BELLSOFT

Не клади все яйца в один контейнер

Дмитрий Чуйко
JVM Team

Who we are

Dmitry Chuyko


@dchuyko

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

Ex-employers

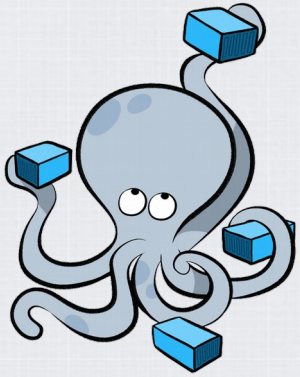
Микросервисы в контейнерах в 2019

Контейнеры

- Контейнеризация в Linux
  - cgroups
  - namespaces
  - Изоляция
  - Управление ресурсами
  - Не виртуализация

- Образы Docker
  - Конфигурация

- Инструменты Docker
  - Управление
  - Мониторинг
  - Оркестрация



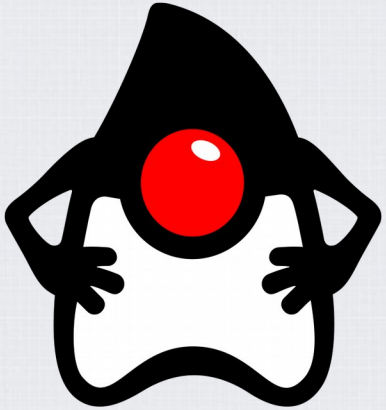

JAVA В КОНТЕЙНЕРЕ
ПРОСТО ЛЕЖАЛА
risovach.ru

JVM

- Виртуальная машина
  - Процесс ОС
  - Runtime
  - JIT/code
  - GC
- Ожидания от контейнеров
  - Конфигурация
  - Test ≃ Prod
  - Изоляция
- Нужны инструменты Java
  - Управление
  - Мониторинг
  - Отладка

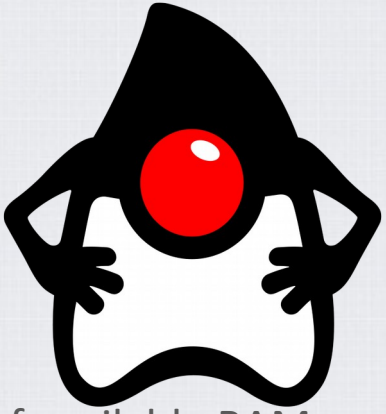
JDK 9

- 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 10

- 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

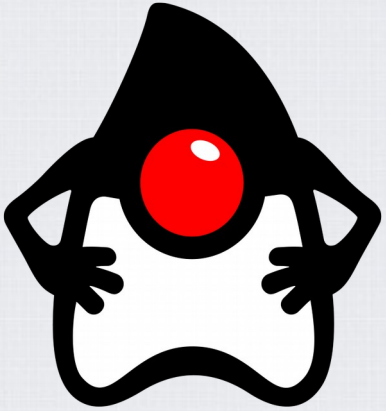


- JDK-8186248 Allow more flexibility in selecting Heap % of available RAM
  - -XX:InitialRAMPercentage
  - -XX:MaxRAMPercentage
  - -XX:MinRAMPercentage

- JDK-8179498 attach in Linux should be relative to /proc/pid/root and namespace aware

JDK 11

- 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

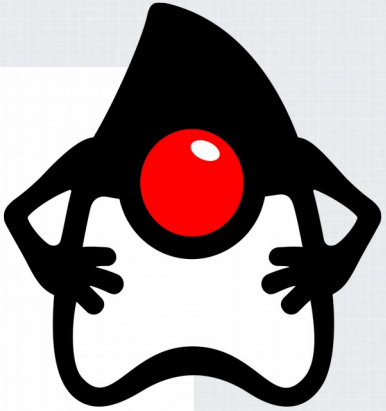
- JDK 8 GA. Один я Д’Артаньян!

- JDK 8u
  - Backports из JDK 9
  - Backports из JDK 10
  - Backports из JDK 11

¯_(ツ)_/¯

JDK 12

- JEP 346: Promptly Return Unused Committed Memory from G1
  - Поможет в случае overcommit



JEP 346: Promptly Return Unused Committed Memory from G1

Authors	Rodrigo Bruno, Thomas Schatzl, Ruslan Synytsky
Owner	Thomas Schatzl
Type	Feature
Scope	Implementation
Status	Closed / Delivered
Release	12
Component	hotspot / gc
Discussion	hotspot dash gc dash dev at openjdk dot java dot net
Effort	M
Duration	S
Reviewed by	Mikael Vidstedt, Stefan Johansson
Endorsed by	Vladimir Kozlov
Created	2018/05/30 14:23
Updated	2019/01/23 14:02
Issue	8204089

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 uncommit of 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.

Motivation

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 tries 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. Even 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 reduce its heap usage during that time, both would benefit.

Shenandoah and OpenJ9's GenCon collector already provide similar functionality.

Tests with a prototype in Bruno et al., section 5.5, shows that based on the real-world utilization of a Tomcat server that serves HTTP requests during the day, and is mostly idle during the night, this solution can reduce the amount of memory committed by the Java VM by 85%.

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 user control, a full GC can be performed to maximize the amount of memory returned.

The application is considered inactive, and G1 triggers a periodic garbage collection if both:

- More than `G1PeriodicGCInterval` milliseconds have passed since any previous garbage collection pause and there is no concurrent cycle in progress at this point. A value of zero indicates that periodic garbage collections to promptly reclaim memory are disabled.
- The average one-minute system load value as returned by the `getloadavg()` call on the JVM host system (e.g. container) is below `G1PeriodicGCSystemLoadThreshold`. This condition is ignored if `G1PeriodicGCSystemLoadThreshold` is zero.

If either of these conditions is not met, the current prospective periodic garbage collection is cancelled. A periodic garbage collection is reconsidered the next time `G1PeriodicGCInterval` time passes.

The type of periodic garbage collection is determined by the value of the `G1PeriodicGCInvokesConcurrent` option: if set, G1 continues or starts a concurrent cycle, otherwise G1 performs a full GC. At the end of either collection, G1 adjusts the current Java heap size, potentially returning memory to the operation system. The new Java heap size is determined by the existing configuration for adjusting the Java heap size, including but not limited to the `MinHeapFreeRatio`, the `MaxHeapFreeRatio`, and minimum and maximum heap size configuration.

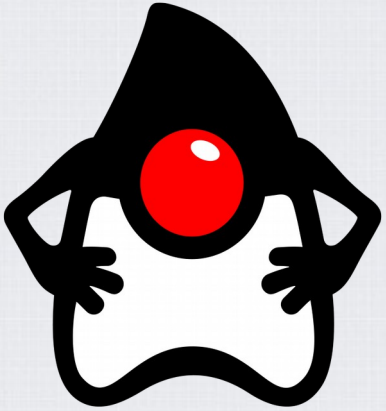
By default, G1 starts or continues a concurrent cycle during this periodic garbage collection. This minimizes disruption of the application, but compared to a full collection may ultimately not be able to return as much memory.

Any garbage collection triggered by this mechanism is tagged with the `G1 Periodic Collection` cause. An example of how such a log could look like is as follows:

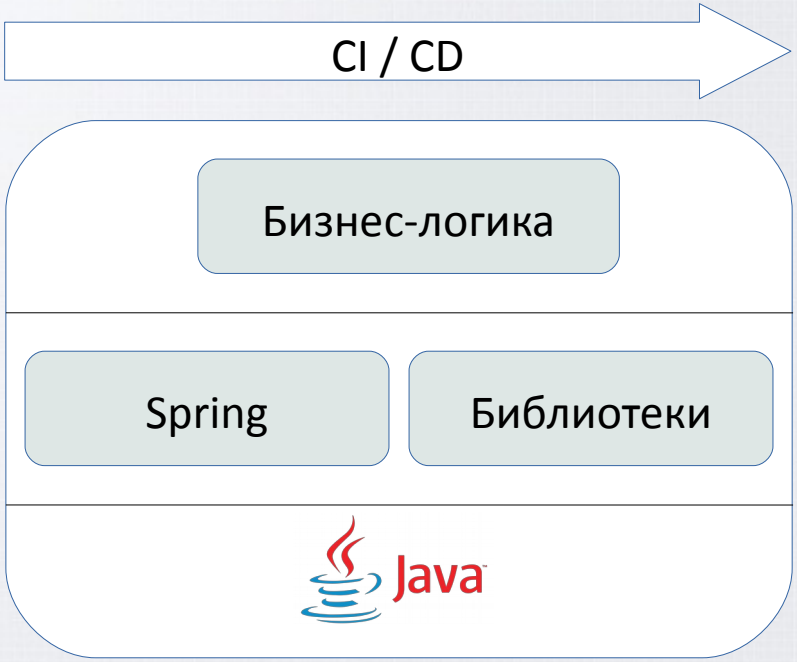
```
(1) [6.084s][debug][gc,periodic ] Checking for periodic GC.
    [6.086s][info ][gc          ] GC(13) Pause Young (Concurrent Start) (G1 Periodic Collection) 37M->36M(78M) 1.786ms
(2) [9.087s][debug][gc,periodic ] Checking for periodic GC.
    [9.088s][info ][gc          ] GC(15) Pause Young (Prepare Mixed) (G1 Periodic Collection) 9M->9M(32M) 0.722ms
```

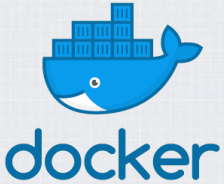
JDK 13+

- 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



Запустим всё по науке





Памяти одного светлого образа

The Best Linux Blog In the Unixve…
@nixcraft
Follow

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 remov…
It is a bummer, but I cannot reach Docker Hub team neither via e-mails, nor via [Docker Hub Feedback...
reddit.com

9:50 AM - 22 Jan 2019

114 Retweets **89** Likes

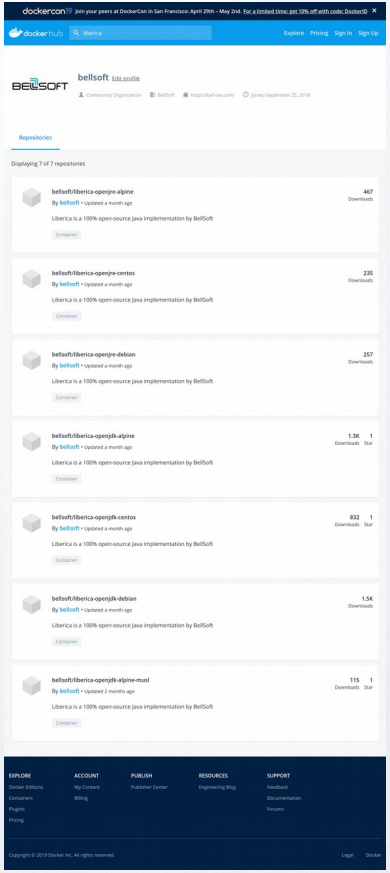
19 114 89

Tweet your reply

Maulwurf @de_maulwurf87 · Jan 22
Replying to @nixcraft @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.

8

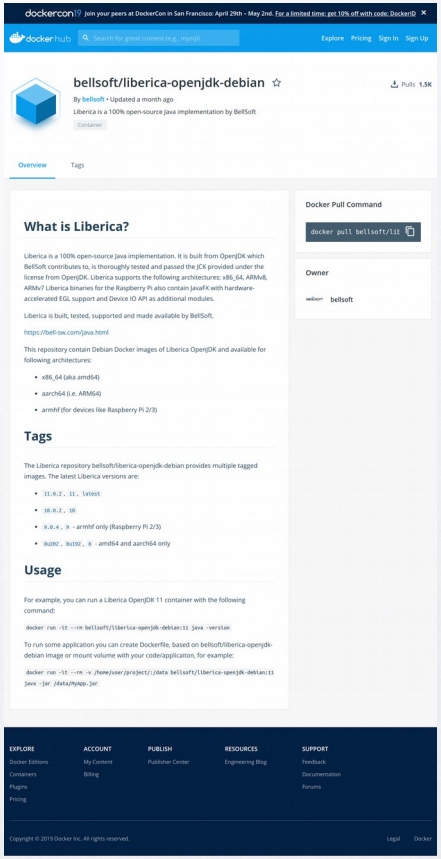
Базовые образы



https://hub.docker.com/u/bellsoft

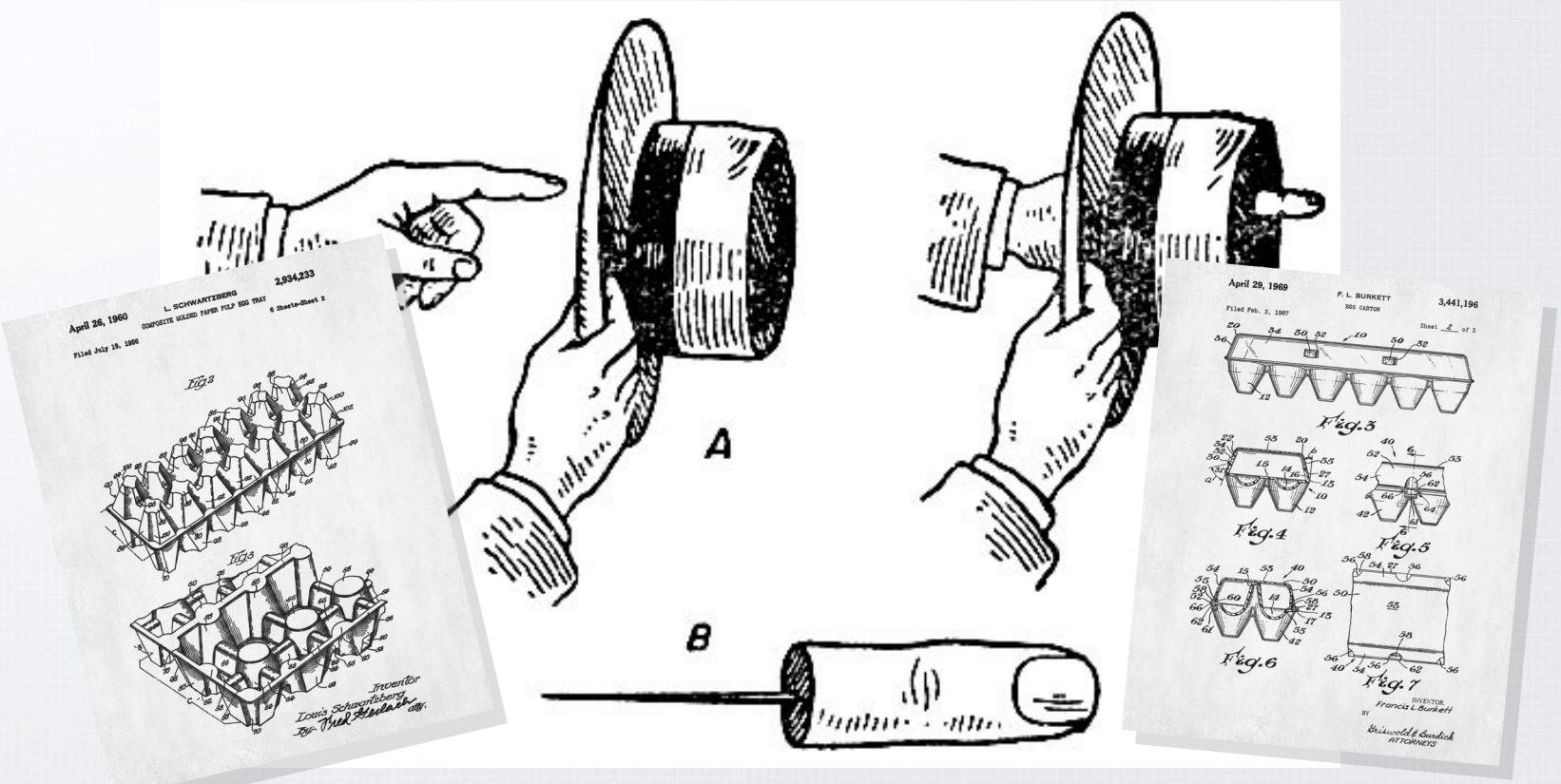
	JRE 8	JDK 11
Debian	227 MB	622 MB
Centos	307 MB	702 MB
Alpine	114 MB	509 MB
Alpine musl base		**37 MB**

Базовые образы



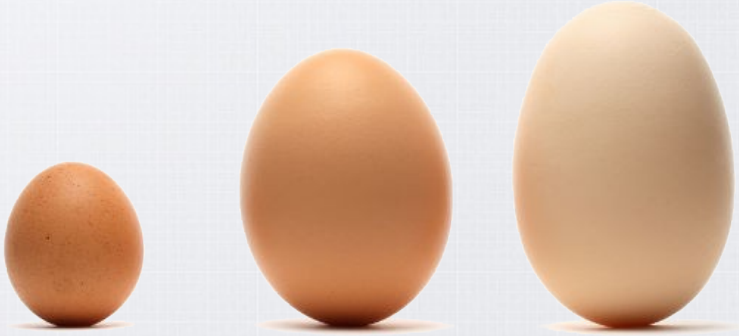
- Версии Java
  - 11
  - 8
- Дистрибутив
  - Debian
  - CentOS
  - Alpine
  - Alpine musl
- Архитектура
  - x86_64
  - ARM64
  - ARM32

Демо


April 26, 1960
L. SCHWARTZBERG
2,934,233
COMPOSITE MOLDED PAPER PULP EGG TRAY
6 Sheets-Sheet 2
Filed July 19, 1956
Fig.2
Fig.3
Inventor
Louis Schwartzberg
A
B
April 29, 1969
F. L. BURKETT
3,441,196
EGG CARTON
Filed Feb. 3, 1967
Sheet 2 of 3
Fig.3
Fig.4
Fig.5
Fig.6
Fig.7
INVENTOR
Francis L. Burkett
BY
ATTORNEYS

Сборка образа alpine-musl


```
$ mkdir ctx; cd ctx
$ wget https://raw.githubusercontent.com/bell-sw/Liberica/master/docker/alpine-musl/Dockerfile
$ docker build . --build-arg LIBERICA_IMAGE_VARIANT=base
```



Что-то случилось?


```
$ 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
```

Что-то случилось

```
$ journalctl -f _TRANSPORT=kernel
```

или

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

Сколько нужно памяти

- -XX:NativeMemoryTracking=summary
- jps
- jcmd

Что-то случается?

```
$ 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
```

Что-то случается

Кто-то толкается локтями

- docker stats
- jstat
- smem, pmap

```
$ 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)
```

AOT


```
$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
     -m 768m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.1 java \
     -XX:+UnlockDiagnosticVMOptions -XX:+LogTouchedMethods \
     -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication

$ jdk-11.0.1/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:11.0.1 jaotc \
     --compile-commands /demo/methods.list --jar $(cat cpv) \
     --info --ignore-errors --output /demo/thin.so
```

cpv – classpath в контейнере. Нет улучшений на старте.

AppCDS

```
$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
     -m 384m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.1 \
     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:11.0.1 \
     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:11.0.1 \
     java -Xshare:on -XX:SharedArchiveFile=/demo/hello-ext.jsa \
     -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication
```

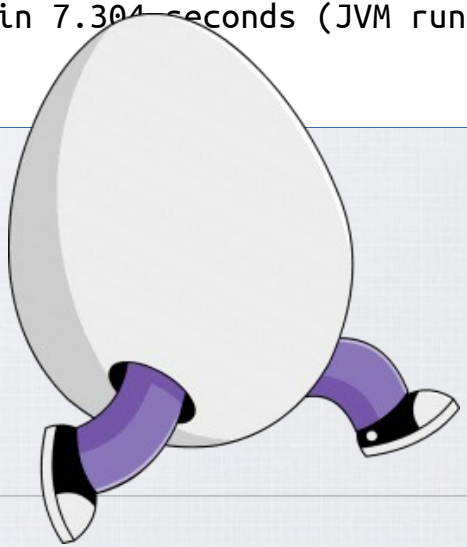
cpv – classpath в контейнере

Улучшения на старте

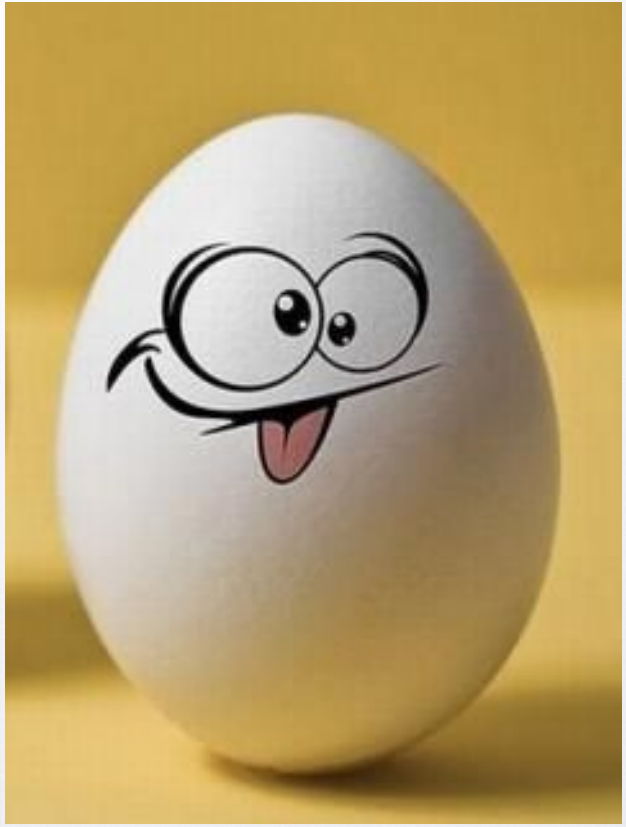
```
$ docker run -it --rm -v /export/dchuyko/demo:/demo -p 9000:8080 \
  -m 128m --memory-swappiness 0 bellsoft/liberica-openjdk-alpine:11.0.1 \
  java -XX:TieredStopAtLevel=1 \
  -Xshare:on -XX:SharedArchiveFile=/demo/hello-ext.jsa \
  -cp /demo/thin.jar:$(cat cpv) hello.HelloWorldApplication
~~~~~
Started HelloWorldApplication in 7.304 seconds (JVM running for 8.283)
```

÷6

x2.5



Выводы



- Java понимает контейнеры и работает в них
- Диагностика контейнеров работает для Java
- Диагностика Java работает для контейнеров
- В контейнере доступны все фичи
  - То, что будет использоваться в контейнере, лучше генерировать в идентичном контейнере
- Используйте свежие релизы и обновления
  - Эффективно
  - Безопасно
- Выберите подходящий базовый образ
- Всё это помогает приложениям
  - Предотвратить падения
  - Ограничить и уменьшить объём памяти
  - Ускорить запуск