

Installing Contributing Sponsoring Developers' Guide Vulnerabilities JDK GA/EA Builds Mailing lists

Wiki · IRC

Bylaws · Census

Workshop

Legal

JEP Process
Source code

Mercurial GitHub **Tools**

Git jtreg harness

Groups
(overview)
Adoption
Build
Client Libraries
Compatibility &

Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization

Specification

JMX
Members
Networking
Porters
Quality
Security
Serviceability
Vulnerability

Projects

Web

(overview, archive) **Amber** Babylon CRaC Caciocavallo Closures Code Tools Coin Common VM Interface Compiler Grammar Detroit Developers' Guide Device I/O Duke Galahad Graal IcedTea JDK 7 JDK 8

JDK 8
JDK 8 Updates
JDK 9
JDK (..., 21, 22, 23)
JDK Updates
JavaDoc.Next
Jigsaw
Kona
Kulla
Lambda
Lanai
Leyden
Lilliput

Locale Enhancement

Loom
Memory Model
Update
Metropolis
Mission Control
Multi-Language VM
Nashorn
New I/O
OpenJFX
Panama
Penrose
Port: AArch32
Port: AArch64
Port: BSD

Port: Haiku
Port: Mac OS X
Port: MIPS
Port: Mobile
Port: PowerPC/AIX
Port: RISC-V
Port: s390x
Portola
SCTP
Shenandoah
Skara

Tsan
Type Annotations
Valhalla
Verona
VisualVM
Wakefield
Zero
ZGC

Tiered Attribution

Sumatra

ORACLE

JEP 386: Alpine Linux Port

Owner Boris Ulasevich
Type Feature
Scope Implementation

Status Closed/Delivered Release 16

Component hotspot

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 2021/08/28 00:30

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. For example, if a target application depends only on the java. base module then a Docker image with Alpine Linux and a Java runtime with just that module and the server VM fits in 38 MB.

The same motivation applies to embedded deployments, which also have size constraints.

Description

This JEP intends to integrate the Portola Project upstream.

This port will not support the attach mechanism of the HotSpot Serviceability Agent.

To build a musl variant of the JDK on Alpine Linux, the following packages are required:

alpine-sdk alsa-lib alsa-lib-dev autoconf bash cups-dev cups-libs fontconfig fontconfig-dev freetype freetype-dev grep libx11 libx11-dev libxext libxext-dev libxrandr libxrandr-dev libxrender libxrender-dev libxt libxt-dev libxtst libxtst-dev linux-headers zip

Once these packages are installed, the JDK build process works as usual.

Musl ports for other architectures may be implemented in follow-up enhancements, if there is demand.

Alternatives

- Keep the musl port downstream in the Portola Project.
- Use the glibc portability layer provided by the system, or build and bundle glibc with the OS image. This alternative has the drawback of increasing image footprint and adding a dependency which is not part of the base OS image. For a cloud deployment scenario, assuming the base Alpine Linux 3.11 musl image is 5.6 MB, the additional glibc layer is 26 MB, and the Java runtime with the java.base module and the server VM is 38 MB, the static footprint overhead of having the glibc portability layer in the image is 30%.

Testing

- Since the port affects shared code, we will run JCK and JTreg tests on glibcbased Linux distributions (x64, AArch64, and ARM32), Windows, and Mac before integration to ensure the absence of regressions.
- We will verify the quality of the musl port by running a relevant subset of JTreg tests on Alpine Linux. We will make sure that a downport of Portola to a current production JDK release passes the JCK on Alpine Linux. We will run sanity testing on OpenWrt.
- We will define a new subset of JTreg tests suitable for musl, in a vm.musl test group.