Docker Who.
Small Containers Through Time and Space

Dmitry Chuyko

2022

Who we are

Dmitry Chuyko

BELLSOFT

Liberica www.bell-sw.com
supported OpenJDK binaries

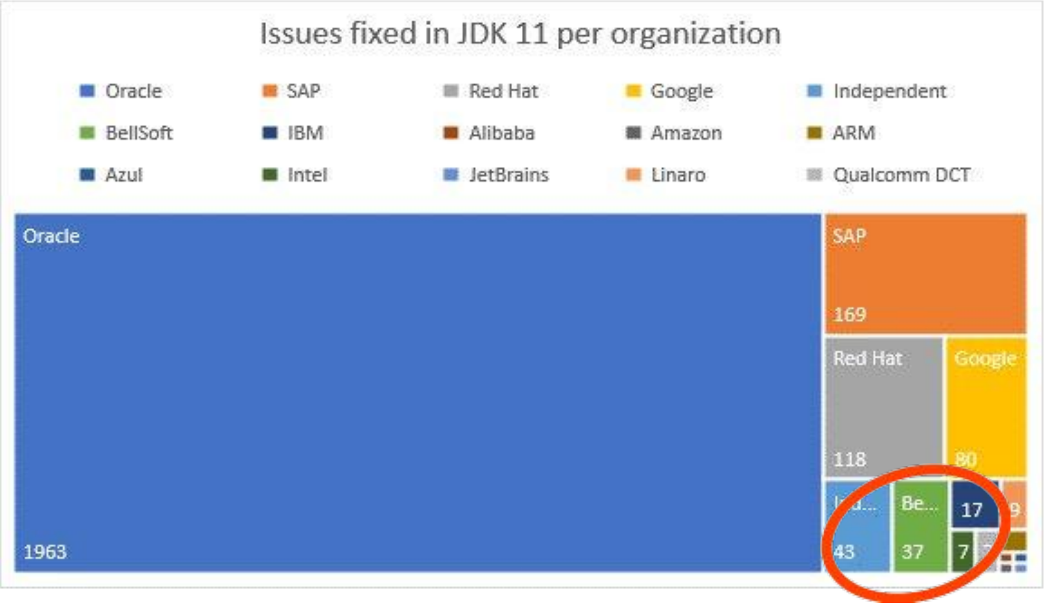
ex-employers:

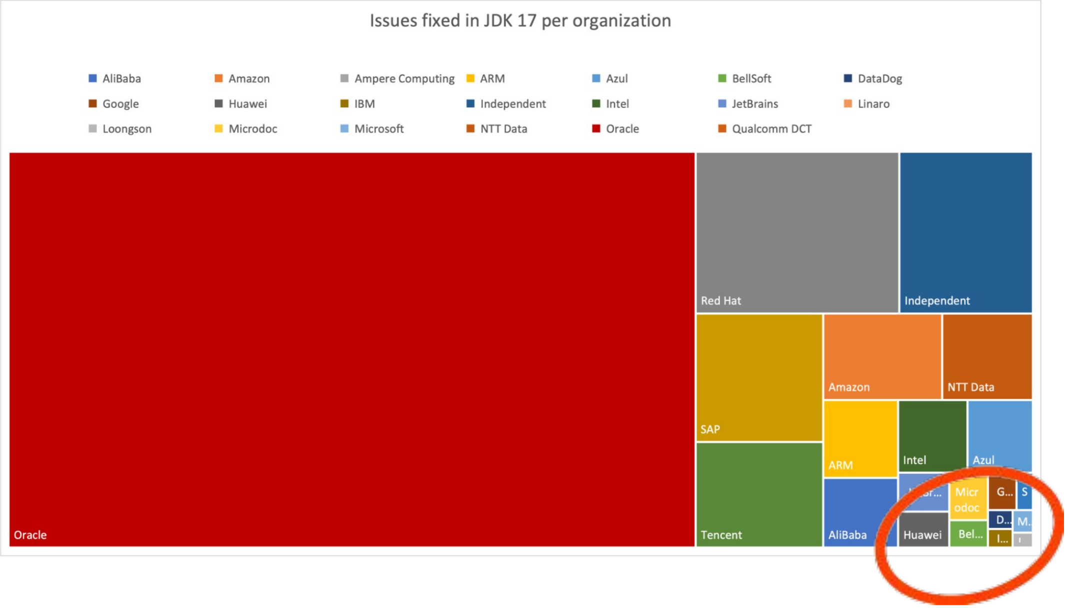
ORACLE

@dchuyko

OpenJDK Contributions

LTS





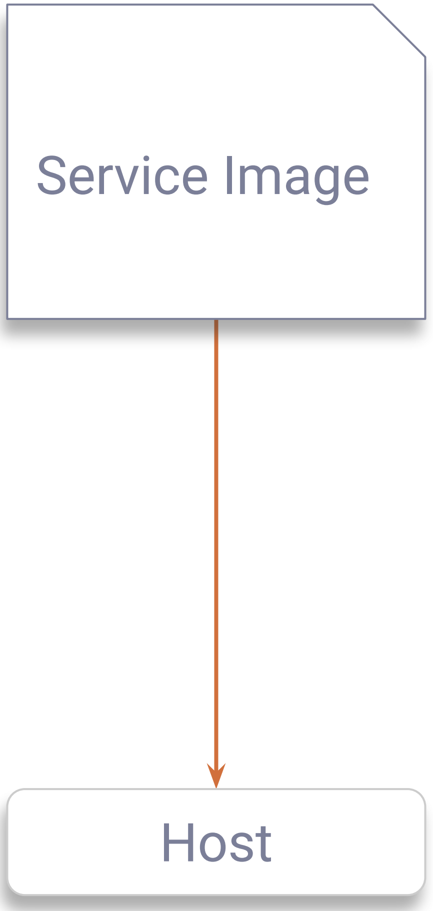
BELLSOFT

Deployment

> *...package an application with all of its dependencies into a standardized unit for software development.*
>
> — Docker

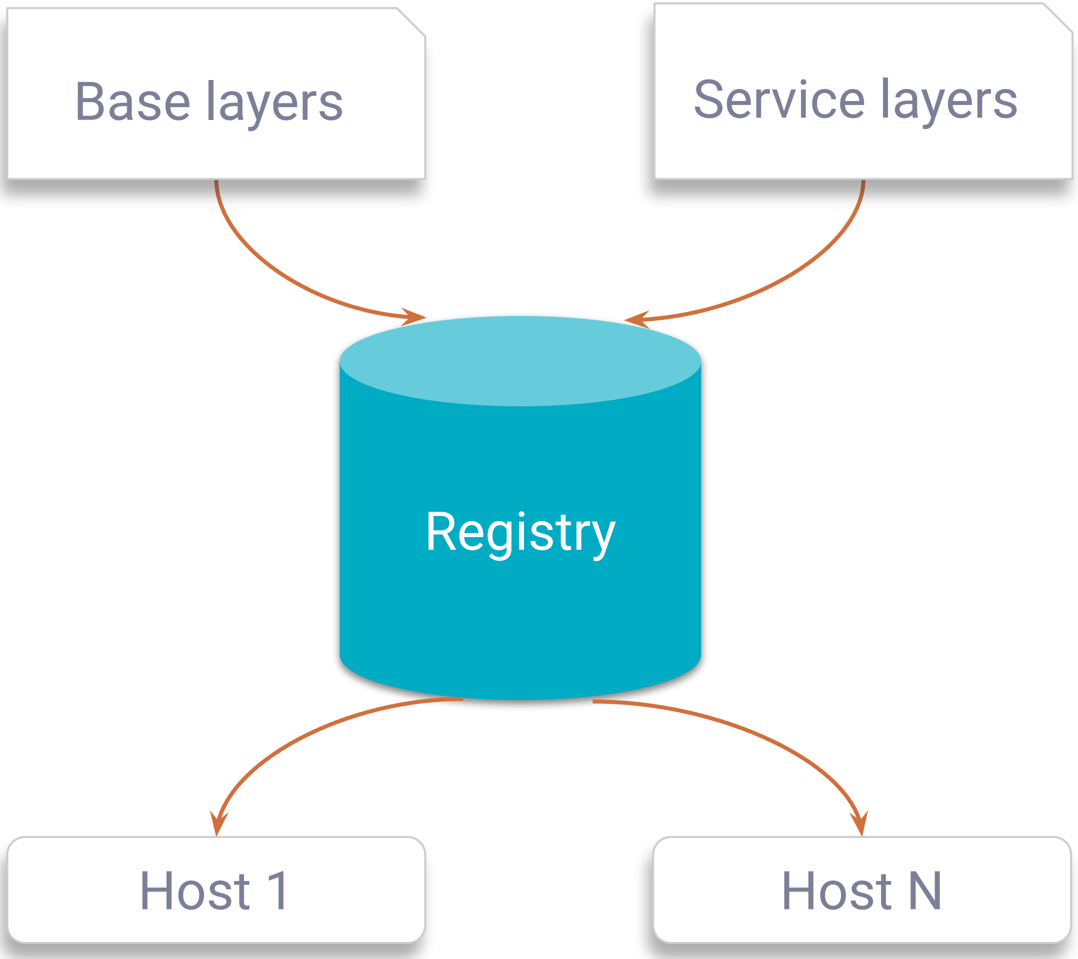
Deploy an image. Direct

- **Participants**
  - User/CI in Dev local or cloud
  - Hosts in the cloud
- **Transfer**
  - Full image every time
- **Custom connection**
- **Custom topology management**

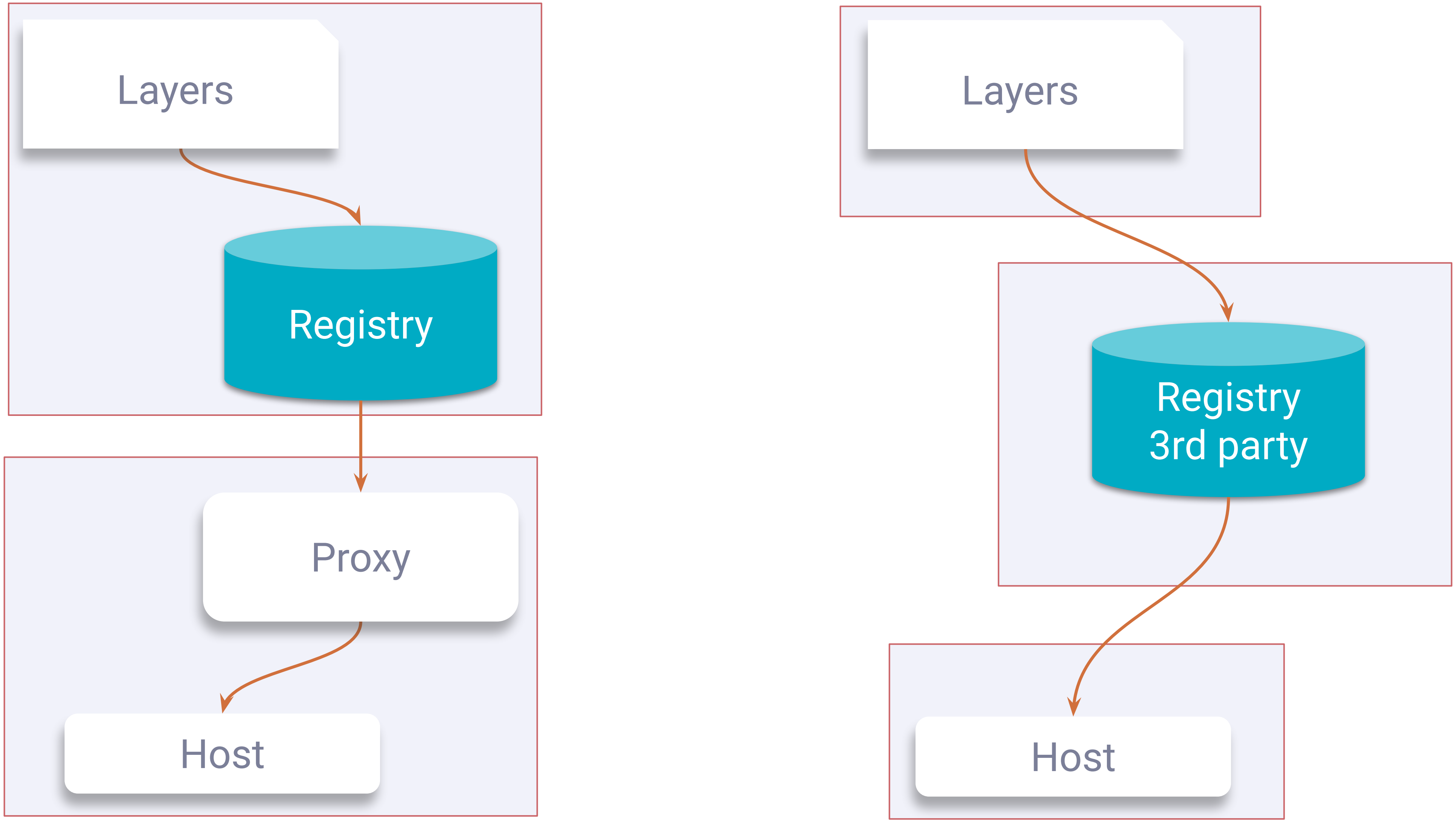


Deploy an image. Registry

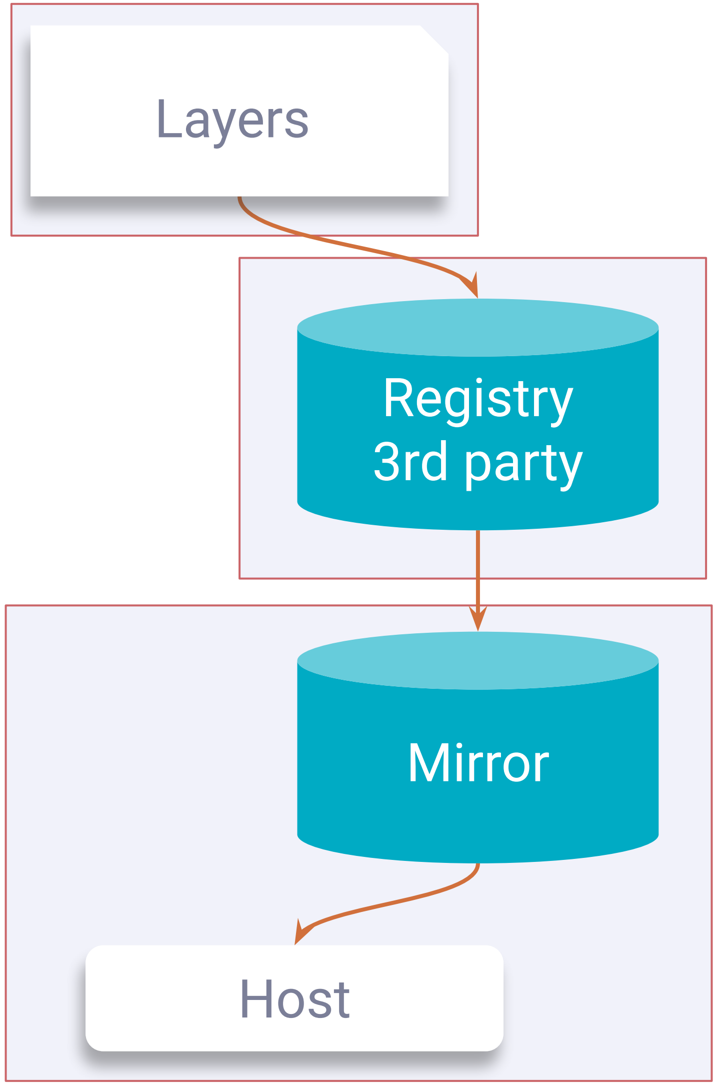
- **Participants**
  - User/CI in Dev local or cloud
  - Hosts in the cloud
  - Registry
    - User/CI in Dev local or cloud (proxy)
    - Cloud
    - Cloud SaaS
    - Public 3rd party SaaS
- **Transfer**
  - All layers for a clean host
  - New layers

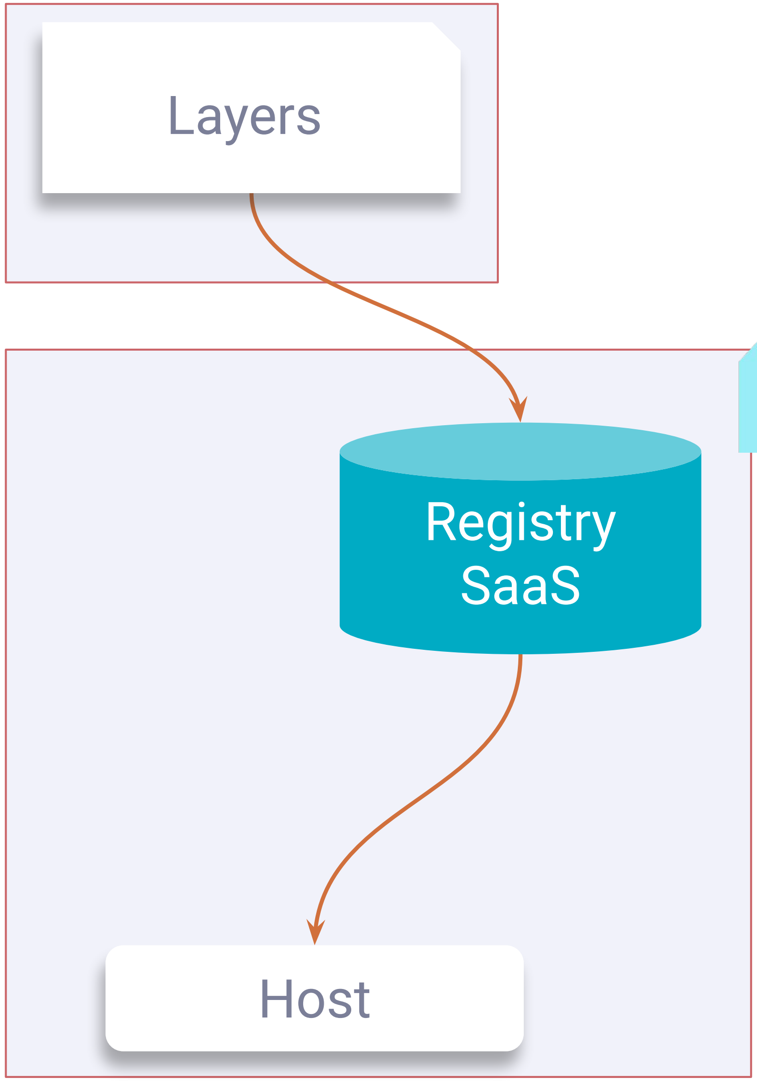


Deploy an image. Networks

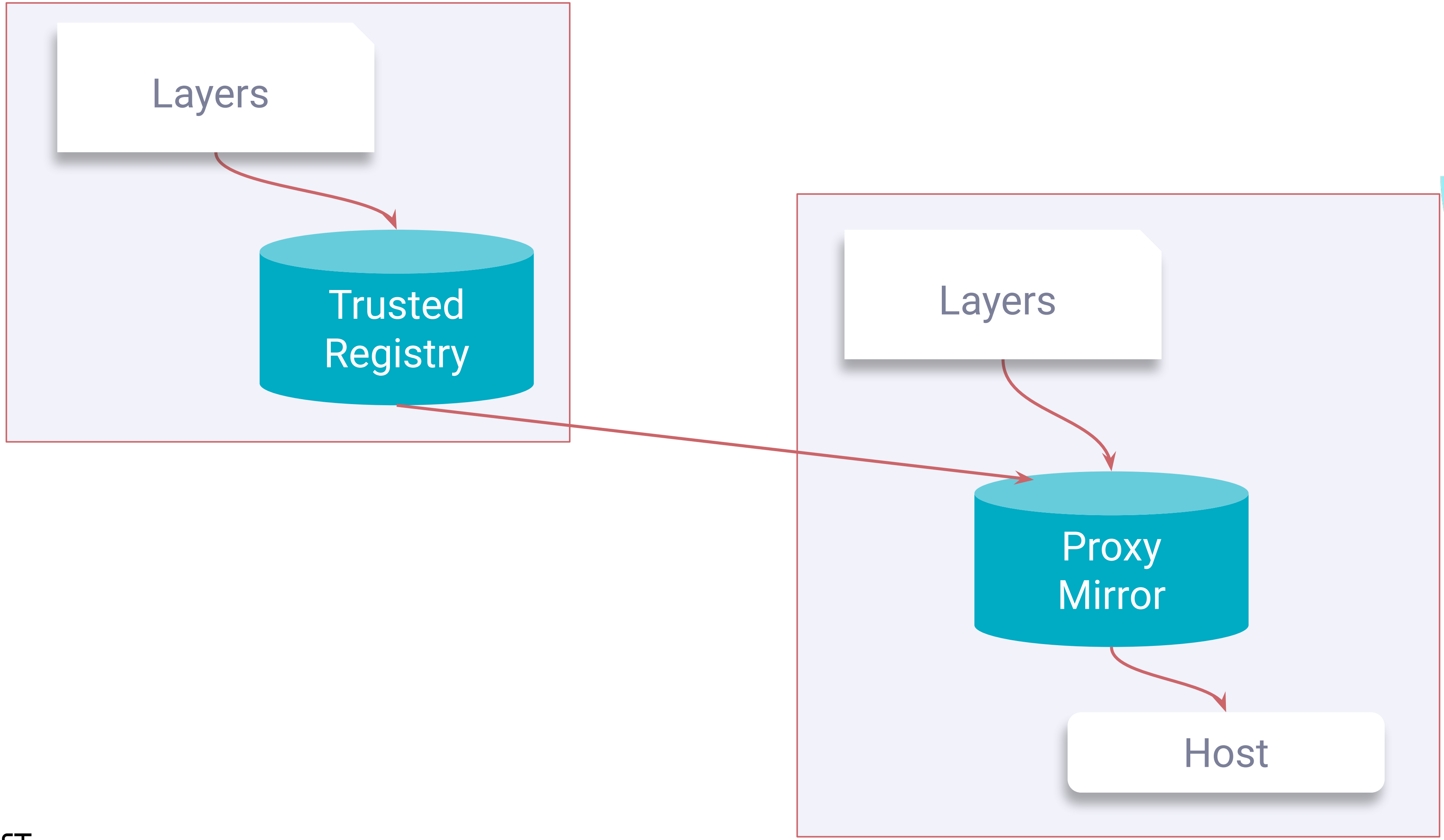


Deploy an image. Networks





Deploy an Image. Networks



Public Services

> ***On November 20, 2020, rate limits anonymous and free authenticated use of Docker Hub went into effect.***
>
> — Dockerhub

Deploy an Image. Not for free

- **Docker Hub Free**
  - Pull rate limits since Nov 2 2020 (200 rqs / 6 hrs)
- **Registry**
  - SaaS or 3rd party
  - Day $, GB $, GB*day $, GB out $$
- **Mirror**
  - A running instance $
  - Maintenance / SLAs $
  - Traffic

Deploy an Image. Not for free

- **Traffic**
  - No direct cost within VPC
  - Cross network, VPNs $$
  - Delays $
  - Machine time $
- **Time**
  - CPU time $
  - Deployment $$
  - Downtime $$$

Smaller containers can help

Images are transferred over the network across domains, so less traffic is cheaper. At the same time, every deployment will go faster.

The paid registry needs to contain less volume of data, and less data is transferred out.

She’s just bigger on the inside

Base Images

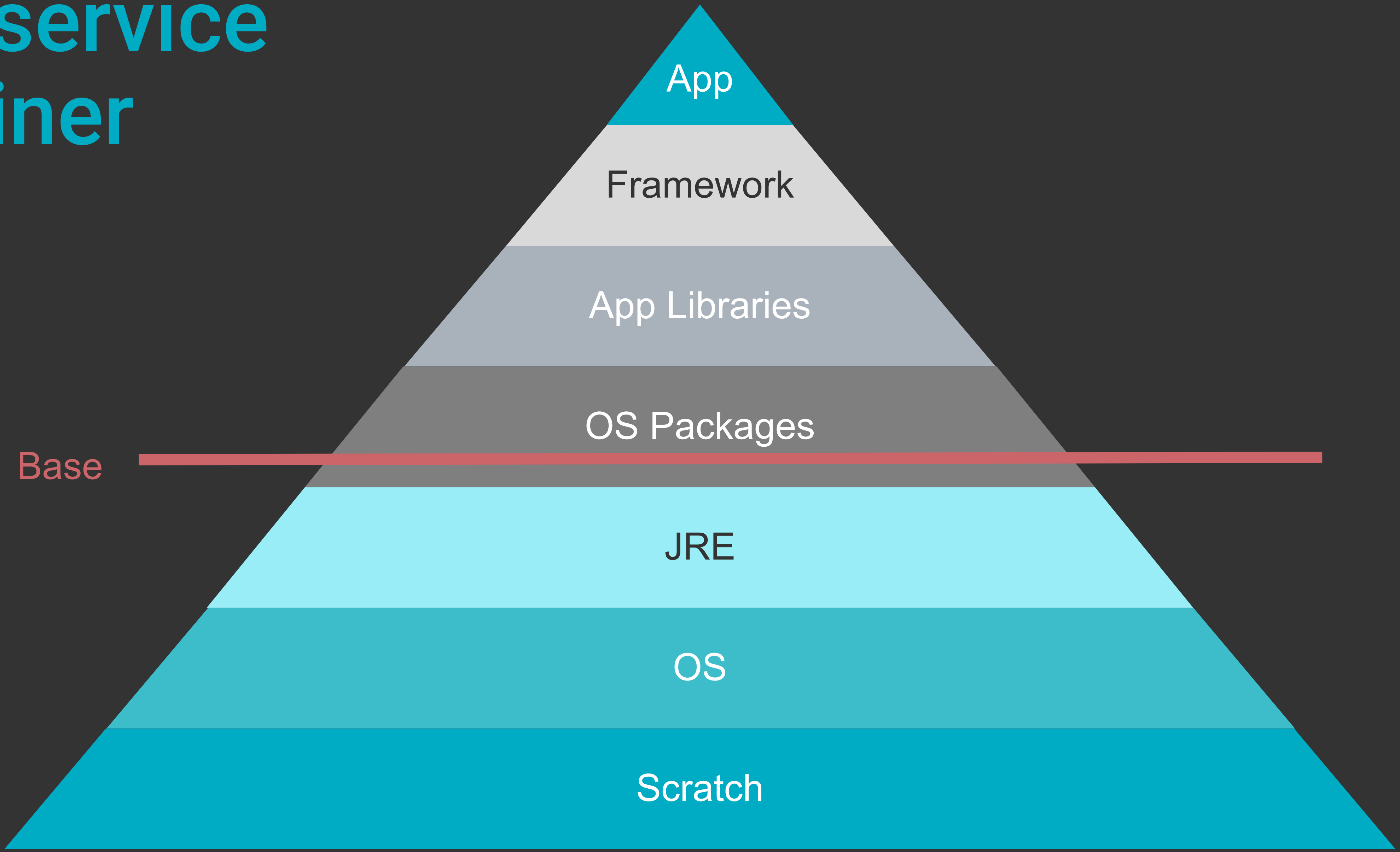
> *Most Dockerfiles start from a parent image.*
>
> — Docker

Base/Parent Images

A **base image** has **FROM scratch** in its Dockerfile.

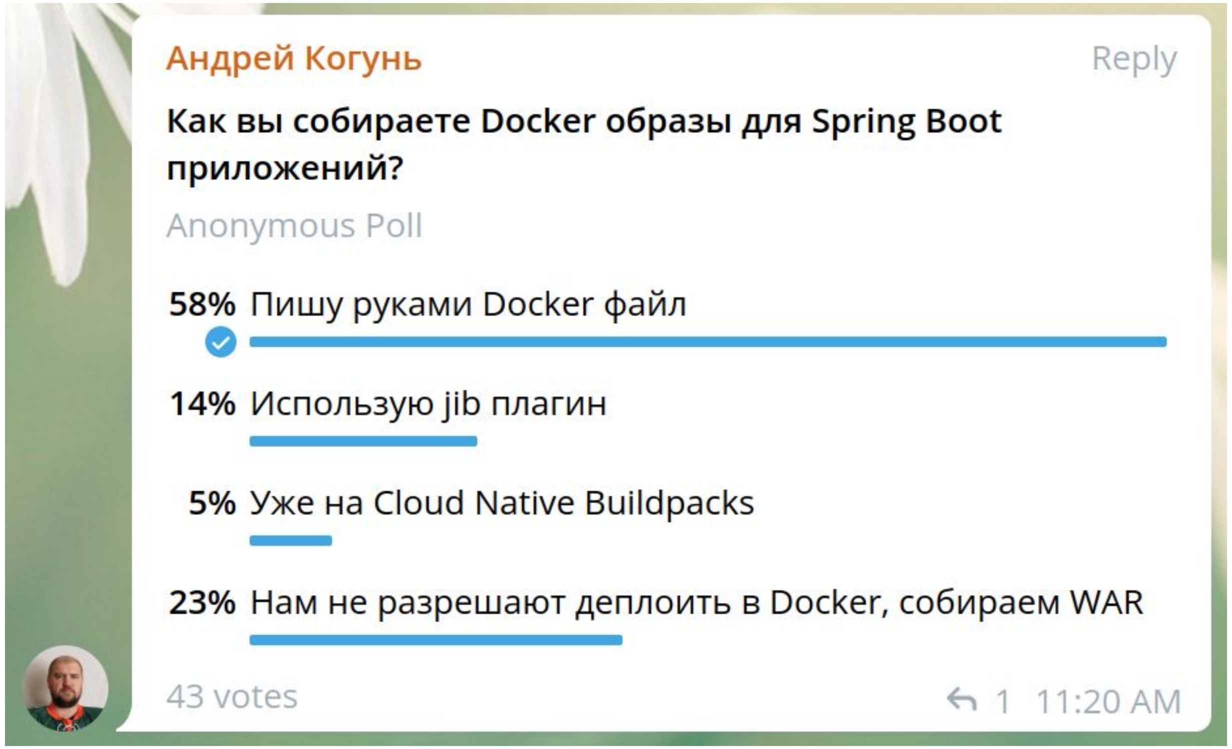
A **parent image** is the one that your image is based on. It refers to the contents of the FROM directive in the Dockerfile. Each subsequent declaration in the Dockerfile modifies this parent image. Most Dockerfiles start from a parent image rather than a base image. **However, the terms are sometimes used interchangeably**.

Microservice container layers



Developer voice

- **Aleksey Nesterov. Spring: Your next Java microframework**
- **Vladimir Plizga. Spring Boot "fat" JAR: Thin parts of a thick artifact**



Андрей Когунь

Как вы собираете Docker образы для Spring Boot приложений?

Anonymous Poll

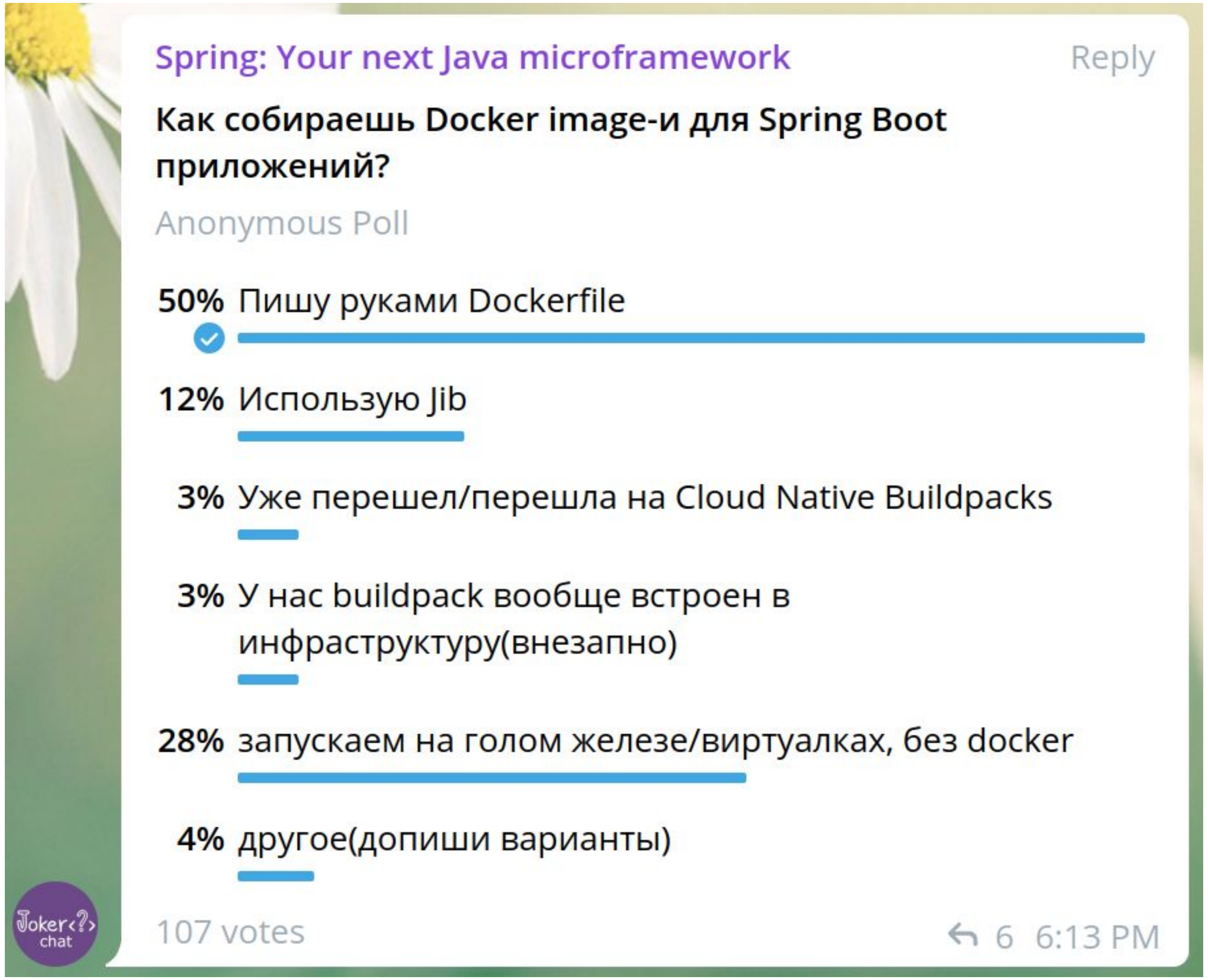
58% Пишу руками Docker файл ✓

14% Использую jib плагин

5% Уже на Cloud Native Buildpacks

23% Нам не разрешают деплоить в Docker, собираем WAR

43 votes



Spring: Your next Java microframework

Как собираешь Docker image-и для Spring Boot приложений?

Anonymous Poll

50% Пишу руками Dockerfile ✓

12% Использую Jib

3% Уже перешел/перешла на Cloud Native Buildpacks

3% У нас buildpack вообще встроен в инфраструктуру(внезапно)

28% запускаем на голом железе/виртуалках, без docker

4% другое(допиши варианты)

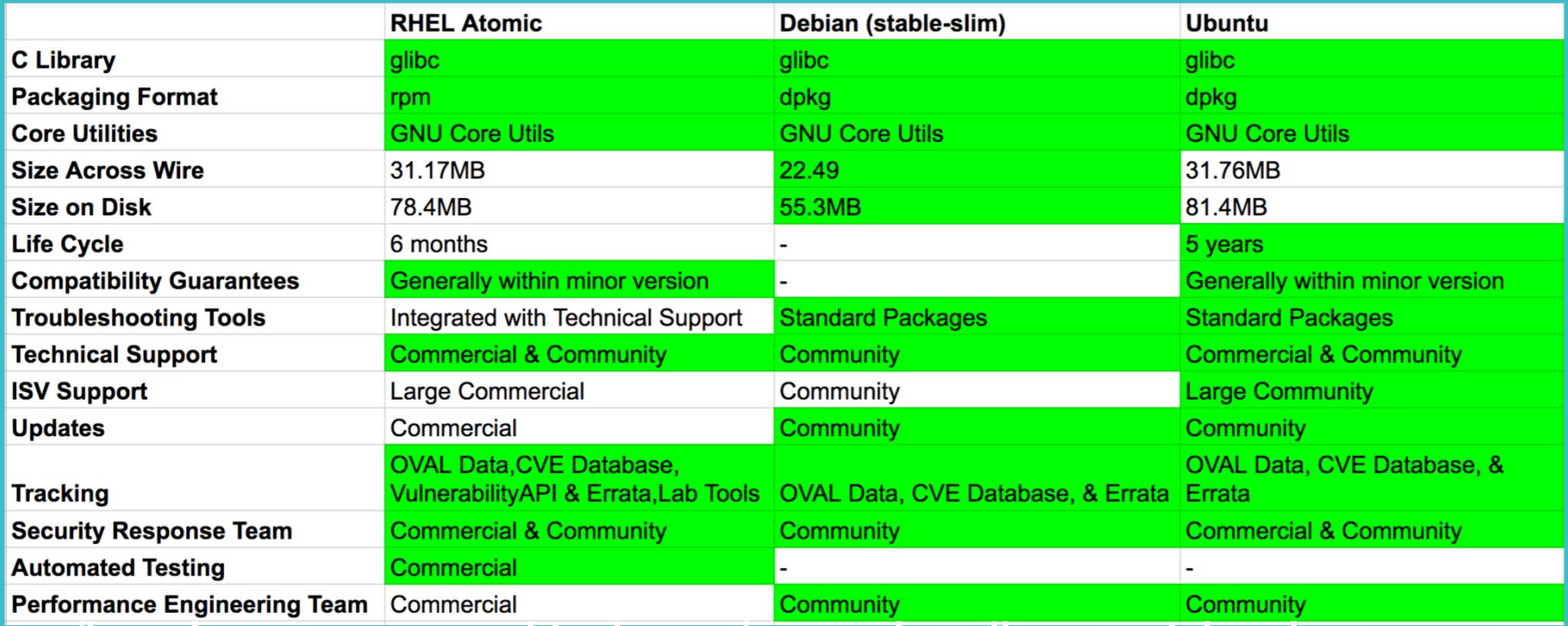
107 votes

Optimize Top

- **Select management system, use generic technics**
- **App**
  - Keep microservices micro
- **Framework & Libraries**
  - You can choose, smaller app = wider choice
  - Also affect app part (so keep it micro)
- **OS Packages**
  - Keep apps micro
  - Add minimal sufficient ones
  - Select OS

Optimize Base. Selection Criteria

- Correctness
- Security and updates
- Maintenance, tools and support
- Size
- Performance



	RHEL Atomic	Debian (stable-slim)	Ubuntu
C Library	glibc	glibc	glibc
Packaging Format	rpm	dpkg	dpkg
Core Utilities	GNU Core Utils	GNU Core Utils	GNU Core Utils
Size Across Wire	31.17MB	22.49	31.76MB
Size on Disk	78.4MB	55.3MB	81.4MB
Life Cycle	6 months	-	5 years
Compatibility Guarantees	Generally within minor version	-	Generally within minor version
Troubleshooting Tools	Integrated with Technical Support	Standard Packages	Standard Packages
Technical Support	Commercial & Community	Community	Commercial & Community
ISV Support	Large Commercial	Community	Large Community
Updates	Commercial	Community	Community
Tracking	OVAL Data,CVE Database, VulnerabilityAPI & Errata,Lab Tools	OVAL Data, CVE Database, & Errata	OVAL Data, CVE Database, & Errata
Security Response Team	Commercial & Community	Community	Commercial & Community
Automated Testing	Commercial	-	-
Performance Engineering Team	Commercial	Community	Community

kubedex.com/base-images crunchtools.com/comparison-linux-container-images

OS + JDK images

- **Based on OS images**
- **JDK package installation**
  - Package manager
  - Package
  - Same vendor
- **JDK binary installation**
  - Requirements
  - Compatibility
- **Ask your provider about testing**

Optimize Base. Size

- **Smaller JRE**
  - Lighter JVM type, proper JDK variant
  - Reduced set of modules, compressed modules
- **No JRE (compile app to native executable)**
  - Going beyond module granularity
  - Closed world
- **OS**
  - Small “OS” images
- **No OS (distroless)**
  - Actually “package manager”-less
- **Scratch only**
  - Only for simple programs

Compressed Size (across wire)

```
~~$ java -XX:+UnlockDiagnosticVMOptions …~~

$ vi ~/.docker/config.json

{
  "experimental": "enabled",
  "debug": true
}

$ docker manifest inspect -v openjdk
```

Compressed Size (across wire)

```
...
layers": [
    {
        "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
        "size": 42097812,
         "digest": "sha256:28587b6e64756a60a354301d011190fb69ea1eed25d7a5180811dab252e16a21"
    },
    {
        "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
        "size": 13511818,
        "digest": "sha256:b1655352c888337fbd3af21c25aa26d4561a0b9b6035a8b22c9ee0c8ae9a3784"
    },
    {
        "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
        "size": 187170401,
         "digest": "sha256:1f9646f00e96d7c315cc4e7a61c091ac85314d738b5c12f5a05566e9af31f65f"

    }
]
...
```

232 MB

Uncompressed Size (disk)

```
$ docker history openjdk
IMAGE                  CREATED                     CREATED BY                               SIZE
ff693b5bd1bb     5 weeks ago    /bin/sh -c #(nop)  CMD ["jshell"]                    0B
<missing>        5 weeks ago    /bin/sh -c set -eux;   arch="$(objdump="$(co…        321MB
<missing>        5 weeks ago    /bin/sh -c #(nop)  ENV JAVA_VERSION=17               0B
<missing>        5 weeks ago    /bin/sh -c #(nop)  ENV LANG=C.UTF-8                  0B
<missing>        5 weeks ago    /bin/sh -c #(nop)  ENV PATH=/usr/java/openjd…        0B
<missing>        5 weeks ago    /bin/sh -c #(nop)  ENV JAVA_HOME=/usr/java/o…        0B
<missing>        5 weeks ago    /bin/sh -c set -eux;  microdnf install   gzi…        39.3MB
<missing>        5 weeks ago    /bin/sh -c #(nop)  CMD ["/bin/bash"]                 0B
<missing>        5 weeks ago    /bin/sh -c #(nop) ADD file:ab982d6f37710c434…        111MB

$ docker images | head -n 1; docker images | grep openjdk
REPOSITORY               TAG                  IMAGE ID             CREATED             SIZE
openjdk                  latest               ff693b5bd1bb         5 weeks ago         471MB
```

471 MB

Pull time (100 Mbps)

```
$ time docker pull openjdk
...
real    0m27.990s
user    0m0.095s
sys 0m0.096s
```

28 s

Deployment costs per instance. Cloud

STORAGE

$0.09/mo

SSD

IOPS

Keep 471 MB

BANDWIDTH

$0*

***From internet**

***From same region**

Pull 232 MB

CPU

≈$0

Efficiently myocardinate
market-driven innovation via
open-source alignments.

Pull 28 s

REGISTRY

$0*

***Same region**.
Good rate limits
Push is cheap.

Keep & Seed

Clean deployment costs. Cloud

REGISTRY

$0.09

Different region.

Seed 251 MB

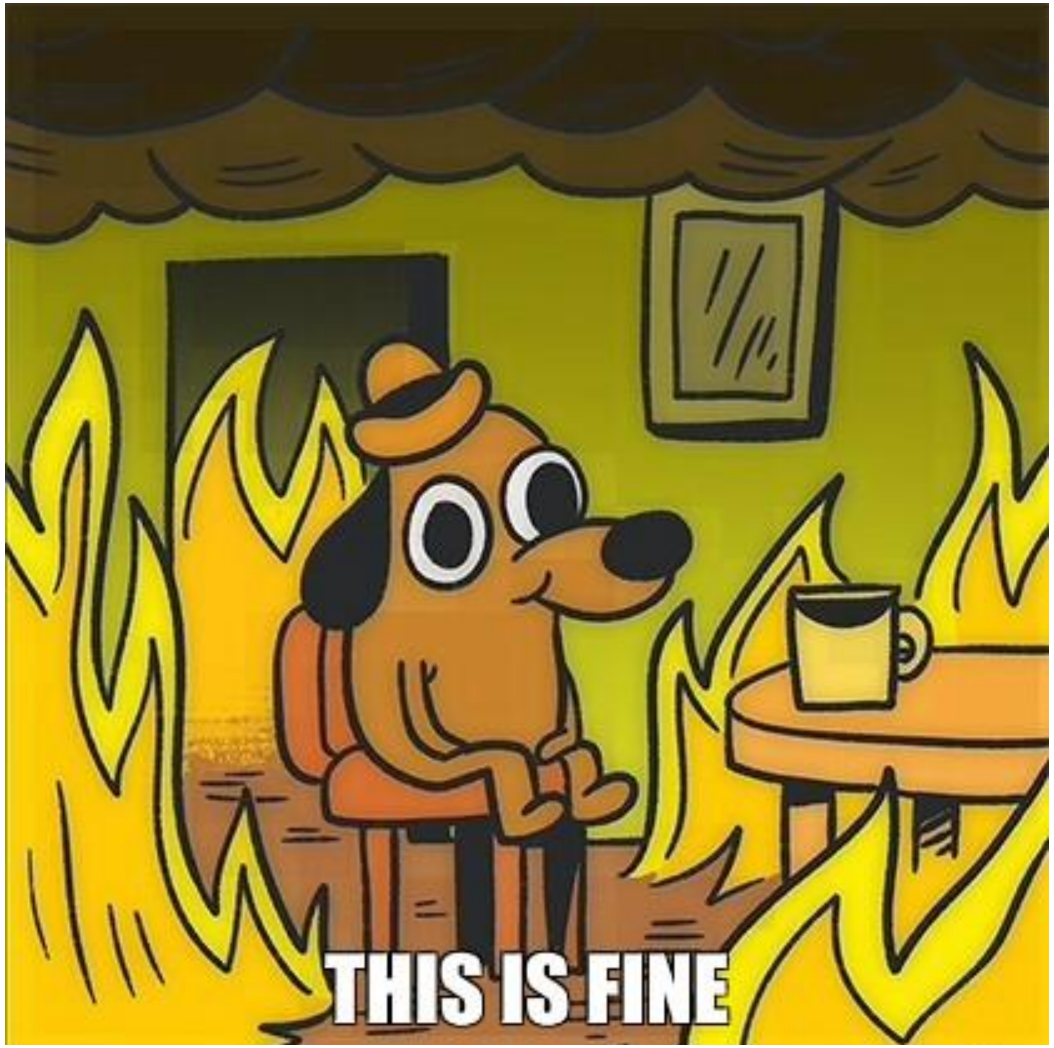
x 0.251 GB
x 1000 deploys
x 29.5 days

= $666

Deployment costs. Cloud

x 0.251 GB
x 1k deploys **= 0.25 TB**

- Tens of seconds for a single pull
- Shared HW
- Shared I/O limits
- Keep old versions
- On-premise / private cloud?
- Elastic fleet
- 10 Mbps



OS Image	Wire	Disk	libc	pkg man	shell
Ubuntu	27 MB	73 MB	glibc	apt	bash
Debian	48 MB	114 MB	glibc	apt	bash
Debian Slim	26 MB	69 MB	glibc	apt	bash
CenOS	71 MB	215 MB	glibc	yum	bash
RHEL Atomic Base	31 MB	78 MB	glibc	microdnf	bash
GCR Distroless base	7.6 MB	17 MB	glibc	—	—
Alpine	**2.7 MB**	**5.6 MB**	musl	apk	ash
GCR Distroless static	0.6 MB	1.8 MB	—	—	—

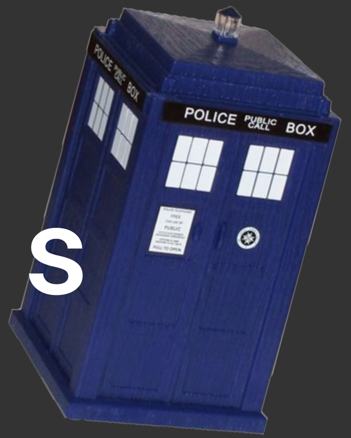
Liberica JDK Images

- hub.docker.com/r/bellsoft

OS + JDK 17 Image	Wire	Disk
bellsoft/liberica-openjdk-debian	131 MB	238 MB
bellsoft/liberica-openjdk-centos	147 MB	314 MB
bellsoft/liberica-openjdk-alpine	79 MB	124 MB
bellsoft/liberica-openjdk-alpine-musl	**71 MB**	**100 MB**

Pull time

```
$ time docker pull bellsoft/liberica-openjdk-alpine-musl:latest
...
real    0m3.957s
user    0m0.026s
sys 0m0.061s
```



4 s

Small containers do help

The amount of transferred data for OS+JDK image can be decreased to 76 MB, overall pull time drops many times (like 28 s → 4 s or 6 s → 0.8 s).

Image contents look unfamiliar.

Alpine Linux

> ***… is a security-oriented, lightweight Linux distribution based on musl libc and busybox.***
>
> **— Alpine**

Musl libc. At a glance

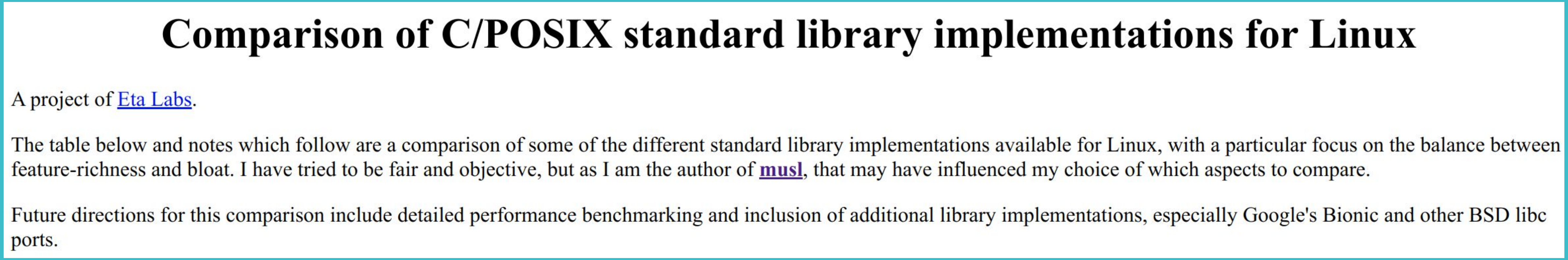
- musl.libc.org
- Built on top of Linux syscall API (C bindings for the OS interfaces)
- Base language standard (ISO C)
- POSIX + widely-agreed extensions
- Lightweight (size), fast, simple, free (MIT)
- Strives to be correct in the sense of standards-conformance and safety

Musl libc. Key Principles

- musl.libc.org/about.html
- **Simplicity**
  - Decoupling, minimize abstractions
  - Favors simple algorithms over more complex ones
  - Readable code
- **Resource efficiency**
  - Minimal size, low overhead, efficient static linking (Nx10kb)
  - Scalable (small stacks)
- **Attention to correctness**
  - Defensive coding, no race conditions
- **Ease of deployment (single binary)**
- **First-class support for UTF-8/multilingual text**

Libc implementations

- etalabs.net/compare_libcs.html
- **Note: outdated**



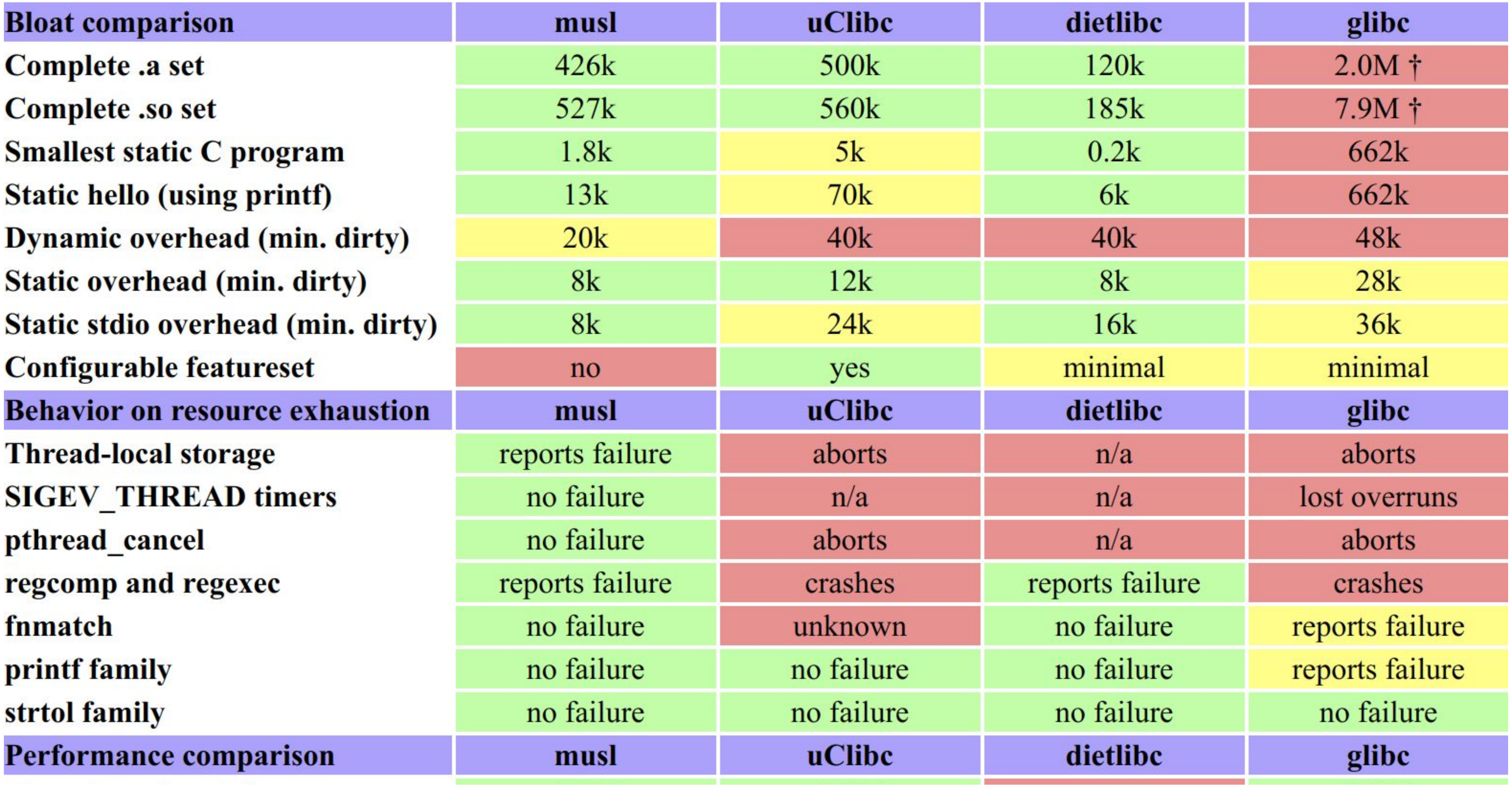
Comparison of C/POSIX standard library implementations for Linux

A project of Eta Labs.

The table below and notes which follow are a comparison of some of the different standard library implementations available for Linux, with a particular focus on the balance between feature-richness and bloat. I have tried to be fair and objective, but as I am the author of **musl**, that may have influenced my choice of which aspects to compare.

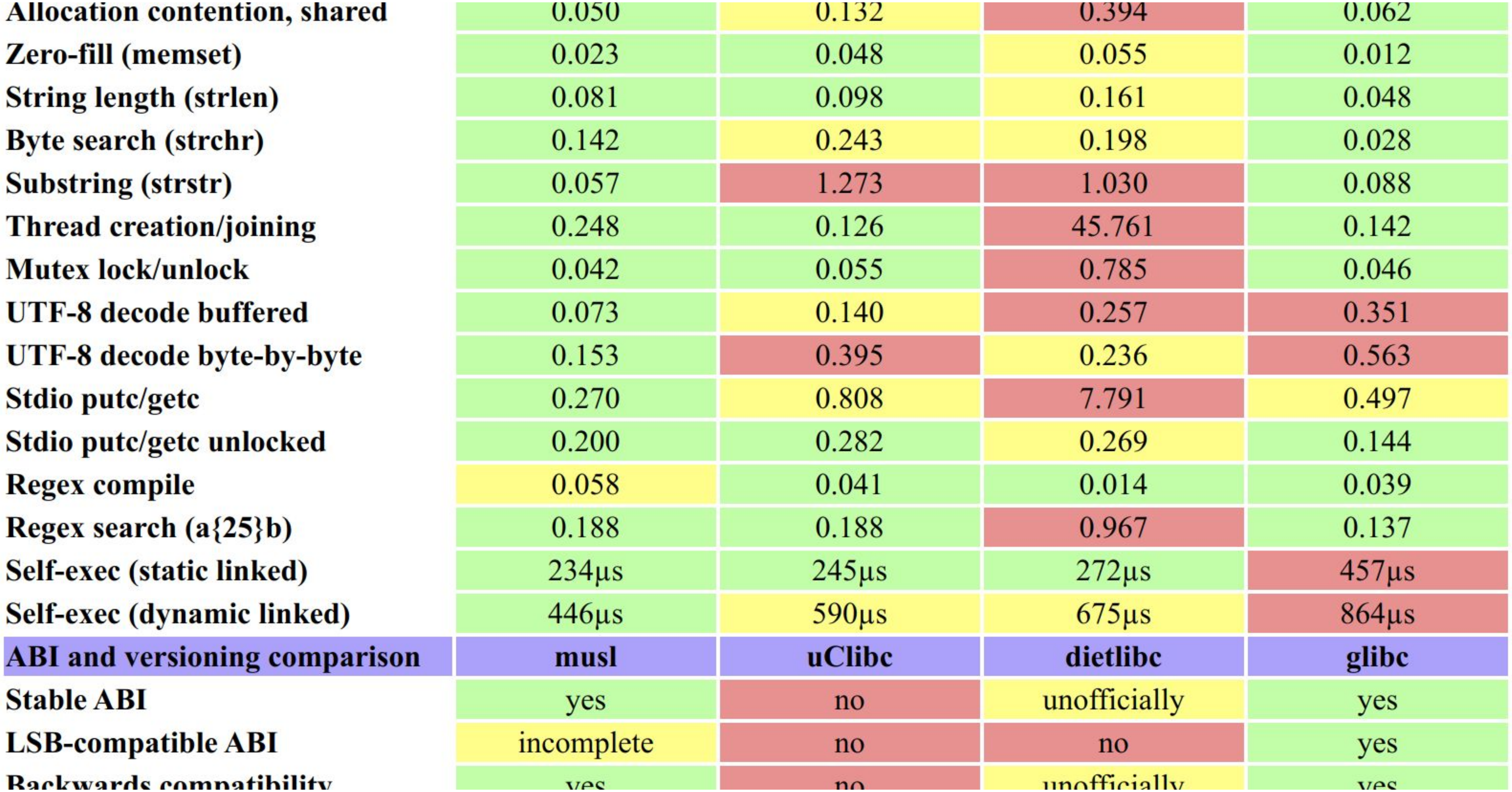
Future directions for this comparison include detailed performance benchmarking and inclusion of additional library implementations, especially Google's Bionic and other BSD libc ports.

Libc implementations



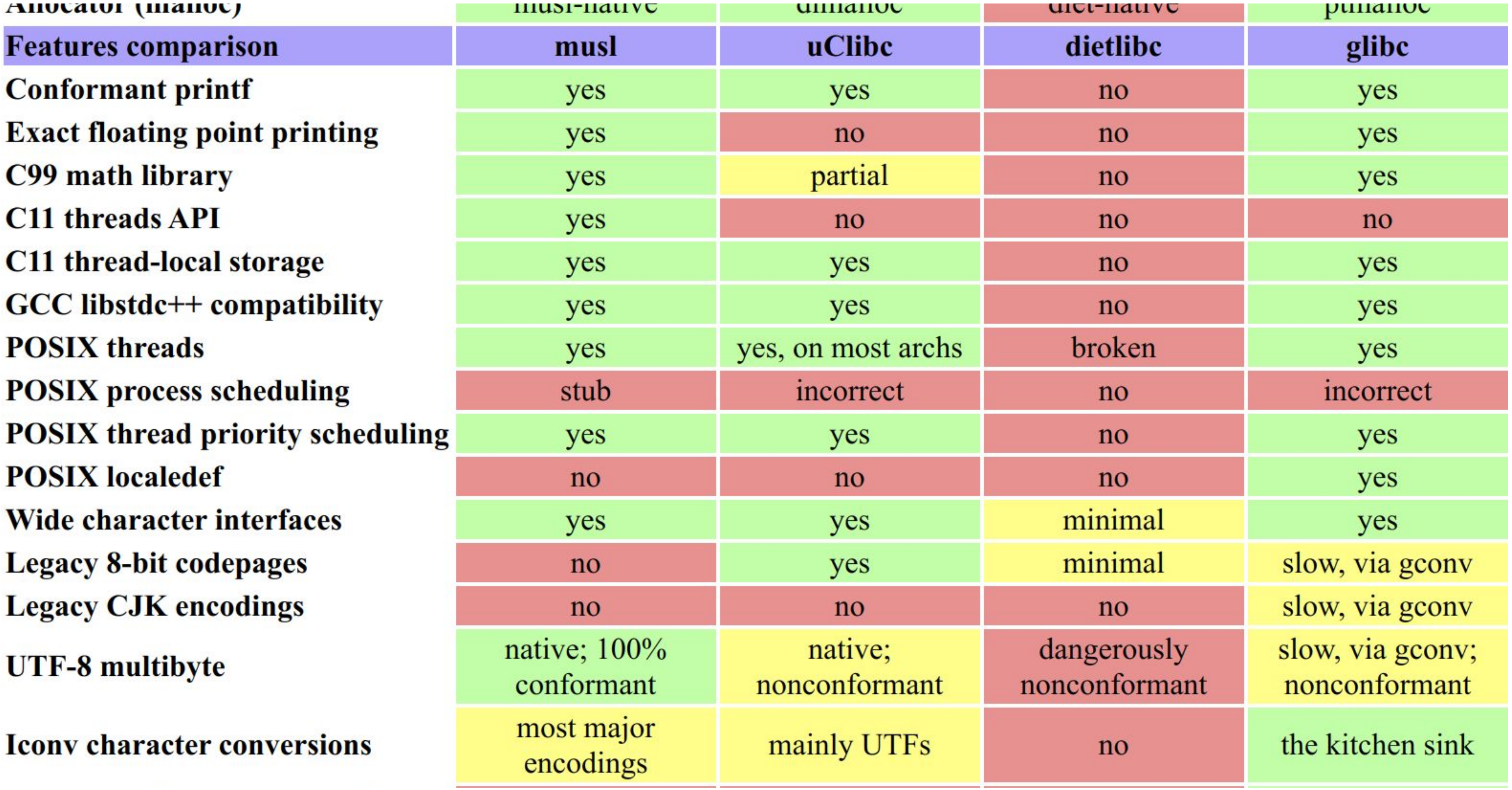
Bloat comparison	musl	uClibc	dietlibc	glibc
Complete .a set	426k	500k	120k	2.0M †
Complete .so set	527k	560k	185k	7.9M †
Smallest static C program	1.8k	5k	0.2k	662k
Static hello (using printf)	13k	70k	6k	662k
Dynamic overhead (min. dirty)	20k	40k	40k	48k
Static overhead (min. dirty)	8k	12k	8k	28k
Static stdio overhead (min. dirty)	8k	24k	16k	36k
Configurable featureset	no	yes	minimal	minimal
Behavior on resource exhaustion	**musl**	**uClibc**	**dietlibc**	**glibc**
Thread-local storage	reports failure	aborts	n/a	aborts
SIGEV_THREAD timers	no failure	n/a	n/a	lost overruns
pthread_cancel	no failure	aborts	n/a	aborts
regcomp and regexec	reports failure	crashes	reports failure	crashes
fnmatch	no failure	unknown	no failure	reports failure
printf family	no failure	no failure	no failure	reports failure
strtol family	no failure	no failure	no failure	no failure
Performance comparison	**musl**	**uClibc**	**dietlibc**	**glibc**

Libc implementations



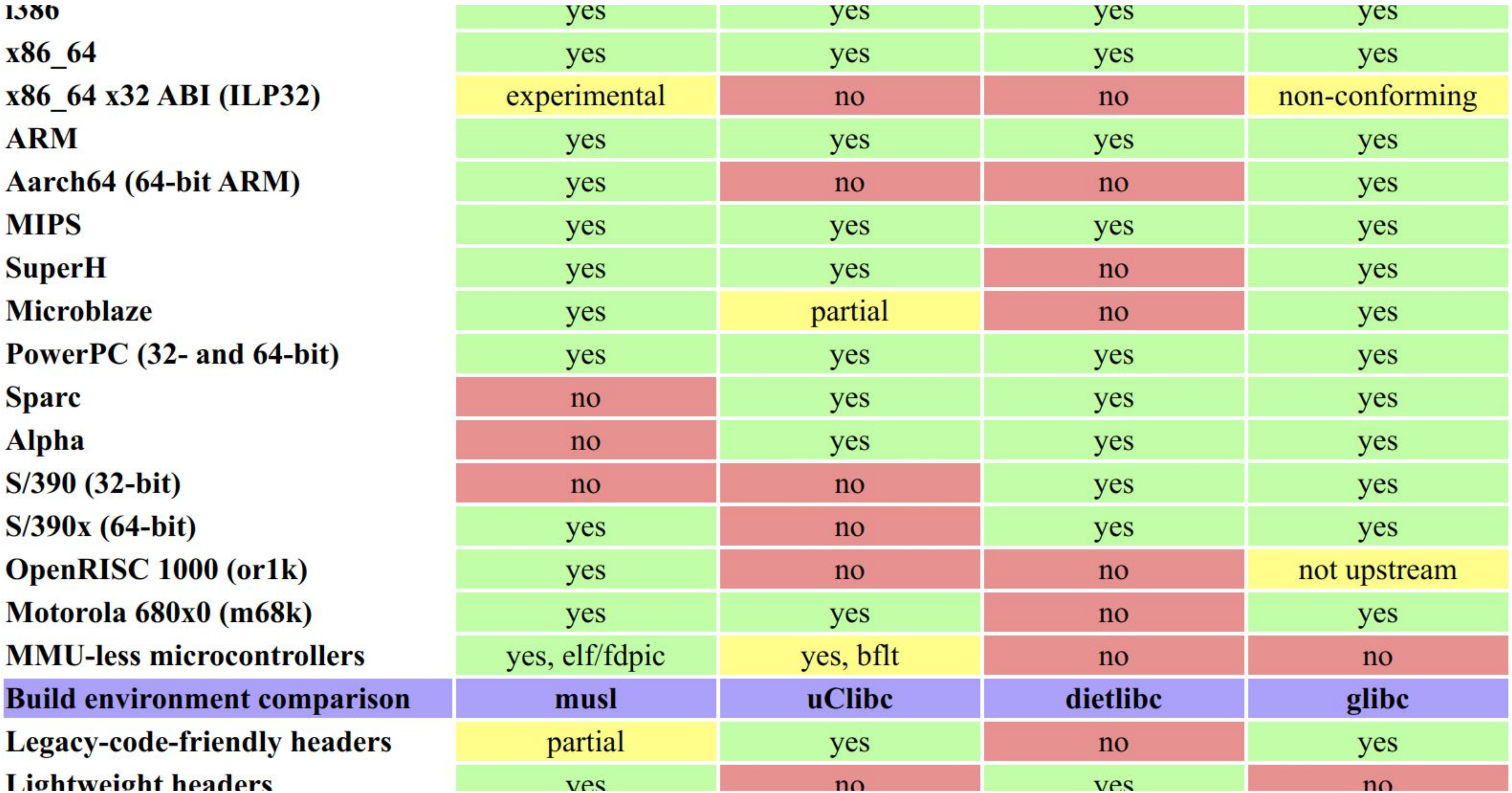
Allocation contention, shared	0.050	0.132	0.394	0.062
Zero-fill (memset)	0.023	0.048	0.055	0.012
String length (strlen)	0.081	0.098	0.161	0.048
Byte search (strchr)	0.142	0.243	0.198	0.028
Substring (strstr)	0.057	1.273	1.030	0.088
Thread creation/joining	0.248	0.126	45.761	0.142
Mutex lock/unlock	0.042	0.055	0.785	0.046
UTF-8 decode buffered	0.073	0.140	0.257	0.351
UTF-8 decode byte-by-byte	0.153	0.395	0.236	0.563
Stdio putc/getc	0.270	0.808	7.791	0.497
Stdio putc/getc unlocked	0.200	0.282	0.269	0.144
Regex compile	0.058	0.041	0.014	0.039
Regex search (a{25}b)	0.188	0.188	0.967	0.137
Self-exec (static linked)	234µs	245µs	272µs	457µs
Self-exec (dynamic linked)	446µs	590µs	675µs	864µs
ABI and versioning comparison	**musl**	**uClibc**	**dietlibc**	**glibc**
Stable ABI	yes	no	unofficially	yes
LSB-compatible ABI	incomplete	no	no	yes
Backwards compatibility	yes	no	unofficially	yes

Libc implementations



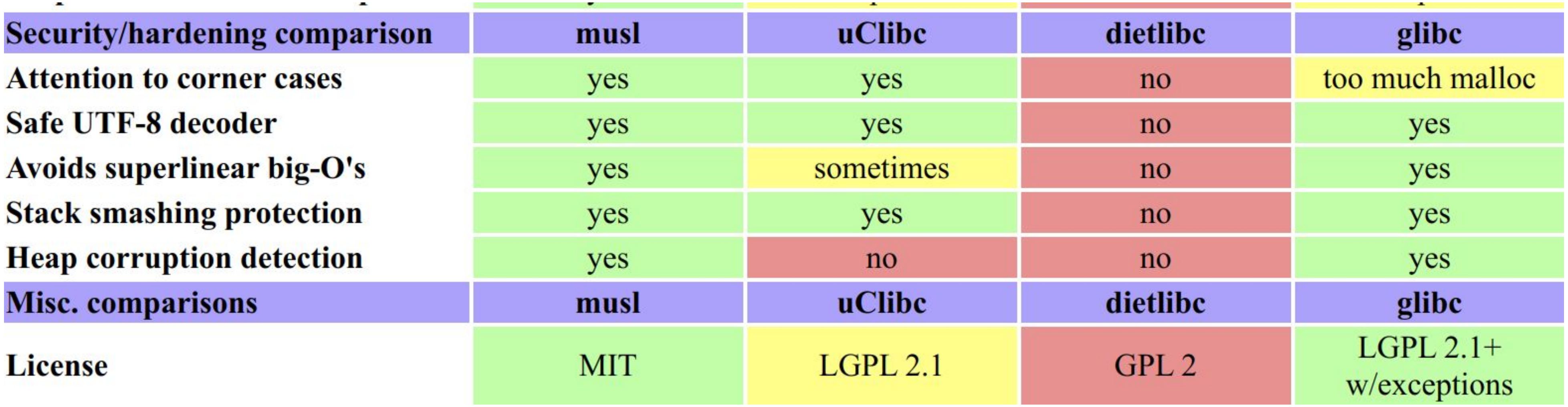
Allocator (malloc)	musl-native	dlmalloc	diet-native	ptmalloc
Features comparison	**musl**	**uClibc**	**dietlibc**	**glibc**
Conformant printf	yes	yes	no	yes
Exact floating point printing	yes	no	no	yes
C99 math library	yes	partial	no	yes
C11 threads API	yes	no	no	no
C11 thread-local storage	yes	yes	no	yes
GCC libstdc++ compatibility	yes	yes	no	yes
POSIX threads	yes	yes, on most archs	broken	yes
POSIX process scheduling	stub	incorrect	no	incorrect
POSIX thread priority scheduling	yes	yes	no	yes
POSIX localedef	no	no	no	yes
Wide character interfaces	yes	yes	minimal	yes
Legacy 8-bit codepages	no	yes	minimal	slow, via gconv
Legacy CJK encodings	no	no	no	slow, via gconv
UTF-8 multibyte	native; 100% conformant	native; nonconformant	dangerously nonconformant	slow, via gconv; nonconformant
Iconv character conversions	most major encodings	mainly UTFs	no	the kitchen sink

Libc implementations



i386	yes	yes	yes	yes
x86_64	yes	yes	yes	yes
x86_64 x32 ABI (ILP32)	experimental	no	no	non-conforming
ARM	yes	yes	yes	yes
Aarch64 (64-bit ARM)	yes	no	no	yes
MIPS	yes	yes	yes	yes
SuperH	yes	yes	no	yes
Microblaze	yes	partial	no	yes
PowerPC (32- and 64-bit)	yes	yes	yes	yes
Sparc	no	yes	yes	yes
Alpha	no	yes	yes	yes
S/390 (32-bit)	no	no	yes	yes
S/390x (64-bit)	yes	no	yes	yes
OpenRISC 1000 (or1k)	yes	no	no	not upstream
Motorola 680x0 (m68k)	yes	yes	no	yes
MMU-less microcontrollers	yes, elf/fdpic	yes, bflt	no	no
Build environment comparison	**musl**	**uClibc**	**dietlibc**	**glibc**
Legacy-code-friendly headers	partial	yes	no	yes
Lightweight headers	yes	no	yes	no

Libc implementations



Security/hardening comparison	musl	uClibc	dietlibc	glibc
Attention to corner cases	yes	yes	no	too much malloc
Safe UTF-8 decoder	yes	yes	no	yes
Avoids superlinear big-O's	yes	sometimes	no	yes
Stack smashing protection	yes	yes	no	yes
Heap corruption detection	yes	no	no	yes
Misc. comparisons	**musl**	**uClibc**	**dietlibc**	**glibc**
License	MIT	LGPL 2.1	GPL 2	LGPL 2.1+ w/exceptions

Musl libc. Key Issues

- **It’s different**

```
$ cat src/hotspot/os/linux/os_linux.cpp

…
# include <stdio.h>
# include <unistd.h>
…
```

Busybox. At a glance

- busybox.net
- **Many Unix utilities in a single executable file**
  - i.e. shell commands and the shell itself
- **Glibc, musl (Alpine), uLibc**
- **GPLv2**
- hub.docker.com/_/busybox

Busybox. Key Principles

- **Swiss army knife, small**
- **Implementation of the standard Linux command line tools**
- **Smallest executable size**
- **Simplest and cleanest implementation**
- **Standards compliant**
- **Minimal run-time memory usage (heap and stack)**
- **Fast**

Busybox. Key Issues

- **It’s different**
- **Single executable**
  - Process binary path
  - Non-modular binary
- **Doesn’t support environment variables with periods in the names**
  - POSIX compliant

Alpine Linux. At a glance

- alpinelinux.org
- **Small**
  - Built around musl libc and busybox
  - Small packages
- **Simple**
  - OpenRC init system
  - apk package manager
- **Secure**
  - Position Independent Executables (PIE) binaries with stack smashing protection

Alpine Linux. Key Issues

- **It’s different**
- **Not desktop-oriented**
- **Package repository**

Alpine Linux is perfect for containers

It is small and secure. All necessary tools are available out of the box or in packages.

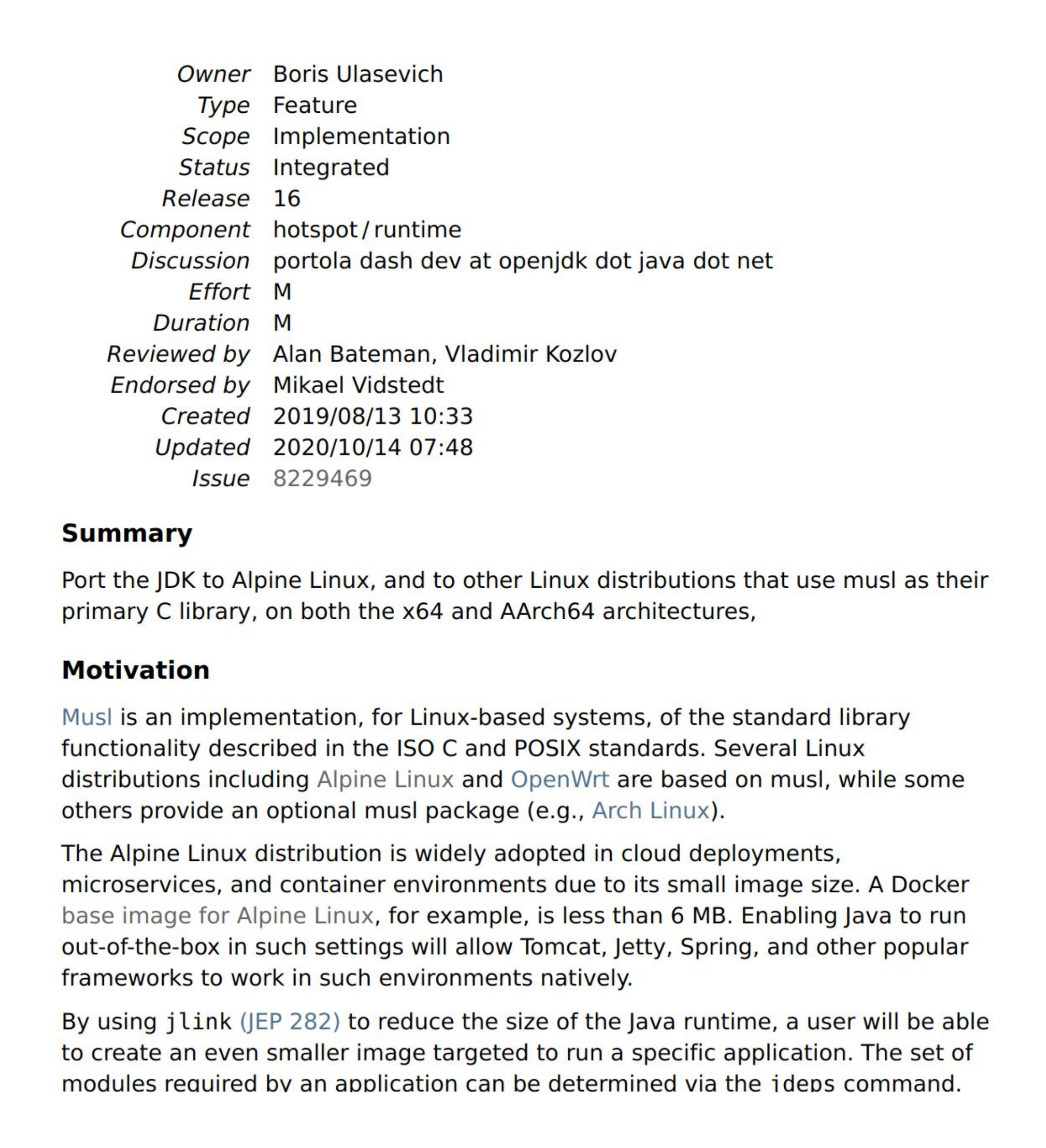
Alpine containers with Java work.

Alpine Linux Port

> *Port the JDK to Alpine Linux, and to other Linux distributions that use musl as their primary C library, on both the x64 and AArch64 architectures.*
>
> — JEP 386

JDK 16

- **JEP 386: Alpine Linux Port**
- openjdk.java.net/jeps/386



Owner	Boris Ulasevich
Type	Feature
Scope	Implementation
Status	Integrated
Release	16
Component	hotspot / runtime
Discussion	portola dash dev at openjdk dot java dot net
Effort	M
Duration	M
Reviewed by	Alan Bateman, Vladimir Kozlov
Endorsed by	Mikael Vidstedt
Created	2019/08/13 10:33
Updated	2020/10/14 07:48
Issue	8229469

Summary

Port the JDK to Alpine Linux, and to other Linux distributions that use musl as their primary C library, on both the x64 and AArch64 architectures,

Motivation

Musl is an implementation, for Linux-based systems, of the standard library functionality described in the ISO C and POSIX standards. Several Linux distributions including Alpine Linux and OpenWrt are based on musl, while some others provide an optional musl package (e.g., Arch Linux).

The Alpine Linux distribution is widely adopted in cloud deployments, microservices, and container environments due to its small image size. A Docker base image for Alpine Linux, for example, is less than 6 MB. Enabling Java to run out-of-the-box in such settings will allow Tomcat, Jetty, Spring, and other popular frameworks to work in such environments natively.

By using `jlink` (JEP 282) to reduce the size of the Java runtime, a user will be able to create an even smaller image targeted to run a specific application. The set of modules required by an application can be determined via the jdeps command.

Project Portola

- openjdk.java.net/projects/portola
- **Port of the JDK to the Alpine Linux distribution, and in particular the musl C library**
- **Started by Mikael Vidstedt from Oracle in 2017**
- **Used for Alpine musl containers with JDK 9+**
- **Integrated into mainline in 2020 with JEP 386**
  - Delivered by BellSoft
  - JDK 16

Project Portola. Build

- **A new port**
  - Determine and distinguish C libraries
  - Conditional compilation
- **Native build**
- **Cross-toolchain for glibc environment**
- **Implement missing functions or make them compatible**
- **Testing environment**
- **Documentation**
  - https://github.com/openjdk/jdk/blob/master/doc/building.md#building-for-musl

JNI. Build

```
$ gcc -std=c99 -I"$JAVA_HOME/include" -I"$JAVA_HOME/include/linux" -shared -o
libhelloworld.so -fPIC JNIHelloWorld.c

16K libhelloworld.so

$ java -Djava.library.path=. JNIHelloWorld

Hello world!


$ docker run -it -v ~/jni:/jni bellsoft/liberica-openjdk-alpine:15 java
-Djava.library.path=/jni -cp /jni JNIHelloWorld

Hello world!


$ docker run -it -v ~/jni:/jni bellsoft/liberica-openjdk-alpine-musl:15 java
-Djava.library.path=/jni -cp /jni JNIHelloWorld

Hello world!
```

JNI. Cross Build

```
$ x86_64-linux-musl-cross/bin/x86_64-linux-musl-gcc -std=c99 -I"$JAVA_HOME/include"
-I"$JAVA_HOME/include/linux" -shared -o libhelloworld.so -fPIC JNIHelloWorld.c

7.7K libhelloworld.so
```

```
$ docker run -it -v ~/jni:/jni bellsoft/liberica-openjdk-alpine-musl:15 java
-Djava.library.path=/jni -cp /jni JNIHelloWorld

Hello world!
```

```
$ docker run -it -v ~/jni:/jni bellsoft/liberica-openjdk-alpine:15 java
-Djava.library.path=/jni -cp /jni JNIHelloWorld

Hello world!
```

```
$ java -Djava.library.path=. JNIHelloWorld

Exception in thread "main" java.lang.UnsatisfiedLinkError: /home/tp/jni/libhelloworld.so:
/usr/lib/x86_64-linux-gnu/libc.so: invalid ELF header
```

Project Portola. Issues

- **LD_PRELOAD is not the same on different platforms**
  - Glibc resolves libs not like musl (or AIX libc)
  - jpackage and other launchers were fixed to still use proper JDK libs
- **Alpine used to have PaX/grsecurity in kernel by default**
  - Attempt to execute JIT code shut down the JVM
  - Added Memory protection check on startup
- **JDWP (Debug) sometimes had troubles with IPv4/IPv6 config**
  - Initialization was made more careful
- **Debugging (gdb)**
  - There's SIGSYNCCALL during JVM init
  - Debug with -XX:-MaxFDLimit

Project Portola. Issues

- **Running AWT in headless mode**
  - You may want to render images
  - Install freetype and fonts
- **Fontmanager**
  - For all real cases load awt lib before fontmanager
- **NMT**
  - Use latest Alpine (3.11+)
- **NUMA detection requires recent libnuma**
  - apk add numactl

Project Portola. Issues

- **lsof does not support ‘-p’ option on busybox**
  - Expect reduced output
- **Musl does not execute scripts that does not have a proper shebang**
  - Write proper # headers in *.sh
  - https://www.openwall.com/lists/musl/2020/02/13/4
- **Serviceability agent (private API) doesn’t work**

Shebang in scripts

```
$ docker run -it bellsoft/liberica-openjdk-alpine-musl:15 ash

-rwxr-xr-x    run.sh

echo "hello"

jshell> Runtime.getRuntime().exec("./run.sh")
|  Exception java.io.IOException: Cannot run program "./run.sh": error=8, Exec format error
```

```
-rwxr-xr-x    run.sh

#!/bin/sh
echo "hello"

jshell> Runtime.getRuntime().exec("./run.sh")
$1 ==> Process[pid=262, exitValue=0]
```

Variables with dots

```
$ docker run -it -e "hibernate.format_sql=true" bellsoft/liberica-openjdk-alpine:15 ash

# set | grep hibernate

hibernate


$ docker run -it -e "hibernate.format_sql=true" bellsoft/liberica-openjdk-debian:15 bash

# set | grep hibernate

<empty>

$ docker run -it -e "hibernate_format_sql=true" bellsoft/liberica-openjdk-alpine-musl:15 ash

# set | grep hibernate

hibernate_format_sql='true'
```

Serviceability Agent

```
$ docker run -it bellsoft/liberica-openjdk-alpine:8 jstack -h
...
Options:
    -F  to force a thread dump. Use when jstack <pid> does not respond (process is hung)
    -m  to print both java and native frames (mixed mode)
    -l  long listing. Prints additional information about locks
    -h or -help to print this help message

$ docker run -it bellsoft/liberica-openjdk-alpine-musl:8 jstack -h
...
Options:
    -l  long listing. Prints additional information about locks
    -h or -help to print this help message


$ docker run -it bellsoft/liberica-openjdk-debian:11 jstack -h
...
Options:
    -l  long listing. Prints additional information about locks
    -h or -help to print this help message
```

Alpine Linux port in upstream

Unifies platform support across community and distributions. Helps maintenance and port development for perfect small containers. Liberica JDK Alpine musl containers are tested and TCK-verified.

Different uses are possible.

She can be seen in various forms

Make More Users Happy

> ***We plan to stay on Java 8.***
>
> ~50% of users

Portola Expansion

- **JDK 11 LTS**
  - Not in mainline (yet)
  - Historical downports in Liberica 9+
  - Liberica 11u on Dockerhub
- **JDK 8 LTS**
  - Liberica 8u on Dockerhub
- **AArch64**
- **OpenWRT**

Linux @bellsoftware

- **Acquired ex-Oracle Linux engineering best talent**
- **Fixed some kernel and libc bugs as part of Liberica JDK support**
- **OSS Contributions**
  - MUSL support for LTP project, ~100 patches
  - MUSL support for OpenJDK
  - MUSL support for GraalVM

Alpaca Linux

> “*...is the operating system optimized for Java deployment, emphasizing high performance, security, small size, and flexibility.*”
>
> — BellSoft

Top 4 features of Alpaca Linux

Enhanced security

The lack of extra components means it is harder to break, and timely, frequent updates reliably remove the vulnerabilities. As a bonus, an additional security hardening is provided by userspace compilation options.

Optimized performance

Alpaca’s features include tuned kernel, optimized libc, and optimized malloc options to boost the performance of your applications without sacrificing stability.

Miniature size

With its 2.9 Mb base image size, Alpaca offers the smallest performant docker images, JDK docker images, and native images, making the deployment faster and memory footprint lesser.

Liberica Lite and Liberica NIK

Liberica Lite, the optimized version of Liberica JDK, enhances the performance and minimizes memory footprint. Liberica NIK allows creating the native images that benefit even more with Alpaca Linux as the foundation.

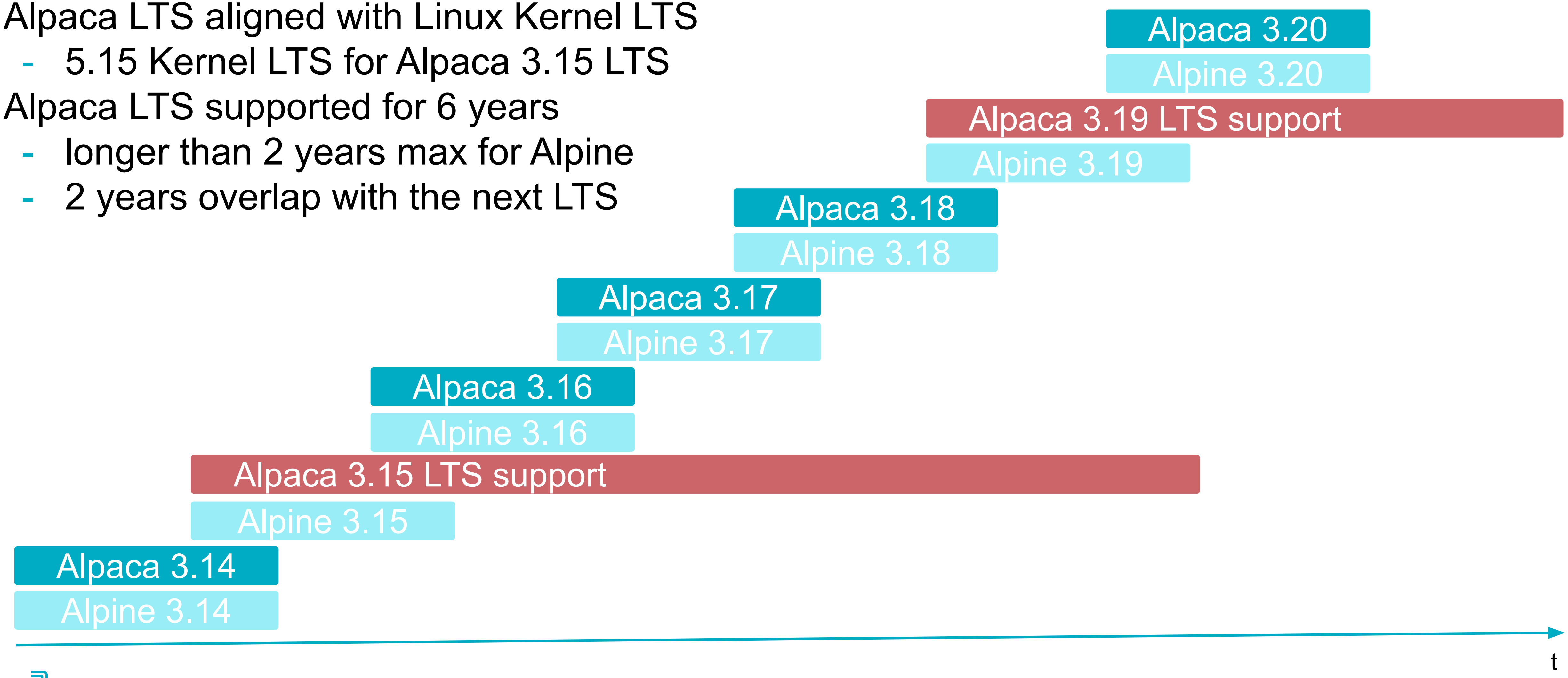
Support cycle

Alpaca LTS aligned with Linux Kernel LTS

- 5.15 Kernel LTS for Alpaca 3.15 LTS

Alpaca LTS supported for 6 years

- longer than 2 years max for Alpine
- 2 years overlap with the next LTS



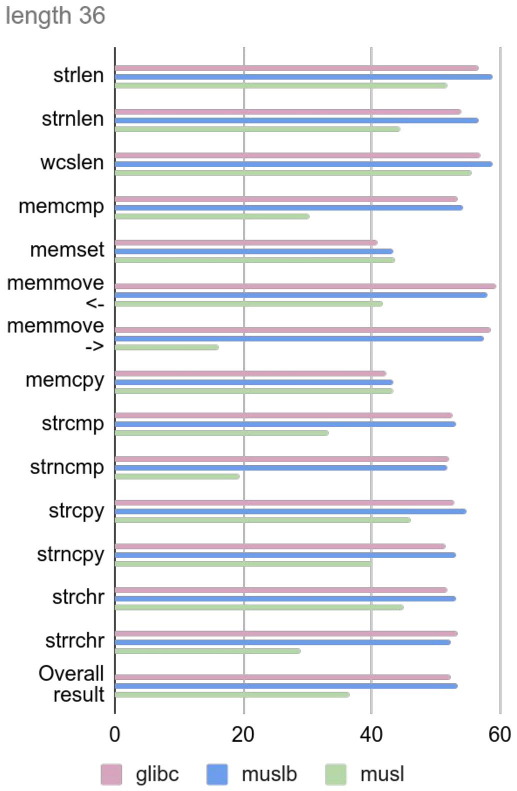
Alpaca. Addressing CVEs

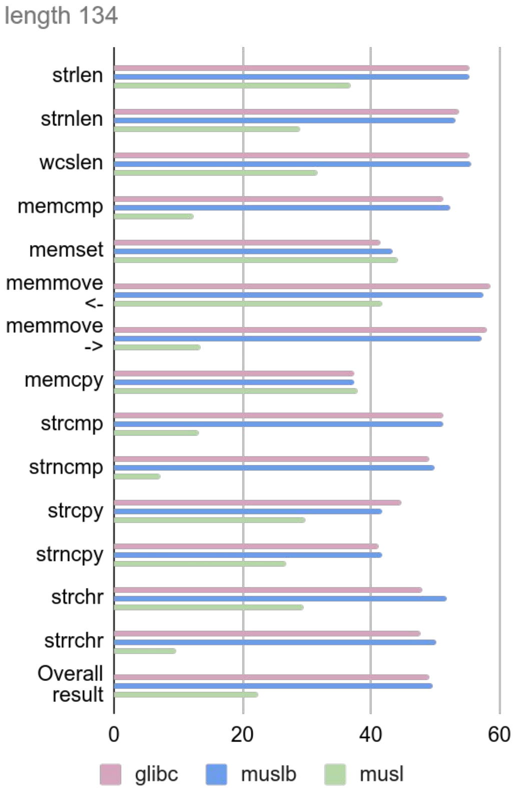
- BellSoft Security Advisory (being built)
  - Full open listings of addressed CVEs and affected components
  - OpenJDK/Liberica JDK security advisory
  - Tooling and scanning to support the efforts
    - Internal CVE DB checking against MITRE/NIST
    - CVE checker
    - Static code and binary analysis tools

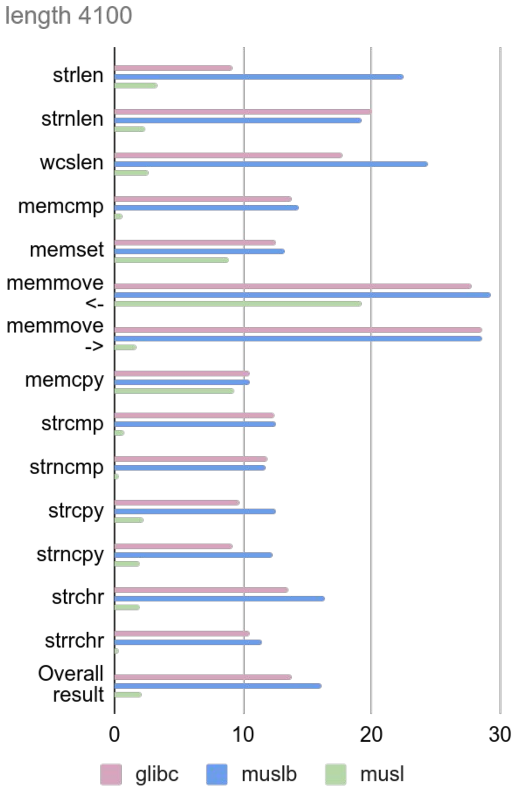
Alpaca. Optimized musl performance

Tests: basic functional string tests with 1 million iterations.
Results are shown in relative avg speed, 1K/ns.
Machine: bare-metal, Intel Core i5-6600 CPU 3.30GHz.

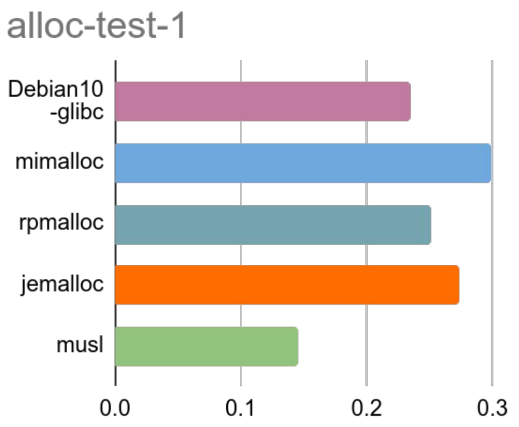
Higher is better:
GLIBC: Debian 10.11 2.28
MUSL-perf: Alpaca musl-1.2.2_p-r1803
MUSL: Alpaca musl-1.2.2-r3

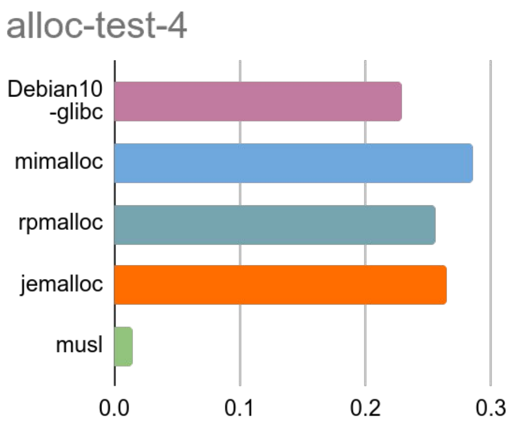




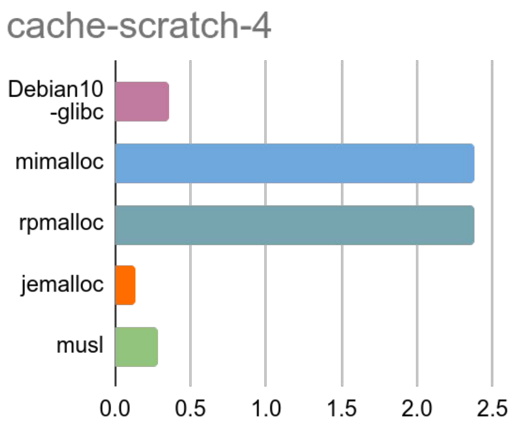


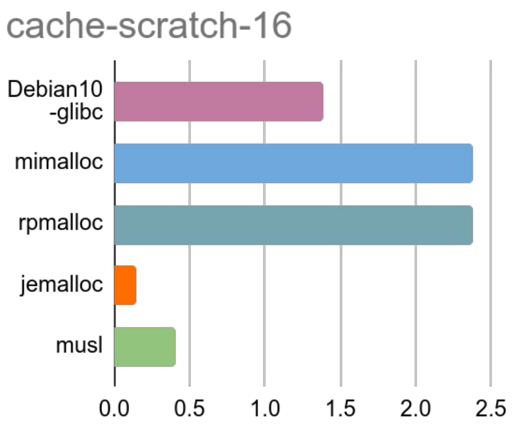
Alpaca. Extra mallocs performance

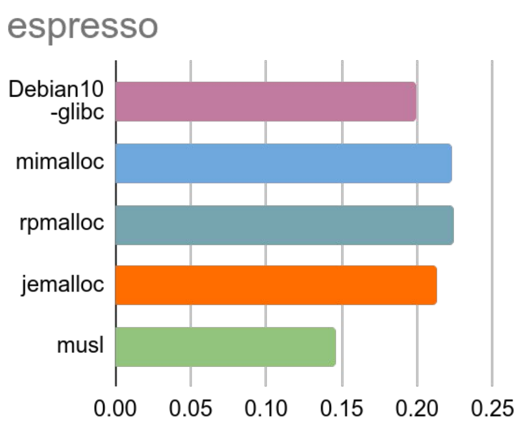




- **alloc-test**: simulates intensive allocation workloads with a Pareto distribution.
- **cache-scratch**: introduced with the Hoard allocator to test for passive-false sharing of cache lines.
- **espresso**: a programmable logic array analyzer in the context of cache aware memory allocation.



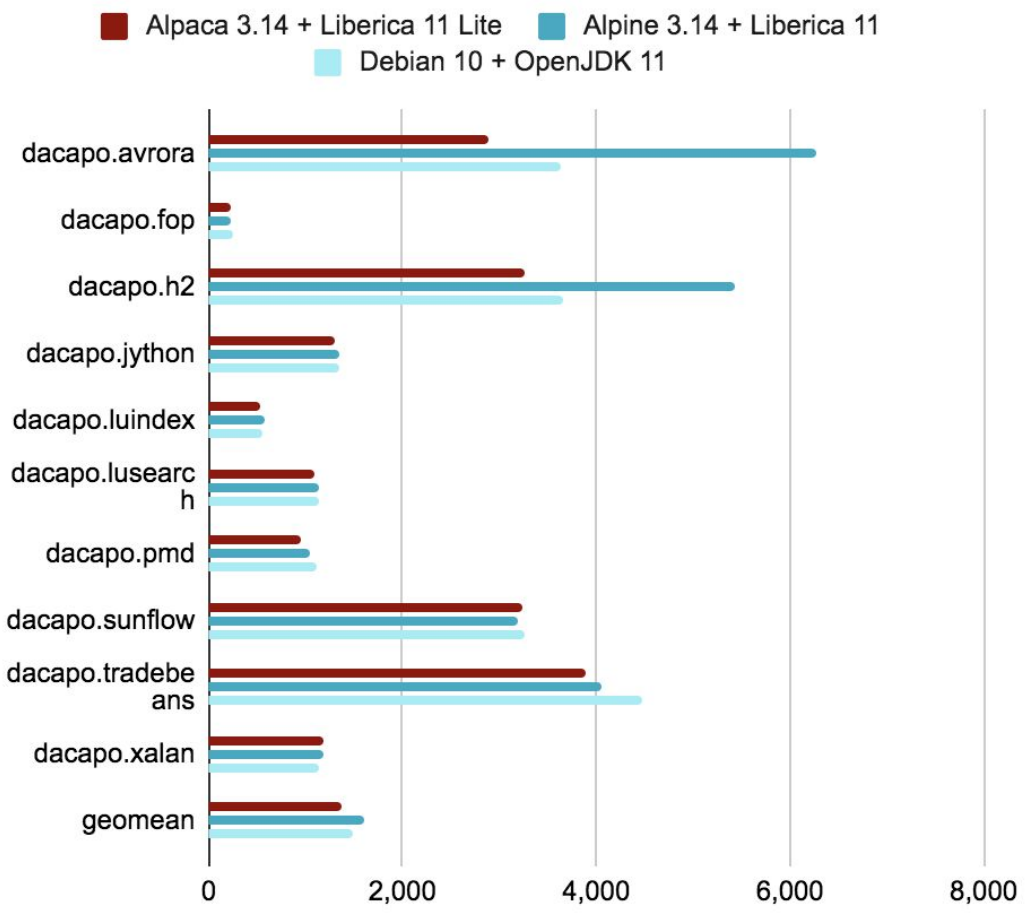




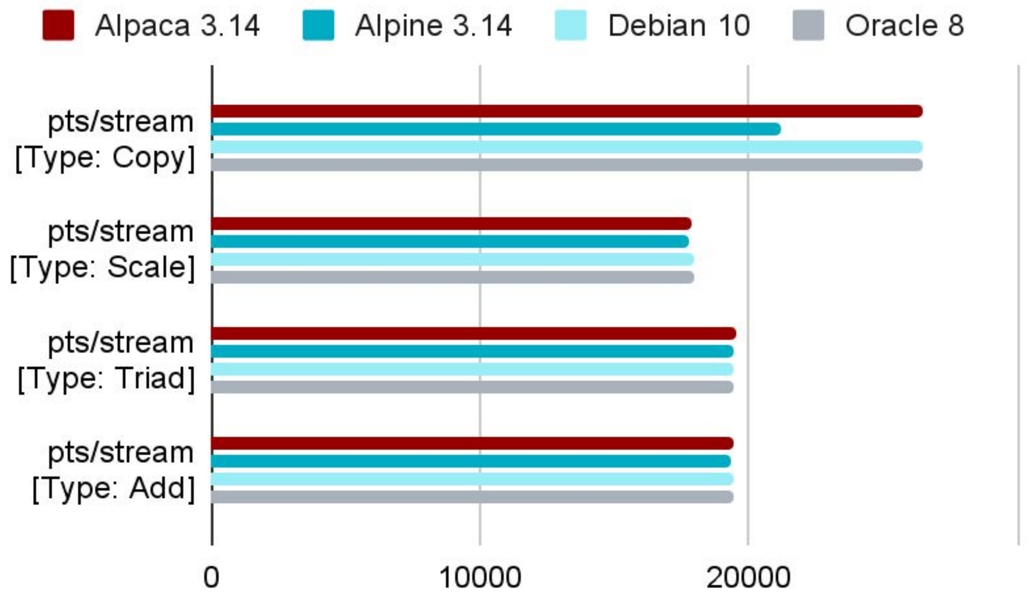
Results in relative speed 1/s,
higher is better

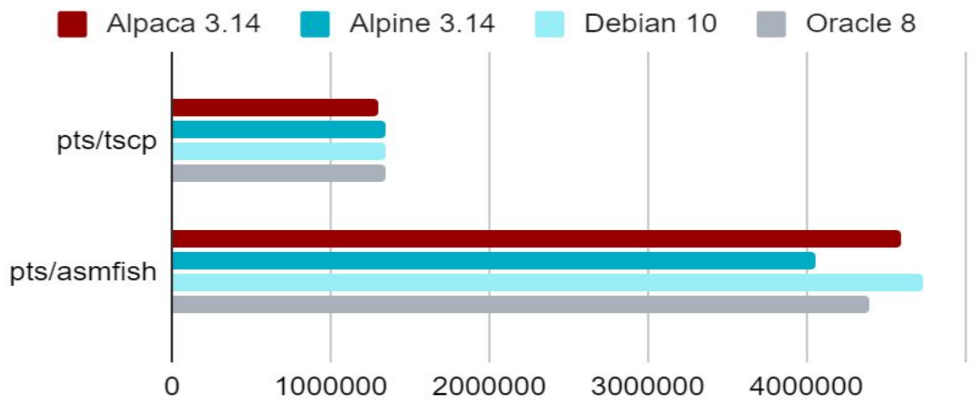
Alpaca. Phoronix benchmarks

Lower is better, by 7% geomean



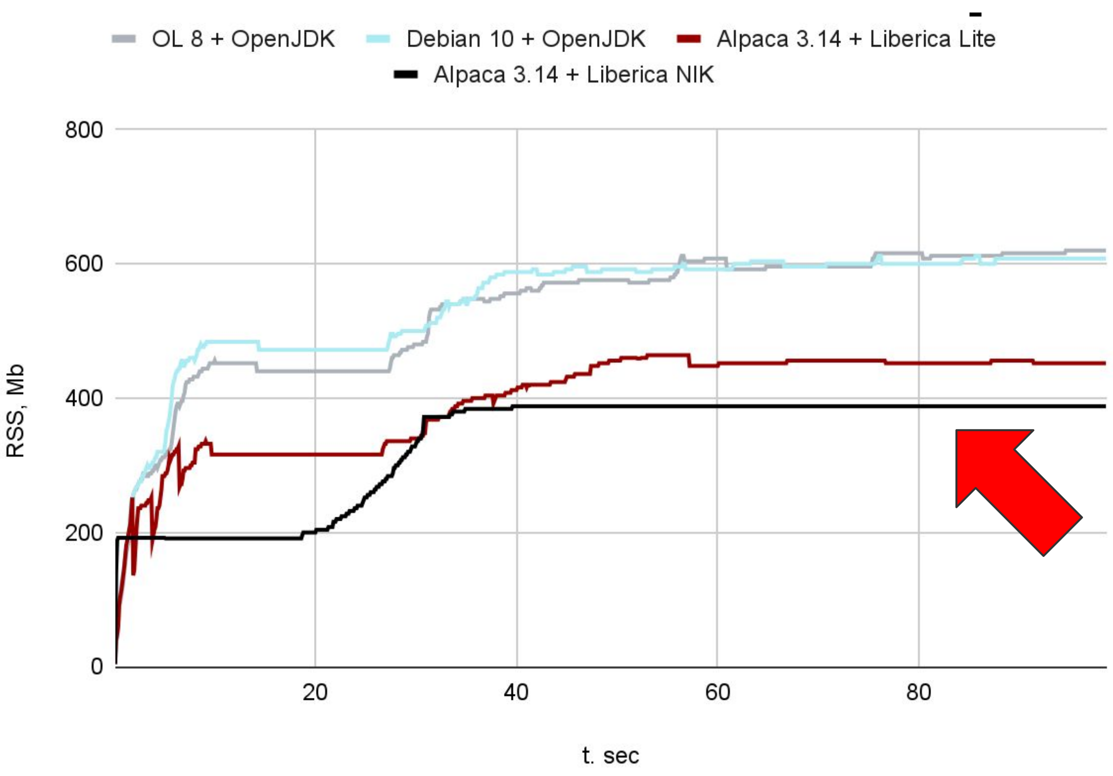
Higher is better

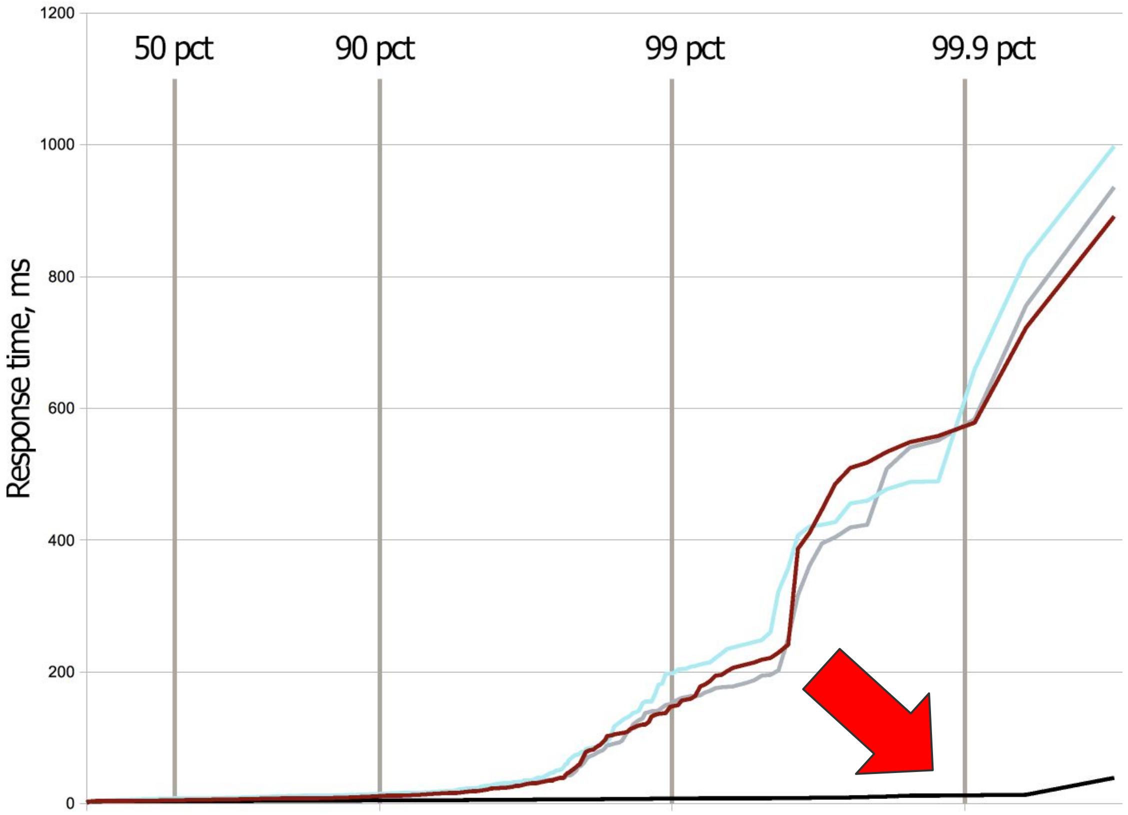




Petclinic RAM footprint & latency

Low-end HW, underutilized CPU, mixed scenario
Low load (133 users, 54 RPS), delayed load start



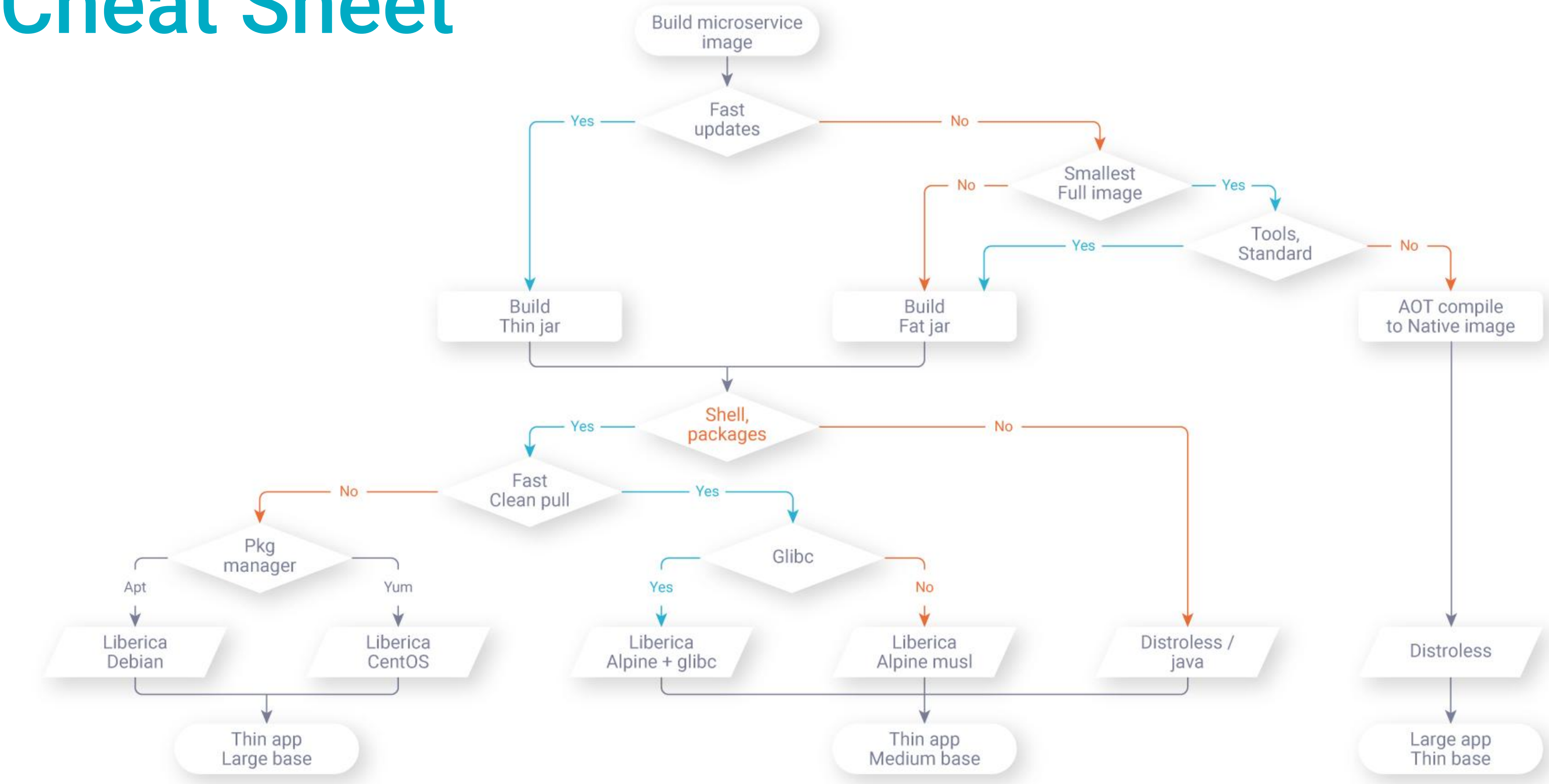


- Up to 50% lower RAM consumption

- NIK vs JDK **including ramp-up**
  - 25x lower latency at 99 pct
  - 46x lower latency at 99.9 pct

BELLSOFT

Cheat Sheet



Build microservice image
Fast updates
Yes
No
Smallest Full image
No
Yes
Tools, Standard
Yes
No
Build Thin jar
Build Fat jar
AOT compile to Native image
Shell, packages
Yes
No
Fast Clean pull
No
Yes
Pkg manager
Apt
Yum
Glibc
Yes
No
Liberica Debian
Liberica CentOS
Liberica Alpine + glibc
Liberica Alpine musl
Distroless / java
Distroless
Thin app Large base
Thin app Medium base
Large app Thin base

Conclusions

- **Container images**
  - Many ways to deliver
  - Many ways to build
  - Small base images help in production
- **Alpine & musl**
  - Smallest OS image with tools
  - Peculiarities
- **OpenJDK port to musl**
  - Officially in OpenJDK
  - Good base images with JDK
- **Alpaca, Liberica Lite, Liberica NIK**
  - **Make it all even more secure and performant**

BELLSOFT

Thank you for your attention!

https://bell-sw.com

dmitry.chuyko@bell-sw.com

@dchuyko