  OpenJDK Runtime Environment 18.9 (build 11.0.7+10)
  OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10, mixed mode)
```

Determining & Setting Heap

```
$ docker run openjdk:11 java -XX:MaxRAMPercentage=50 -XshowSettings:vm -version

  VM settings:
      Max. Heap Size (Estimated): 3.84G
      Using VM: OpenJDK 64-Bit Server VM

  openjdk version "11.0.7" 2020-04-14
  OpenJDK Runtime Environment 18.9 (build 11.0.7+10)
  OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10, mixed mode)
```

Determining & Setting Heap

```
$ podman run -m 1GB openjdk:8 java -XshowSettings:vm -version
  VM settings:
      Max. Heap Size (Estimated): 247.50M
      Ergonomics Machine Class: server
      Using VM: OpenJDK 64-Bit Server VM

  openjdk version "1.8.0_252"
  OpenJDK Runtime Environment (build 1.8.0_252-b09)
  OpenJDK 64-Bit Server VM (build 25.252-b09, mixed mode)
```

Determining & Setting Heap

```
$ docker run openjdk:11 java -Xms512M -Xmx1G -XshowSettings:vm -version
VM settings:
    Min. Heap Size: 512.00M
    Max. Heap Size: 1.00G
    Using VM: OpenJDK 64-Bit Server VM

openjdk version "11.0.7" 2020-04-14
OpenJDK Runtime Environment 18.9 (build 11.0.7+10)
OpenJDK 64-Bit Server VM 18.9 (build 11.0.7+10, mixed mode)
```

Other Useful Flags: `-XX:+UseContainerSupport`

- Introduced in Java 10
- Backported to Java 8 (u191)

```
$ docker run openjdk:8 java -XX:+PrintFlagsFinal -version | grep ContainerSupport
  bool UseContainerSupport         = true          {product}
  openjdk version "1.8.0_252"
  OpenJDK Runtime Environment (build 1.8.0_252-b09)
  OpenJDK 64-Bit Server VM (build 25.252-b09, mixed mode)
```

Native Memory Tracking (NMT)

- NMT instruments and categorizes all internal VM allocations:

```
-XX:+UnlockDiagnosticVMOptions  # Enables the feature
-XX:NativeMemoryTracking=        # args can be off|detail|summary
-XX:+PrintNMTStatistics          # Will print the JVM process statistics on exit
```

- Enabling NMT will result in a 5-10 percent JVM performance drop and memory usage for NMT as it adds 2 machine words to all malloc memory as malloc header
- https://shipilev.net/jvm/anatomy-quarks/12-native-memory-tracking/

Native Memory Tracking (NMT)

```
$ java -XX:+UnlockDiagnosticVMOptions -XX:NativeMemoryTracking=summary HelloFX

$ jcmd

20917 HelloFX
20968 jdk.jcmd/sun.tools.jcmd.JCmd

$ jcmd 20917 VM.native_memory summary

20917:

Native Memory Tracking:

Total: reserved=3489518KB, committed=220658KB

-                 Java Heap (reserved=2015232KB, committed=129024KB)
                            (mmap: reserved=2015232KB, committed=129024KB)
 ...
```

jcmd

Description	Command
List Java Processes	`jcmd`
Heap Dumps	`jcmd <pid> GC.heap_dump`
Heap Usage Histogram	`jcmd <pid> GC.class_histogram`
Thread Dump	`jcmd <pid> Thread.print`
List System Properties	`jcmd <pid> VM.system_properties`
VM process Info	`jcmd <pid> VM.info`

Other Useful Commands ...

To check physical memory, you can run the following command line:

```
$ oc adm top pod <pod_name>
```

And inside the pod:

```
sh-4.2$ cat /sys/fs/cgroup/memory/memory.stat
```

Tuning

Garbage Collection

Garbage Collector Primer

- When does the choice of a garbage collector matter?
- For some applications, the answer is never
- That is, the application can perform well in the presence of garbage collection with pauses of modest frequency and duration
- However, this is not the case for a large class of applications, particularly those with large amounts of data (multiple gigabytes), many threads, and high transaction rates

Garbage Collector Primer

- The Garbage collector (GC) is a memory management tool
- It achieves automatic memory management through the following operations:
  - Allocating objects to a young generation and promoting aged objects into an old generation
  - Finding live objects in the old generation through a concurrent (parallel) marking phase
    - The VM triggers the marking phase when the total Java heap occupancy exceeds the default threshold
  - Recovering free memory by compacting live objects through parallel copying

GC Summary

- Tuning garbage collection is not easy
- Tuning the GC may result in negligible pause times
  - Test the guidelines
- Enable GC logging to determine where it’s spending time
- G1 Collector may be appropriate for your workload

Your mileage may vary

Other Tuning Options

Application Class Data-Sharing (CDS)

- Class-Data Sharing (CDS) was introduced to make the metadata of the pre-loaded classes available in a shared file, which could be shared in multiple instances of the JVM
- Application Class-Data Sharing (JEP 310) was introduced in Java 10+
- An extension of CDS that aims to allow pre-loading of metadata files, bootstrap classes and other JDK and application classes
- CDS only works for classes loaded from modules or JAR files
- Now supported with Quarkus 1.6 (Hotspot VM)

Application Class Data-Sharing (CDS)

- Generate the file with the class metadata for your application.

```
$ java -XX:DumpLoadedClassList=app-classes.txt -jar your-app.jar
```

- Convert the file app-classes.txt to a file with the class metadata that can be understood by the JVM:

```
$ java -Xshare:dump -XX:SharedClassListFile=app-classes.txt \
-XX:SharedArchiveFile=app-classes.jsa --class-path your-app.jar

$ java -XX:SharedArchiveFile=app-classes.jsa -jar your-app.jar
```

Checkpointing (CRIU)

- Checkpoint/Restore In Userspace, or CRIU can freeze a running container (or an individual application) and checkpoint its state to disk
- The data saved can be used to restore the application and run it exactly as it was during the time of the freeze
- Using this functionality, application or container live migration, snapshots, remote debugging, and many other things are now possible.
- Demo: https://asciinema.org/a/FsTbx9mZkzeuhCM2pFOr1tujM

`jlink`

- Tool that generates a custom Java runtime image that contains only the platform modules that are required for a given application
- Runtime image acts exactly like the JRE but contains only the modules we picked and the dependencies they need to function
- The concept of modular runtime images was introduced in JEP 220
- Requires Java 9+

jlink

```
$ jdeps Hello.class

HelloWorld.class -> java.base
   <unnamed>           -> java.io           java.base
   <unnamed>           -> java.lang         java.base


$ jlink --add-modules java.base --output customjre
```

Using `jdeps`

- `jdeps` is a Java Class Dependency Analyzer
- Java 8+ JDKs
- Analyzes the dependencies by class or package (default) level
- Not just for migrating to modularity

Using jdeps

# View a list of dependencies of your application

```
$ jdeps <path to jar>
```

# Shows dependencies at the class level, useful when refactoring code to avoid internal APIs

```
$ jdeps -v <path to jar>
```

# Shows only dependencies belonging to a certain package

```
$ jdeps -v -p java.lang <path to jar>
```

# Shows packages nested within java.lang

```
$ jdeps -v -e java.lang.* <path to jar>
```

# Filters out dependencies by regex pattern

```
$ jdeps -v -filter java.lang.* <path to jar>
```

# Show only dependencies on JDK internal classes

```
$ jdeps -jdkinternals <path to jar>
```

Spring Tips

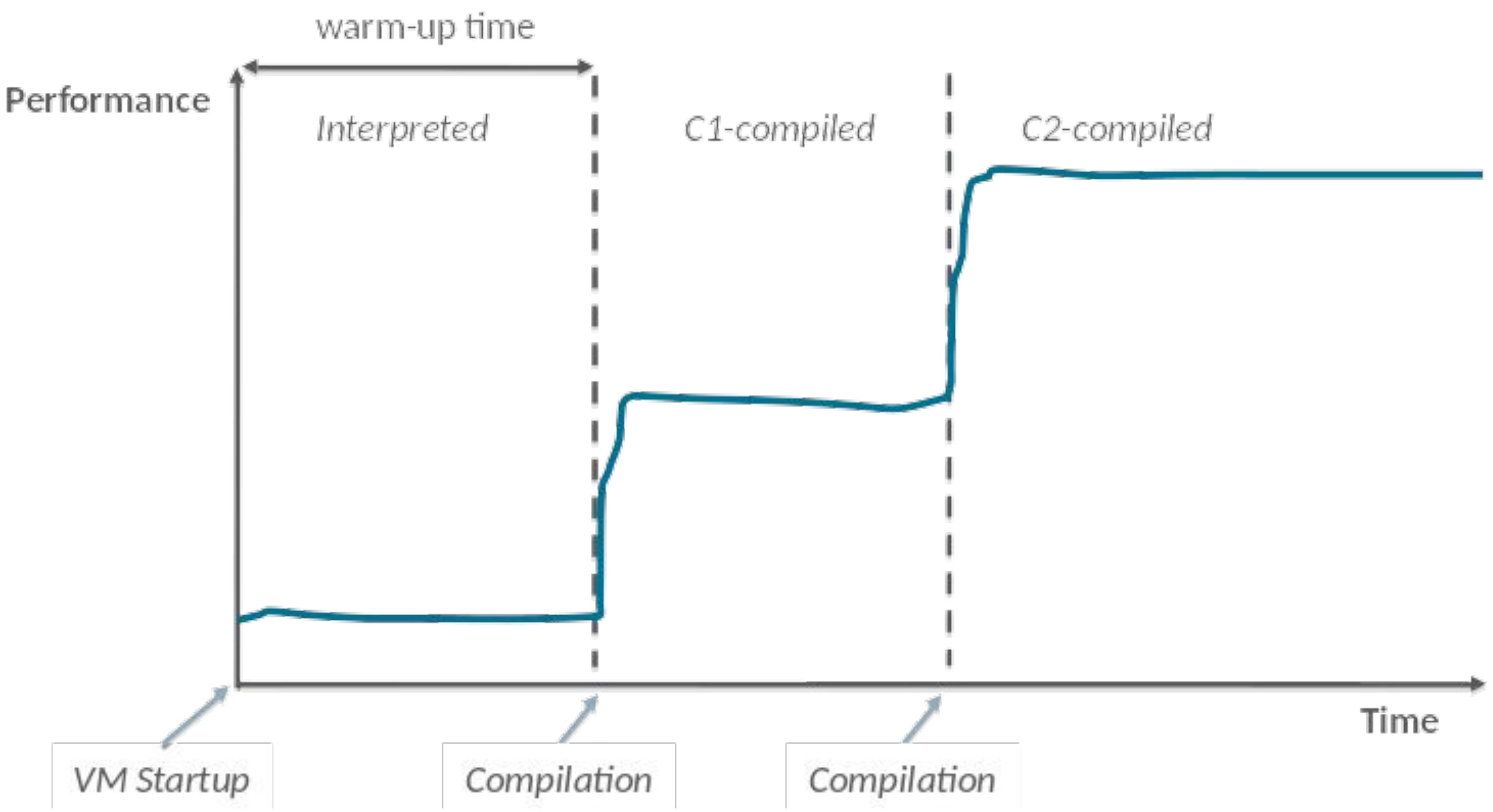
- Disable JMX, you may or may not need it in your container:
  `spring.jmx.enabled=false`

- Run the JVM with `-noverify`
  - Make certain the app dependencies match the JDK version used to build the application
  - Be aware disabling verification can lead to potential security compromises or crashes
  - When possible, use the latest dependency versions rather than `-noverify`

- Consider `-XX:TieredStopAtLevel=1`
  - Enabling this option will slow the JIT (for long-running apps) at the expense of faster startup time … so it's a tradeoff

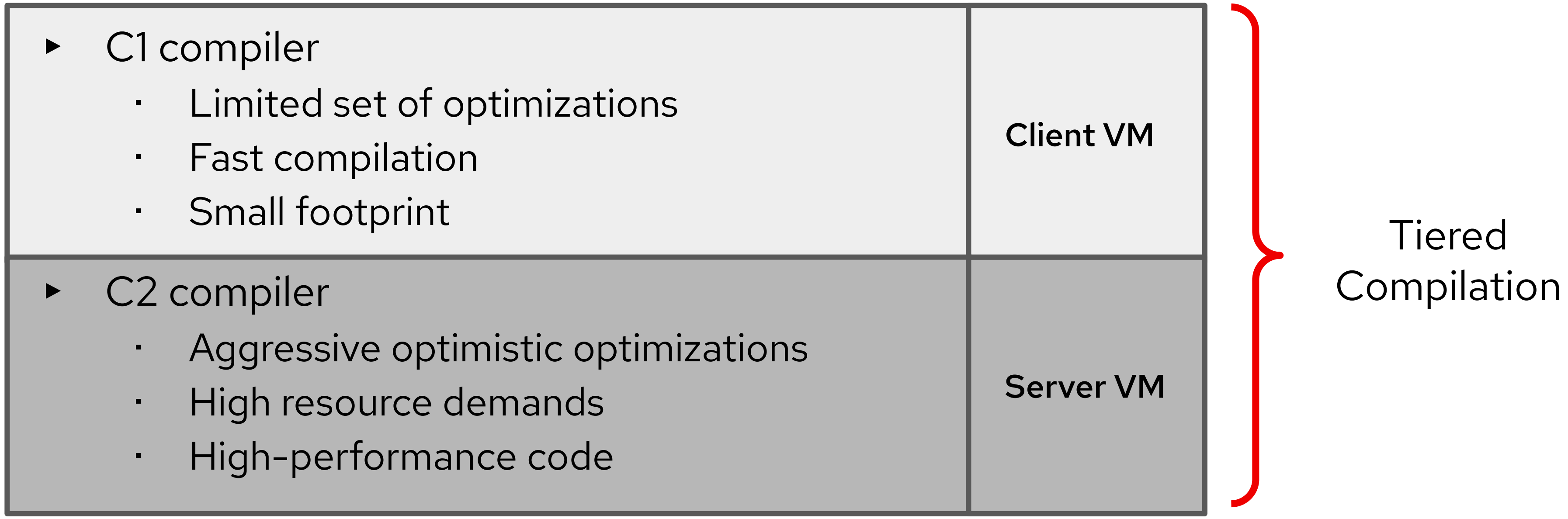
- Use the container memory hints for Java 8 (prior to u181):
  - `-XX:+UnlockExperimentalVMOptions -XX:+UseCGroupMemoryLimitForHeap`
  - Enabled with later updates of Java 8 (u181+), enabled by default with Java 11
  - You'll encounter some issues if your OS is using cgroups v2

Tiered Compilation



Credit: Tobias Hartman, Oracle

Tiered Compilation



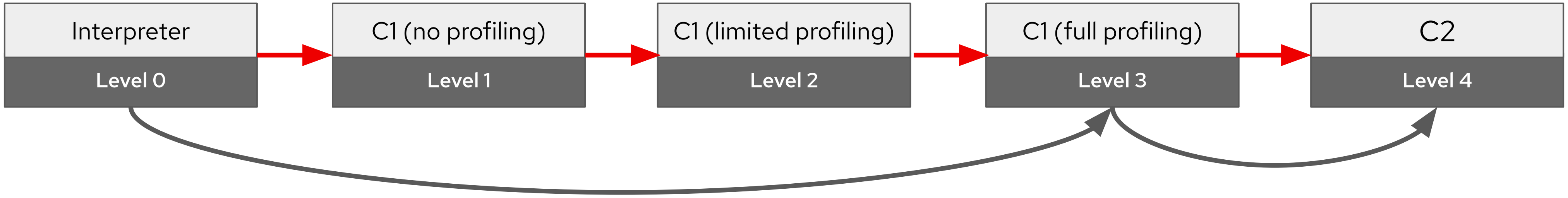
- C1 compiler
  - Limited set of optimizations
  - Fast compilation
  - Small footprint

Client VM

- C2 compiler
  - Aggressive optimistic optimizations
  - High resource demands
  - High-performance code

Server VM

Tiered Compilation



Interpreter
Level 0

C1 (no profiling)
Level 1

C1 (limited profiling)
Level 2

C1 (full profiling)
Level 3

C2
Level 4

Tiered Compilation

```
$ java -jar target/spring-petclinic-2.3.1.BUILD-SNAPSHOT.jar

Started PetClinicApplication in 18.307 seconds (JVM running for 19.322)


$ java -XX:TieredStopAtLevel=1 -jar target/spring-petclinic-2.3.1.BUILD-SNAPSHOT.jar

Started PetClinicApplication in 9.489 seconds (JVM running for 10.125)


$ java -noverify -XX:TieredStopAtLevel=1 -jar \
target/spring-petclinic-2.3.1.BUILD-SNAPSHOT.jar

Started PetClinicApplication in 6.32 seconds (JVM running for 6.779)
```

Ahead of Time Compiler (AOT)

- AOT compiler's primary capability is to generate machine code for an application without having to run the application, allowing a future run of the application to pick the generated code
- Similarly, to C1 and C2, `jaotc` compiles Java bytecode to native code
- The primary motivation behind using AOT in Java is to bypass the interpreter
- It is generally faster for the machine to execute machine code than it is to execute the code via the bytecode interpreter
- In many cases, it is a definite advantage, especially for code that needs to be executed even just a few times
- AOT is a use case for short running programs, which finish execution before any JIT compilation occurs

Ahead of Time Compiler (AOT)

```
$ javac HelloWordAOT.java

$ java HelloWorldAOT

Hello, World


$ jaotc --compile-for-tiered --output libHelloWorldAOT.so --verbose HelloWorldAOT

Compiling libHelloWorldAOT.so…

...

$ java -XX:+UnlockExperimentalVMOptions -XX:AOTLibrary=./libHelloWorldAOT.so HelloWorldAOT

Hello, World
```

JVM Config Options Tool

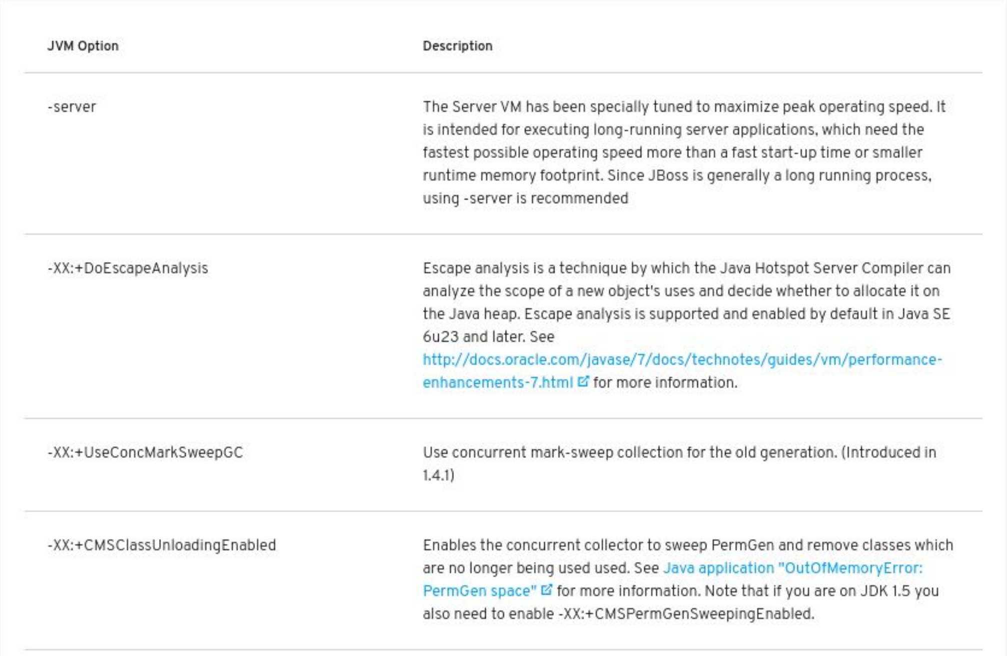
https://access.redhat.com/labsinfo/jvmconfig

- JVM config tool helps tune to avoid common problems
- Generates optimized settings for your application, based on our experience with a wide range of synthetic and real-world applications
- We recommend using this tool to provide a baseline JVM configuration*
- Also explains why each option is generated and links back to the Red Hat knowledge base for known issues and solutions

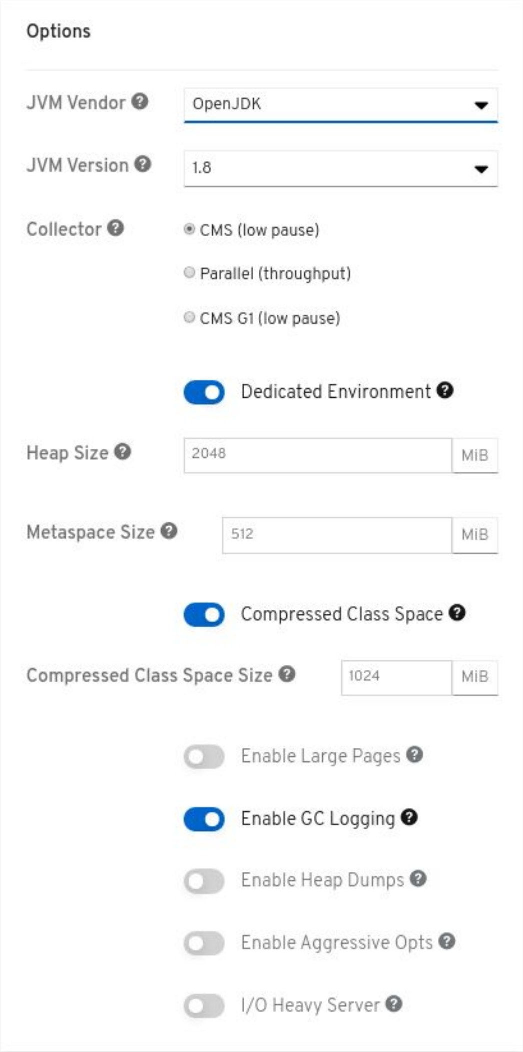
* Additional JVM tuning requires running the application under simulated load with garbage collection enabled and analyzing the garbage collection logging maximum pause, overall throughput, and throughput bottlenecks

JVM Config Options Tool

https://access.redhat.com/labsinfo/jvmconfig



JVM Option	Description
-server	The Server VM has been specially tuned to maximize peak operating speed. It is intended for executing long-running server applications, which need the fastest possible operating speed more than a fast start-up time or smaller runtime memory footprint. Since JBoss is generally a long running process, using -server is recommended
-XX:+DoEscapeAnalysis	Escape analysis is a technique by which the Java Hotspot Server Compiler can analyze the scope of a new object's uses and decide whether to allocate it on the Java heap. Escape analysis is supported and enabled by default in Java SE 6u23 and later. See http://docs.oracle.com/javase/7/docs/technotes/guides/vm/performance-enhancements-7.html for more information.
-XX:+UseConcMarkSweepGC	Use concurrent mark-sweep collection for the old generation. (Introduced in 1.4.1)
-XX:+CMSClassUnloadingEnabled	Enables the concurrent collector to sweep PermGen and remove classes which are no longer being used used. See Java application "OutOfMemoryError: PermGen space" for more information. Note that if you are on JDK 1.5 you also need to enable -XX:+CMSPermGenSweepingEnabled.



Options

JVM Vendor: OpenJDK

JVM Version: 1.8

Collector:
● CMS (low pause)
○ Parallel (throughput)
○ CMS G1 (low pause)

☑ Dedicated Environment

Heap Size: 2048 MiB

Metaspace Size: 512 MiB

☑ Compressed Class Space

Compressed Class Space Size: 1024 MiB

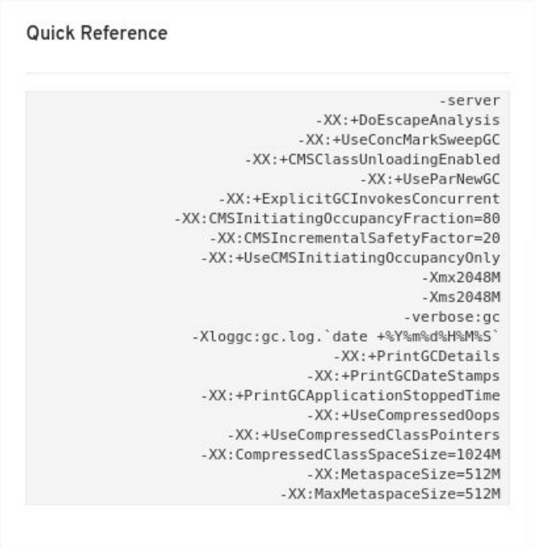
☐ Enable Large Pages

☑ Enable GC Logging

☐ Enable Heap Dumps

☐ Enable Aggressive Opts

☐ I/O Heavy Server



Quick Reference

```
                              -server
                -XX:+DoEscapeAnalysis
              -XX:+UseConcMarkSweepGC
        -XX:+CMSClassUnloadingEnabled
                     -XX:+UseParNewGC
     -XX:+ExplicitGCInvokesConcurrent
-XX:CMSInitiatingOccupancyFraction=80
    -XX:CMSIncrementalSafetyFactor=20
   -XX:+UseCMSInitiatingOccupancyOnly
                            -Xmx2048M
                            -Xms2048M
                          -verbose:gc
  -Xloggc:gc.log.`date +%Y%m%d%H%M%S`
                  -XX:+PrintGCDetails
               -XX:+PrintGCDateStamps
   -XX:+PrintGCApplicationStoppedTime
               -XX:+UseCompressedOops
      -XX:+UseCompressedClassPointers
   -XX:CompressedClassSpaceSize=1024M
               -XX:MetaspaceSize=512M
            -XX:MaxMetaspaceSize=512M
```

Optimizing Container Builds

Container Image Size

Repository	Tag	Size
openjdk	8	510 MB
openjdk	8-jdk-slim	285 MB
openjdk	8-jre	265 MB
openjdk	8-jre-slim	184 MB

stats

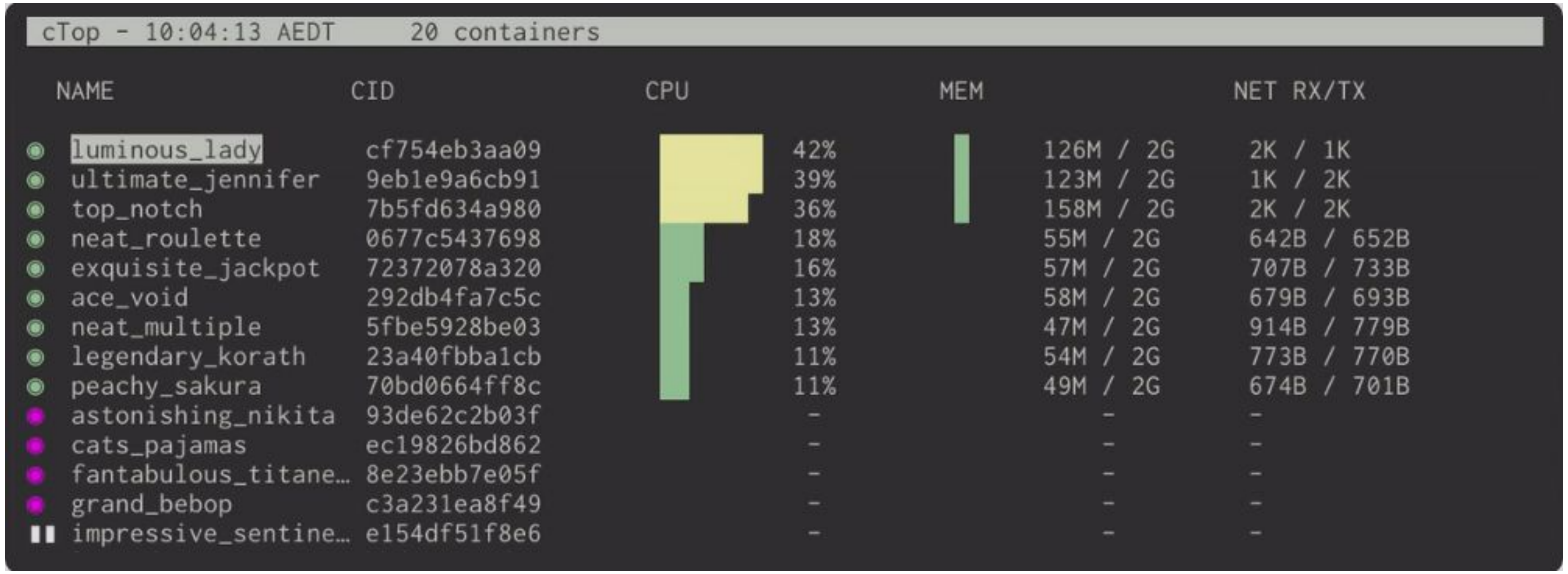
```
$ docker stats
ID             NAME             CPU %   MEM USAGE / LIMIT   MEM %   NET IO    BLOCK IO    PIDS
1d60aed50b65   brave_roentgen   7.38%   400.5MB / 24.85GB   1.61%   -- / --   -- / --     64
...
```

image history

```
$ podman image history spring-boot-podman
ID             CREATED        CREATED BY                                         SIZE
44bf8274d128   5 weeks ago    /bin/sh -c #(nop) ENTRYPOINT ["java","-cp"...
16.74MB
f5de33dc9079   5 weeks ago    /bin/sh -c #(nop) COPY dir:fbf8b3938d1d0ee...      0B
<missing>      5 weeks ago    /bin/sh -c #(nop) COPY dir:60ee44dab014f44...      0B
<missing>      5 weeks ago    /bin/sh -c #(nop) COPY dir:923792c91b1ae43...      0B
<missing>      5 weeks ago    /bin/sh -c #(nop) ARG DEPENDENCY=target/de...      0B
<missing>      5 weeks ago    /bin/sh -c #(nop) VOLUME /tmp                      0B
<missing>      7 weeks ago    /bin/sh -c #(nop) CMD ["jshell"]                   0B
<missing>      7 weeks ago    /bin/sh -c set -eux; dpkgArch="$(dpkg --pr...
323.2MB
<missing>      7 weeks ago    /bin/sh -c #(nop) ENV JAVA_URL_VERSION=11....      0B
<missing>      7 weeks ago    /bin/sh -c #(nop) ENV JAVA_BASE_URL=https:...      0B
<missing>      7 weeks ago    /bin/sh -c #(nop) ENV JAVA_VERSION=11.0.7          0B
<missing>      7 weeks ago    /bin/sh -c { echo '#/bin/sh'; echo 'echo "...
3.584kB
...
```

Other Useful Commands … ctop (docker)

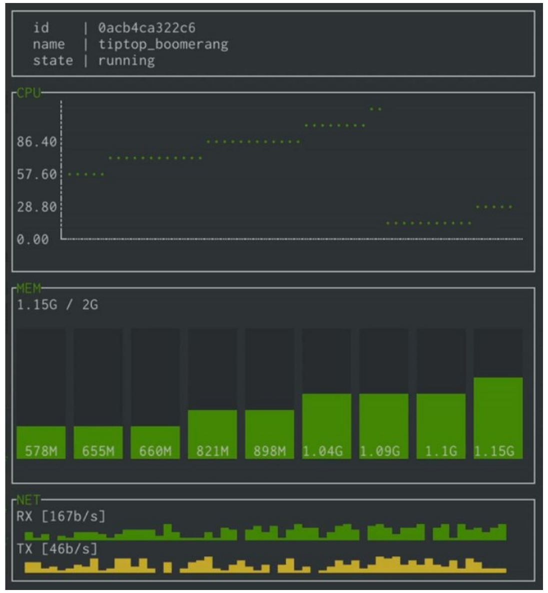
- Command-line monitoring for containers



```
cTop - 10:04:13 AEDT      20 containers

  NAME                CID              CPU                 MEM                 NET RX/TX

● luminous_lady       cf754eb3aa09          42%                126M / 2G    2K / 1K
● ultimate_jennifer   9eb1e9a6cb91          39%                123M / 2G    1K / 2K
● top_notch           7b5fd634a980          36%                158M / 2G    2K / 2K
● neat_roulette       0677c5437698          18%                55M / 2G     642B / 652B
● exquisite_jackpot   72372078a320          16%                57M / 2G     707B / 733B
● ace_void            292db4fa7c5c          13%                58M / 2G     679B / 693B
● neat_multiple       5fbe5928be03          13%                47M / 2G     914B / 779B
● legendary_korath    23a40fbba1cb          11%                54M / 2G     773B / 770B
● peachy_sakura       70bd0664ff8c          11%                49M / 2G     674B / 701B
● astonishing_nikita  93de62c2b03f          -                   -            -
● cats_pajamas        ec19826bd862          -                   -            -
● fantabulous_titane… 8e23ebb7e05f          -                   -            -
● grand_bebop         c3a231ea8f49          -                   -            -
▮▮ impressive_sentine… e154df51f8e6          -                   -            -
```

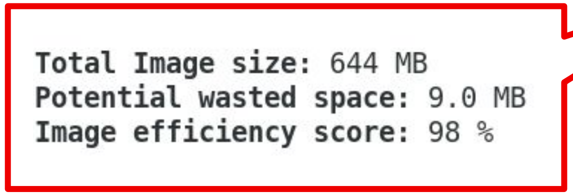


```
id    | 0acb4ca322c6
name  | tiptop_boomerang
state | running
```

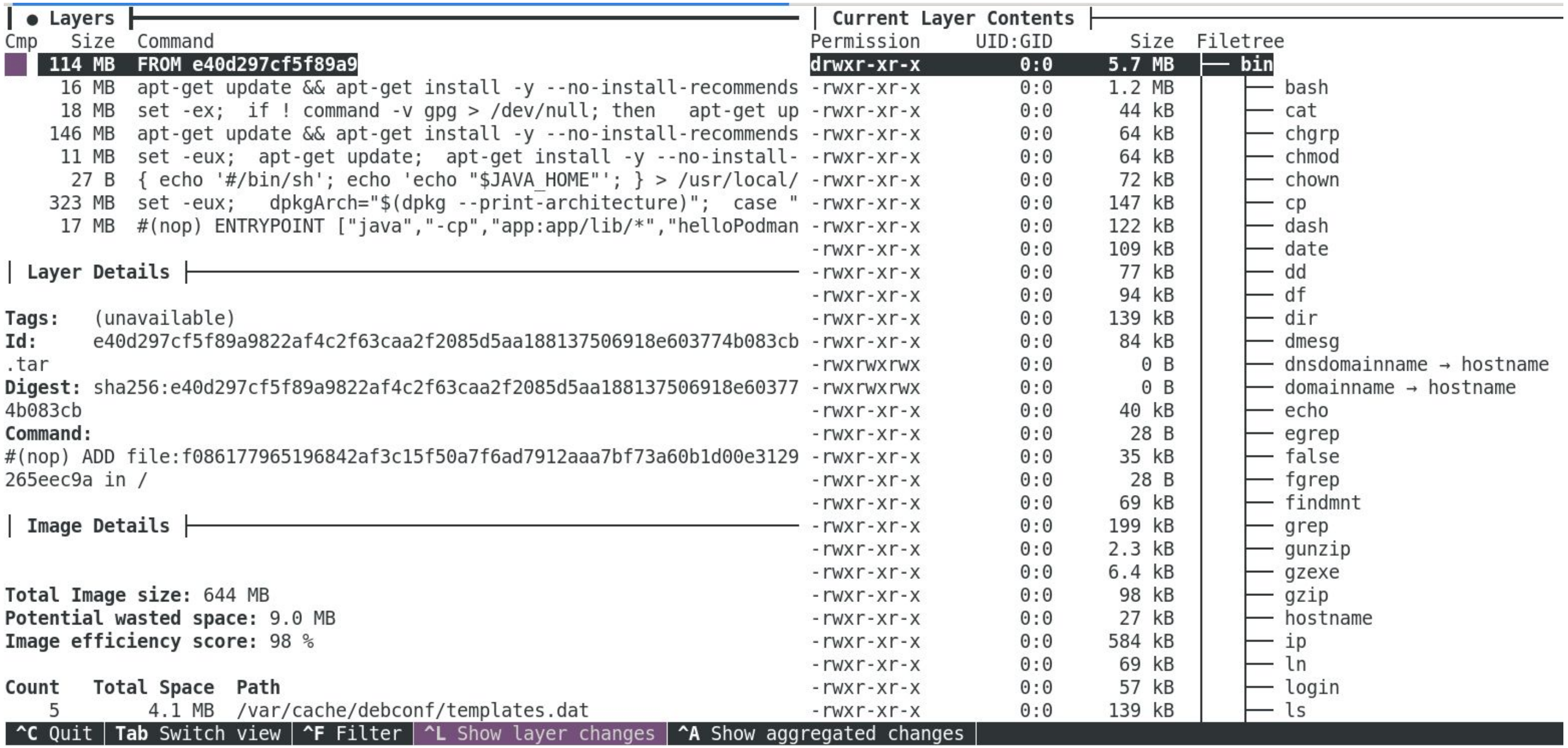
- https://ctop.sh/

dive

- A tool for exploring a container image, layer contents, and discovering ways to shrink the size of your Docker/OCI image



```
Total Image size: 644 MB
Potential wasted space: 9.0 MB
Image efficiency score: 98 %
```



```
│ ● Layers ├──────────────────────────────────────────────────────────── │ Current Layer Contents ├──────────────────────────
Cmp   Size  Command                                                      Permission     UID:GID       Size  Filetree
█   114 MB  FROM e40d297cf5f89a9                                         drwxr-xr-x         0:0     5.7 MB  ├── bin
     16 MB  apt-get update && apt-get install -y --no-install-recommends -rwxr-xr-x         0:0     1.2 MB  │   ├── bash
     18 MB  set -ex;  if ! command -v gpg > /dev/null; then   apt-get up -rwxr-xr-x         0:0      44 kB  │   ├── cat
    146 MB  apt-get update && apt-get install -y --no-install-recommends -rwxr-xr-x         0:0      64 kB  │   ├── chgrp
     11 MB  set -eux;  apt-get update;  apt-get install -y --no-install- -rwxr-xr-x         0:0      64 kB  │   ├── chmod
      27 B  { echo '#/bin/sh'; echo 'echo "$JAVA_HOME"'; } > /usr/local/ -rwxr-xr-x         0:0      72 kB  │   ├── chown
    323 MB  set -eux;   dpkgArch="$(dpkg --print-architecture)";  case " -rwxr-xr-x         0:0     147 kB  │   ├── cp
     17 MB  #(nop) ENTRYPOINT ["java","-cp","app:app/lib/*","helloPodman -rwxr-xr-x         0:0     122 kB  │   ├── dash
                                                                         -rwxr-xr-x         0:0     109 kB  │   ├── date
│ Layer Details ├─────────────────────────────────────────────────────── -rwxr-xr-x         0:0      77 kB  │   ├── dd
                                                                         -rwxr-xr-x         0:0      94 kB  │   ├── df
Tags:    (unavailable)                                                   -rwxr-xr-x         0:0     139 kB  │   ├── dir
Id:      e40d297cf5f89a9822af4c2f63caa2f2085d5aa188137506918e603774b083cb -rwxr-xr-x         0:0      84 kB  │   ├── dmesg
.tar                                                                     -rwxrwxrwx         0:0        0 B  │   ├── dnsdomainname → hostname
Digest: sha256:e40d297cf5f89a9822af4c2f63caa2f2085d5aa188137506918e60377 -rwxrwxrwx         0:0        0 B  │   ├── domainname → hostname
4b083cb                                                                  -rwxr-xr-x         0:0      40 kB  │   ├── echo
Command:                                                                 -rwxr-xr-x         0:0       28 B  │   ├── egrep
#(nop) ADD file:f086177965196842af3c15f50a7f6ad7912aaa7bf73a60b1d00e3129 -rwxr-xr-x         0:0      35 kB  │   ├── false
265eec9a in /                                                            -rwxr-xr-x         0:0       28 B  │   ├── fgrep
                                                                         -rwxr-xr-x         0:0      69 kB  │   ├── findmnt
│ Image Details ├─────────────────────────────────────────────────────── -rwxr-xr-x         0:0     199 kB  │   ├── grep
                                                                         -rwxr-xr-x         0:0     2.3 kB  │   ├── gunzip
                                                                         -rwxr-xr-x         0:0     6.4 kB  │   ├── gzexe
Total Image size: 644 MB                                                 -rwxr-xr-x         0:0      98 kB  │   ├── gzip
Potential wasted space: 9.0 MB                                           -rwxr-xr-x         0:0      27 kB  │   ├── hostname
Image efficiency score: 98 %                                             -rwxr-xr-x         0:0     584 kB  │   ├── ip
                                                                         -rwxr-xr-x         0:0      69 kB  │   ├── ln
Count   Total Space  Path                                                -rwxr-xr-x         0:0      57 kB  │   ├── login
    5        4.1 MB  /var/cache/debconf/templates.dat                    -rwxr-xr-x         0:0     139 kB  │   ├── ls
^C Quit │ Tab Switch view │ ^F Filter │ ^L Show layer changes │ ^A Show aggregated changes │
```

https://github.com/wagoodman/dive

gnomon

```
$ podman build . | gnomon
   0.0076s   STEP 1: FROM registry.access.redhat.com/ubi8/ubi-minimal
   0.5017s   STEP 2: RUN microdnf install java-11-openjdk --nodocs
   1.1076s   STEP 2: RUN microdnf install java-11-openjdk --nodocsnf.conf":
   2.0081s   Downloading metadata...
   2.3106s   Downloading metadata...
   1.3181s   Downloading metadata...
   0.0001s   Package                                            Repository          Size
   0.0001s   Installing:
   0.0000s    abattis-cantarell-fonts-0.0.25-4.el8.noarch       ubi-8-appstream 159.0  kB
   0.0000s    acl-2.2.53-1.el8.x86_64                           ubi-8-baseos      83.0  kB
   ...
   0.0489s   STEP 8: COMMIT
   0.3988s   --> cb2d530c0f9
   0.0229s   cb2d530c0f9b680f450c8aadcd098a48881d0ab02acb43d7d9472320a73f3427
   0.0004s
     Total   63.4261s
```

https://github.com/paypal/gnomon

Resources

Additional Resources

Links and such …

- Java garbage collection long pause times
  - https://access.redhat.com/solutions/19932
- G1 Collector Tuning
  - https://access.redhat.com/solutions/2162391
- How do I analyze Java garbage collection logging?
  - https://access.redhat.com/solutions/23735
- Improving OpenJDK Garbage Collection Performance
  - https://access.redhat.com/articles/1192773

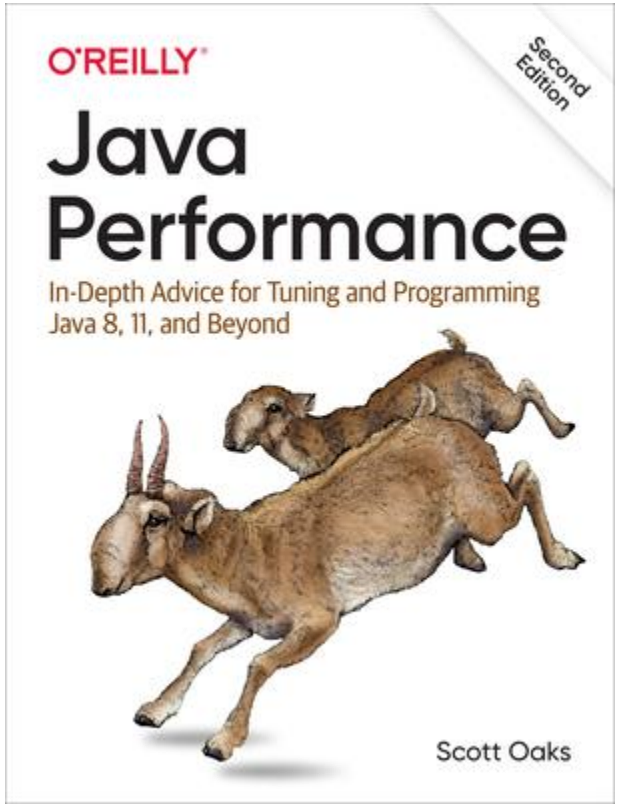
Additional Resources

Links and such ...

- How do I enable Java garbage collection logging?
  - https://access.redhat.com/solutions/18656
- How do I analyze Java garbage collection logging?
  - https://access.redhat.com/solutions/23735
- VM Options Explorer
  - https://chriswhocodes.com/vm-options-explorer.html
  - https://chriswhocodes.com/hotspot_option_differences.html
- Java Command Line Inspector
  - https://chriswhocodes.com/vm-options-explorer.html

Additional Resources

Links and such …


O'REILLY
Second Edition
Java Performance
In-Depth Advice for Tuning and Programming Java 8, 11, and Beyond
Scott Oaks

Action ...

- We can help you assess your current Java environment and provide guidance on how you can achieve your performance goals
- Whether you want to focus on minimizing application response times, maximizing throughput or improving startup performance
- Schedule an in-depth session to discuss Java performance optimization and best practices

Q & A

Thank you

Red Hat is the world's leading provider of enterprise open source software solutions. Award-winning support, training, and consulting services make Red Hat a trusted adviser to the Fortune 500.

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat