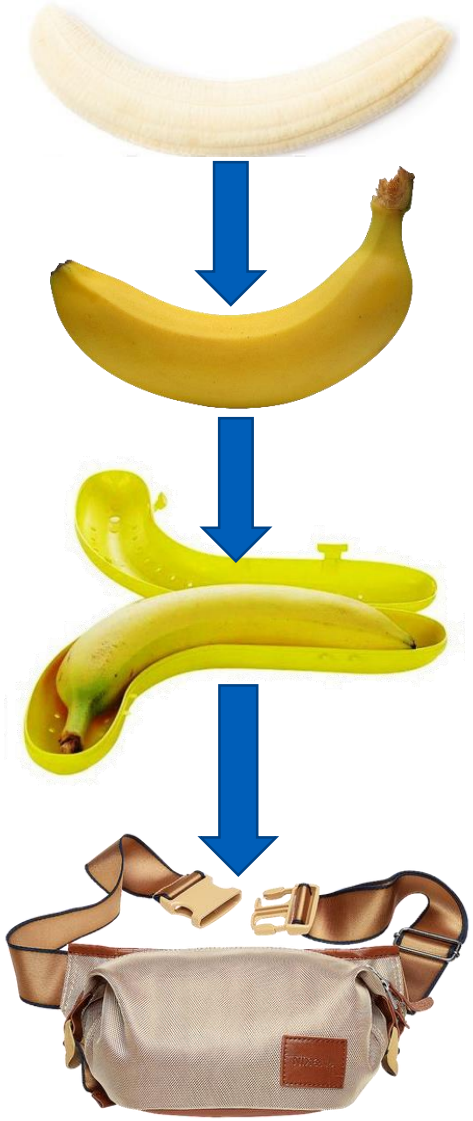
Java containers in production

Mastering the "banana box principle"

Roland Brackmann
Solutions Engineer - Amadeus
18th May 2018



Java Application

JVM – Box 1

Container – Box 2

Linux machine hosting containers – Box 3

2018.rivieradev.fr/session/338
github.com/rbrackma/banana-box/talk.pdf

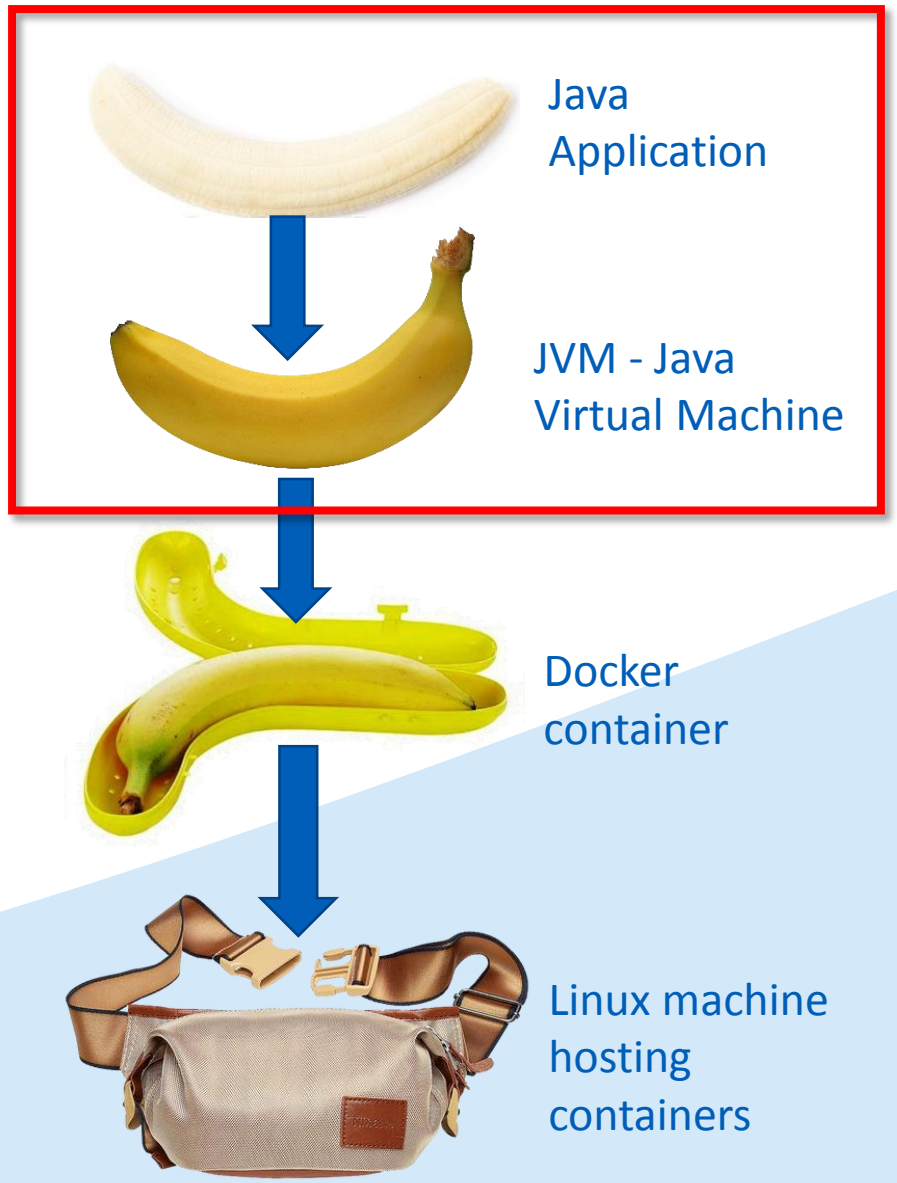
Problem statement

Banana box principle

_How to optimize memory & CPU resources for containerized Java applications?

1.

JVM memory & CPU management



JVM memory management

JVM sizing

```
root@56210c715dd9:/# java -XX:+PrintFlagsFinal -XX:+UnlockExperimentalVMOptions -version | grep ergo
     intx CICompilerCount                          = 3                                         {product} {ergonomic}
   size_t InitialHeapSize                          = 8388608                                   {product} {ergonomic}
   size_t MaxHeapSize                              = 132120576                                 {product} {ergonomic}
   size_t MaxNewSize                               = 44040192                                  {product} {ergonomic}
   size_t MinHeapDeltaBytes                        = 196608                                    {product} {ergonomic}
   size_t NewSize                                  = 2752512                                   {product} {ergonomic}
    uintx NonNMethodCodeHeapSize                   = 5830092                                {pd product} {ergonomic}
    uintx NonProfiledCodeHeapSize                  = 122914074                              {pd product} {ergonomic}
   size_t OldSize                                  = 5636096                                   {product} {ergonomic}
    uintx ProfiledCodeHeapSize                     = 122914074                              {pd product} {ergonomic}
    uintx ReservedCodeCacheSize                    = 251658240                              {pd product} {ergonomic}
     bool SegmentedCodeCache                       = true                                      {product} {ergonomic}
     bool UseCompressedClassPointers               = true                                  {lp64_product} {ergonomic}
     bool UseCompressedOops                        = true                                  {lp64_product} {ergonomic}
     bool UseSerialGC                              = true                                      {product} {ergonomic}
openjdk version "11" 2018-09-25
```

_JVM Ergonomics [1]

- JVM autotuning mechanism based initially or
- Autotune examples : MaxHeapSize (-Xmx), ThreadStackSize (-Xss), CICompilerCount, …

_JVM Flags

LTS JDK	GA	JVM Flags
8	March 2014	722 – standard
		103 – diagnostic
		41 – experimental
11 [2]	~ Sept 2018	655 – standard
		154 – diagnostic
		60 – experimental

[1] https://docs.oracle.com/javase/8/docs/technotes/guides/vm/gctuning/ergonomics.html

[2] not final yet for OpenJDK11

JVM memory management

Memory used during banana app lifecycle

_ Use case

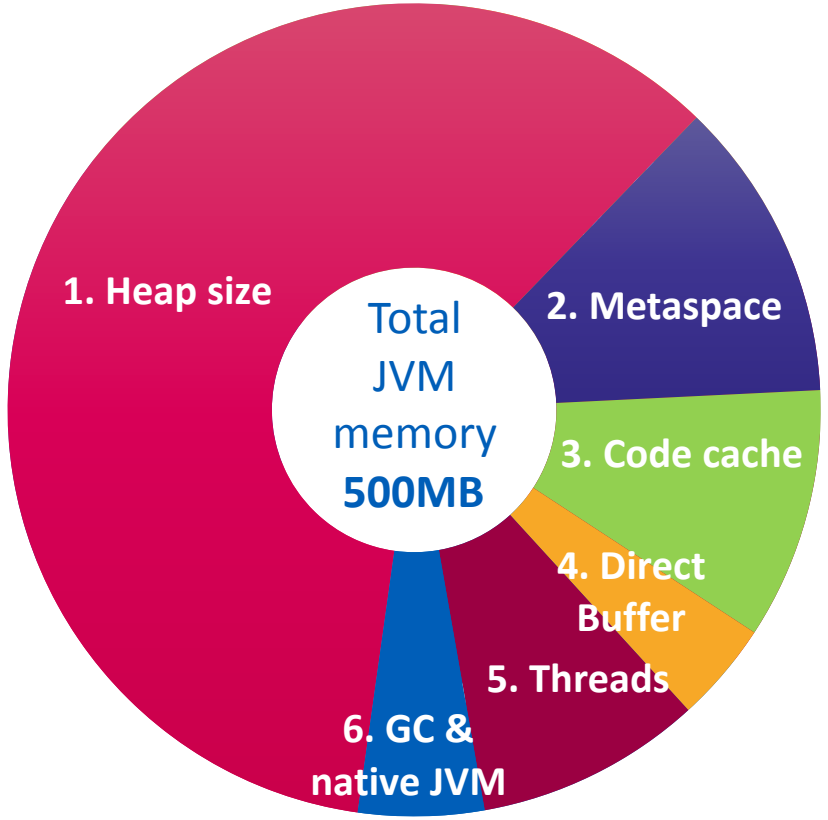
- Our banana application is started on a JVM and receives traffic, where does the memory go ?

_ JVM Memory usage

- Loading BananaServer class : 2
- Client calls our thread/request endpoint : 5
- Program logic calls new BananaServer() : 1
- BananaServer.slip() (after n^{th} run, JIT optimized) : 3
- BananaServer bypasses Heap with DirectByteBuffer : 4
- JVM garbage collector kicks in : 6

_ What is the memory distribution for e.g. 20 TPS?

- For 1, 2 & 3 & 4 use gc.logs / JMXViewer [1]
- For 5 & 6 use tool “jcmd <pid> VM.native_memory” [2]



[1] http://gceasy.io / JVisualVM – JConsole - JMC

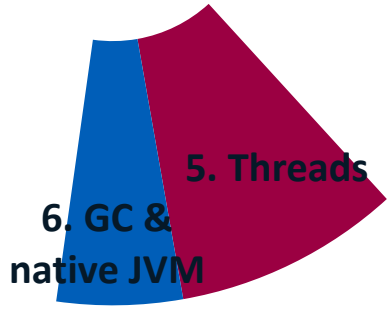
[2] https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/tooldescr007.html & link

JVM CPU management

Threads created by banana JVM

_Use case

- Our banana application is started on a JVM and receives traffic, what impact has the CPU (cores) on the thread creation ?



_Application threads based on # of cores : 5

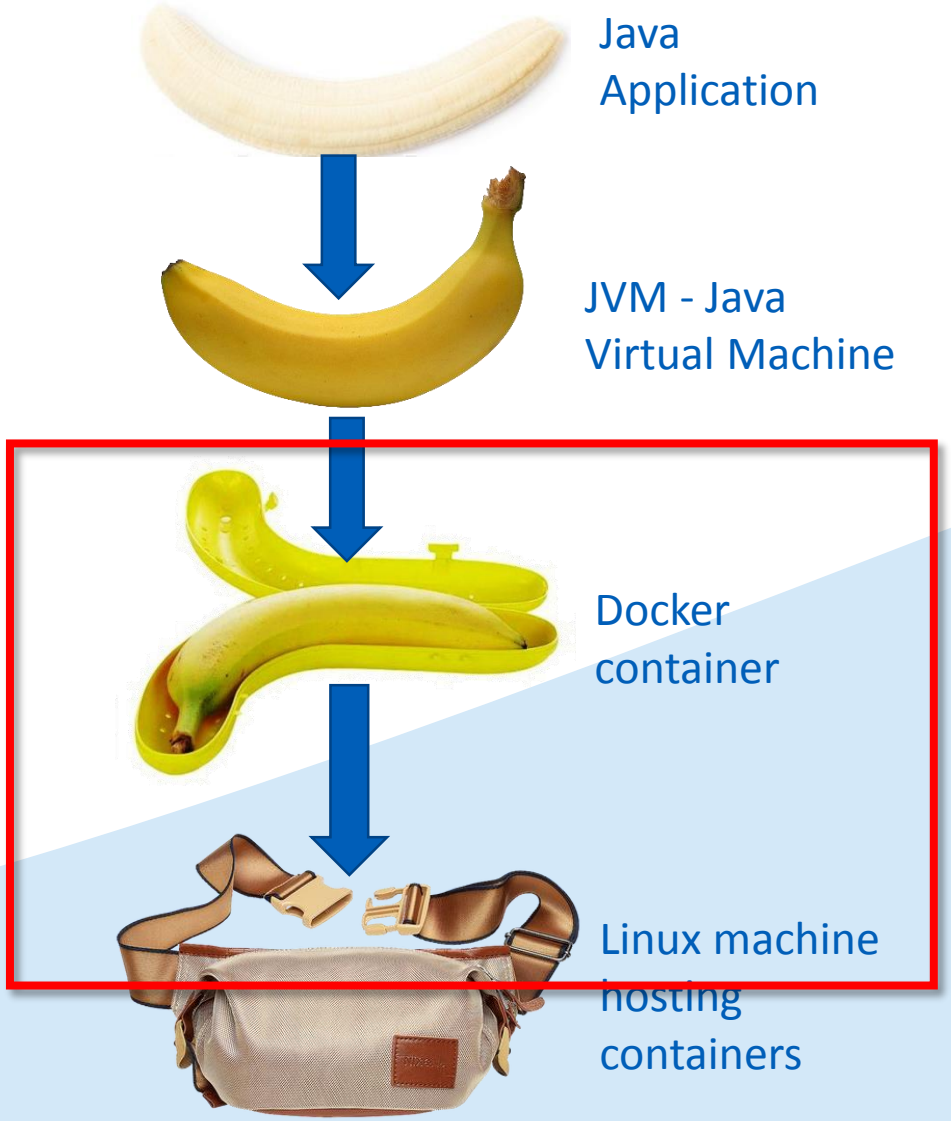
- App server : e.g. JBoss EAP 6 – logging- , EJB-timer- and HTTP thread pool
- 3rd party lib used : e.g. Oracle Coherence
- In-house libraries : using Runtime.getRuntime().availableProcessors() & api.java.util.concurrent.ForkJoinPool

_GC threads : 6

- JVM uses core information to determine # of Compiler & GC threads

2.

Container memory & CPU management



Container internals

Linux namespaces

_Definition “Namespaces”

- Linux kernel feature that makes a global resource appear local
- Namespace types : Mount, PID, User, …

_Activating namespaces for containers

- Out of the box
- Most Linux CLI commands respect namespaces

Container internals

Linux control groups “cgroups”

_Definition “cgroups”

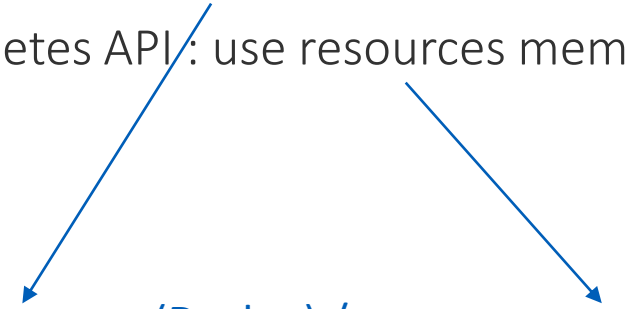
- Linux kernel feature that is a resource-limiter for processes
- cgroups settings in /sys/fs/cgroup/ (on host machine and in container)

```
cat /sys/fs/cgroup/.../memory.limit_in_bytes
524288000
cat /sys/fs/cgroup/.../cpu.shares
2048
```

_Activating cgroups for memory

- Docker API : `--memory=500M` (`--memory-reservation=500M`)
- Kubernetes API : use resources memory limits

```
# Replication controller
name: jvm-docker
resources:
  requests:
    memory: 500M
  limits:
    memory: 500M
```



Values set by --memory (Docker) / memory resource limits (Kubernetes) are picked up in JDK11 ! Not in JDK8 !!!

Container internals

Linux control groups “cgroups”

_Activating cgroups for CPU

- Docker API : `--cpu-shares=2048` `(--cpu-quota=1000; --cpu-set=1,2)`
- Kubernetes API : use resource limits

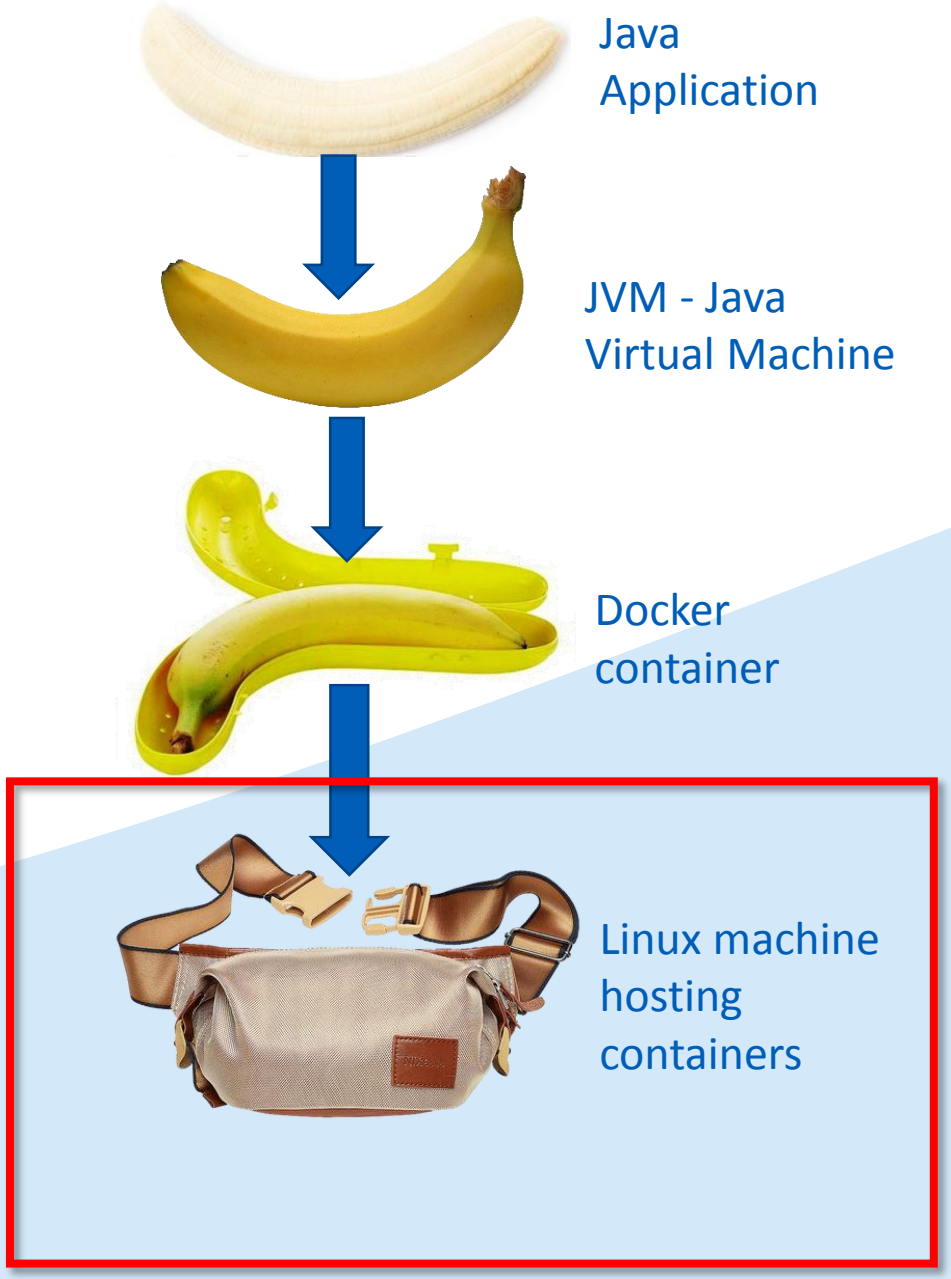
```
# Replication controller
name: jvm-docker
resources:
  requests:
    cpu: 1024m
  limits:
    cpu: 2048m
```

Values set by –cpu-shares (Docker) / memory resource limits (Kubernetes)
are picked up in JDK11 via Runtime.getRuntime().availableProcessors() ! Not in JDK8 !!!

cgroups limit not enforced

3.

Host resource management



Kubernetes scheduling decisions

when host memory gets low

_3 QoS classes

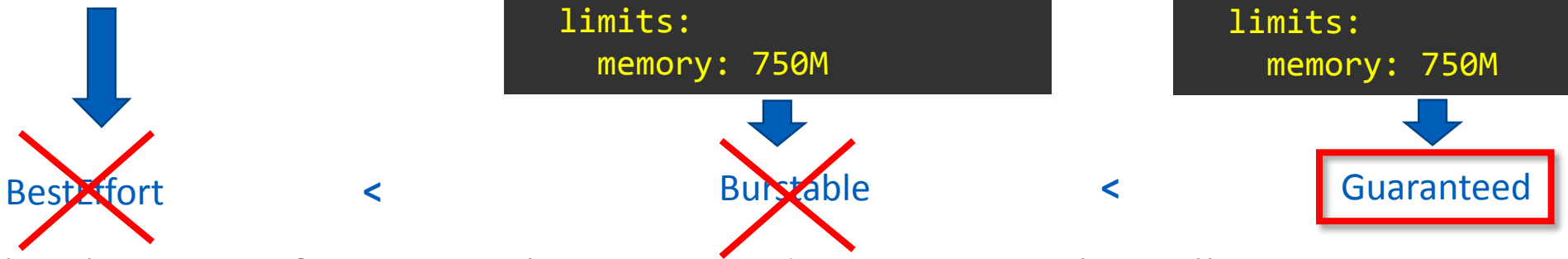
- A QoS class maps to a Linux OOM score
- OOM score used to make decisions about scheduling and evicting pods

_Q1 : Which of the 3 containers/pods are scheduled if 1GB is left on a host ?

```
# Replication controller
name: apple-application-A
image : A
resources:
```

```
# Replication controller
name: apple-application-B
image : B
resources:
  requests:
    memory: 200M
  limits:
    memory: 750M
```

```
# Replication controller
name: banana-docker-jvm-C
image : C
resources:
  requests:
    memory: 750M
  limits:
    memory: 750M
```



_Q2 : What happens if every application needs 600MB ? Who will survive ?

Kubernetes node configuration

Memory options for nodes running containers

_Disable swapping

- Docker API : `--memory-swappiness=-1`
  `-1 = use host settings (default);  100=most likely to swap;  0=no swapping`
- Kubernetes API : swapping option NOT supported → inherited from non-swapping host machine [1]

_Allocate maximum memory for containers

- Allows to reserve memory for Kubernetes/OS processes

_OOM killer setup

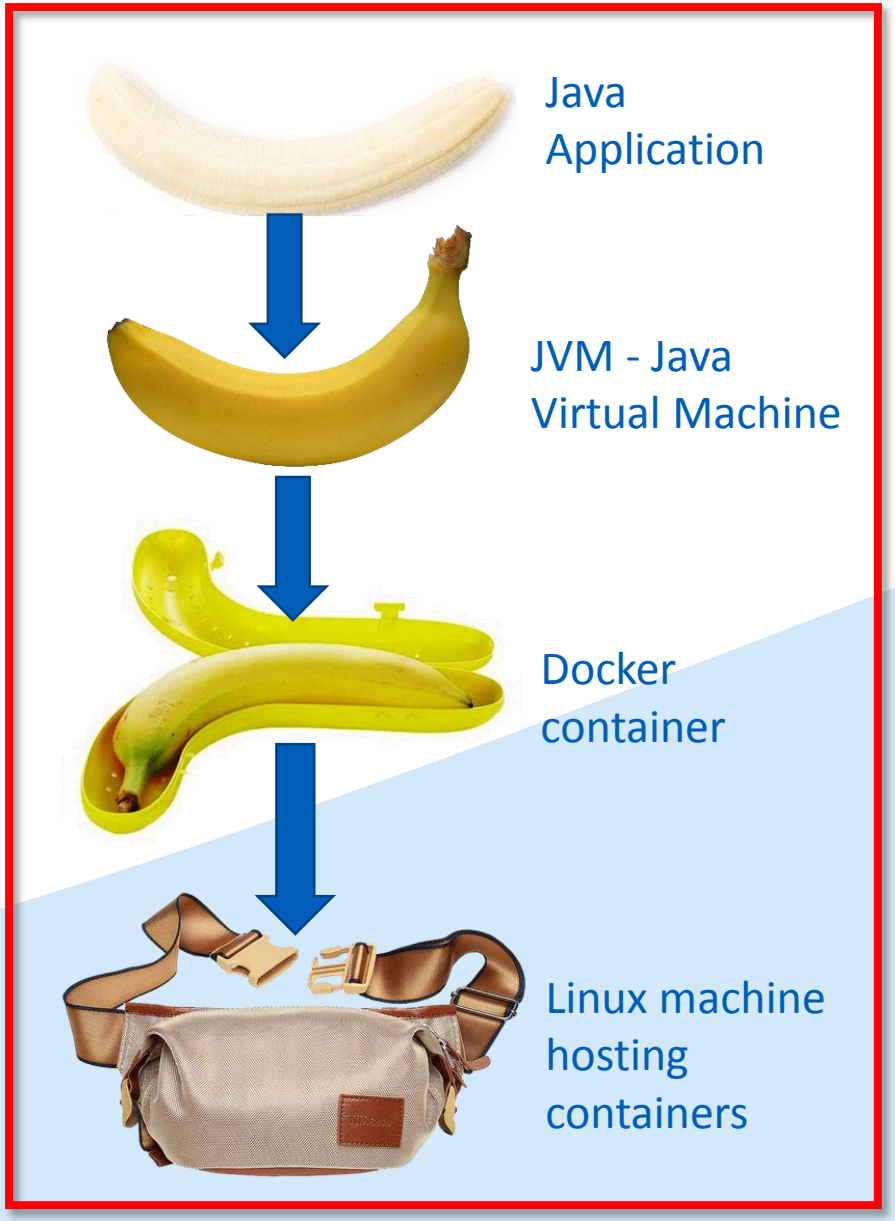
- Disable kernel panic - Kernel OOM killer will kill on priority given by QoS class. [2]

[1] On host check “cat /proc/sys/vm/swappiness“ = 0

[2] On host check “cat /proc/sys/vm/panic_on_oom” = 0

4.

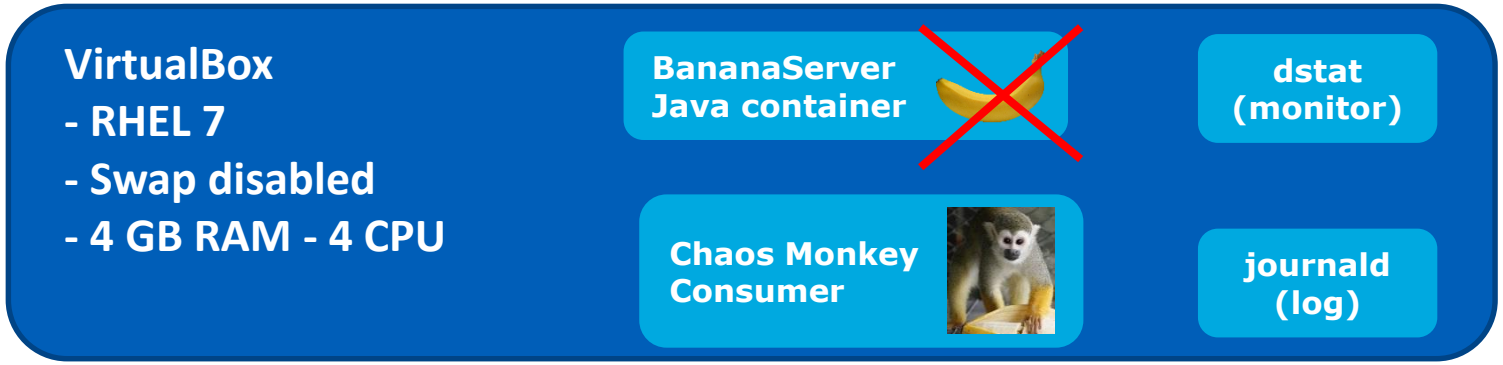
Running Java containers locally (demo)



Live demo – OOM - hitting max host memory

Docker with no memory limits

No JVM flag set (using OpenJDK 8 container)



_Error

- Kernel killed the Banana container process after hitting **host memory limit** of 4GB

```
journalctl -n 1000
dockerd-current: Max. Heap Size (Estimated): 880.00M
kernel: Out of memory: Kill process 50237 (java) score 1070 or sacrifice child
```

_Issue

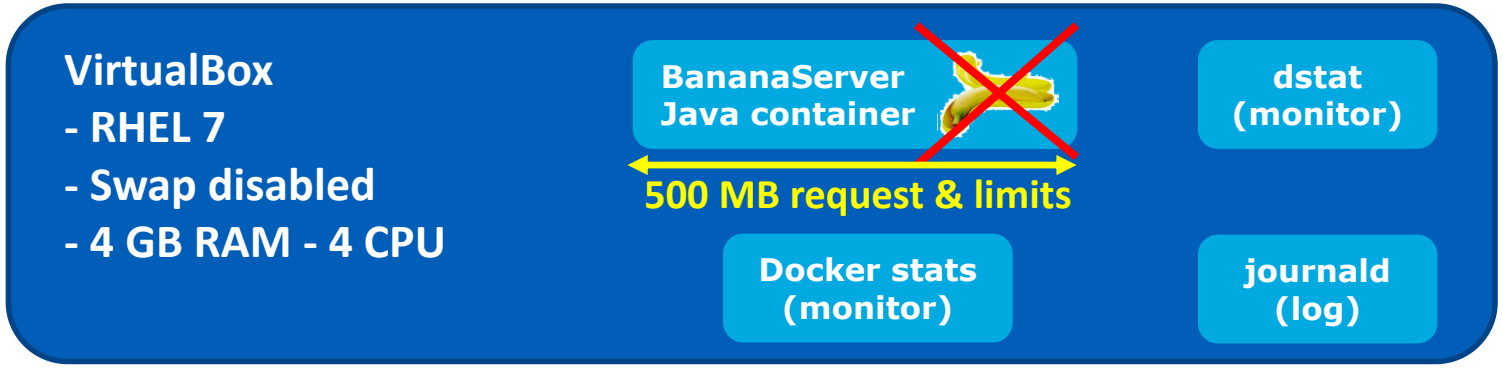
- We can not investigate
- Banana container was only running as “Best Effort” container (monkey won with better OOM score)

github.com/rbrackma/banana-box

Live demo – OOM - hitting cgroup max memory

Running as “Guaranteed” container (docker with memory & CPU limits)

No JVM flag set (using OpenJDK 8 container)



_Error

- Kernel killed the Banana container process after hitting **cgroup max memory** of 500MB

```
journalctl -n 1000
dockerd-current: Max. Heap Size (Estimated): 880.00M
kernel: Memory cgroup out of memory: Kill process 3593 (java) score 1009 or sacrifice child
```

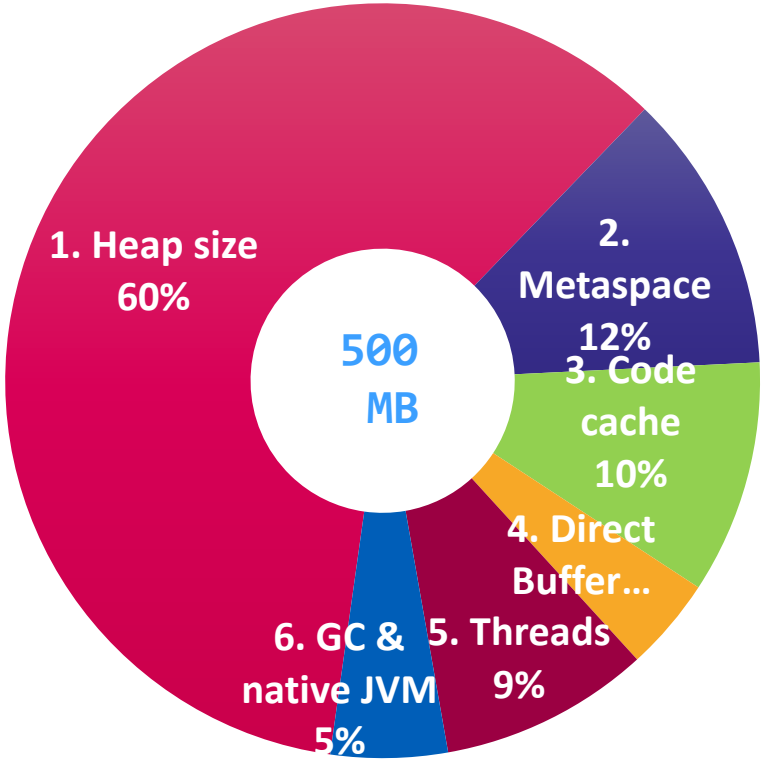
_Issue

- We can still not investigate
- JVM Ergonomics of JDK8 are not cgroups aware out-of-the-box

github.com/rbrackma/banana-box

Reviewing our JVM segments

Try to limit the JVM from growing over the container limits




```
docker run -d docker.io/openjdk:8-jdk \
--memory=500MB \
java \
  -Xms300m -Xmx300m \                  # 1. ergonomics
  -XX:MaxMetaspaceSize=60m \           # 2. unlimited
  -XX:ReservedCodeCacheSize=50m \      # 3. 256MB
  -XX:MaxDirectMemorySize=2MB \        # 4. 64MB
  -XX:ThreadStackSize=228 \            # 5. 1024KB
-v ${LOCAL_JAR_REPO}:/deployments \
-jar /deployments/BananaServer.jar
```

Can we avoid hard coding Heap size ?
Make the configuration dynamic ? 60% ?

→ **java.lang.OutOfMemoryError: Java heap space**

→ **java.lang.OutOfMemoryError: Metaspace**

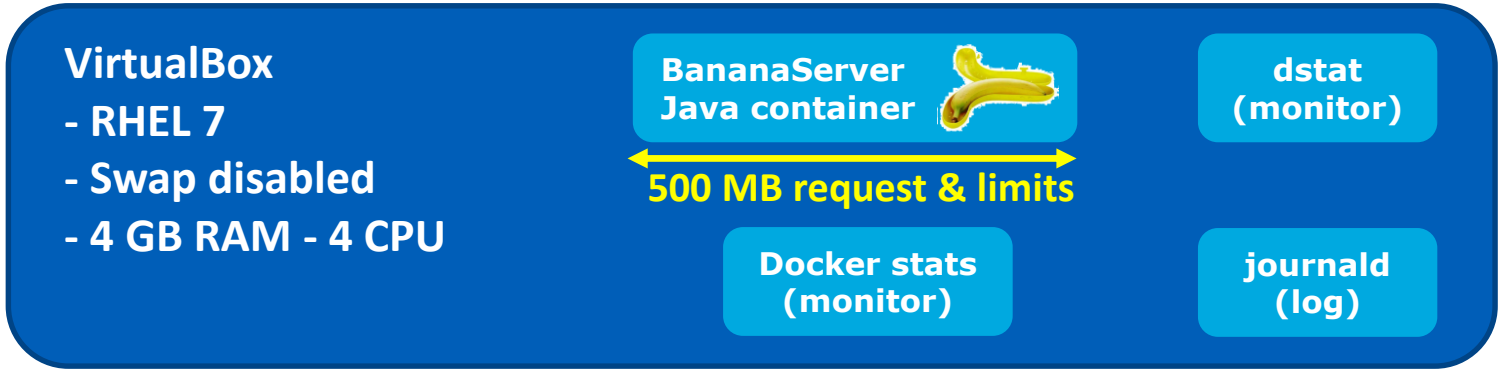
→ **No error - flushed**

→ **java.lang.OutOfMemoryError: Direct buffer memory**

→ **Exception in thread "main" java.lang.StackOverflowError**

Live demo – OutOfMemoryError in JDK8

Running as “Guaranteed” container (docker with memory & CPU limits)
JVM flags set by magic start script [1] (using Red Hat OpenJDK 8 container)



_Expected error

```
journalctl -n 1000
dockerd-current:     Max. Heap Size: 300.00M       <-- sized according to container limit
dockerd-current: Terminating due to java.lang.OutOfMemoryError: Java heap space
```

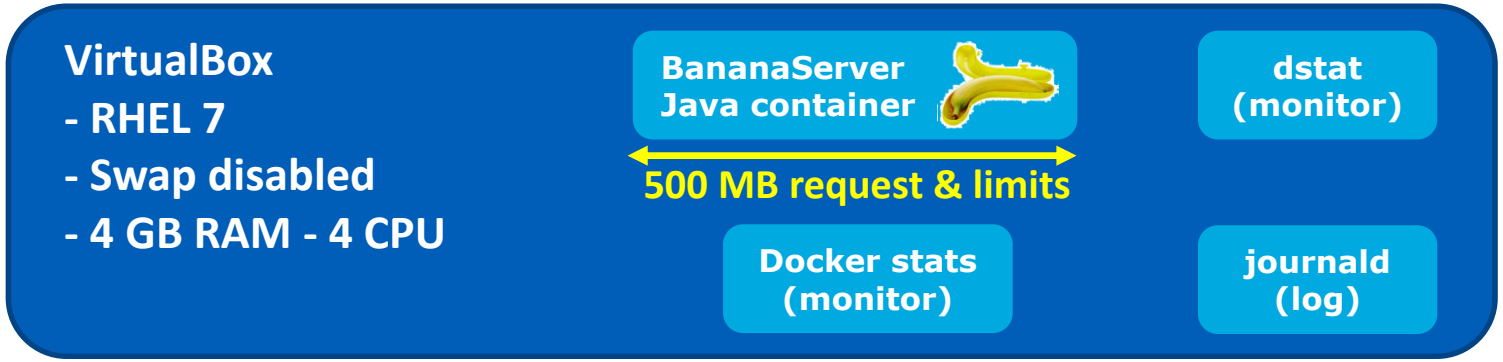
_Analysis

- We can analyse the Java OOME with heapdumps!
- With the info from the JVM segment section, we can use JAVA_OPTIONS & JAVA_MAX_MEM_RATIO
  $USER_{RATIO} * cgroups_{memlimit} = MaxHeapSize$ (0.60 *500MB = 300MB) [1]

[1] github.com/jboss-openshift/cct_module/blob/master/os-java-run/added/container-limits

Live demo – OutOfMemoryError in JDK11

Running as “Guaranteed” container (docker with memory & CPU limits)
JDK 11 cgroup aware for memory & CPU (using OpenJDK 11 container)



_Expected error

```
journalctl -n 1000
dockerd-current[881]:     Max. Heap Size (Estimated): 290.00M
dockerd-current: Terminating due to java.lang.OutOfMemoryError: Java heap space
```

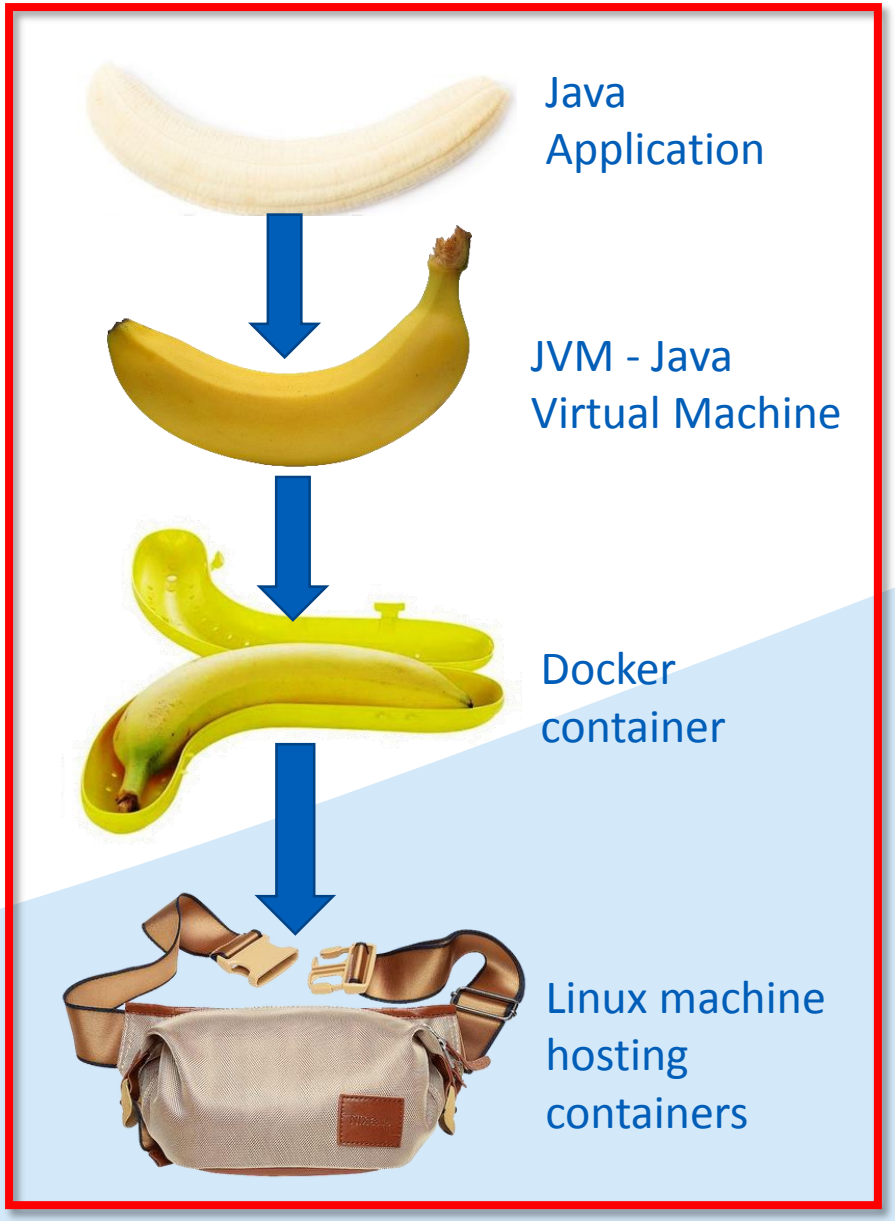
_Analysis

- We can analyse the Java OOME with heapdumps!
- With the info from the segment section, we can pass in “-XX:MaxRAMPercentage=60“ & Runtime.getRuntime().availableProcessors() returns cgroups limit

github.com/rbrackma/banana-box

5.

Running Java containers in production (OpenShift)



Production feedback

Our Banana box container

_Monolith EAR application on JBoss EAP 6

_Red Hat container

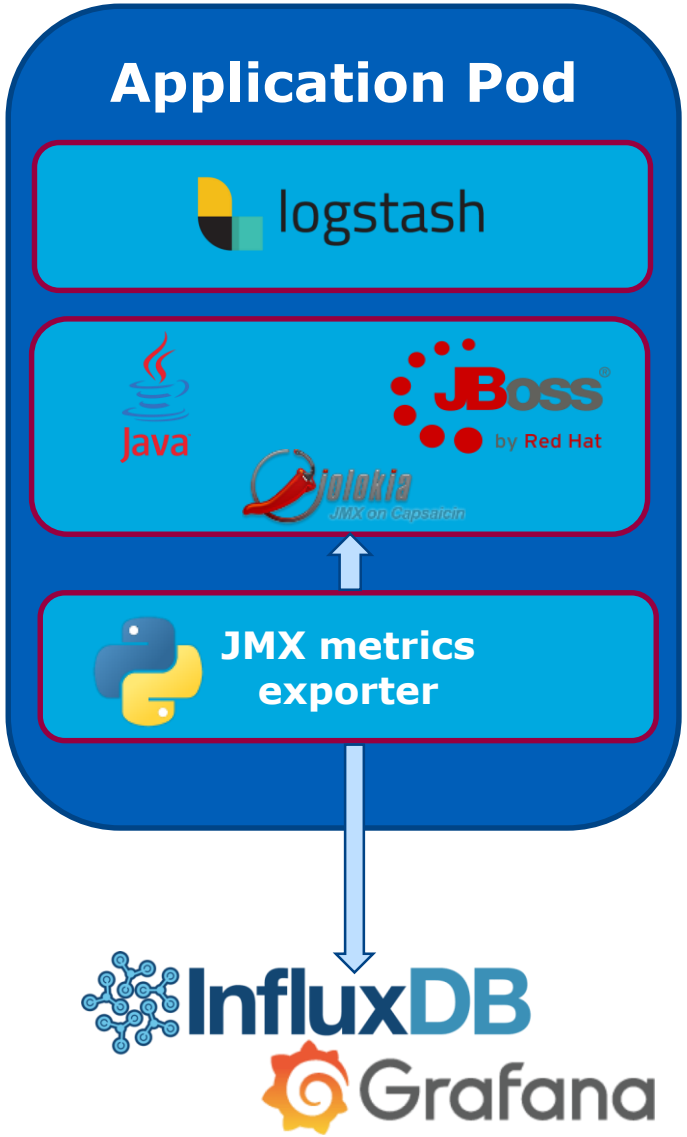
- OpenJDK 1.8 – garbage collector G1
- JAVA_MAX_MEM_RATIO=50 with a Xmx of 3,5 GB

_Pod setup

- Log exporter container
- JBoss container exposes Jolokia JMX interface
- JMX exporter container pushes to InfluxDB/Grafana

_OpenShift setup

- Custom Amadeus deployment unit loaded via Pipelines
- 40 pods deployed

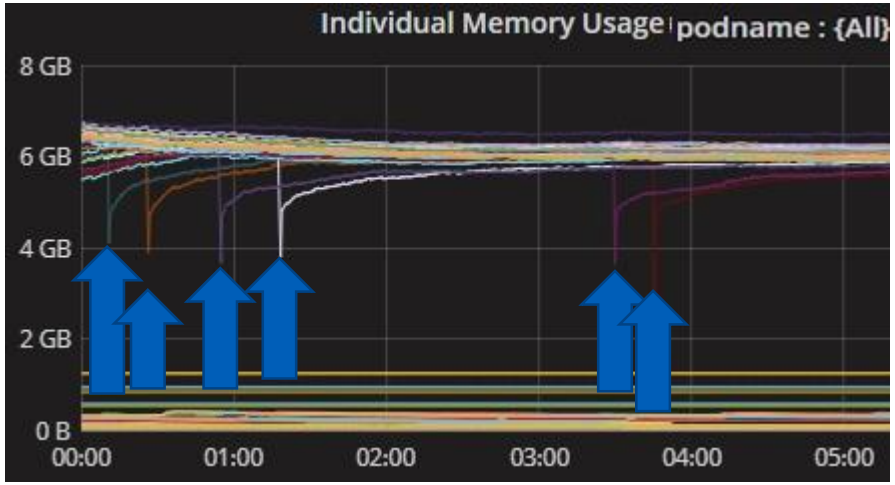


Production feedback

V1 - JDK Bug

_Go-live with v1 !

- JVM started core-dumping!



- Bug identified in garbage collector G1 !
  Fixed in java-1.8.0-openjdk-devel-1.8.0.151-**5**.b12

```
oc logs --previous=true pod1 -c jvm-docker
# A fatal error has been detected by the JRE:
#
#  SIGSEGV (0xb) at pc=0x00007f552121a673
#
# JRE version: OpenJDK Runtime Environment
(8.0_131-b12) (build 1.8.0_131-b12)
# Java VM: OpenJDK 64-Bit Server VM (25.131-
b12 mixed mode linux-amd64 compressed oops)
# Problematic frame:
# V  [libjvm.so+0x584673]
#
# Core dump written. Default location:
/home/jboss/core or core.364
```

_ Let's fix our banana box

- Fixed by yum update, NRE tested, Stress-tested, manually canary-tested for a week in production …
- Now let's roll out v2!

Production feedback

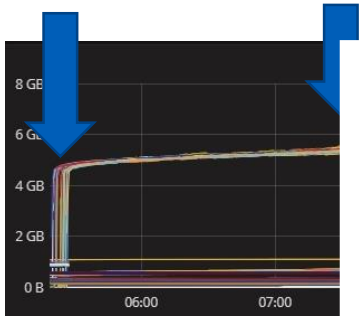
V2 - JVM OOME limit

_We loaded v2 at 5h20 (fixing the core dump bug)

- 2h later JVM container kepts restarting … every hour!

```
oc logs --previous=true pod1 -c jvm-docker
java.lang.OutOfMemoryError: Metaspace
Dumping heap to /tmp/rc-22-x4xlc.hprof ...
Heap dump file created [1055484480 bytes in 5.243 secs]
```

Load v2



Load v3

We used “-XX:+PrintFlagsFinal” …

JVM parameters of v1 (MetaSpace unlimited)

```
11 FILTERED LINES
1723    uintx MaxMetaspaceSize                  = 18446744073709547520
11 FILTERED LINES
1735    uintx MinHeapFreeRatio                  = 40
418 FILTERED LINES
2154     bool UseParallelGC                     = false
66 FILTERED LINES
```

JVM parameters of v2 (MetaSpace limited to 256MB)

```
11 FILTERED LINES
2339    uintx MaxMetaspaceSize                  := 268435456
11 FILTERED LINES
2351    uintx MinHeapFreeRatio                  := 20
418 FILTERED LINES
2770     bool UseParallelGC                     := false
66 FILTERED LINES
```

_Problem : Human error

- Updated Red Hat’s minor version of the JBoss EAP 6 base container env variable GC_MAX_METASPACE_SIZE

Production feedback

General

_What helped us to stay HA ?

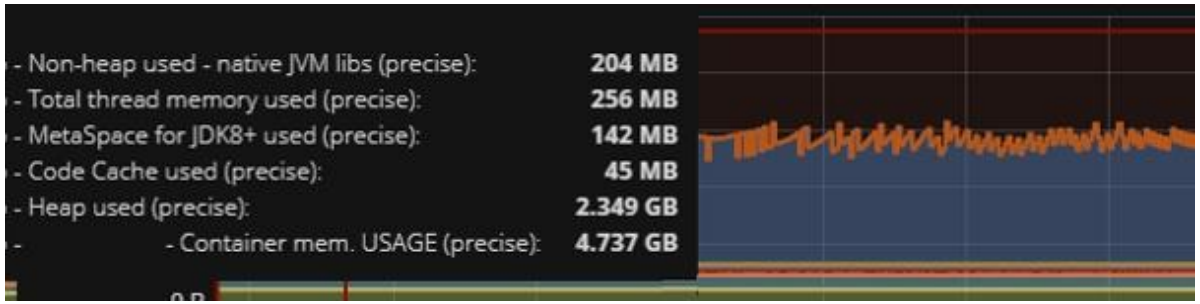
- For session management we offload to Redis cluster [1]

_What helps us to stay PCI/DSS compliant ?

- We use automated container health checks using Red Hat's website[2]

_How do we avoid JVM memory segment problems now ?

- Banana Box Grafana dashboard



_Next steps

- For core- & heapdumps we will use the CoreInterceptor [3]

[1] github.com/AmadeusITGroup/HttpSessionReplacer
[2] https://access.redhat.com/containers/#/registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift
[3] github.com/AmadeusITGroup/ContainerCoreInterceptor

Key takeaways

_Use a Heap ratio mechanism

_LTS JDK8 vs JDK11 (~Sept 2018)

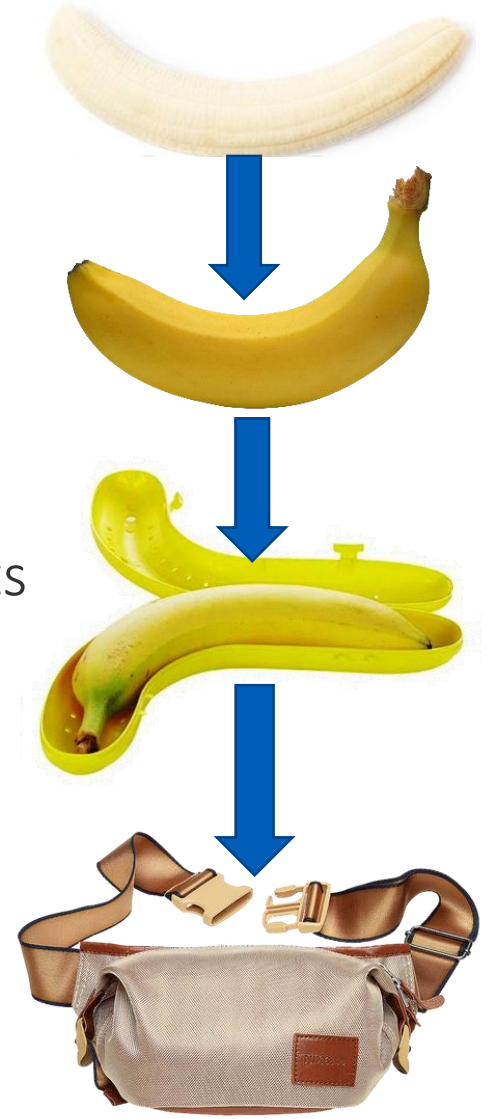
_Limit the # of threads

_Set your container memory & CPU limits

_Disable swapping on hosts

_Don’t forget the banana box

- 2018.rivieradev.fr/session/338
- github.com/rbrackma/banana-box/talk.pdf



Question and answers & next steps

Interested in the topic ?

_Want to watch related videos ?

- Devoxx 2016 talk that this one got a lot of inspiration from https://youtu.be/6ePUiQuaUos
- Red Hat summit talk 2018 “Why you’re going to FAIL running Java on docker”: https://www.youtube.com/watch?v=UrAE0hD1_pM
- DevoxxFR 2018 (French) https://www.youtube.com/watch?v=vzpU2jxrxJ8
- VJUG 2018 https://www.youtube.com/watch?v=2TwjNcrfjKM

_Want to read articles on the subject ?

- https://mjg123.github.io/2018/01/10/Java-in-containers-jdk10.html
- https://developers.redhat.com/blog/2017/03/14/java-inside-docker/
- https://developers.redhat.com/blog/2017/04/04/openjdk-and-containers/
- https://shipilev.net/jvm-anatomy-park/12-native-memory-tracking/

Extra slide - Memory/CPU flags to use for a Java 11 container

JVM flags for LTS OpenJDK 11*1

Type	Banana Box JVM Flags	Description
INFO	-XshowSettings:vm	Displays Heap size & Thread size
INFO	-XX:+PrintFlagsFinal	Prints all ~1000 JVM flags
MEM	-XX:ThreadStackSize=228	Application max thread size in KB (aka -Xss)
MEM	-XX:MaxRAMPercentage=80 [2]	
CPU	-XX:ActiveProcessorCount=count [2]	This count overrides any other automatic CPU detection logic in the JVM.
CPU	-XX:+UseContainerSupport [2] (default=true) -XX:+PreferContainerQuotaForCPUCount*1 (default=true)	Nothing to do

[1] not final yet for JDK11

[2] introduced for JDK10

Find out more JVM options : https://chriswhocodes.com/hotspot_options_jdk11.html

Extra slide - Memory/CPU flags to use for a Java 8 contair

JVM flags for LTS OpenJDK 8

```
MaxRamFraction
    Avail - MaxHeap
1 : 400MB - 386.69M
2 : 400MB - 193.38M
3 : 400MB - 129.56M
4 : 400MB - 121.81M
```

Type	Banana Box JVM Flags	Description
INFO	-XshowSettings:vm	Displays Heap size & Thread size
INFO	-XX:+PrintFlagsFinal	Prints all ~1000 JVM flags
MEM	-XX:ThreadStackSize=228	Application max thread size in KB (aka -Xss)
MEM	~~-XX:+UnlockExperimentalVMOptions~~ ~~-XX:+UseCGroupMemoryLimitForHeap~~ ~~-XX:MaxRAMFraction=2~~	Only use if you are forced to, as MaxRAMFraction flag is too unprecise (deprecated in JDK10+) → **use custom start script instead with % HEAP**
CPU	-XX:CICompilerCount=2	Force # compiler threads (to avoid ergonomics) → **use custom start script** (-XX:CICompilerCount=2)
CPU	-XX:ParallelGCThreads=2	Force # GC threads (to avoid ergonomics) → **use custom start script**
CPU	“Runtime.getRuntime().availableProcessors()”-ISSUE	Every application using this value might spawn too many threads!

Find out more JVM options : https://chriswhocodes.com/hotspot_options_jdk11.html

Extra slide - Chosing your base docker image

Maintained JDK containers

	Vendor	Docker pull	URL	Dockerfile	Properly versioned / maintained?	Added value ? Magic start Script
1	OpenJDK community	docker pull docker.io/openjdk:**8u171-jdk**	store.docker.com	github.com		
2	Fabric8 community	docker pull docker.io/fabric8/java-alpine-openjdk8-jdk:**1.4.0**	hub.docker.com	github.com		
3	IBM – J9	docker pull docker.io/ibmjava:8-sdk	store.docker.com	github.com		
4	Oracle - OpenJDK	docker pull docker.io/oracle/openjdk:8	hub.docker.com	github.com		
5	Oracle - Hotspot	docker pull docker.io/store/oracle/serverjre:8 (registration needed)	store.docker.com	---		
6	Red Hat - OpenJDK	docker pull registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift:**1.3-7**	access.redhat.com	github.com		
7	SAP - Machine	docker pull docker.io/sapmachine/jdk11:**11.0.0.12.0** (JDK8 not available)	hub.docker.com	github.com		

Extra slide - How to decide which docker image to use ?

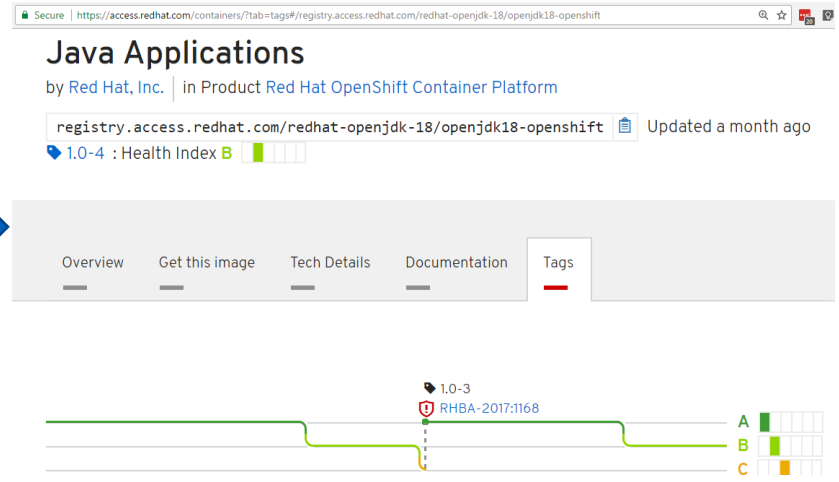
Questions I could ask myself

_Do I want to maintain the JDK updates myself – PCI/DSS ?

_Are the base containers versioned ?

_Are the containers updated regularly ?

_Does the container registry provide a health check ? [1]



_What is the added value?

- e.g. images include JMX agent, “cgroup-aware start scripts”
- you know which RPM’s are installed

[1] https://access.redhat.com/containers/#/registry.access.redhat.com/redhat-openjdk-18/openjdk18-openshift

Extra slide - Running Java containers

Useful debugging tools

Low level tool	Tool family	Respecting Namespace	Cgroups aware
jcmd *1	JVM swiss army knife		
docker stats / systemd-cgtop	Container monitor		
dstat*2	Host monitor		

*1 binary of OpenJDK

*2 not installed by default on RHEL. Hint: use “sudo yum install dstat -y“ then “dstat -m 3 50”

Extra slide - Hint for writing your own script for JDK8

Make sure it get's killed

_Making sure the start script is verbose

```
set -x              # prints every command executed to STDOUT
export SHELLOPTS    # exports first option to all subshells
```

_Print your

```
env    # prints all Linux variables passed into the container
```

_Print your config files to STDOUT

- Hint : the JVM option position counts, if they are in double → first one wins!

_Make sure your script get's killed by SIGTERM

- On uprade Openshift will not use SIGKILL to stop a container, but SIGTERM

```
Use "exec" in your start script
# or
```

Extra slide - JVM segment view

HotSpot JVM options cheatsheet

Java 8

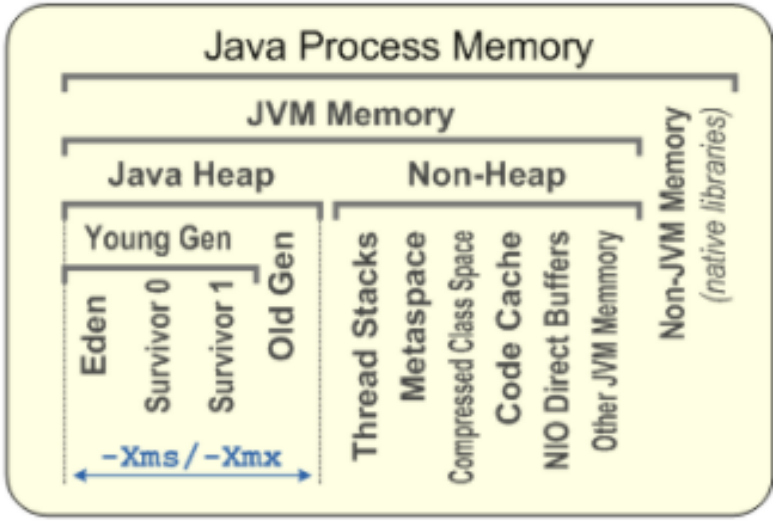
All concrete numbers in JVM options in this card are for illustrational purposes only!

Alexey Ragozin – http://blog.ragozin.info

Available combinations of garbage collection algorithms in HotSpot JVM

Young collector	Old collectior	JVM Flags
Serial (DefNew)	Serial Mark Sweep Compact	-XX:+UseSerialGC
Parallel scavenge (PSYoungGen)	Serial Mark Sweep Compact (PSOldGen)	-XX:+UseParallelGC
Parallel scavenge (PSYoungGen)	Parallel Mark Sweep Compact (ParOldGen)	-XX:+UseParallelOldGC
Parallel (ParNew)	Serial Mark Sweep Compact	-XX:+UseParNewGC
Serial (DefNew)	Concurrent Mark Sweep	-XX:-UseParNewGC[1] -XX:+UseConcMarkSweepGC
Parallel (ParNew)	Concurrent Mark Sweep	-XX:+UseParNewGC -XX:+UseConcMarkSweepGC
Garbage First (G1)		-XX:+UseG1GC

1 - Notice minus before UseParNewGC, which is explicitly disables parallel mode



_Correction : Compressed Class Space is part of Metaspace !!!

https://3.bp.blogspot.com/-6P98YNSxuF4/WA7KEsctHvI/AAAAAAAANrI/Guxu7Th7k0sbz5gg5eIpVssajaACjYRmQCPcB/s1600/Java%2B8%2B-%2BGC%2Bcheatsheet.png