Liberica JDK
Choosing Liberica JDK flavor

Liberica JDK
Revision 1.0
April 2024

bellsoft

BellSoft software contains open source software. Additional information about third party code is available at https://bell-sw.com/third_party_licenses. You can also get more information on how to get a copy of source code by contacting info@bell-sw.com.

The use of any software product referenced in this document is governed by the applicable license agreement, which is not modified in any way by the terms of this notice.

Contents

1. Introduction

Liberica JDK is compatible with the broadest range of system configurations on the market. In addition, we provide three different flavors: Standard, Full, Perf, and Lite, as well as JRE and JDK builds. In this document, we provide an overview of available packages so that you can choose the most optimal configuration for your project.

2. Liberica JDK packages: Full, Standard, Perf, Lite

Liberica JDK Standard

Liberica JDK Standard is a standard OpenJDK distribution with only minor adjustments to the codebase. This package includes all the components necessary for writing, compiling, debugging, and running Java applications. The libraries are implemented as is (vanilla builds), with a guarantee of no vendor lock-in. Liberica JDK Standard is perfect for enterprises looking for a tried-and-true, classical JDK for desktop and server deployment.

Liberica JDK Full

Liberica JDK Full is based on Liberica JDK Standard but also includes LibericaFX, an OpenJFX implementation for building desktop and GUI applications. In addition, the package also contains a Minimal VM for the development of applications for embedded systems.

Liberica JDK Perf

Liberica JDK Performance Edition (or liberica-perf) brings some of the performance of JVM 17 to projects that cannot afford complete migration out of JDK 8 or JDK 11. Liberica JDK Performance Edition is only available for Linux x86_64. For more information, see Liberica JDK: Performance Edition Overview.

Liberica JDK Lite

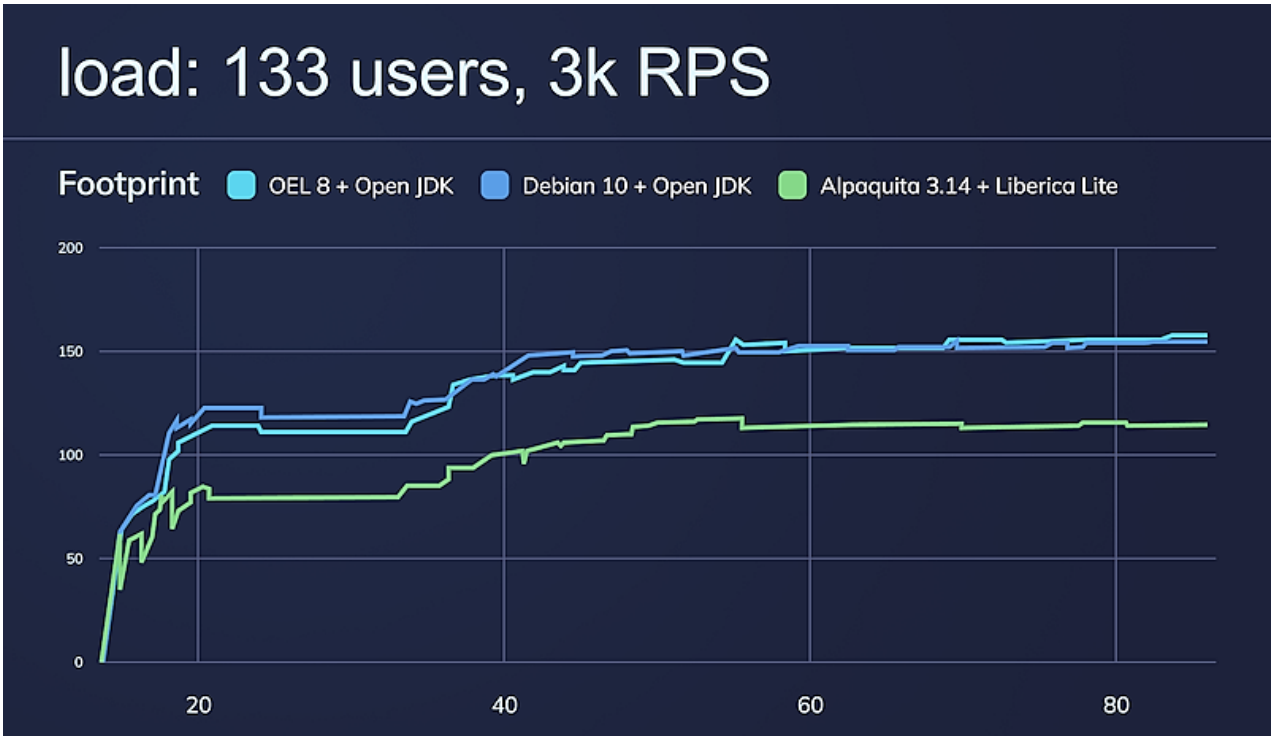
Liberica JDK Lite is a JDK optimized for cloud instances with a minimal footprint. It is a full-fledged, Java SE-compliant runtime but much smaller than any standard Java distribution. We didn't remove any components to trim down its size; instead, we introduced multiple enhancements, such as better

compression for modules. As a result, Liberica Lite has a higher compression ratio for modules than a classic JDK, thus reducing static footprint. Furthermore, we integrated several backports from the recent versions into Liberica JDK 11 and 17, including:

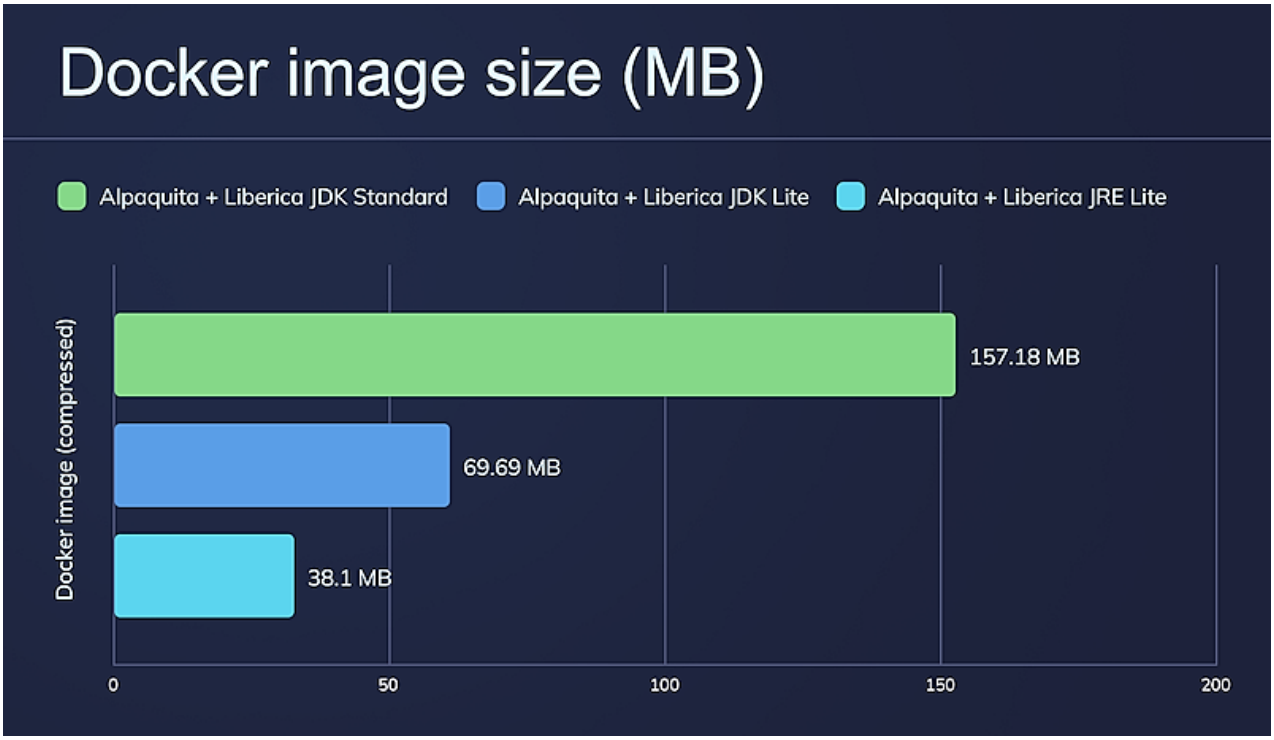
- For Liberica JDK 11:
  - JEP 346: Promptly Return Unused Committed Memory from G1, — enables G1 GC to automatically return unused Java heap memory to the operating system in the case of low application activity, — and its dependent changes.
  - JDK-8203469: Faster safepoints, — increases the average CPU time for JavaThreads between safepoints thus accelerating allocation rate for better performance, — and its dependent changes.
  - LTO (link-time optimization) build. Link-time optimization enables the GCC (the GNU Compiler Collection) to write unoptimized intermediate code to object files, which are then optimized by linker as a single module at link time. LTO support in JDK was added as part of JEP 297: Unified arm32/arm64 Port and is aimed at improving performance and reducing the static footprint of shared libraries.
- For Liberica JDK 17:
  - String deduplication for ZGC, ParallelGC, and SerialGC, first introduced in JEP 192: String deduplication for G1, and then extended to other garbage collectors in Java 18. If the strings are duplicated, i.e., contain the same bytes in their code and value fields, the GC deletes all but one byte arrays and reassigns references, thus reducing memory footprint.
  - LTO build.
  - Supporting CDS archived heap objects for ParallelGC and SerialGC helps to improve startup because there's no need to create numerous expensive data structures (e.g., the module graph) from scratch.

Note that all Liberica Lite binaries go through rigorous Q&A tests before each release, including performance regression testing, so there's no difference between Standard and Lite flavors in terms of performance.

The graph below demonstrates how optimized Liberica Lite coupled with Alpaquita Linux helps to reduce the footprint of the Petclinic Spring application compared with other popular Linux and OpenJDK distributions.



Liberica Lite is the build we use to create the smallest Java containers on the market. Take a look at the following numbers (as of March 20, 2023):



The striking difference on the graph raises a question: why don't we use a JRE-based container if it is four times smaller than the regular one? Well, it depends on your goals. The section below reveals the

distinguishing features of JDK and JRE.

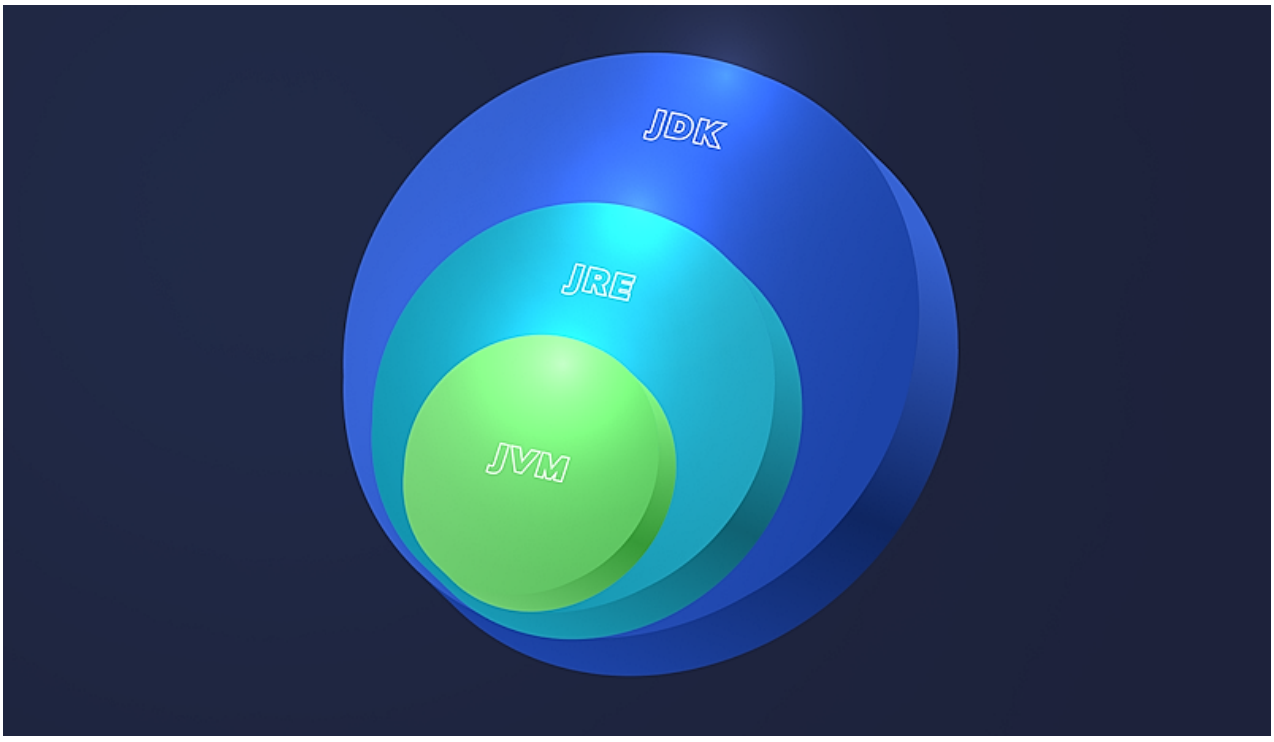
3. Difference between JDK and JRE packages

JDK and JRE are the essential concepts of Java programming. In the hearts of them both is a JVM, Java Virtual Machine. JVM provides a runtime environment to execute the Java code. It makes the key feature of Java programs (WORA or ``write once, run anywhere'') a reality by compiling the Java bytecode into machine-specific one.

JRE, or Java Runtime Environment, provides a set of libraries and binaries to execute Java applications. It doesn't contain any development tools such as javac, Java debugger, JShell, etc., so it is an optimal choice if you plan to only run Java apps, for instance, in the cloud.

JDK, or Java Development Kit, is the superset of JRE. It contains everything that is in JRE plus multiple development tools such as a Java interpreter (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), a debugger (jdb), and others for developing and debugging Java applications.

The diagram below shows the critical difference between JVM, JRE, and JDK concepts.



To keep your containers slim, use JDK for development and JRE for execution. The Dockerfile below

demonstrates how we can build a project with Liberica JDK in a container and then copy the image into another container with Liberica JRE:

```
FROM bellsoft/liberica-runtime-container:jdk-17-stream-musl as builder
WORKDIR /home/myapp
ADD docker-image-demo /home/myapp/docker-image-demo
RUN cd docker-image-demo && ./mvnw package
FROM bellsoft/liberica-runtime-container:jre-17-stream-musl
WORKDIR /home/myapp
COPY –from=builder /home/myapp/docker-image-demo/target .
CMD ["java", "-jar", "docker-image-demo-0.0.1-SNAPSHOT.jar"]
```

The resulting JRE-based container with lightweight Alpaquita Linux will be three times smaller than the one with Oracle JDK and Debian. The size reduction is especially relevant for Spring Boot apps, which sometimes can be associated with significant memory footprint.

Liberica JDK

Choosing Liberica JDK flavor

bell//soft