Alpine Musl Containers
Now Upstream

Dmitry Chuyko

2021
Dmitry Chuyko

Liberica [www.bell-sw.com](http://www.bell-sw.com) supported OpenJDK binaries

ex-employers:

[@dchuyko](https://twitter.com/dchuyko)
OpenJDK Contributions

JDK 11
Deployment

“...package an application with all of its dependencies into a standardized unit for software development.”

— Docker
Microservice container layers

- Scratch
- OS
- JRE
- OS Packages
- App Libraries
- Framework
- App
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.**
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
Pull time (100 Mbps)

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

28 s
## Uncompressed Size (disk)

```
$ docker history openjdk

<table>
<thead>
<tr>
<th>IMAGE</th>
<th>CREATED</th>
<th>CREATED BY</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>95b80f783bd2</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;jshell&quot;]</td>
<td>0B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c set -eux; objdump=&quot;${(command -v...</td>
<td>336MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) ENV JAVA_VERSION=15.0.1</td>
<td>0B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) ENV PATH=/usr/java/openj...</td>
<td>0B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) ENV JAVA_HOME=/usr/java/o...</td>
<td>0B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) ENV LANG=C.UTF-8</td>
<td>0B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c set -eux; microdnf install gzi...</td>
<td>40.1MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) CMD [&quot;/bin/bash&quot;]</td>
<td>0B</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>12 days ago</td>
<td>/bin/sh -c #(nop) ADD file:ca74b6a4572ba9ecd...</td>
<td>148MB</td>
</tr>
<tr>
<td>&lt;missing&gt;</td>
<td>8 weeks ago</td>
<td>/bin/sh -c #(nop) LABEL org.opencontainers....</td>
<td>0B</td>
</tr>
</tbody>
</table>

$ docker images | head -n 1; docker images | grep openjdk

<table>
<thead>
<tr>
<th>REPOSITORY</th>
<th>TAG</th>
<th>IMAGE ID</th>
<th>CREATED</th>
<th>SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td>openjdk</td>
<td>latest</td>
<td>95b80f783bd2</td>
<td>12 days ago</td>
<td>524MB</td>
</tr>
</tbody>
</table>
```

524 MB
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.
<table>
<thead>
<tr>
<th>OS Layer</th>
<th>Wire</th>
<th>Disk</th>
<th>libc</th>
<th>pkg man</th>
<th>shell</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu</td>
<td>27 MB</td>
<td>73 MB</td>
<td>glibc</td>
<td>apt</td>
<td>bash</td>
</tr>
<tr>
<td>Debian</td>
<td>48 MB</td>
<td>114 MB</td>
<td>glibc</td>
<td>apt</td>
<td>bash</td>
</tr>
<tr>
<td>Debian Slim</td>
<td>26 MB</td>
<td>69 MB</td>
<td>glibc</td>
<td>apt</td>
<td>bash</td>
</tr>
<tr>
<td>CenOS</td>
<td>71 MB</td>
<td>215 MB</td>
<td>glibc</td>
<td>yum</td>
<td>bash</td>
</tr>
<tr>
<td>RHEL Atomic Base</td>
<td>31 MB</td>
<td>78 MB</td>
<td>glibc</td>
<td>microdnf</td>
<td>bash</td>
</tr>
<tr>
<td>GCR Distroless base</td>
<td>7.6 MB</td>
<td>17 MB</td>
<td>glibc</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>Alpine</td>
<td>2.7 MB</td>
<td>5.6 MB</td>
<td>musl</td>
<td>apk</td>
<td>ash</td>
</tr>
<tr>
<td>GCR Distroless static</td>
<td>0.6 MB</td>
<td>1.8 MB</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>
Alpine Linux

... is a security-oriented, lightweight Linux distribution based on musl libc and busybox.

— Alpine
Alpine Linux. At a glance

- [alpinelinux.org](http://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
Musl libc. At a glance

- [musl.libc.org](http://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 mustl, 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.
Busybox. At a glance

- [busybox.net](https://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](https://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
Alpine Linux is perfect for containers

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

What about JDK layer?
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](openjdk.java.net/jeps/386)

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 `jlink` command.
Project Portola

- [openjdk.java.net/projects/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](https://github.com/openjdk/jdk/blob/master/doc/building.md#building-for-musl)
**JNI. Cross Build**

```bash
$ 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

- *ls\(fof\) 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*

- *Serviceability agent (private API) doesn’t work*
$ 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

$ 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'
$ 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.
<table>
<thead>
<tr>
<th>OS + JDK 15 Image</th>
<th>Wire</th>
<th>Disk</th>
</tr>
</thead>
<tbody>
<tr>
<td>bellsoft/liberica-openjdk-debian</td>
<td>126 MB</td>
<td>231 MB</td>
</tr>
<tr>
<td>bellsoft/liberica-openjdk-centos</td>
<td>183 MB</td>
<td>322 MB</td>
</tr>
<tr>
<td>bellsoft/liberica-openjdk-alpine</td>
<td>78 MB</td>
<td>132 MB</td>
</tr>
<tr>
<td>bellsoft/liberica-openjdk-alpine-musl</td>
<td>76 MB</td>
<td>107 MB</td>
</tr>
</tbody>
</table>
Pull time

$ time docker pull bellsoft/liberica-openjdk-alpine-musl:latest
...
real  0m3.957s
user  0m0.026s
sys   0m0.061s
Alpine 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).
We plan to stay on Java 8.

— NN% of users
Portola Expansion

- **JDK 11 LTS**
  - Not in mainline (yet)
  - Historical downports in Liberica 9+
- **JDK 8 LTS**
  - Liberica 8u on Dockerhub
- **AArch64**
- **OpenWRT**
  - One more flavor of Raspberry Pi