Red Hat build of OpenJDK 11

Migrating Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11

Last Updated: 2024-10-03

Legal Notice

Abstract

The Migrating Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11 guide provides information on how to upgrade your Red Hat build of OpenJDK 8 applications to Red Hat build of OpenJDK 11.

Table of Contents

PROVIDING FEEDBACK ON RED HAT BUILD OF OPENJDK DOCUMENTATION

To report an error or to improve our documentation, log in to your Red Hat Jira account and submit an issue. If you do not have a Red Hat Jira account, then you will be prompted to create an account.

Procedure

1. Click the following link to **create a ticket**.
2. Enter a brief description of the issue in the **Summary**.
3. Provide a detailed description of the issue or enhancement in the **Description**. Include a URL to where the issue occurs in the documentation.
4. Clicking **Create** creates and routes the issue to the appropriate documentation team.

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message.

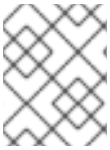
CHAPTER 1. MIGRATION TO RED HAT BUILD OF OPENJDK 11

The *Migrating Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11* guide provides information on how to upgrade your Java applications in Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11.

This guide describes changes in the Red Hat build of OpenJDK 11 release, including new features and deprecated or removed APIs, that might impact your migration from Red Hat build of OpenJDK 8.

The OpenJDK project is known for its conservative approach to providing updates and for providing backward compatibility. However, to guarantee the evolution, security and stability of the project, the Red Hat build of OpenJDK project might sometimes introduce a few incompatibilities across major releases of Red Hat build of OpenJDK. These incompatibilities are relevant for the following scenarios:

- When you use APIs that are considered obsolete or unsecure
- When you access internals of the project that are considered implementation details and not public or supported API details



NOTE

Red Hat recommends that you migrate to the latest supported Red Hat build of OpenJDK distribution.

1.1. ABOUT RED HAT BUILD OF OPENJDK DISTRIBUTIONS

OpenJDK is the free and open source reference implementation of the Java Platform, Standard Edition (Java SE). Red Hat build of OpenJDK distributions are based on the upstream OpenJDK 8u, OpenJDK 11u, OpenJDK 17u, and OpenJDK 21u projects. The Shenandoah Garbage Collector is included in all supported versions of Red Hat build of OpenJDK.

All versions of Red Hat build of OpenJDK provide the following benefits:

Multi-platform

Red Hat build of OpenJDK is supported on RHEL and Microsoft Windows, so you can standardize applications on a single Java platform across desktop, data center, and hybrid cloud environments.

Frequent releases

Red Hat delivers quarterly updates of JRE and JDK for Red Hat build of OpenJDK 8, Red Hat build of OpenJDK 11, Red Hat build of OpenJDK 17, and Red Hat build of OpenJDK 21 distributions. These updates are available as archive, RPM, and Windows MSI-based installer files and container images.

Long-term support

Red Hat supports the recently released Red Hat build of OpenJDK 8, Red Hat build of OpenJDK 11, Red Hat build of OpenJDK 17, and Red Hat build of OpenJDK 21 distributions.

Additional resources

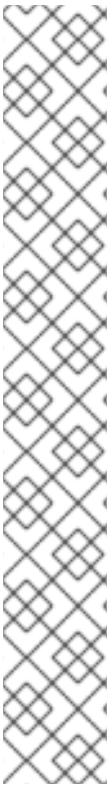
- For more information about the support lifecycle, see OpenJDK Life Cycle and Support Policy .

CHAPTER 2. MAJOR DIFFERENCES BETWEEN RED HAT BUILD OF OPENJDK 8 AND RED HAT BUILD OF OPENJDK 11

Before migrating your Java applications from Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11, familiarize yourself with the changes in Red Hat build of OpenJDK 11. These changes might require that you reconfigure your existing Red Hat build of OpenJDK installation before you migrate to version 11.

One of the major differences between Red Hat build of OpenJDK 8 and Red Hat build of OpenJDK 11 is the inclusion of a module system in Red Hat build of OpenJDK 11. If you are migrating from Red Hat build of OpenJDK 8, consider moving your application's libraries and modules from the Red Hat build of OpenJDK 8 class path to the module class in Red Hat build of OpenJDK 11. This change can improve the class-loading capabilities of your application.

Red Hat build of OpenJDK 11 includes new features and enhancements that can improve the performance of your application, such as enhanced memory usage, improved startup speed, and increased container integration.



NOTE

Some features might differ between Red Hat build of OpenJDK and other upstream community or third-party versions of OpenJDK. For example:

- The Shenandoah garbage collector is available in all versions of Red Hat build of OpenJDK, but this feature might not be available by default in other builds of OpenJDK.
- JDK Flight Recorder (JFR) support in OpenJDK 8 has been available from version 8u262 onward and enabled by default from version 8u272 onward, but JFR might be disabled in certain builds. Because JFR functionality was backported from the open source version of JFR in OpenJDK 11, the JFR implementation in Red Hat build of OpenJDK 8 is largely similar to JFR in Red Hat build of OpenJDK 11 or later. This JFR implementation is different from JFR in Oracle JDK 8, so users who want to migrate from Oracle JDK to Red Hat build of OpenJDK 8 or later need to be aware of the command-line options for using JFR.
- 32-bit builds of OpenJDK are generally unsupported in OpenJDK 8 or later, and they might not be available in later versions. 32-bit builds are unsupported in all versions of Red Hat build of OpenJDK.

2.1. CRYPTOGRAPHY AND SECURITY

Certain minor cryptography and security differences exist between Red Hat build of OpenJDK 8 and Red Hat build of OpenJDK 11. However, both versions of Red Hat build of OpenJDK have many similar cryptography and security behaviors.

Red Hat builds of OpenJDK use system-wide certificates, and each build obtains its list of disabled cryptographic algorithms from a system's global configuration settings. These settings are common to all versions of Red Hat build of OpenJDK, so you can easily change from Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11.

In FIPS mode, Red Hat build of OpenJDK 8 and Red Hat build of OpenJDK 11 releases are self-configured, so that either release uses the same security providers at startup.

The TLS stacks in Red Hat build of OpenJDK 8 and Red Hat build of OpenJDK 11 are identical, because the SunJSSE engine from Red Hat build of OpenJDK 11 was backported to Red Hat build of OpenJDK 8. Both Red Hat build of OpenJDK versions support the TLS 1.3 protocol.

The following minor cryptography and security differences exist between Red Hat build of OpenJDK 8 and Red Hat build of OpenJDK 11:

Red Hat build of OpenJDK 8	Red Hat build of OpenJDK 11
TLS clients do not use TLSv1.3 for communication with the target server by default. You can change this behavior by setting the **jdk.tls.client.protocols** system property to **-Djdk.tls.client.protocols=TLSv1.3**.	TLS clients use TLSv.1.3 by default.
This release does not support the use of the **X25519** and **X448** elliptic curves in the Diffie-Hellman key exchange.	This release supports the use of the **X25519** and **X448** elliptic curves in the Diffie-Hellman key exchange.
This release still supports the legacy KRB5-based cipher suites, which are disabled for security reasons. You can enable these cipher suites by changing the **jdk.tls.client.cipherSuites** and **jdk.tls.server.cipherSuites** system properties.	This release does not support the legacy KRB5-based cipher suites.
This release does not support the Datagram Transport Layer Security (DTLS) protocol.	This release supports the DTLS protocol.
The **max_fragment_length** extension, which is used by DTLS, is not available for TLS clients.	The **max_fragment_length** extension is available for both clients and servers.

2.2. GARBAGE COLLECTOR

For garbage collection, Red Hat build of OpenJDK 8 uses the Parallel collector by default, whereas Red Hat build of OpenJDK 11 uses the Garbage-First (G1) collector by default.

Before you choose a garbage collector, consider the following details:

- If you want to improve throughput, use the Parallel collector. The Parallel collector maximizes throughput but ignores latency, which means that garbage collection pauses could become an issue if you want your application to have reasonable response times. However, if your application is performing batch processing and you are not concerned about pause times, the Parallel collector is the best choice. You can switch to the Parallel collector by setting the **-XX:+UseParallelGC** JVM option.
- If you want a balance between throughput and latency, use the G1 collector. The G1 collector can achieve great throughput while providing reasonable latencies with pause times of a few hundred milliseconds. If you notice throughput issues when migrating applications from Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11, you can switch to the Parallel collector as described above.
- If you want low-latency garbage collection, use the Shenandoah collector.

You can select the garbage collector type that you want to use by specifying the **-XX:+<gc_type>** JVM option at startup. For example, the **-XX:+UseParallelGC** option switches to the Parallel collector.

2.3. GARBAGE COLLECTOR LOGGING OPTIONS

Red Hat build of OpenJDK 11 includes a new and more powerful logging framework that works more effectively than the old logging framework. Red Hat build of OpenJDK 11 also includes unified JVM logging options and unified GC logging options.

The logging system for Red Hat build of OpenJDK 11 activates the -**XX:+PrintGCTimeStamps** and **-XX:+PrintGCDateStamps** options by default. Because the logging format in Red Hat build of OpenJDK 11 is different from Red Hat build of OpenJDK 8, you might need to update any of your code that parses garbage collector logs.

Modified options in Red Hat build of OpenJDK 11

The old logging framework options are deprecated in Red Hat build of OpenJDK 11. These old options are still available only as aliases for the new logging framework options. If you want to work more effectively with Red Hat build of OpenJDK 11, use the new logging framework options.

The following table outlines the changes in garbage collector logging options between Red Hat build of OpenJDK versions 8 and 11:

Options in Red Hat build of OpenJDK 8	Options in Red Hat build of OpenJDK 11
-verbose:gc	**-Xlog:gc**
-XX:+PrintGC	**-Xlog:gc**
-XX:+PrintGCDetails	**-Xlog:gc*** or **-Xlog:gc+$tags**
-Xloggc:$FILE	**-Xlog:gc:file=$FILE**

When using the **-XX:+PrintGCDetails** option, pass the **-Xlog:gc*** flag, where the asterisk (***) activates more detailed logging. Alternatively, you can pass the **-Xlog:gc+$tags** flag.

When using the **-Xloggc** option, append the **:file=$FILE** suffix to redirect log output to the specified file. For example **-Xlog:gc:file=$FILE**.

Removed options in Red Hat build of OpenJDK 11

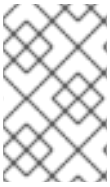
Red Hat build of OpenJDK 11 does not include the following options, which were deprecated in Red Hat build of OpenJDK 8:

- **-Xincgc**
- **-XX:+CMSIncrementalMode**
- **-XX:+UseCMSCompactAtFullCollection**

- **-XX:+CMSFullGCsBeforeCompaction**
- **-XX:+UseCMSCollectionPassing**

Red Hat build of OpenJDK 11 also removes the following options because the printing of timestamps and datestamps is automatically enabled:

- **-XX:+PrintGCTimeStamps**
- **-XX:+PrintGCDateStamps**



NOTE

In Red Hat build of OpenJDK 11, unless you specify the **-XX:+IgnoreUnrecognizedVMOptions** option, the use of any of the preceding removed options results in a startup failure.

Additional resources

- For more information about the common framework for unified JVM logging and the format of **Xlog** options, see JEP 158: Unified JVM Logging .
- For more information about deprecated and removed options, see JEP 214: Remove GC Combinations Deprecated in JDK 8.
- For more information about unified GC logging, see JEP 271: Unified GC Logging .

2.4. OPENJDK GRAPHICS

Before version 8u252, Red Hat build of OpenJDK 8 used Pisces as the default rendering engine. From version 8u252 onward, Red Hat build of OpenJDK 8 uses Marlin as the new default rendering engine. Red Hat build of OpenJDK 11 and later releases also use Marlin by default. Marlin improves the handling of intensive application graphics.

Because the rendering engines produce the same results, users should not observe any changes apart from improved performance.

2.5. WEBSTART AND APPLETS

You can use Java WebStart by using the IcedTea-Web plug-in with Red Hat build of OpenJDK 8 or Red Hat build of OpenJDK 11 on RHEL 7, RHEL 8, and Microsoft Windows operating systems. The IcedTea-Web plug-in requires that Red Hat build of OpenJDK 8 is installed as a dependency on the system.

Applets are not supported on any version of Red Hat build of OpenJDK. Even though some applets can be run on RHEL 7 by using the IcedTea-web plug-in with OpenJDK 8 on a Netscape Plugin Application Programming Interface (NPAPI) browser, Red Hat build of OpenJDK does not support this behavior.

NOTE

The upstream community version of OpenJDK does not support applets or Java Webstart. Support for these technologies is deprecated and they are not recommended for use.

2.6. JPMS

The Java Platform Module System (JPMS), which was introduced in OpenJDK 9, limits or prevents access to non-public APIs. JPMS also impacts how you can start and compile your Java application (for example, whether you use a class path or a module path).

Internal modules

By default, Red Hat build of OpenJDK 11 restricts but still permits access to JDK internal modules. This means that most applications can continue to work without requiring changes, but these applications will emit a warning. As a workaround for this restriction, you can enable your application to access an internal package by passing a **--add-opens <module-name>/<package-in-module>=ALL-UNNAMED** option to the **java** command.

For example:

```
--add-opens java.base/jdk.internal.math=ALL-UNNAMED
```

Additionally, you can check illegal access cases by passing the **--illegal-access=warn** option to the **java** command. This option changes the default behavior of Red Hat build of OpenJDK.

ClassLoader

The JPMS refactoring changes the **ClassLoader** hierarchy in Red Hat build of OpenJDK 11.

In Red Hat build of OpenJDK 11, the system class loader is no longer an instance of **URLClassLoader**. Existing application code that invokes **ClassLoader::getSystemClassLoader** and casts the result to a **URLClassLoader** in Red Hat build of OpenJDK 11 will result in a runtime exception.

In Red Hat build of OpenJDK 8, when you create a class loader, you can pass **null** as the parent of this class loader instance. However, in Red Hat build of OpenJDK 11, applications that pass **null** as the parent of a class loader might prevent the class loader from locating platform classes.

Red Hat build of OpenJDK 11 includes a new class loader that can control the loading of certain classes. This improves the way that a class loader can locate all of its required classes. In Red Hat build of OpenJDK 11, when you create a class loader instance, you can set the platform class loader as its parent by using the **ClassLoader.getPlatformClassLoader()** API.

Additional resources

- For more information about JPMS, see JEP 261: Module System .

2.7. EXTENSION AND ENDORSED OVERRIDE MECHANISMS

In Red Hat build of OpenJDK 11, both the extension mechanism, which supported optional packages, and the endorsed standards override mechanism are no longer available.

These changes mean that any libraries that are added to the **<JAVA_HOME>/lib/ext** or **<JAVA_HOME>/lib/endorsed** directory are no longer used, and Red Hat build of OpenJDK 11 generates an error if these directories exist.

Additional resources

- For more information about the removed mechanisms, see JEP 220: Modular Run-Time Images .

2.8. JFR FUNCTIONALITY

JDK Flight Recorder (JFR) support was backported to Red Hat build of OpenJDK 8 starting from version 8u262. JFR support was subsequently enabled by default from Red Hat build of OpenJDK 8u272 onward.

NOTE

The term *backporting* describes when Red Hat takes an update from a more recent version of upstream software and applies that update to an older version of the software that Red Hat distributes.

Backported JFR features

The JFR backport to Red Hat build of OpenJDK 8 included all of the following features:

- A large number of events that are also available in Red Hat build of OpenJDK 11
- Command-line tools such as **jfr** and the Java diagnostic command (**jcmd**) that behave consistently across Red Hat build of OpenJDK versions 8 and 11
- The Java Management Extensions (JMX) API that you can use to enable JFR by using the JMX beans interfaces either programmatically or through **jcmd**
- The **jdk.jfr** namespace

NOTE

The JFR APIs in the **jdk.jfr** namespace are not considered part of the Java specification in Red Hat build of OpenJDK 8, but these APIs are part of the Java specification in Red Hat build of OpenJDK 11. Because the JFR API is available in all supported Red Hat build of OpenJDK versions, applications that use JFR do not require any special configuration to use the JFR APIs in Red Hat build of OpenJDK 8 and later versions.

JDK Mission Control, which is distributed separately, was also updated to be compatible with Red Hat build of OpenJDK 8.

Applications that need to be compatible with other OpenJDK versions

If your applications need to be compatible with any of the following OpenJDK versions, you might need to adapt these applications:

- OpenJDK versions earlier than 8u262
- OpenJDK versions from other vendors that do not support JFR
- Oracle JDK

To aid this effort, Red Hat has developed a special compatibility layer that provides an empty implementation of JFR, which behaves as if JFR was disabled at runtime. For more information about the JFR compatibility API, see openjdk8-jfr-compat. You can install the resulting **.jar** file in the **jre/lib/ext** directory of an OpenJDK 8 distribution.

Some applications might need to be updated if these applications were filtering out OpenJDK 8 by checking only for the version number instead of querying the MBeans interface.

2.9. JRE AND HEADLESS PACKAGES

All Red Hat build of OpenJDK versions for RHEL platforms are separated into the following types of packages. The following list of package types is sorted in order of minimality, starting with the most minimal.

Java Runtime Environment (JRE) headless

Provides the library only without support for graphical user interface but supports offline rendering of images

JRE

Adds the necessary libraries to run for full graphical clients

JDK

Includes tooling and compilers

Red Hat build of OpenJDK versions for Windows platforms do not support headless packages. However, the Red Hat build of OpenJDK packages for Windows platforms are also divided into JRE and JDK components, similar to the packages for RHEL platforms.

NOTE

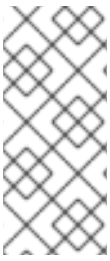
The upstream community version of OpenJDK 11 or later does not separate packages in this way and instead provides one monolithic JDK installation.

OpenJDK 9 introduced a modularised version of the JDK class libraries divided by their namespaces. From Red Hat build of OpenJDK 11 onward, these libraries are packaged into **jmods** modules. For more information, see Jmods.

2.10. JMODS

OpenJDK 9 introduced **jmods**, which is a modularized version of the JDK class libraries, where each module groups classes from a set of related packages. You can use the **jlink** tool to create derivative runtimes that include only some subset of the modules that are needed to run selected applications.

From Red Hat build of OpenJDK 11 onward, Red Hat build of OpenJDK versions for RHEL platforms place the **jmods** files into a separate RPM package that is not installed by default. If you want to create standalone OpenJDK images for your applications by using **jlink**, you must manually install the **jmods** package (for example, **java-11-openjdk-jmods**).



NOTE

On RHEL platforms, OpenJDK is dynamically linked against system libraries, which means the resulting **jlink** images are not portable across different versions of RHEL or other systems. If you want to ensure portability, you must use the portable builds of Red Hat build of OpenJDK that are released through the Red Hat Customer Portal. For more information, see Installing Red Hat build of OpenJDK on RHEL by using an archive .

2.11. DEPRECATED AND REMOVED FUNCTIONALITY IN RED HAT BUILD OF OPENJDK 11

Red Hat build of OpenJDK 11 has either deprecated or removed some features that Red Hat build of OpenJDK 8 supports.

CORBA

Red Hat build of OpenJDK 11 does not support the following Common Object Request Broker Architecture (CORBA) tools:

- **ldlj**
- **orbd**
- **servertool**
- **tnamesrv**

Logging framework

Red Hat build of OpenJDK 11 does not support the following APIs:

- **java.util.logging.LogManager.addPropertyChangeListener**
- **java.util.logging.LogManager.removePropertyChangeListener**
- **java.util.jar.Pack200.Packer.addPropertyChangeListener**
- **java.util.jar.Pack200.Packer.removePropertyChangeListener**
- **java.util.jar.Pack200.Unpacker.addPropertyChangeListener**
- **java.util.jar.Pack200.Unpacker.removePropertyChangeListener**

Java EE modules

Red Hat build of OpenJDK 11 does not support the following APIs:

- **java.activation**
- **java.corba**
- **java.se.ee (aggregator)**
- **java.transaction**
- **java.xml.bind**
- **java.xml.ws**
- **java.xml.ws.annotation**

java.awt.peer

Red Hat build of OpenJDK 11 sets the **java.awt.peer** package as internal, which means that applications cannot automatically access this package by default. Because of this change, Red Hat build of OpenJDK 11 removed a number of classes and methods that refer to the peer API, such as the **Component.getPeer** method.

The following list outlines the most common use cases for the peer API:

- Writing of new graphics ports
- Checking if a component can be displayed

- Checking if a component is either lightweight or backed by an operating system native UI component resource such as an Xlib XWindow

From Java 1.1 onward, the **Component.isDisplayable()** method provides the functionality to check whether a component can be displayed. From Java 1.2 onward, the **Component.isLightweight()** method provides the functionality to check whether a component is lightweight.

javax.security and java.lang APIs

Red Hat build of OpenJDK 11 does not support the following APIs:

- **javax.security.auth.Policy**
- **java.lang.Runtime.runFinalizersOnExit(boolean)**
- **java.lang.SecurityManager.checkAwtEventQueueAccess()**
- **java.lang.SecurityManager.checkMemberAccess(java.lang.Class,int)**
- **java.lang.SecurityManager.checkSystemClipboardAccess()**
- **java.lang.SecurityManager.checkTopLevelWindow(java.lang.Object)**
- **java.lang.System.runFinalizersOnExit(boolean)**
- **java.lang.Thread.destroy()**
- **java.lang.Thread.stop(java.lang.Throwable)**

Sun.misc

The **sun.misc package** has always been considered internal and unsupported. In Red Hat build of OpenJDK 11, the following packages are deprecated or removed:

- **sun.misc.BASE64Encoder**
- **sun.misc.BASE64Decoder**
- **sun.misc.Unsafe**
- **sun.reflect.Reflection**

Consider the following information:

- Red Hat build of OpenJDK 8 added the **java.util.Base64** package as a replacement for the **sun.misc.BASE64Encoder** and **sun.misc.BASE64Decoder** APIs. You can use the **java.util.Base64** package rather than these APIs, which have been removed from Red Hat build of OpenJDK 11.
- Red Hat build of OpenJDK 11 deprecates the **sun.misc.Unsafe** package, which is scheduled for removal. For more information about a new set of APIs that you can use as a replacement for **sun.misc.Unsafe**, see JEP 193.
- Red Hat build of OpenJDK 11 removes the **sun.reflect.Reflection** package. For more information about new functionality for stack walking that replaces the **sun.reflect.Reflection.getCallerClass** method, see JEP 259.

Additional resources

Additional resources

- For more information about the removed Java EE and CORBA modules and the potential replacements for these modules, see JEP 320: Remove the Java EE and CORBA Modules .

2.12. ADDITIONAL RESOURCES (OR NEXT STEPS)

- For more information about Red Hat build of OpenJDK 8 features, see JDK 8 Features.
- For more information about OpenJDK 9 features inherited by Red Hat build of OpenJDK 11, see JDK 9.
- For more information about OpenJDK 10 features inherited by Red Hat build of OpenJDK 11, see JDK 10.
- For more information about Red Hat build of OpenJDK 11 features, see JDK 11.
- For more information about a list of all available JEPs, see JEP 0: JEP Index .

CHAPTER 3. PREPARATION FOR MIGRATION

Red Hat build of OpenJDK 11 includes changes that might require you to reconfigure your applications, which were already successfully deployed on Red Hat build of OpenJDK 8.

You can ensure an effective migration plan by completing the following tasks:

- Review the Major differences between Red Hat build of OpenJDK 8 and Red Hat build of OpenJDK 11 section.
- Integrate the differences into your migration plan.

Red Hat provides a migration toolkit for applications (MTA) tool that you can use to help with your migration tasks. You can use the MTA tool to migrate Java applications from Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11.

Additional resources

- For more information about the installation of Red Hat build of OpenJDK 11 on RHEL, see the *Installing and using Red Hat build of OpenJDK 11 on RHEL* guide.
- For more information about the installation of Red Hat build of OpenJDK 11 on Microsoft Windows, see the *Installing and using Red Hat build of OpenJDK 11 for Windows* guide.
- For more information about switching between Red Hat build of OpenJDK versions on RHEL, see Interactively selecting a system-wide Red Hat build of OpenJDK version of RHEL in the *Configuring Red Hat build of OpenJDK 11 on RHEL* guide.
- For more information about switching between Red Hat build of OpenJDK versions on Microsoft Windows, see Selecting a specific Red Hat build of OpenJDK from the installed versions of an application in the *Configuring Red Hat build of OpenJDK 11 on Microsoft Windows* guide.
- For more information about the MTA tool, see the *Introduction to the Migration Toolkit for Applications* guide.

CHAPTER 4. TOOLS FOR APPLICATION MIGRATION

Before you migrate your applications from Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11, you can use tools to test the suitability of your application to run on Red Hat build of OpenJDK 11.

You can use the following steps to enhance your testing process:

- Update third-party libraries.
- Compile your application code.
- Run **jdeps** on your application's code.
- Use the migration toolkit for applications (MTA) tool to migrate Java applications from Red Hat build of OpenJDK 8 to Red Hat build of OpenJDK 11.

Additional resources

- For more information about the MTA tool, see the *Introduction to the Migration Toolkit for Applications* guide.