Do not put all eggs in one container
Who we are

Dmitry Chuyko

@dchuyko

BELLSOFT

Liberica JDK – verified OpenJDK binary
http://bell-sw.com

Ex-employers

ORACLE
2019. Microservices are in containers
2019. Microservices are in containers
Containers

- Linux containers
  - cgroups
  - namespaces
  - Isolation
  - Resource management
  - Not a virtualization

- Docker images
  - Configuration

- Docker tools
  - Management
  - Monitoring
  - Orchestration
Careless processes in containers

I POOPED MY PANTS

BUT THAT'S NONE OF MY BUSINESS
Virtually Machine
- OS process
- Runtime
- JIT/code
- GC

Expectations from containers
- Configuration
- Test ≠ Prod
- Isolation

We need Java tools
- Management
- Monitoring
- Debug
• **JDK-6515172** Runtime.availableProcessors() ignores Linux `taskset` command
  • `docker --cpuset-cpus`

• **JDK-8161993** G1 crashes if `active_processor_count` changes during startup

• **JDK-8170888** Experimental support for cgroup memory limits in container (ie Docker) environments
  • `-XX:+UseCGroupMemoryLimitForHeap`
  • `docker --memory`
• **JDK-8146115** Improve docker container detection and resource configuration usage
  - -XX:+UseContainerSupport
  - -XX:ActiveProcessorCount=N
  - -Xlog:os+container=trace
  - --cpus --cpu-quota --cpu-period
  - Deprecate experimental

• **JDK-8186248** Allow more flexibility in selecting Heap % of available RAM
  - -XX:InitialRAMPercentage
  - -XX:MaxRAMPercentage
  - -XX:MinRAMPercentage

• **JDK-8179498** attach in Linux should be relative to /proc/pid/root and namespace aware
• **JDK-8197867** Update CPU count algorithm when both cpu shares and quotas are used
  - `-XX:+PreferContainerQuotaForCPUCount`
  - `--cpu-shares`

• **JDK-8194086** Remove deprecated experimental flag `UseCGroupMemoryLimitForHeap`

• **JDK-8203357** Container Metrics
  - `-XshowSettings:system`

• **JDK-8193710** `jcmd -l` and `jps` commands do not list Java processes running in Docker containers
JDK 8 (AKA the best release)

• JDK 8 GA. “...none of my business”

• JDK 8u
  ✷ Backports from JDK 9
  ✷ Backports from JDK 10
  ✷ Backports from JDK 11
JDK 8 (AKA the best release)

- JDK 8 GA. “...none of my business”
- JDK 8u
  - Backports from JDK 9
  - Backports from JDK 10
  - Backports from JDK 11

¯\_(ツ)_/¯
JDK 12

- JEP 346: Promptly Return Unused Committed Memory from G1
  - May help in case of overcommit

JEP 346: Promptly Return Unused Committed Memory from G1

Authors: Rodrigo Bruno, Thomas Schatz, Rakan Synytsky

Status: Closed/Delivered

Component: HotSpot/GC

Description: Promptly return unused committed memory to the application.

Summary: Enhance the G1 garbage collector to automatically return Java heap memory to the operating system when idle.

Non-Goals:
- Sharing of committed but empty pages between Java processes. Memory should be returned (uncommitted) to the operating system.
- The process of giving back memory does not need to be frugal with CPU resources, nor does it need to be instantaneous.
- Use of different methods to return memory other than available uncommitted memory.
- Support for other collectors than G1.

Success Metrics:
G1 should release unused Java heap memory within a reasonable period of time if there is very low application activity.

Implementation:
Currently the G1 garbage collector may not return committed Java heap memory to the operating system in a timely manner. G1 only returns memory from the Java heap at either a full GC or during a concurrent cycle. Since G1 times hard to completely avoid full GCs, and only triggers a concurrent cycle based on Java heap occupancy and allocation activity, it will not return Java heap memory in many cases unless forced to do so externally. This behavior is particularly disadvantageous in container environments where resources are paid by use, and during phases where the VM only uses a fraction of its assigned memory resources due to inactivity, G1 will retain all of the Java heap. This results in customers paying for all resources all the time, and cloud providers not being able to fully utilize their hardware.

If the VM were able to detect phases of Java heap under-utilization (“idle” phases), and automatically release its heap usage during that time, both would benefit.

Implementation:
G1 should release unused Java heap memory within a reasonable period of time if there is very low application activity.

Description:
To accomplish the goal of returning a maximum amount of memory to the operating system, G1 will, during inactivity of the application, periodically try to continue or trigger a concurrent cycle to determine overall Java heap usage. This will cause it to automatically return unused portions of the Java heap back to the operating system. Optionally, under-use controls, a GC policy can be configured to monitor the amount of memory retained.

The implementation is considered inactive, and G1 triggers a periodic garbage collection if both:
- More than half of the memory is utilized. JVM monitors have passed and any previous garbage collection phases is to the next concurrent cycle in progress at the current time. A value of zero indicates that periodic garbage collections to promptly reclaim memory are disabled.
- The average user system load value as returned to the getSystem() call on the JVM system (e.g. a constant is below 50%) is below the configured threshold (x). This condition is removed if the JVM system load value exceeds the threshold (x). If either of these conditions are not met, the current prospective periodic garbage collection is cancelled. A periodic garbage collection is recomputed the next time JVM monitors time passes.

The type of periodic garbage collection is determined by the value of the JVM system load value (e.g. constant), which is monitored, or starts a concurrent cycle, otherwise G1 performs a full GC. At the end of either collection, G1 adjusts the current heap size, potentially returning memory to the operating system.

The new Java heap size is determined by the existing configuration for adjusting the Java heap size, including but not limited to the MaxPermSize and the MaxHeapSize configuration.

For instance, if G1 starts and continues a concurrent cycle during this periodic garbage collection, this minimizes disruption of the application, but compared to a full collection may ultimately not be able to return as much memory.

Any garbage collection triggered by this mechanism is tagged with the "Periodic" collection name. An example of how such a heap collect looks like is as follows:

<table>
<thead>
<tr>
<th>GC Name</th>
<th>Collectors</th>
<th>Mode</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>PS Parallel Young (Partial)</td>
<td>G1 Parallel Collection</td>
<td>2MB-4MB/2MB</td>
<td>1.79GB</td>
</tr>
<tr>
<td>PS Parallel Young (Complete)</td>
<td>G1 Parallel Collection</td>
<td>3MB-4MB/2MB</td>
<td>1.79GB</td>
</tr>
</tbody>
</table>

WWW.BELL-SW.COM
• JDK-8199944 Add Container MBean to JMX

• JDK-8203359 Create new events, and adjust existing events, to account for host/container reporting of resources

• JMC-5901 Utilize information from the host/container

• JDK-8198715 Investigate adding NUMA container support to hotspot
  • --cpuset-mems
Service Deployment

- CI / CD
- Business logic
- Spring
- Libraries
- Java

Maven
Gradle
spring boot
docker
In memory of one cherished image

Oracle is forcing @Docker HUB to delete an image that contains Oracle software such as JDK, oracle-xe-11g etc.

My image with 10M+ pulls has just gone (completely removed)
github.com/docker/hub-fee …

Reddit thread

My image with 10M+ pulls has just gone (completely removed).
It is a bummer, but I cannot reach Docker Hub team neither via e-mails, nor via Docker Hub Feedback.
red35.com

9:50 AM - 22 Jan 2019
114 Retweets 89 Likes

Mauluru @de_mauluru87 · Jan 22

Replying to @crockar @Docker

That's hard, but not the biggest surprise after changes to licenses. I remove it everywhere I can and replace software if needed. We also reject software companies with software relying on it. Oracle is not trustworthy enough to be sure software works in 2 years or is affordable.
## Base images

Visit [https://hub.docker.com/u/bellsoft](https://hub.docker.com/u/bellsoft)

<table>
<thead>
<tr>
<th>Base Images</th>
<th>JRE 8u222</th>
<th>JDK 13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian</td>
<td>227 MB</td>
<td>227 MB</td>
</tr>
<tr>
<td>Centos</td>
<td>307 MB</td>
<td>307 MB</td>
</tr>
<tr>
<td>Alpine</td>
<td>133 MB</td>
<td>134 MB</td>
</tr>
<tr>
<td>Alpine musl base</td>
<td></td>
<td>39 MB</td>
</tr>
</tbody>
</table>
Base images

- **Java versions**
  - 13
  - 8, 11

- **Linux distribution**
  - Debian
  - CentOS
  - Alpine
  - Alpine musl

- **Arch**
  - x86_64
  - ARM64
  - ARM32
Demo
$ mkdir ctx; cd ctx
$ wget https://github.com/bell-sw/Liberica/blob/master/docker/repos/
liberica-openjdk-alpine-musl/11/Dockerfile
$ docker build . --build-arg LIBERICA_IMAGE_VARIANT=base
What happened?

$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:9000 \
  -m 5m debian /demo/jdk8u121/bin/java \
  -jar /demo/gs-actuator-service-0.1.0.jar
What happened

$ journalctl -f _TRANSPORT=kernel

or

$ docker inspect test -f '{json .State}'

How much memory is enough

- `XX:NativeMemoryTracking=summary`
- `jps`
- `jcmd`
What’s happening?

$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:9000 \\
  -m 128m debian /demo/jdk8u121/bin/java \\
  -jar /demo/gs-actuator-service-0.1.0.jar

$ jmeter.sh -n -t micro.jmx
What’s happening

Someone is careless

- docker stats
- jstat
- smem, pmap

```bash
$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:9000 \
    -m 768m --memory-swappiness 0 debian /demo/jdk8u121/bin/java \
    -jar /demo/gs-actuator-service-0.1.0.jar

~~~~~~
Started HelloWorldApplication in 18.584 seconds (JVM running for 20.425)
```
$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
   -m 768m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.4 java \
   -XX:+UnlockDiagnosticVMOptions -XX:+LogTouchedMethods \
   -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication

$ jdk-11.0.4/bin/jcmd 35647 VM.print_touched_methods \
   | grep -v "35647" | grep -v "#" >methods.log

$ cat methods.log | grep -v SystemModules.hashes | grep -v SystemModules.descriptors \
   | tr -d ':' | awk -F "(" '{gsub(/\//,"."),$1);print $1("$2"}' \
   | awk -F ")" '{gsub(/\//,"."),$2);print "compileOnly "$1")"$2}' >methods.list

$ docker run -it --rm -v /export/dchuyko/demo:/demo -m 768m --memory-swappiness 0 \
   bellsoft/liberica-openjdk-alpine:11.0.4 jaotc \
   --compile-commands /demo/methods.list --jar $(cat cpv) \ 
   --info --ignore-errors --output /demo/thin.so

cpv – classpath in container. Startup is not faster.
AppCDS

$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
   -m 384m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.4 \
   java -XX:DumpLoadedClassList=/demo/hello-ext.classlist \ 
   -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication

$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
   -m 384m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.4 \
   java -Xshare:dump -XX:SharedClassListFile=/demo/hello-ext.classlist \ 
   -XX:SharedArchiveFile=/demo/hello-ext.jsa \ 
   -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication

$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
   -m 256m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.4 \
   java -Xshare:on -XX:SharedArchiveFile=/demo/hello-ext.jsa \ 
   -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication

cpv – classpath in container.
$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
÷6
-m 128m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.4 \\
java -XX:TieredStopAtLevel=1 \\
-Xshare:on -XX:SharedArchiveFile=/demo/hello-ext.jsa \\
-cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication

~~~~~~

Started HelloWorldApplication in 7.304 seconds (JVM running for 8.283) x2.5
Summary

- Java works in containers and knows the limits
- Container diagnostics works for Java
- Java diagnostics works for containers
- All JVM features work in container
  - Use similar environment to generate things in advance
- Use latest releases and updates
  - Security
  - Effectiveness
- Choose base image wisely
- Help your services
  - Prevent failures
  - Limit and decrease footprint
  - Shorten startup