ORACLE

JVMs in Containers

QCon LONDON by InfoQ

David Delabassée

@delabassee

DevRel Java Platform Group

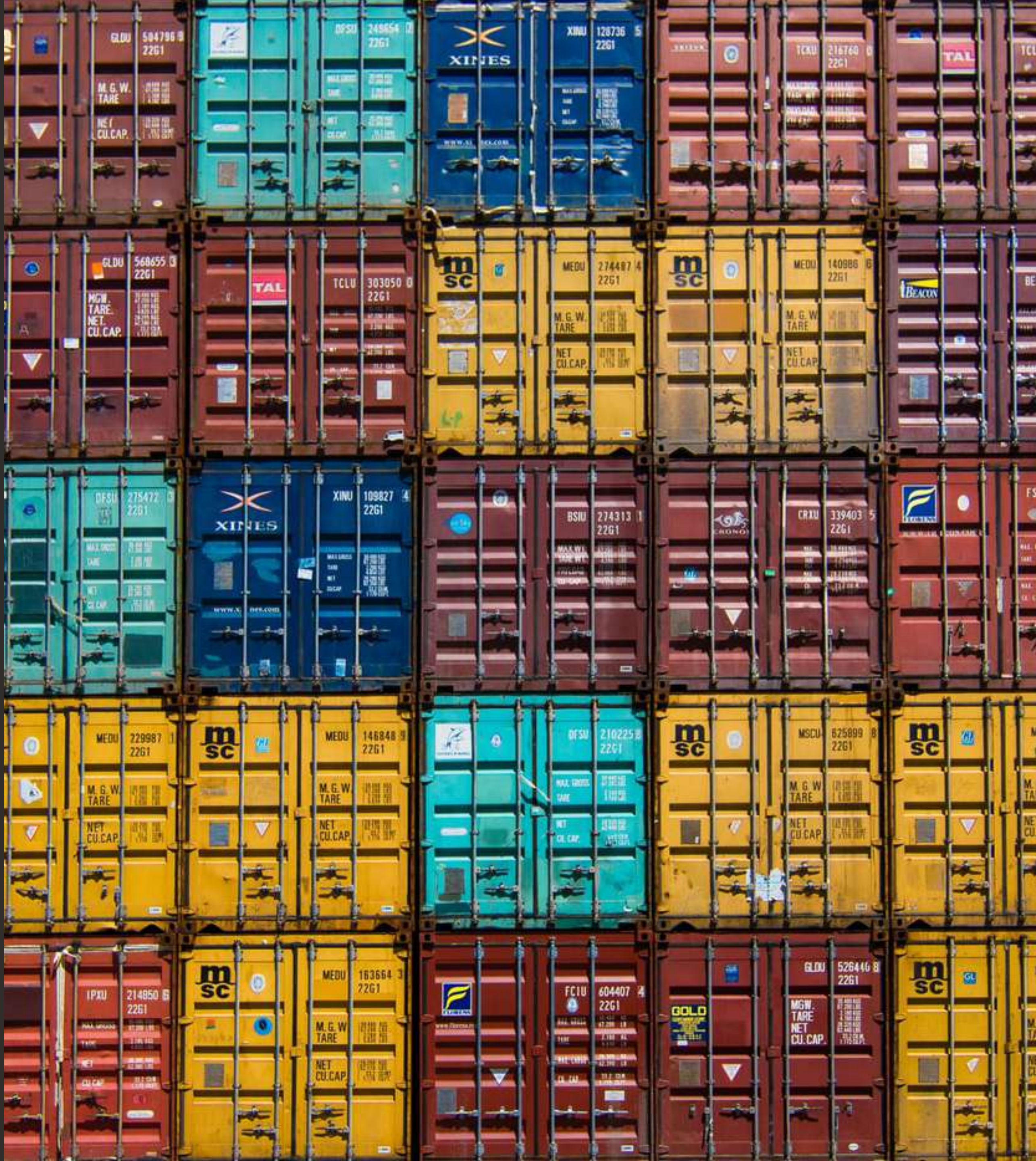
March 2020

Safe harbor statement

Developer productivity
Application performance

In the face of constantly-evolving
programming paradigms,
application styles,
hardware
and deployment styles.

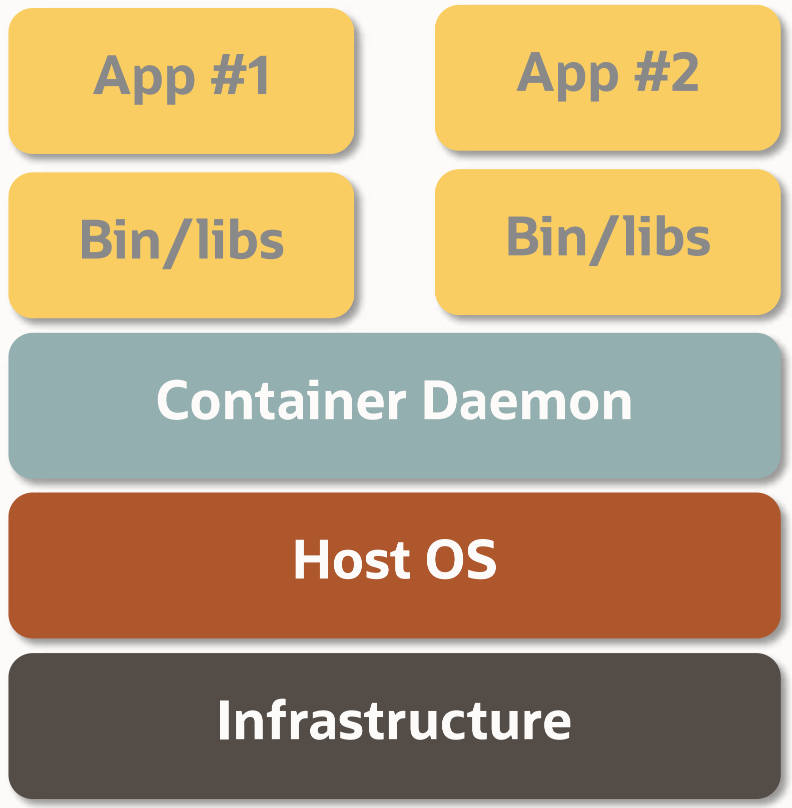
Containers

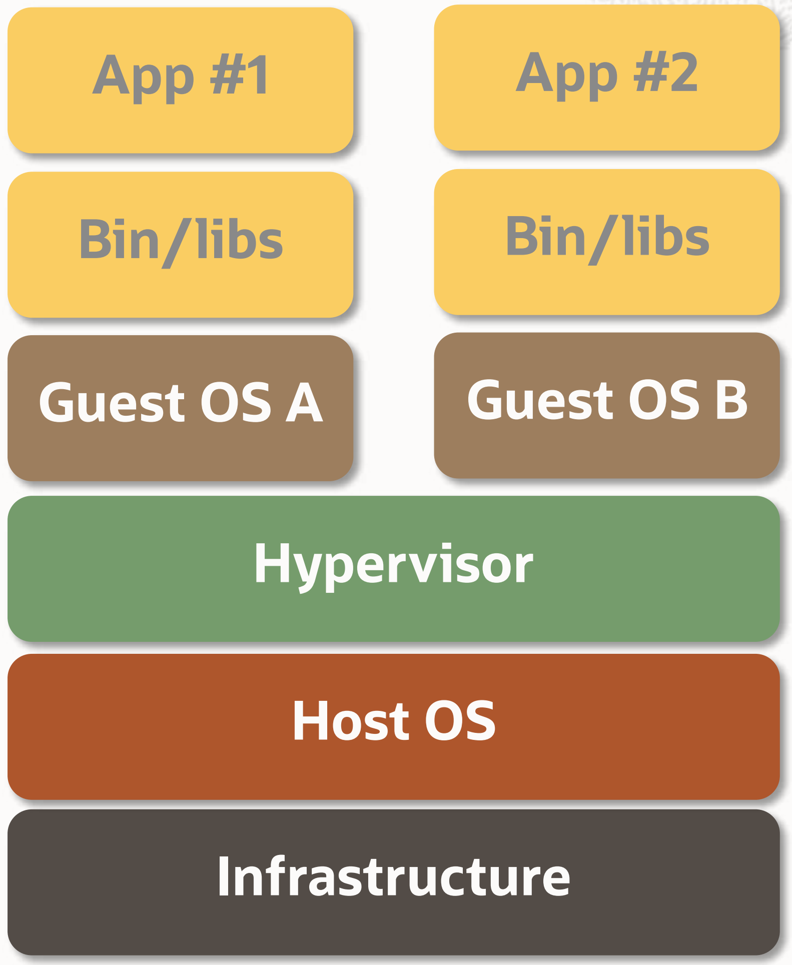


Container

- Package Software into Standardized Units
  - Development
  - Shipment
  - Deployment
- Runtimes
  - Docker, CRI-O, LXC, Rkt, runC, systemd-nspawn, OpenVZ, etc.

Container vs. VM





JVM

JVM Container Landscape

Tools

- docker-maven-plugin
- jib + jib-maven-plugin
- Testcontainers
- IDE
- …

Frameworks

- Helidon
- Quarkus
- Micronaut
- Jhipster
- Spring Boot
- …

FaaS

- Fn Project
- OpenFaaS
- OpenWhisk
- …

JVM Container Awareness

JDK-8186248	More flexibility in selecting Heap % of available RAM (8u144)
JDK-8179498	attach should be relative to `/proc/pid/root` and namespace aware as `jcmd`, `jstack`, ... fail to attach (10)
JDK-8146115	Improve Docker container detection & resource config usage (10)
JDK-8193710	`jcmd -l` & `jps` do not list Java processes running in containers (11)
JDK-8203357	Container Metrics (11)
JDK-8220786	Create new switch to redirect error reporting output to `stdout` or `stderr` (13)
JDK-8203359	JFR `jdk.CPUInformation` event reports incorrect information when running in container (in progress)
…	…

https://bugs.openjdk.java.net

JVM Ergonomics

- The JVM tunes itself based on the system it runs on
- Behavior-Based Tuning dynamically optimizes the sizes of the heap to meet an expected behavior
  - Maximum Pause-time (`-XX:MaxGCPauseMillis`)
  - Or Application Throughput (`-XX:GCTimeRatio`)
- Sets defaults for the GC, heap size, and runtime compiler

https://docs.oracle.com/en/java/javase/13/gctuning/ergonomics.html

Hello Container

Performance

“Latency”

“Latency”

Container Startup

Stack of Layers

3 ‘core’ layers

- Java application and its dependencies
- Java Runtime
- Operating System

⇨ Reduce layers size

Java Application Layer

- Dependencies!
- Leverage Container cache layer mechanism
  - Fat JAR?
  - Anything that is (relatively) static in its own layer
  - CDS Shared Archive

Java Runtime Layer

9

Serverless Java function (Fn) - openjdk:13

	Modules	jlink flags	MB		
JDK	Whole JDK!		**316**	**100%**	
Runtime image	All (explicit)	`--add-modules $(java --list-modules)`	**178**	56%	**100%**
Custom runtime image	Only required modules	`--add-modules $(jdeps --print-module-deps …)`	50	16%	28%
		`… --no-header-files --no-man-pages --strip-java-debug-attributes`	44	14%	25%
		`… --compress=1`	37	12%	21%
		`… --compress=2`	**34**	**11%**	**19%**

316 MB → 178 MB → 50 MB → 34 MB

Operating System Layer

- Slim distros
  - debian: bullseye (117 MB) vs. debian: bullseye-slim (71 MB)
- Distroless distros
  - gcr.io/distroless/java:11 (195 MB - Java included)
- Docker-slim
  - *“Don't change anything in your Docker container image and minify it by up to 30x”* (?)
- …

Operating System Layer

- Alpine - Security-oriented, lightweight Linux distro
- musl - Lightweight, fast, free, C standard library implementation

- alpine-pkg-glibc - glibc compatibility layer package for Alpine
  https://github.com/sgerrand/alpine-pkg-glibc
- Project Portola - Runs OpenJDK on musl (*)
  https://openjdk.java.net/projects/portola/

JLINK Alpine

Java Runtime Layer

Minecraft server

java.base, java.compiler, java.desktop, java.management, java.naming, java.rmi, java.scripting, java.sql, jdk.sctp, jdk.unsupported, jdk.zipfs

openjdk:13 (*) (12 modules)	88 MB
`--strip-debug --strip-java-debug-attributes`	-14 MB
`--compress=1`	-18 MB
`--compress=2`	-31 MB
`--no-header-file --no-man-pages`	0 MB

(*) Oracle OpenJDK builds on OEL - YMMV!

Java Runtime Layer

13

Base Image	Java	Module	Custom Runtime
openjdk:13	Inc. Oracle OpenJDK 13	java.base	48 MB
debian:buster	+ Debian openjdk-13-jdk	java.base	491 MB

`--strip-native-debug-symbols` (*)

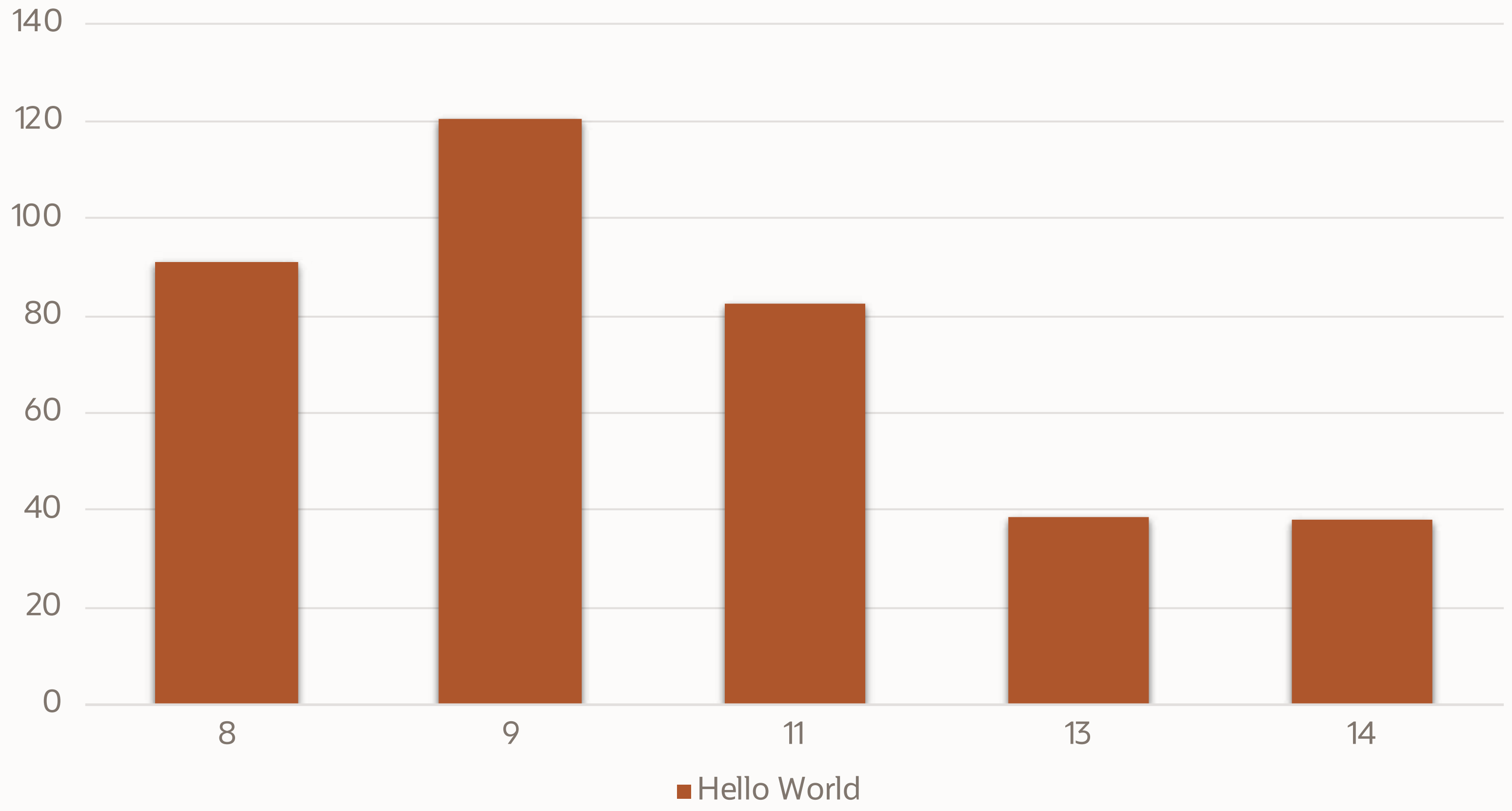
debian:buster	+ Debian openjdk-13-jdk	java.base	51 MB

(*) JDK 13 https://bugs.openjdk.java.net/browse/JDK-8219257

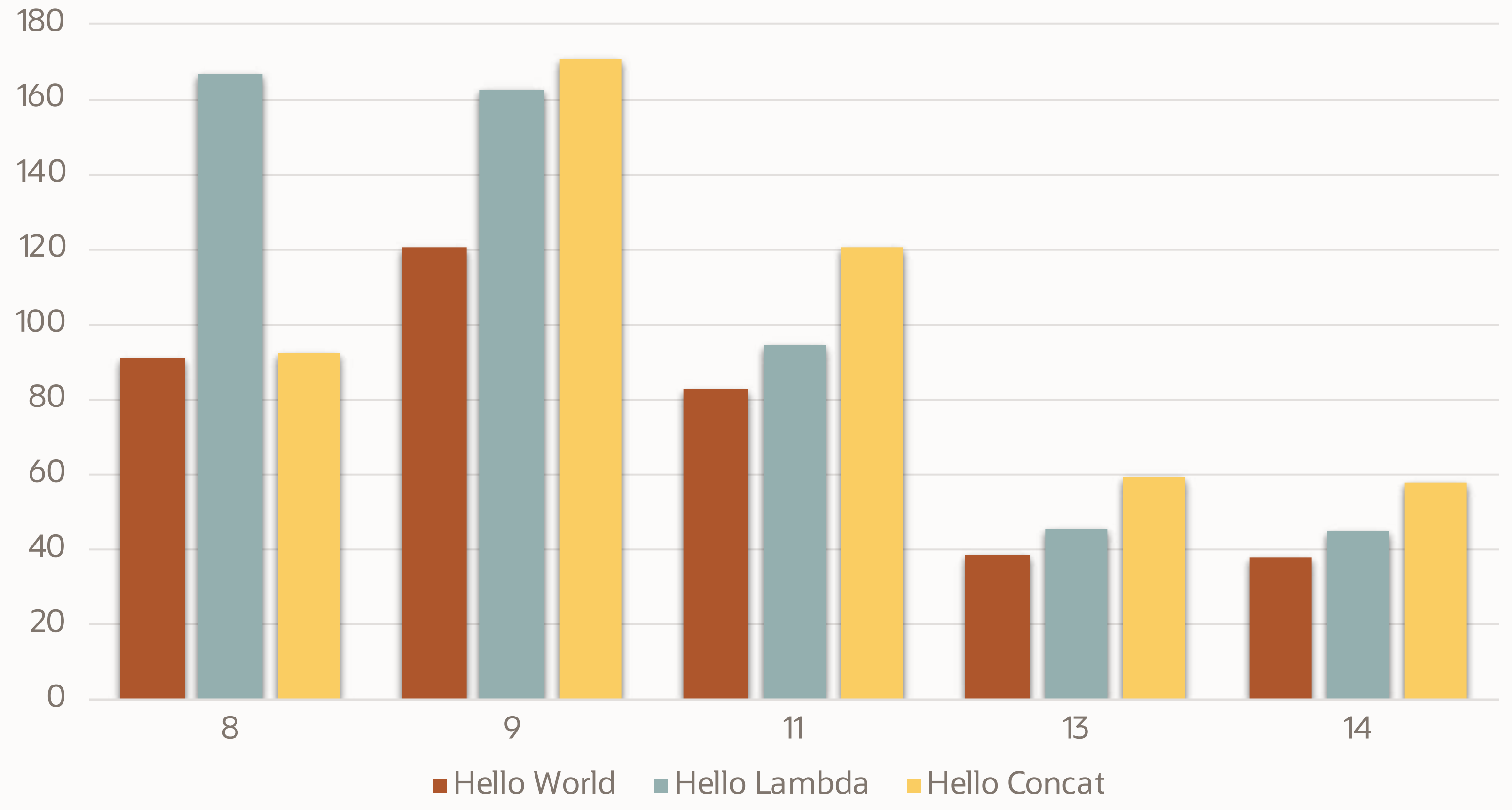
“Latency”

Application Startup

Java - Startup Time



Java - Startup Time



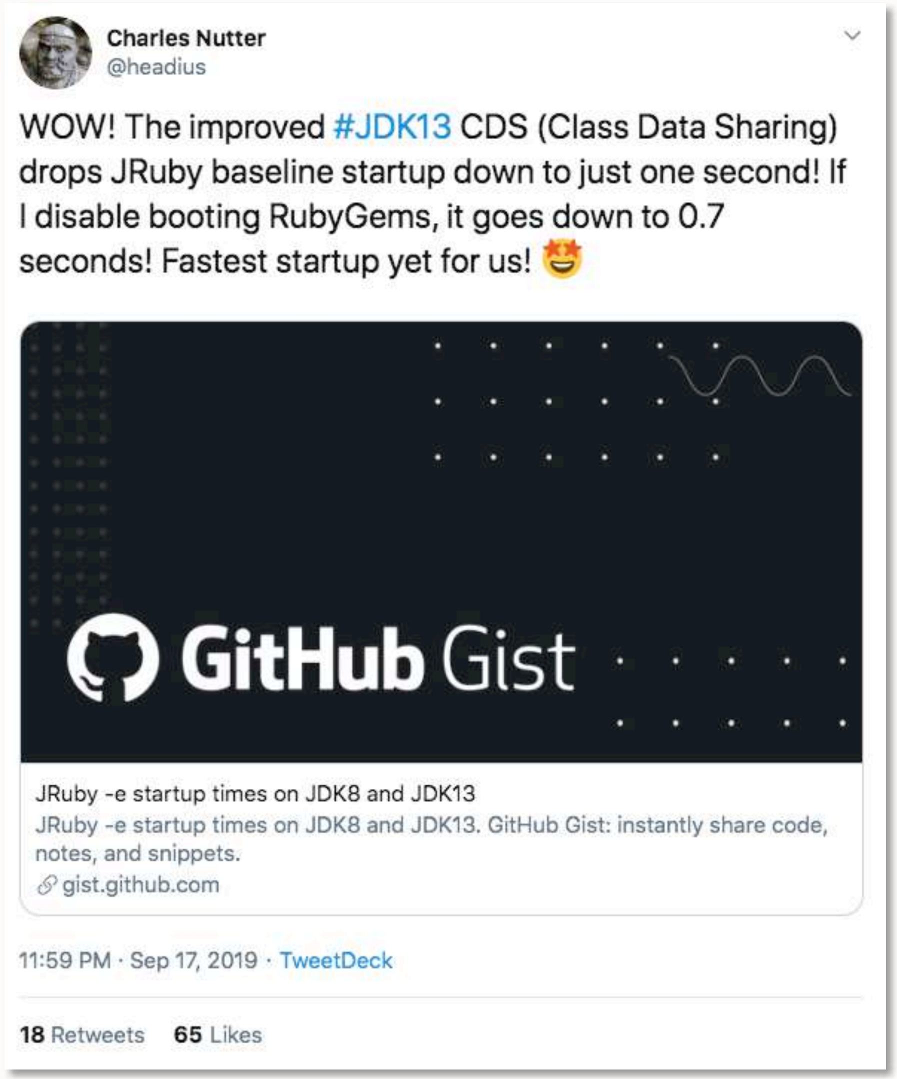
https://cl4es.github.io

Class Data Sharing

- Reduce memory footprint between multiple JVMs by sharing common class metadata
- Improve startup time
- How?
  - Loads classes from JAR file into a private internal representation
  - Dumps it to a shared archive
  - When JVMs (re)starts, the archive is memory-mapped to allow sharing of R/O JVM metadata for these classes among multiple JVMs

CDS

Application CDS


Charles Nutter
@headius

WOW! The improved #JDK13 CDS (Class Data Sharing) drops JRuby baseline startup down to just one second! If I disable booting RubyGems, it goes down to 0.7 seconds! Fastest startup yet for us! 🤩

JRuby -e startup times on JDK8 and JDK13
JRuby -e startup times on JDK8 and JDK13. GitHub Gist: instantly share code, notes, and snippets.
gist.github.com

11:59 PM · Sep 17, 2019 · TweetDeck

18 Retweets 65 Likes

Application CDS

jdk-08-u202-b08-hotspot

```
jruby -e 1
    real      0m1.601s
    …

jruby --dev -e 1
    real      0m1.216s
    …

jruby --disable-gems --dev -e 1
    real      0m0.853s
    …
```

jdk-13.jdk

```
… -J-XX:SharedArchiveFile=jruby.jsa
    real      0m1.491s
    …

… -J-XX:SharedArchiveFile=jruby.jsa
    real      0m1.089s
    …

… -J-XX:SharedArchiveFile=jruby.jsa
    real      0m0.717s
    …
```

Class Data Sharing

- Java 5 - Limited to system classes and serial GC
- Java 9 - Application CDS and other GCs (commercial feature + JEP 250)
- Java 10 - Application CDS (JEP 310)
- Java 12 - Default CDS Archives (JEP 341)
- Java 13 - Dynamic CDS Archives (JEP 350)

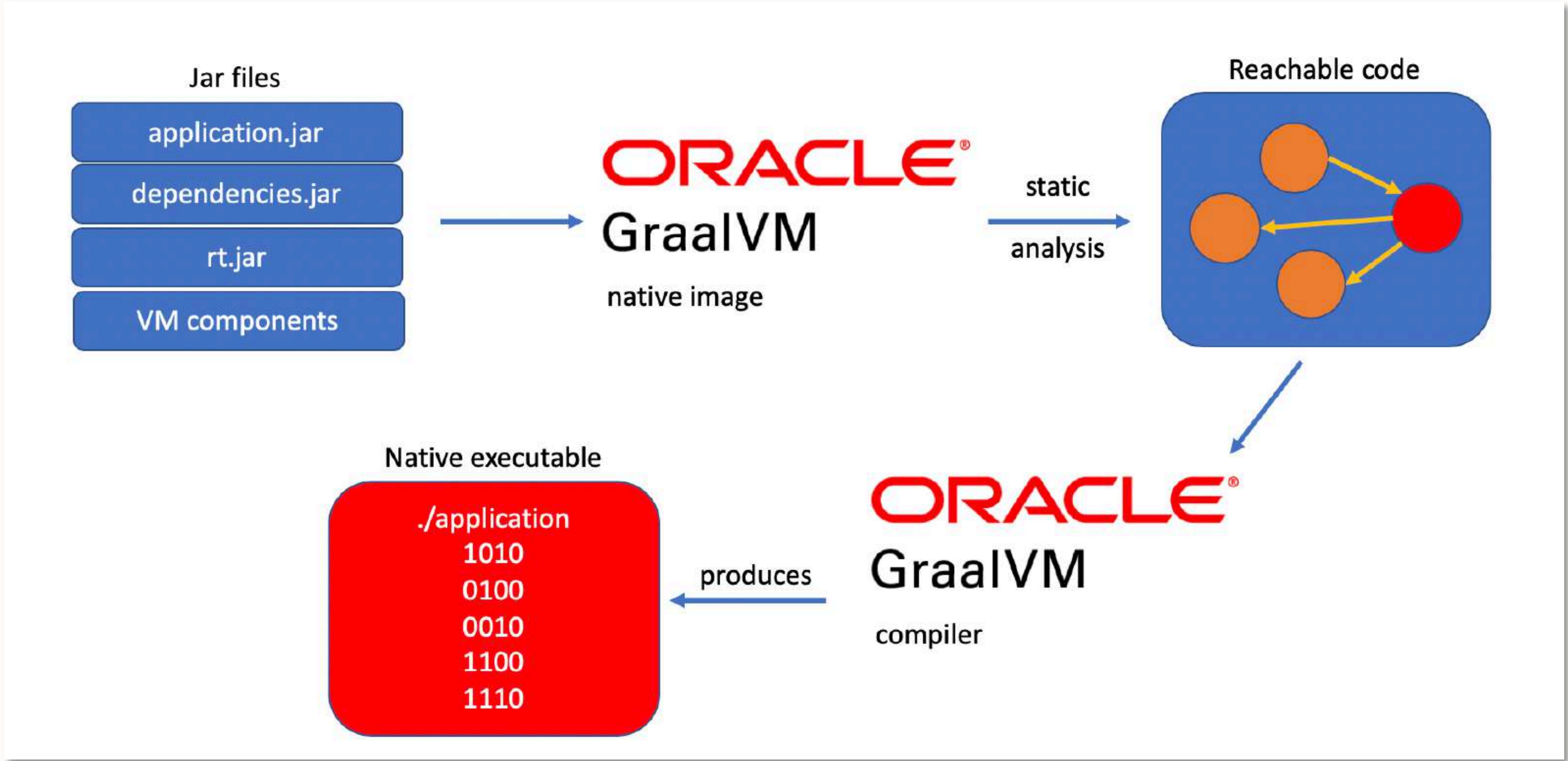
GraalVM

- High-performance polyglot VM
- …
- Polyglot API
- JIT Compiler
- AOT Compiler - native-image
  - Reduced startup time
  - Improved foot-print
  - Reduced image size

GraalVM™

https://www.graalvm.org

GraalVM - native-image



GraalVM - native-image limitations

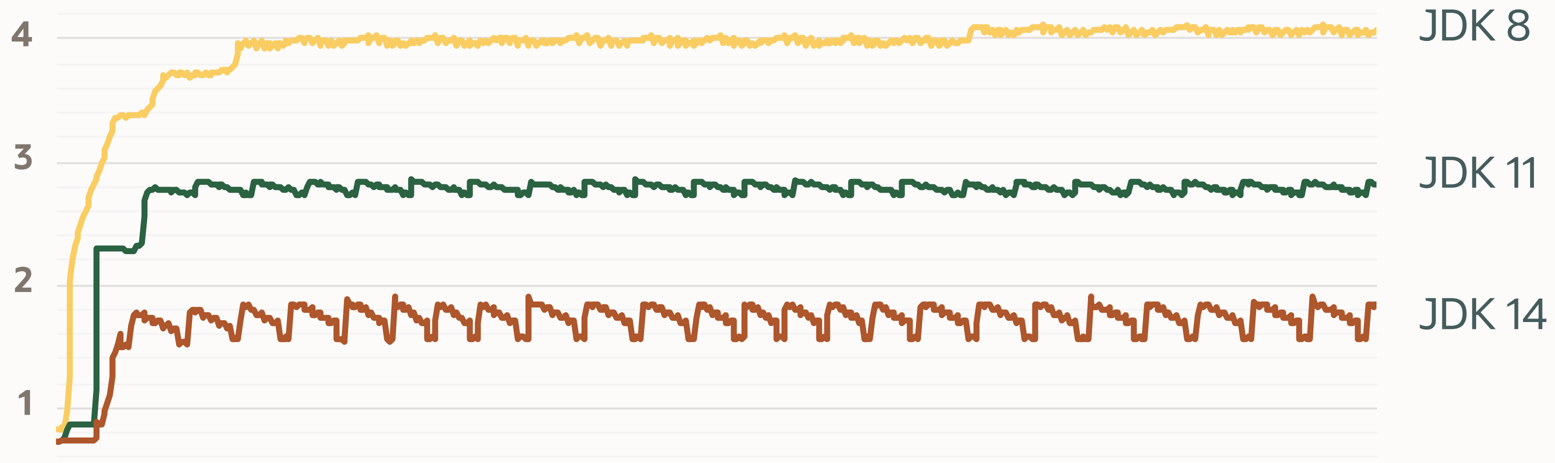
- Java 8 & 11
- Mostly supported
  - Reflections, Dynamic Proxy, JNI, Unsafe Memory Access, Static Initializers, References
- Not supported
  - InvokeDynamic (*) and Method Handles, Dynamic Class Un/Loading, Finalizers, Security Manager, Serialisation
  - Native VM interfaces (JVMTI, JMX, etc.)

https://github.com/oracle/graal/blob/master/substratevm/LIMITATIONS.md

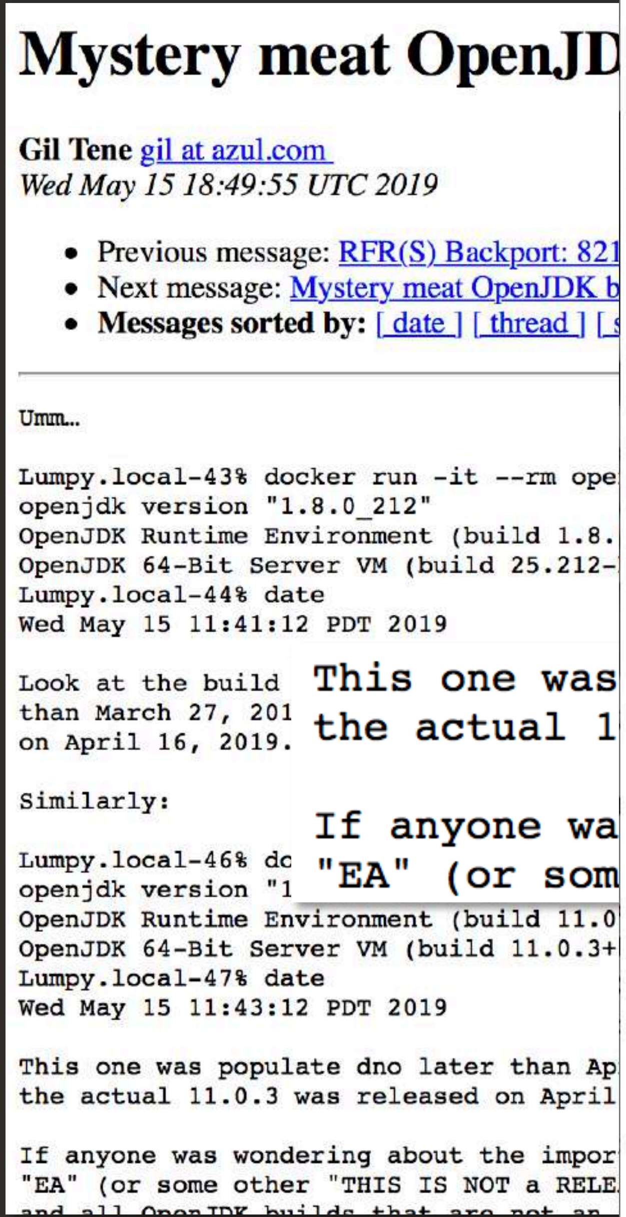
G1 GC

14

- NUMA-Aware Memory Allocation for G1 - JEP 345
- ~ 700 enhancements since JDK 8, across all areas!
  - Across all areas ⇨ significant improvements
- Ex. Native memory usage over time (GB)
  - BigRamTester, w. 16GB heap



Security



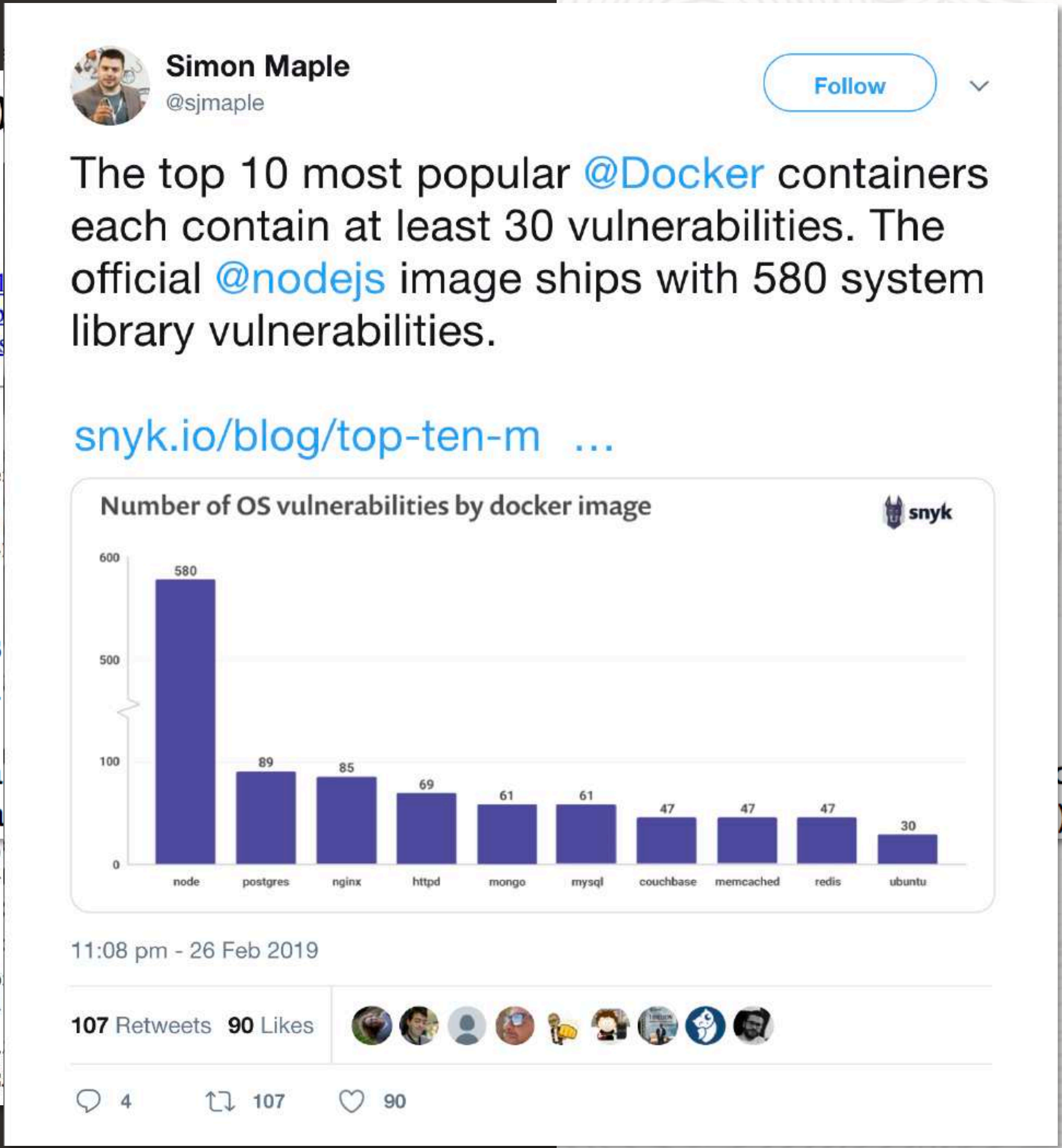
Mystery meat OpenJD

Gil Tene gil at azul.com
Wed May 15 18:49:55 UTC 2019

- Previous message: RFR(S) Backport: 821
- Next message: Mystery meat OpenJDK b
- **Messages sorted by:** [date] [thread] [s

```
Umm...

Lumpy.local-43% docker run -it --rm ope
openjdk version "1.8.0_212"
OpenJDK Runtime Environment (build 1.8.
OpenJDK 64-Bit Server VM (build 25.212-
Lumpy.local-44% date
Wed May 15 11:41:12 PDT 2019

Look at the build
than March 27, 201
on April 16, 2019.

Similarly:

Lumpy.local-46% dc
openjdk version "1
OpenJDK Runtime Environment (build 11.0
OpenJDK 64-Bit Server VM (build 11.0.3+
Lumpy.local-47% date
Wed May 15 11:43:12 PDT 2019

This one was populate dno later than Ap
the actual 11.0.3 was released on April

If anyone was wondering about the impor
"EA" (or some other "THIS IS NOT a RELE
and all OpenJDK builds that are not an
```

```
This one was
the actual 1

If anyone wa
"EA" (or som
```

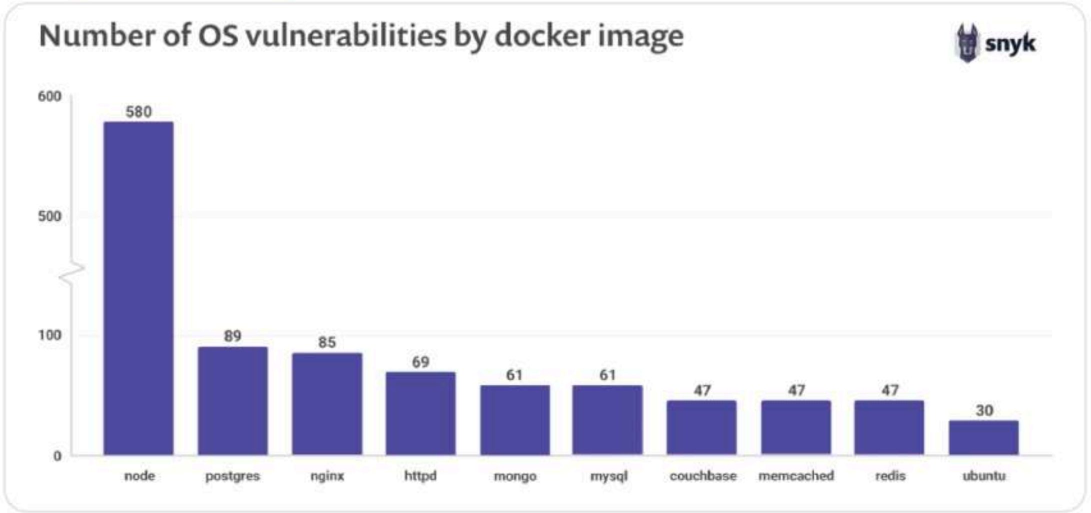


Simon Maple
@sjmaple

Follow

The top 10 most popular @Docker containers each contain at least 30 vulnerabilities. The official @nodejs image ships with 580 system library vulnerabilities.

snyk.io/blog/top-ten-m …

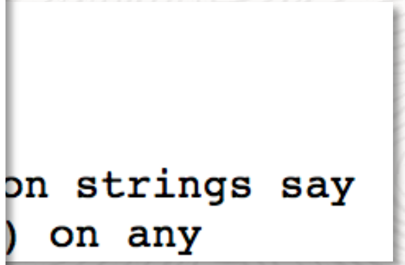


node	580
postgres	89
nginx	85
httpd	69
mongo	61
mysql	61
couchbase	47
memcached	47
redis	47
ubuntu	30

11:08 pm - 26 Feb 2019

107 Retweets 90 Likes

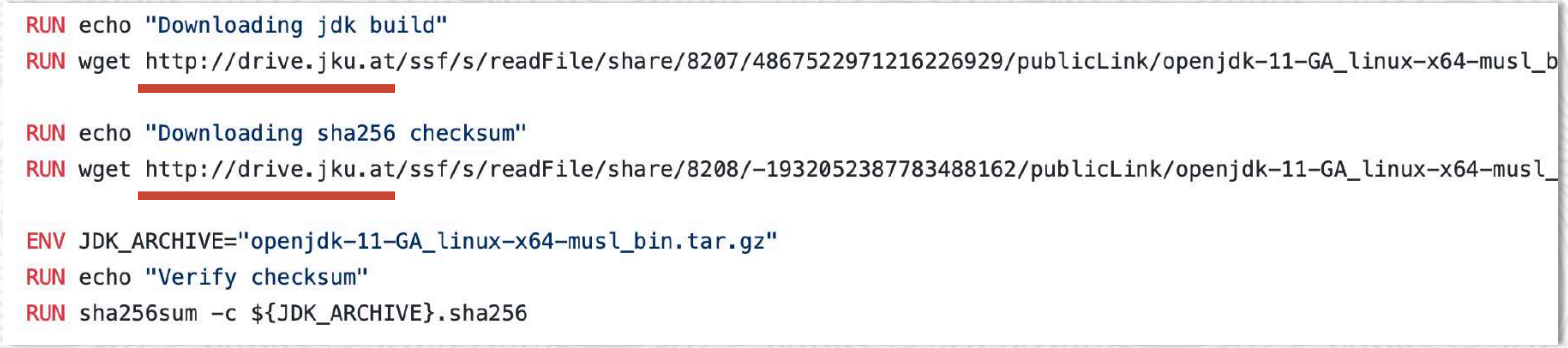
4 107 90



```
on strings say
) on any
```

Where in the World Is openjdk-11-GA_linux-x64-musl?

"… so you can consider it as the (OpenJDK 11 Alpine) General-Availability Release"



```
RUN echo "Downloading jdk build"
RUN wget http://drive.jku.at/ssf/s/readFile/share/8207/4867522971216226929/publicLink/openjdk-11-GA_linux-x64-musl_b

RUN echo "Downloading sha256 checksum"
RUN wget http://drive.jku.at/ssf/s/readFile/share/8208/-1932052387783488162/publicLink/openjdk-11-GA_linux-x64-musl_

ENV JDK_ARCHIVE="openjdk-11-GA_linux-x64-musl_bin.tar.gz"
RUN echo "Verify checksum"
RUN sha256sum -c ${JDK_ARCHIVE}.sha256
```

Choose your base image wisely !
And secure it!

Rootless container

- Ideally containers should be managed and run by the respective container runtime without root privileges
- Docker Rootless mode (experimental)
  - https://docs.docker.com/engine/security/rootless/

Rootless container

15 (*)

- Unified Control Groups Hierarchy aka “cgroups v2”
  - Linux kernel 3.16 (Aug. 2014)
  - Enabled by default on Fedora 31 (Oct. 2019)
- Pod Man Rootless containers
  - https://podman.io/blogs/2019/10/29/podman-crun-f31.html
- JDK 15 - cgroups v2 Container awareness
  - JDK-8230305
  - Memory, cpu, cpuset
  - Fall back to cgroups v1 container support

And common sense!

- Docker-bench-security, Snyk, Clair, Anchore, etc.
- Certificates!
- Processes in containers should not run as Root
- Rely on an actively maintained Java runtime
- Reduce the potential surface attack
  - jlink’s Custom Runtime Image
- …

Observability

Observability

14

- JDK tools
  - `jcmd`, `jinfo`, `jps`, `jmap` …

  ⇨ `docker exec <container> <jdk_command> …`
- JDK Flight Recorder
  - Low overhead event based tracing framework built into the JVM
  - Keeps history of tracing data, enables "after-the-fact" analysis
- JFR Event Streaming - JDK 14
  - Stream event data as it is being produced, enables continuous monitoring
  - API for the continuous consumption of events
  - In-process and out-of-process

Wrap-up

JVMs in Containers

- JVM behaves as a good (Container) citizen
- Reduce "latency"
  - Container Startup
  - Application Startup
- All OpenJDK investments "leaks" into containers too!
  - Features
  - Performance
  - Footprint
  - Etc.

Innovating for the Future

ZGC
Create a scalable low latency garbage collector capable of handling large heaps

Loom
Massively scale lightweight threads, making concurrency simple again

Amber
Continuously improve developer productivity through evolutions of the Java language

Panama
Higher performance and easier development of I/O intensive applications through Java-native platform enhancements

Valhalla
Higher density and performance of machine learning and big data applications through the introduction of Value Types

Metropolis
Implement more of the JVM in Java starting with the JIT complier “Java-on-Java”

https://openjdk.java.net

JVMs in Containers

- Choose your base image wisely!
- Use the latest Java version, never java:latest !!!
- Only rely on actively-supported versions!
  - They are Container aware!
  - `-XX:+UseContainerSupport`
- Use a JRE/Java runtime image instead of a JDK

Thanks!

David Delabassée
@delabassee

ORACLE