



Red Hat build of Quarkus 3.2

Developing and compiling your Red Hat build of Quarkus applications with Apache Maven

Red Hat build of Quarkus 3.2 Developing and compiling your Red Hat build of Quarkus applications with Apache Maven

Legal Notice

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide describes how to develop and compile Red Hat build of Quarkus applications by using the Apache Maven tool.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	3
CHAPTER 1. DEVELOPING AND COMPILING YOUR RED HAT BUILD OF QUARKUS APPLICATIONS WITH APACHE MAVEN	4
1.1. ABOUT RED HAT BUILD OF QUARKUS	4
1.2. ABOUT APACHE MAVEN AND RED HAT BUILD OF QUARKUS	4
1.2.1. Configuring the Maven settings.xml file for the online repository	5
1.3. CREATING A RED HAT BUILD OF QUARKUS PROJECT ON THE COMMAND LINE	6
1.4. CREATING A RED HAT BUILD OF QUARKUS PROJECT BY CONFIGURING THE POM.XML FILE	9
1.5. CREATING THE GETTING STARTED PROJECT BY USING CODE.QUARKUS.REDHAT.COM	11
1.6. CONFIGURING THE JAVA COMPILER	14
1.7. INSTALLING AND MANAGING EXTENSIONS	15
1.8. IMPORTING YOUR PROJECT INTO AN IDE	16
1.9. CONFIGURING THE RED HAT BUILD OF QUARKUS PROJECT OUTPUT	18
1.10. TESTING YOUR RED HAT BUILD OF QUARKUS APPLICATION IN JVM MODE WITH A CUSTOM PROFILE	19
1.11. LOGGING THE RED HAT BUILD OF QUARKUS APPLICATION BUILD CLASSPATH TREE	20
1.12. PRODUCING A NATIVE EXECUTABLE	21
1.12.1. Producing a native executable by using an in-container build	22
1.12.2. Producing a native executable by using a local-host build	23
1.12.3. Creating a container manually	24
1.13. TESTING THE NATIVE EXECUTABLE	26
1.14. USING RED HAT BUILD OF QUARKUS DEVELOPMENT MODE	29
1.15. DEBUGGING YOUR RED HAT BUILD OF QUARKUS PROJECT	30
1.16. ADDITIONAL RESOURCES	31

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. DEVELOPING AND COMPILING YOUR RED HAT BUILD OF QUARKUS APPLICATIONS WITH APACHE MAVEN

As an application developer, you can use Red Hat build of Quarkus to create microservices-based applications written in Java that run on OpenShift Container Platform and serverless environments. Applications compiled to native executables have small memory footprints and fast startup times.

Use the Quarkus Apache Maven plugin to create a Red Hat build of Quarkus project.



NOTE

Where applicable, alternative instructions for using the Quarkus command-line interface (CLI) are provided. The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Prerequisites

- You have installed OpenJDK 11 or 17.
 - To download Red Hat build of OpenJDK, log in to the Red Hat Customer Portal and go to [Software Downloads](#).
- You have set the **JAVA_HOME** environment variable to specify the location of the Java SDK.
- You have installed Apache Maven 3.8.6 or later.
 - To download Maven, go to the [Apache Maven Project](#) website.

1.1. ABOUT RED HAT BUILD OF QUARKUS

Red Hat build of Quarkus is a Kubernetes-native Java stack optimized for containers and Red Hat OpenShift Container Platform. Quarkus is designed to work with popular Java standards, frameworks, and libraries such as Eclipse MicroProfile, Eclipse Vert.x, Apache Camel, Apache Kafka, Hibernate ORM with Jakarta Persistence, and RESTEasy Reactive (Jakarta REST).

As a developer, you can choose the Java frameworks you want for your Java applications, which you can run in Java Virtual Machine (JVM) mode or compile and run in native mode. Quarkus provides a container-first approach to building Java applications. The container-first approach facilitates the containerization and efficient execution of microservices and functions. For this reason, Quarkus applications have a smaller memory footprint and faster startup times.

Quarkus also optimizes the application development process with capabilities such as unified configuration, automatic provisioning of unconfigured services, live coding, and continuous testing that gives you instant feedback on your code changes.

For information about the differences between the Quarkus community version and Red Hat build of Quarkus, see [Differences between the Red Hat build of Quarkus community version and Red Hat build of Quarkus](#).

1.2. ABOUT APACHE MAVEN AND RED HAT BUILD OF QUARKUS

Apache Maven is a distributed build automation tool that is used in Java application development to create, manage, and build software projects. Maven uses standard configuration files called Project Object Model (POM) files to define projects and manage the build process. POM files describe the

module and component dependencies, build order, and targets for the resulting project packaging and output by using an XML file, ensuring that the project gets built correctly and uniformly.

Maven repositories

A Maven repository stores Java libraries, plugins, and other build artifacts. The default public repository is the Maven 2 Central Repository, but repositories can be private and internal within a company to share common artifacts among development teams. Repositories are also available from third parties.

You can use the Red Hat-hosted Maven repository with your Quarkus projects, or you can download the Red Hat build of Quarkus Maven repository.

Maven plugins

Maven plugins are defined parts of a POM file that run one or more tasks. Red Hat build of Quarkus applications use the following Maven plugins:

- *Quarkus Maven plugin* (***quarkus-maven-plugin***): Enables Maven to create Quarkus projects, packages your applications into JAR files, and provides a dev mode.
- *Maven Surefire plugin* (***maven-surefire-plugin***): When Quarkus enables the **test** profile, the Maven Surefire plugin is used during the **test** phase of the build lifecycle to run unit tests on your application. The plugin generates text and XML files that contain the test reports.

Additional resources

- [Configuring your Red Hat build of Quarkus applications](#)

1.2.1. Configuring the Maven settings.xml file for the online repository

To use the Red Hat-hosted Quarkus repository with your Quarkus Maven project, configure the **settings.xml** file for your user. Maven settings that are used with a repository manager or a repository on a shared server offer better control and manageability of projects.

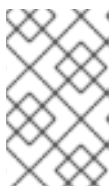


NOTE

When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects. If you want to apply the configuration to a specific project only, use the **-s** option and specify the path to the project-specific **settings.xml** file.

Procedure

1. Open the Maven **\$HOME/.m2/settings.xml** file in a text editor or an integrated development environment (IDE).



NOTE

If no **settings.xml** file is present in the **\$HOME/.m2/** directory, copy the **settings.xml** file from the **\$MAVEN_HOME/conf/** directory into the **\$HOME/.m2/** directory.

2. Add the following lines to the **<profiles>** element of the **settings.xml** file:

```
<!-- Configure the Red Hat build of Quarkus Maven repository -->
```

```

<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>>true</enabled>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>https://maven.repository.redhat.com/ga/</url>
      <releases>
        <enabled>>true</enabled>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>

```

3. Add the following lines to the `<activeProfiles>` element of the `settings.xml` file and save the file.

```
<activeProfile>red-hat-enterprise-maven-repository</activeProfile>
```

1.3. CREATING A RED HAT BUILD OF QUARKUS PROJECT ON THE COMMAND LINE

Use the Red Hat build of Quarkus Maven plugin on the command line to create a Quarkus project by providing attributes and values on the command line or by using the plugin in interactive mode. You can also create a Quarkus project by using the Quarkus command-line interface (CLI). The resulting project includes the following elements:

- The Maven structure
- An associated unit test
- A landing page that is accessible on **http://localhost:8080** after you start the application
- Example **Dockerfile** files for JVM and native mode in **src/main/docker**
- The application configuration file

Prerequisites

- You have installed OpenJDK 11 or 17.

- To download Red Hat build of OpenJDK, log in to the Red Hat Customer Portal and go to [Software Downloads](#).
- You have set the **JAVA_HOME** environment variable to specify the location of the Java SDK.
- You have installed Apache Maven 3.8.6 or later.
 - To download Maven, go to the [Apache Maven Project](#) website.
- You have installed the Quarkus command-line interface (CLI), which is one of the methods you can use to create a Quarkus project. For more information, see [Installing the Quarkus CLI](#).



NOTE

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Procedure

1. In a command terminal, enter the following command to verify that Maven is using OpenJDK 11 or 17 and that the Maven version is 3.8.6 or later:

```
mvn --version
```

2. If the preceding command does not return OpenJDK 11 or 17, add the path to OpenJDK 11 or 17 to the PATH environment variable and enter the preceding command again.
3. To use the Quarkus Maven plugin to create a project, use one of the following methods:
 - Enter the following command:

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.2.11.Final-redhat-00001:create \
  -DprojectId=<project_group_id> \
  -DprojectId=<project_artifact_id> \
  -DplatformGroupId=com.redhat.quarkus.platform \
  -DplatformArtifactId=quarkus-bom \
  -DplatformVersion=3.2.11.Final-redhat-00001 \
  -DpackageName=getting.started
```

In this command, replace the following values:

- **<project_group_id>**: A unique identifier of your project
- **<project_artifact_id>**: The name of your project and your project directory
- Create the project in interactive mode:

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.2.11.Final-redhat-00001:create
```

When prompted, enter the required attribute values.



NOTE

You can also create your project by using the default values for the project attributes by entering the following command:

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:3.2.11.Final-redhat-00001:create -B
```

- Create the project by using the Red Hat build of Quarkus CLI:

```
quarkus create app my-groupId:my-artifactId --package-name=getting.started
```

- You can also get the list of available options with:

```
quarkus create app --help
```

The following table lists the attributes that you can define with the **create** command:

Attribute	Default Value	Description
projectGroupId	org.acme	A unique identifier of your project.
projectArtifactId	code-with-quarkus	The name of your project and your project directory. If you do not specify the projectArtifactId attribute, the Maven plugin starts the interactive mode. If the directory already exists, the generation fails.
projectVersion	1.0-SNAPSHOT	The version of your project.
platformGroupId	com.redhat.quarkus.platform	The group ID of your platform. All the existing platforms are provided by com.redhat.quarkus.platform . However, you can change the default value.
platformArtifactId	quarkus-bom	The artifact ID of your platform BOM.
platformVersion	The latest platform version, for example, 3.2.11.Final-redhat-00001 .	The version of the platform you want to use for your project. When you provide a version range, the Maven plugin uses the latest version.
packageName	[]	The name of the getting started package, getting.started .

Attribute	Default Value	Description
extensions	[]	The list of extensions you want to add to your project, separated by a comma.



NOTE

By default, the Quarkus Maven plugin uses the latest **quarkus-bom** file. The **quarkus-bom** file aggregates extensions so that you can reference them from your applications to align the dependency versions. When you are offline, the Quarkus Maven plugin uses the latest locally available version of the **quarkus-bom** file. If Maven finds the **quarkus-bom** version 2.0 or earlier, it uses the platform based on the **quarkus-bom**.

1.4. CREATING A RED HAT BUILD OF QUARKUS PROJECT BY CONFIGURING THE POM.XML FILE

You can create a Quarkus project by configuring the Maven **pom.xml** file.

Procedure

1. Open the **pom.xml** file in a text editor.
2. Add the configuration properties that contain the following items:
 - The Maven Compiler Plugin version
 - The Quarkus BOM **groupId**, **artifactId**, and **version**
 - The Maven Surefire Plugin version
 - The **skipITs** property.

```
<properties>
  <compiler-plugin.version>3.11.0</compiler-plugin.version>
  <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
  <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
  <quarkus.platform.version>3.2.11.Final-redhat-00001</quarkus.platform.version>
  <surefire-plugin.version>3.1.2</surefire-plugin.version>
  <skipITs>true</skipITs>
</properties>
```

3. Add the Quarkus GAV (group, artifact, version) and use the **quarkus-bom** file to omit the versions of the different Quarkus dependencies:

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>${quarkus.platform.artifact-id}</artifactId>
      <version>${quarkus.platform.version}</version>
      <type>pom</type>
```

```

    <scope>import</scope>
  </dependency>
</dependencies>
</dependencyManagement>

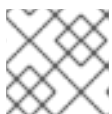
```

4. Add the Quarkus Maven plugin, the Maven Compiler plugin, and the Maven Surefire plugin:

```

<build>
  <plugins>
    <plugin>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus.platform.version}</version>
      <extensions>true</extensions>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
            <goal>generate-code</goal>
            <goal>generate-code-tests</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler-plugin.version}</version>
      <configuration>
        <compilerArgs>
          <arg>-parameters</arg>
        </compilerArgs>
      </configuration>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemPropertyVariables>
          <java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
          <maven.home>${maven.home}</maven.home>
        </systemPropertyVariables>
      </configuration>
    </plugin>
  </plugins>
</build>

```



NOTE

The **maven-surefire-plugin** runs the unit tests for your application.

5. Optional: To build a native application, add a specific native profile that includes the **Maven Failsafe Plugin**:

```

<build>
  <plugins>
    ...
    <plugin>
      <artifactId>maven-failsafe-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>integration-test</goal>
            <goal>verify</goal>
          </goals>
          <configuration>
            <systemPropertyVariables>
              <native.image.path>${project.build.directory}/${project.build.finalName}-
runner
              </native.image.path>

            <java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
              <maven.home>${maven.home}</maven.home>
            </systemPropertyVariables>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
...
<profiles>
  <profile>
    <id>native</id>
    <activation>
      <property>
        <name>native</name>
      </property>
    </activation>
    <properties>
      <skipITs>>false</skipITs>
      <quarkus.package.type>native</quarkus.package.type>
    </properties>
  </profile>
</profiles>

```

- Tests that include **IT** in their names and contain the **@NativeImageTest** annotation are run against the native executable.
- For more details about how native mode differs from JVM mode, see [Difference between JVM and native mode](#) in the Quarkus "Getting Started" guide.

1.5. CREATING THE GETTING STARTED PROJECT BY USING CODE.QUARKUS.REDHAT.COM

As an application developer, you can use `code.quarkus.redhat.com` to generate a Quarkus Maven project and automatically add and configure the extensions that you want to use in your application. In addition, `code.quarkus.redhat.com` automatically manages the configuration parameters that are required to

compile your project into a native executable.

You can generate a Quarkus Maven project, including the following activities:

- Specifying basic details about your application
- Choosing the extensions that you want to include in your project
- Generating a downloadable archive with your project files
- Using custom commands for compiling and starting your application

Prerequisites

- You have a web browser.
- You have prepared your environment to use Apache Maven. For more information, see [Preparing your environment](#).
- You have configured your Quarkus Maven repository. To create a Quarkus application with Maven, use the Red Hat-hosted Quarkus repository. For more information, see [Configuring the Maven settings.xml file for the online repository](#).
- **Optional:** You have installed the Quarkus command-line interface (CLI), which is one of the methods you can use to start Quarkus in dev mode. For more information, see [Installing the Quarkus CLI](#).



NOTE

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Procedure

1. On your web browser, navigate to <https://code.quarkus.redhat.com>.
2. Specify basic details about your project:

The screenshot shows a form titled "CONFIGURE YOUR APPLICATION" with three input fields:

Group	org.acme
Artifact	code-with-quarkus
Build Tool	Maven

- a. Enter a group name for your project. The name format follows the Java package naming convention; for example, **org.acme**.

- b. Enter a name for the Maven artifacts generated by your project, such as **code-with-quarkus**.
- c. Select the build tool you want to use to compile and start your application. The build tool that you choose determines the following setups:
 - The directory structure of your generated project
 - The format of configuration files that are used in your generated project
 - The custom build script and command for compiling and starting your application that code.quarkus.redhat.com displays for you after you generate your project

**NOTE**

Red Hat provides support for using code.quarkus.redhat.com to create Quarkus Maven projects only.

3. Specify additional details about your application project:
 - a. To display the fields that contain further application details, select *More options*.
 - b. Enter a version you want to use for artifacts generated by your project. The default value of this field is **1.0.0-SNAPSHOT**. Using [semantic versioning](#) is recommended; however, you can choose to specify a different type of versioning.
 - c. Select whether you want code.quarkus.redhat.com to add starter code to your project. When you add extensions that are marked with "STARTER-CODE" to your project, you can enable this option to automatically create example class and resource files for those extensions when you generate your project. However, this option does not affect your generated project if you do not add any extensions that provide an example code.

CONFIGURE YOUR APPLICATION

Group	org.acme	Version	1.0.0-SNAPSHOT
Artifact	code-with-quarkus	Java Version	11 ▼
Build Tool	Maven ▼	Starter Code	Yes

CLOSE


**NOTE**

The code.quarkus.redhat.com application automatically uses the latest release of Red Hat build of Quarkus. However, should you require, it is possible to manually change to an earlier BOM version in the pom.xml file after you generate your project, but this is not recommended.







4. Select the extensions that you want to use. The extensions you select are included as dependencies of your Quarkus application. The Quarkus platform also ensures these extensions are compatible with future versions.

**IMPORTANT**

Do not use the **RESTEasy** and the **RESTEasy Reactive** extensions in the same project.

The quark icon () next to an extension indicates that the extension is part of the Red Hat build of Quarkus platform release. Red Hat recommends using extensions from the same platform because they are tested and verified together and are therefore easier to use and upgrade.

You can enable the option to automatically generate starter code for extensions marked with "STARTER-CODE".

Web	
<input type="checkbox"/>	RESTEasy JAX-RS [quarkus-resteasy]  STARTER-CODE SUPPORTED
	REST endpoint framework implementing JAX-RS and more
<input type="checkbox"/>	RESTEasy Jackson [quarkus-resteasy-jackson]  SUPPORTED
	Jackson serialization support for RESTEasy
<input type="checkbox"/>	RESTEasy JSON-B [quarkus-resteasy-jsonb]  SUPPORTED
	JSON-B serialization support for RESTEasy
<input type="checkbox"/>	Eclipse Vert.x GraphQL [quarkus-vertx-graphql]  TECH-PREVIEW
	Query the API using GraphQL
<input type="checkbox"/>	gRPC [quarkus-grpc]  STARTER-CODE SUPPORTED
	Serve and consume gRPC services
<input type="checkbox"/>	Hibernate Validator [quarkus-hibernate-validator]  SUPPORTED
	Validate object properties (field, getter) and method parameters for your beans (REST, CDI, JPA)
<input type="checkbox"/>	Mutiny support for REST Client [quarkus-rest-client-mutiny]  TECH-PREVIEW PREVIEW
	Enable Mutiny for the REST client
<input type="checkbox"/>	Reactive Routes [quarkus-reactive-routes]  SUPPORTED
	REST framework offering the route model to define non blocking endpoints
<input type="checkbox"/>	REST Client [quarkus-rest-client]  STARTER-CODE SUPPORTED

- To confirm your choices, select **Generate your application**. The following items are displayed:
 - A link to download the archive that contains your generated project
 - A custom command that you can use to compile and start your application
- To save the archive with the generated project files to your machine, select **Download the ZIP**.
- Extract the contents of the archive.
- Go to the directory that contains your extracted project files:

```
cd <directory_name>
```

- To compile and start your application in dev mode, use one of the following ways:

- Using Maven:

```
./mvnw quarkus:dev
```

- Using the Quarkus CLI:

```
quarkus dev
```

Additional resources

[Support levels for Red Hat build of Quarkus extensions](#)

1.6. CONFIGURING THE JAVA COMPILER

By default, the Quarkus Maven plugin passes compiler flags to **javac** command from **maven-compiler-plugin**.

Procedure

- To customize the compiler flags used in development mode, add a **configuration** section to the **plugin** block and set the **compilerArgs** property. You can also set **source**, **target**, and **jvmArgs**. For example, to pass **-verbose** to the JVM and **javac** commands, add the following configuration:

```
<plugin>
  <groupId>com.redhat.quarkus.platform</groupId>
  <artifactId>quarkus-maven-plugin</artifactId>
  <version>${quarkus.platform.version}</version>

  <configuration>
    <source>${maven.compiler.source}</source>
    <target>${maven.compiler.target}</target>
    <compilerArgs>
      <arg>-verbose</arg>
    </compilerArgs>
    <jvmArgs>-verbose</jvmArgs>
  </configuration>

  ...
</plugin>
```

1.7. INSTALLING AND MANAGING EXTENSIONS

In Red Hat build of Quarkus, you can use extensions to expand your application's functionality and configure, boot, and integrate a framework into your application. This procedure shows you how to find and add extensions to your Quarkus project.

Prerequisites

- You have created a Quarkus Maven project.
- You have installed the Quarkus command-line interface (CLI), which is one of the methods you can use to manage your Quarkus extensions. For more information, see [Installing the Quarkus CLI](#).



NOTE

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Procedure

- Navigate to your Quarkus project directory.
- List all of the available extensions by using one of the following ways:
 - Using Maven:


```
./mvnw quarkus:list-extensions
```
 - Using the Quarkus CLI:


```
-
```

```
quarkus extension --installable
```

3. Add an extension to your project by using one of the following ways:

- Using Maven, enter the following command where **<extension>** is the group, artifact, and version (GAV) of the extension that you want to add:

```
./mvnw quarkus:add-extension -Dextensions="<extension>"
```

For example, to add the Agroal extension, enter the following command:

```
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-agroal"
```

- Using the Quarkus CLI, enter the following command where **<extension>** is the group, artifact, and version (GAV) of the extension that you want to add:

```
quarkus extension add '<extension>'
```

4. To search for a specific extension, enter the extension name or partial name after **-Dextensions=**. The following example searches for extensions that contain the text **agroal** in the name:

```
./mvnw quarkus:add-extension -Dextensions=agroal
```

This command returns the following result:

```
[SUCCESS] Extension io.quarkus:quarkus-agroal has been installed
```

Similarly, with the Quarkus CLI, you might enter:

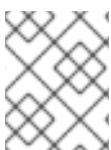
```
quarkus extension add 'agroal'
```

1.8. IMPORTING YOUR PROJECT INTO AN IDE

Although you can develop your Red Hat build of Quarkus project in a text editor, you might find using an integrated development environment (IDE) easier. The following instructions show you how to import your project into specific IDEs.

Prerequisites

- You have a Quarkus Maven project.
- You have installed the Quarkus command-line interface (CLI), which is required to start your project in dev mode. For more information, see [Installing the Quarkus CLI](#).



NOTE

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Procedure

Complete the required procedure for your IDE.

CodeReady Studio or Eclipse

1. In CodeReady Studio or Eclipse, click **File**>***Import***.
2. Select **Maven** → **Existing Maven Project**
3. Next, select the root location of the project. A list of the available modules appears.
4. Select the generated project, and click **Finish**.
5. To compile and start your application, use one of the following ways:

- Using Maven:

```
┆ ./mvnw quarkus:dev
```

- Using the Quarkus CLI:

```
┆ quarkus dev
```

IntelliJ

1. In IntelliJ, complete one of the following tasks:
 - Select **File** > **New** > **Project From Existing Sources**
 - On the **Welcome** page, select **Import project**.
2. Select the project root directory.
3. Select **Import project from external model**, and then select **Maven**.
4. Review the options, and then click **Next**.
5. Click **Create**.
6. To compile and start your application, use one of the following ways:

- Using Maven:

```
┆ ./mvnw quarkus:dev
```

- Using the Quarkus CLI:

```
┆ quarkus dev
```

Apache NetBeans

1. Select **File** > **Open Project**.
2. Select the project **root** directory.
3. Click **Open Project**.

4. To compile and start your application, use one of the following ways:

- Using Maven:

```
./mvnw quarkus:dev
```

- Using the Quarkus CLI:

```
quarkus dev
```

Visual Studio Code

1. Install the Java Extension Pack.
2. In Visual Studio Code, open your project directory.

Verification

The project loads as a Maven project.

1.9. CONFIGURING THE RED HAT BUILD OF QUARKUS PROJECT OUTPUT

Before you build your application, you can control the build command output by changing the default values of the properties in the **application.properties** file.

Prerequisites

- You have created a Quarkus Maven project.

Procedure

1. Go to the **{project}/src/main/resources** folder, and open the **application.properties** file in a text editor.
2. Edit the values of properties that you want to change and save the file.
The following table lists the properties that you can change:

Property	Description	Type	Default
quarkus.package.main-class	The entry point of the application. In most cases, you must change this value.	string	io.quarkus.runner.GeneratedMain
quarkus.package.type	The requested output type for the package, which you can set to 'jar' (uses 'fast-jar'), 'legacy-jar' for the pre-1.12 default jar packaging, 'uber-jar', 'native', or 'native-sources'.	string	jar

Property	Description	Type	Default
quarkus.package.manifest.add-implementation-entries	Determines whether the implementation information must be included in the runner JAR file's MANIFEST.MF file.	boolean	true
quarkus.package.user-configured-ignored-entries	Files that must not be copied to the output artifact.	string (list)	
quarkus.package.runner-suffix	The suffix that is applied to the runner JAR file.	string	-runner
quarkus.package.output-directory	The output folder for the application build. This is resolved relative to the build system target directory.	string	
quarkus.package.output-name	The name of the final artifact.	string	

1.10. TESTING YOUR RED HAT BUILD OF QUARKUS APPLICATION IN JVM MODE WITH A CUSTOM PROFILE

Similar to any other running mode, configuration values for testing are read from the **src/main/resources/application.properties** file.

By default, the **test** profile is active during testing in JVM mode, meaning that properties prefixed with **%test** take precedence. For example, when you run a test with the following configuration, the value returned for the property **message** is **Test Value**.

```
message=Hello
%test.message=Test Value
```

If the **%test** profile is inactive (dev, prod), the value returned for the property **message** is **Hello**.

For example, your application might require multiple test profiles to run a set of tests against different database instances. To do this, you must override the testing profile name, which can be done by setting the system property **quarkus.test.profile** when executing Maven. By doing so, you can control which sets of configuration values are active during the test.

To learn more about standard testing with the 'Starting With Quarkus' example, see [Testing your Red Hat build of Quarkus application with JUnit](#) in the Getting Started guide.

Prerequisites

- A Quarkus project created with Apache Maven.

Procedure

When running tests on a Quarkus application, the **test** configuration profile is set as active by default. However, you can change the profile to a custom profile by using the **quarkus.test.profile** system property.

1. Run the following command to test your application:

```
mvn test -Dquarkus.test.profile=__<profile-name>__
```



NOTE

You cannot use a custom test configuration profile in native mode. Native tests always run under the **prod** profile.

1.11. LOGGING THE RED HAT BUILD OF QUARKUS APPLICATION BUILD CLASSPATH TREE

The Quarkus build process adds deployment dependencies of the extensions that you use in the application to the original application classpath. You can see which dependencies and versions are included in the build classpath. The **quarkus-bootstrap** Maven plugin includes the **build-tree** goal, which displays the build dependency tree for the application.

Prerequisites

- You have created a Quarkus Maven application.

Procedure

- To list the build dependency tree of your application, enter the following command:

```
./mvnw quarkus:dependency-tree
```

Example output. The exact output you see will differ from this example.

```
[INFO] └─ io.quarkus:quarkus-resteasy-deployment:jar:3.2.11.Final-redhat-00001 (compile)
[INFO]   └─ io.quarkus:quarkus-resteasy-server-common-deployment:jar:3.2.11.Final-
redhat-00001 (compile)
[INFO]     └─ io.quarkus:quarkus-resteasy-common-deployment:jar:3.2.11.Final-redhat-
00001 (compile)
[INFO]       └─ io.quarkus:quarkus-resteasy-common:jar:3.2.11.Final-redhat-00001
(compile)
[INFO]         └─ org.jboss.resteasy:resteasy-core:jar:6.2.4.Final-redhat-00003 (compile)
[INFO]           └─ jakarta.xml.bind:jakarta.xml.bind-api:jar:4.0.0.redhat-00008 (compile)
[INFO]             └─ org.jboss.resteasy:resteasy-core-spi:jar:6.2.4.Final-redhat-00003
(compile)
[INFO]               └─ org.reactivestreams:reactive-streams:jar:1.0.4.redhat-00003
(compile)
[INFO]                 └─ com.ibm.async:asyncutil:jar:0.1.0.redhat-00010 (compile)
...

```




NOTE

The `mvn dependency:tree` command displays only the runtime dependencies of your application

1.12. PRODUCING A NATIVE EXECUTABLE

A native binary is an executable that is created to run on a specific operating system and CPU architecture.

The following list outlines some examples of a native executable:

- An ELF binary for Linux AMD 64 bits
- An EXE binary for Windows AMD 64 bits
- An ELF binary for ARM 64 bits

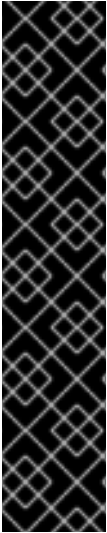
When you build a native executable, one advantage is that your application and dependencies, including the JVM, are packaged into a single file. The native executable for your application contains the following items:

- The compiled application code.
- The required Java libraries.
- A reduced version of the Java virtual machine (JVM) for improved application startup times and minimal disk and memory footprint, which is also tailored for the application code and its dependencies.

To produce a native executable from your Quarkus application, you can select either an in-container build or a local-host build. The following table explains the different building options that you can use:

Table 1.1. Building options for producing a native executable

Building option	Requires	Uses	Results in	Benefits
In-container build - Supported	A container runtime, for example, Podman or Docker	The default registry.access.redhat.com/quarkus/mandrel-23-rhel8:23.0 builder image	A Linux 64-bit executable using the CPU architecture of the host	GraalVM does not need to be set up locally, which makes your CI pipelines run more efficiently
Local-host build - Only supported upstream	A local installation of GraalVM or Mandrel	Its local installation as a default for the quarkus.native.builder-image property	An executable that has the same operating system and CPU architecture as the machine on which the build is executed	An alternative for developers that are not allowed or do not want to use tools such as Docker or Podman. Overall, it is faster than the in-container build approach.



IMPORTANT

- Red Hat build of Quarkus 3.2 only supports the building of native Linux executables by using a Java 17-based [Red Hat build of Quarkus Native builder](#) image, which is a productized distribution of [Mandrel](#). While other images are available in the community, they are not supported in the product, so you should not use them for production builds that you want Red Hat to provide support for.
- Applications whose source is written based on Java 11, with no Java 12 - 17 features used, can still compile a native executable of that application using the Java 17-based Mandrel 23.0 base image.
- Building native executables by using Oracle GraalVM Community Edition (CE), Mandrel community edition, or any other distributions of GraalVM is not supported for Red Hat build of Quarkus.

1.12.1. Producing a native executable by using an in-container build

To create a native executable and run the native image tests, use the **native** profile that is provided by Red Hat build of Quarkus for an in-container build.

Prerequisites

- Podman or Docker is installed.
- The container has access to at least 8GB of memory.

Procedure

1. Open the Getting Started project **pom.xml** file, and verify that the project includes the **native** profile:

```
<profiles>
  <profile>
    <id>native</id>
    <activation>
      <property>
        <name>native</name>
      </property>
    </activation>
    <properties>
      <skipITs>>false</skipITs>
      <quarkus.package.type>native</quarkus.package.type>
    </properties>
  </profile>
</profiles>
```

2. Build a native executable by using one of the following ways:
 - Using Maven:
 - For Docker:

```
./mvnw package -Dnative -Dquarkus.native.container-build=true
```

- For Podman:

```
./mvnw package -Dnative -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman
```

- Using the Quarkus CLI:

- For Docker:

```
quarkus build --native -Dquarkus.native.container-build=true
```

- For Podman:

```
quarkus build --native -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman
```

Step results

These commands create a ***-runner** binary in the **target** directory, where the following applies:

- The ***-runner** file is the built native binary produced by Quarkus.
- The **target** directory is a directory that Maven creates when you build a Maven application.



IMPORTANT

Compiling a Quarkus application to a native executable consumes a large amount of memory during analysis and optimization. You can limit the amount of memory used during native compilation by setting the **quarkus.native.native-image-xxmx** configuration property. Setting low memory limits might increase the build time.

3. To run the native executable, enter the following command:

```
./target/*-runner
```

Additional resources

- [Native executable configuration properties](#)

1.12.2. Producing a native executable by using a local-host build

If you are not using Docker or Podman, use the Quarkus local-host build option to create and run a native executable.

Using the local-host build approach is faster than using containers and is suitable for machines that use a Linux operating system.



IMPORTANT

Using the following procedure in production is not supported by Red Hat build of Quarkus. Use this method only when testing or as a backup approach when Docker or Podman is not available.

Prerequisites

- A local installation of Mandrel or GraalVm, correctly configured according to the [Building a native executable](#) guide.
 - Additionally, for a GraalVM installation, **native-image** must also be installed.

Procedure

1. For GraalVM or Mandrel, build a native executable by using one of the following ways:

- Using Maven:

```
./mvnw package -Dnative
```

- Using the Quarkus CLI:

```
quarkus build --native
```

Step results

These commands create a ***-runner** binary in the **target** directory, where the following applies:

- The ***-runner** file is the built native binary produced by Quarkus.
- The **target** directory is a directory that Maven creates when you build a Maven application.



NOTE

When you build the native executable, the **prod** profile is enabled unless modified in the **quarkus.profile** property.

2. Run the native executable:

```
./target/*-runner
```

Additional resources

For more information, see the [Producing a native executable](#) section of the "Building a native executable" guide in the Quarkus community.

1.12.3. Creating a container manually

This section shows you how to manually create a container image with your application for Linux AMD64. When you produce a native image by using the Quarkus Native container, the native image creates an executable that targets Linux AMD64. If your host operating system is different from Linux AMD64, you cannot run the binary directly and you need to create a container manually.

Your Quarkus Getting Started project includes a **Dockerfile.native** in the **src/main/docker** directory with the following content:

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.8
WORKDIR /work/
RUN chown 1001 /work \
    && chmod "g+rwX" /work \
    && chown 1001:root /work
COPY --chown=1001:root target/*-runner /work/application

EXPOSE 8080
USER 1001

ENTRYPOINT ["/application", "-Dquarkus.http.host=0.0.0.0"]
```

NOTE

Universal Base Image (UBI)

The following list displays the suitable images for use with Dockerfiles.

- Red Hat Universal Base Image 8 (UBI8). This base image is designed and engineered to be the base layer for all of your containerized applications, middleware, and utilities.

```
registry.access.redhat.com/ubi8/ubi:8.8
```
- Red Hat Universal Base Image 8 Minimal (UBI8-minimal). A stripped-down UBI8 image that uses [microdnf](#) as a package manager.

```
registry.access.redhat.com/ubi8/ubi-minimal:8.8
```
- All Red Hat Base images are available on the [Container images](#) catalog site.

Procedure

1. Build a native Linux executable by using one of the following methods:

- Docker:

```
./mvnw package -Dnative -Dquarkus.native.container-build=true
```

- Podman:

```
./mvnw package -Dnative -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman
```

2. Build the container image by using one of the following methods:

- Docker:

```
docker build -f src/main/docker/Dockerfile.native -t quarkus-quickstart/getting-started .
```

- Podman

```
podman build -f src/main/docker/Dockerfile.native -t quarkus-quickstart/getting-started .
```

3. Run the container by using one of the following methods:

- Docker:

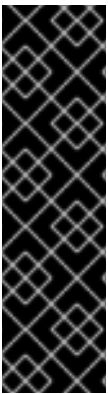
```
docker run -i --rm -p 8080:8080 quarkus-quickstart/getting-started
```

- Podman:

```
podman run -i --rm -p 8080:8080 quarkus-quickstart/getting-started
```

1.13. TESTING THE NATIVE EXECUTABLE

Test the application in native mode to test the functionality of the native executable. Use the `@QuarkusIntegrationTest` annotation to build the native executable and run tests against the HTTP endpoints.



IMPORTANT

The following example shows how to test a native executable with a local installation of GraalVM or Mandrel. Before you begin, consider the following points:

- This scenario is not supported by Red Hat build of Quarkus, as outlined in [Producing a native executable](#).
- The native executable you are testing with here must match the operating system and architecture of the host. Therefore, this procedure will not work on a macOS or an in-container build.

Procedure

1. Open the `pom.xml` file and verify that the `build` section has the following elements:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>${surefire-plugin.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
      <configuration>
        <systemPropertyVariables>
          <native.image.path>${project.build.directory}/${project.build.finalName}-
runner</native.image.path>
        </systemPropertyVariables>
      </configuration>
    </execution>
  </executions>
</plugin>
<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
  <maven.home>${maven.home}</maven.home>
</systemPropertyVariables>
</configuration>
```

```

    </execution>
  </executions>
</plugin>

```

- The Maven Failsafe plugin (**maven-failsafe-plugin**) runs the integration test and indicates the location of the native executable that is generated.
2. Open the **src/test/java/org/acme/GreetingResourceIT.java** file and verify that it includes the following content:

```

package org.acme;

import io.quarkus.test.junit.QuarkusIntegrationTest;

@QuarkusIntegrationTest 1
public class GreetingResourceIT extends GreetingResourceTest { 2

    // Execute the same tests but in native mode.
}

```

- 1** Use another test runner that starts the application from the native file before the tests. The executable is retrieved by using the **native.image.path** system property configured in the Maven Failsafe plugin.
- 2** This example extends the **GreetingResourceTest**, but you can also create a new test.

3. Run the test:

```
./mvnw verify -Dnative
```

The following example shows the output of this command:

```

./mvnw verify -Dnative
....

GraalVM Native Image: Generating 'getting-started-1.0.0-SNAPSHOT-runner' (executable)...
=====
=====
[1/8] Initializing... (6.6s @ 0.22GB)
Java version: 17.0.7+7, vendor version: Mandrel-23.0.0.0-Final
Graal compiler: optimization level: 2, target machine: x86-64-v3
C compiler: gcc (redhat, x86_64, 13.2.1)
Garbage collector: Serial GC (max heap size: 80% of RAM)
2 user-specific feature(s)
- io.quarkus.runner.Feature: Auto-generated class by Red Hat build of Quarkus from the
existing extensions
- io.quarkus.runtime.graal.DisableLoggingFeature: Disables INFO logging during the
analysis phase
[2/8] Performing analysis... [*****] (40.0s @
2.05GB)
10,318 (86.40%) of 11,942 types reachable
15,064 (57.36%) of 26,260 fields reachable
52,128 (55.75%) of 93,501 methods reachable
3,298 types, 109 fields, and 2,698 methods registered for reflection

```


**NOTE**

Quarkus waits 60 seconds for the native image to start before automatically failing the native tests. You can change this duration by configuring the **quarkus.test.wait-time** system property.

You can extend the wait time by using the following command where **<duration>** is the wait time in seconds:

```
./mvnw verify -Dnative -Dquarkus.test.wait-time=<duration>
```

**NOTE**

- Native tests run using the **prod** profile by default unless modified in the **quarkus.test.native-image-profile** property.

1.14. USING RED HAT BUILD OF QUARKUS DEVELOPMENT MODE

Development mode enables hot deployment with background compilation, which means that when you modify your Java or resource files and then refresh your browser, the changes automatically take effect. This also works for resource files such as the configuration property file. You can use either Maven or the Quarkus command-line interface (CLI) to start Quarkus in development mode.

Prerequisites

- You have created a Quarkus Maven application.
- You have installed the Quarkus CLI, which is one of the methods you can use to start Quarkus in development mode. For more information, see [Installing the Quarkus CLI](#).

**NOTE**

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Procedure

1. Switch to the directory that contains your Quarkus application **pom.xml** file.
2. To compile and start your Quarkus application in development mode, use one of the following methods:
 - Using Maven:


```
./mvnw quarkus:dev
```
 - Using the Quarkus CLI:


```
quarkus dev
```
3. Make changes to your application and save the files.
4. Refresh the browser to trigger a scan of the workspace.

If any changes are detected, the Java files are recompiled and the application is redeployed. Your request is then serviced by the redeployed application. If there are any issues with compilation or deployment, an error page appears.

In development mode, the debugger is activated and listens on port **5005**.

- Optional: To wait for the debugger to attach before running the application, include **-Dsuspend**:

```
./mvnw quarkus:dev -Dsuspend
```

- Optional: To prevent the debugger from running, include **-Ddebug=false**:

```
./mvnw quarkus:dev -Ddebug=false
```

1.15. DEBUGGING YOUR RED HAT BUILD OF QUARKUS PROJECT

When Red Hat build of Quarkus starts in development mode, debugging is enabled by default, and the debugger listens on port **5005** without suspending the JVM. You can enable and configure the debugging feature of Quarkus from the command line or by configuring the system properties. You can also use the Quarkus CLI to debug your project.

Prerequisites

- You have created a Red Hat build of Quarkus Maven project.
- You have installed the Quarkus command-line interface (CLI), which is one of the methods you can use to compile and debug your project. For more information, see [Installing the Quarkus CLI](#).



NOTE

The Quarkus CLI is intended for dev mode only. Red Hat does not support using the [Quarkus CLI](#) in production environments.

Procedure

Use one of the following methods to control debugging:

Controlling the debugger by configuring system properties

- Change one of the following values of the **debug** system property where **PORT** is the port that the debugger is listening on:
 - false**: The JVM starts with debug mode disabled.
 - true**: The JVM starts in debug mode and is listening on port **5005**.
 - client**: The JVM starts in client mode and tries to connect to **localhost:5005**.
 - PORT**: The JVM starts in debug mode and is listening on **PORT**.
- To suspend the JVM while running in debug mode, set the value of the **suspend** system property to one of the following values:

- **y** or **true**: The debug mode JVM launch suspends.
- **n** or **false**: The debug mode JVM starts without suspending.

Controlling the debugger from the command line

- To compile and start your Quarkus application in debug mode with a suspended JVM, use one of the following ways
 - Using Maven:

```
./mvnw quarkus:dev -Dsuspend
```

- Using the Quarkus CLI:

```
quarkus dev -Dsuspend
```

Enabling the debugger for specific host network interfaces

In development mode, by default, for security reasons, Quarkus sets the debug host interface to **localhost**.

To enable the debugger for a specific host network interface, you can use the **-DdebugHost** option by using one of the following ways:

- Using Maven:

```
./mvnw quarkus:dev -DdebugHost=<host-ip-address>
```

- Using the Quarkus CLI:

```
quarkus dev -DdebugHost=<host-ip-address>
```

Where **<host-ip-address>** is the IP address of the host network interface that you want to enable debugging on.



NOTE

To enable debugging on all host interfaces, replace **<host-ip-address>** with the following value:

```
0.0.0.0
```

1.16. ADDITIONAL RESOURCES

- [Getting Started with Quarkus](#)
- [Apache Maven project](#)

Revised on 2024-04-04 11:42:33 UTC

