



Red Hat build of Quarkus 1.3

Developing and compiling your Quarkus applications with Apache Maven

Red Hat build of Quarkus 1.3 Developing and compiling your Quarkus applications with Apache Maven

Legal Notice

Copyright © 2020 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 create Quarkus applications with the Apache Maven framework.

Table of Contents

| | |
|--|----|
| PREFACE | 3 |
| CHAPTER 1. RED HAT BUILD OF QUARKUS | 4 |
| CHAPTER 2. APACHE MAVEN AND QUARKUS | 5 |
| 2.1. CONFIGURING THE MAVEN SETTINGS.XML FILE FOR THE ONLINE REPOSITORY | 5 |
| 2.2. DOWNLOADING AND CONFIGURING THE QUARKUS MAVEN REPOSITORY | 6 |
| CHAPTER 3. CREATING A QUARKUS PROJECT ON THE COMMAND LINE | 9 |
| CHAPTER 4. CREATING A QUARKUS PROJECT BY CONFIGURING THE POM.XML FILE | 12 |
| CHAPTER 5. CONFIGURING THE JAVA COMPILER | 13 |
| CHAPTER 6. INSTALLING AND MANAGING JAVA EXTENSIONS WITH QUARKUS APPLICATIONS | 14 |
| CHAPTER 7. IMPORTING YOUR QUARKUS PROJECT INTO AN IDE | 15 |
| CHAPTER 8. CONFIGURING THE QUARKUS PROJECT OUTPUT | 17 |
| CHAPTER 9. TESTING YOUR QUARKUS APPLICATION | 18 |
| CHAPTER 10. LOGGING THE QUARKUS APPLICATION BUILD CLASSPATH TREE | 19 |
| CHAPTER 11. USING QUARKUS DEVELOPMENT MODE | 20 |
| CHAPTER 12. DEBUGGING YOUR QUARKUS PROJECT | 21 |
| CHAPTER 13. ADDITIONAL RESOURCES | 22 |

PREFACE

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

This guide describes how to create a Quarkus project using the Apache Maven plug-in.

Prerequisites

- OpenJDK (JDK) 11 is installed and the **JAVA_HOME** environment variable specifies the location of the Java SDK.
 - Log in to the Red Hat Customer Portal to download Red Hat build of Open JDK from the [Software Downloads](#) page.
- Apache Maven 3.6.2 or higher is installed.
 - Download Maven from the [Apache Maven Project](#) website.

CHAPTER 1. RED HAT BUILD OF QUARKUS

Red Hat build of Quarkus is a Kubernetes-native Java stack that is optimized for use with containers and Red Hat OpenShift Container Platform. Quarkus is designed to work with popular Java standards, frameworks, and libraries such as Eclipse MicroProfile, Apache Kafka, RESTEasy (JAX-RS), Hibernate ORM (JPA), Spring, Infinispan, and Apache Camel.

The Quarkus dependency injection solution is based on CDI (contexts and dependency injection) and includes an extension framework to expand functionality and to configure, boot, and integrate a framework into your application.

Quarkus provides a container-first approach to building Java applications. This approach makes it much easier to build microservices-based applications written in Java as well as enabling those applications to invoke functions running on serverless computing frameworks. For this reason, Quarkus applications have small memory footprints and fast start-up times.

CHAPTER 2. APACHE MAVEN AND QUARKUS

Apache Maven is a distributed build automation tool 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 using an XML file. This ensures that the project is built in a correct and uniform manner.

Maven repositories

A Maven repository stores Java libraries, plug-ins, 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 online Maven repository with your Quarkus projects or you can download the Red Hat build of Quarkus Maven repository.

Maven plug-ins

Maven plug-ins are defined parts of a POM file that achieve one or more goals. Quarkus applications use the following Maven plug-ins:

- Quarkus Maven plug-in (**quarkus-maven-plugin**): Enables Maven to create Quarkus projects, supports the generation of uber-JAR files, and provides a development mode.
- Maven Surefire plug-in (**maven-surefire-plugin**): Used during the test phase of the build lifecycle to execute unit tests on your application. The plug-in generates text and XML files that contain the test reports.

2.1. CONFIGURING THE MAVEN `SETTINGS.XML` FILE FOR THE ONLINE REPOSITORY

You can use the online Quarkus repository with your Quarkus Maven project by configuring your user **settings.xml** file. This is the recommended approach. Maven settings used with a repository manager or repository on a shared server provide 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.

Procedure

1. Open the Maven `~/.m2/settings.xml` file in a text editor or integrated development environment (IDE).



NOTE

If there is not a **settings.xml** file in the `~/.m2/` directory, copy the **settings.xml** file from the `$MAVEN_HOME/.m2/conf/` directory into the `~/.m2/` directory.

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

```

<!-- Configure the 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>

```

2.2. DOWNLOADING AND CONFIGURING THE QUARKUS MAVEN REPOSITORY

If you do not want to use the online Maven repository, you can download and configure the Quarkus Maven repository to create a Quarkus application with Maven. The Quarkus Maven repository contains many of the requirements that Java developers typically use to build their applications. This procedure describes how to edit the **settings.xml** file to configure the Quarkus Maven repository.



NOTE

When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Procedure

1. Download the Quarkus Maven repository ZIP file from the [Software Downloads](#) page of the Red Hat Customer Portal (login required).
2. Expand the downloaded archive.

3. Change directory to the `~/.m2/` directory and open the Maven `settings.xml` file in a text editor or integrated development environment (IDE).
4. Add the following lines to the `<profiles>` element of the `settings.xml` file, where `QUARKUS_MAVEN_REPOSITORY` is the path of the Quarkus Maven repository that you downloaded. The format of `QUARKUS_MAVEN_REPOSITORY` must be `file://$PATH`, for example `file:///home/userX/rh-quarkus-1.3.4.SP1-maven-repository/maven-repository`.

```
<!-- Configure the Quarkus Maven repository -->
<profile>
  <id>red-hat-enterprise-maven-repository</id>
  <repositories>
    <repository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>red-hat-enterprise-maven-repository</id>
      <url>QUARKUS_MAVEN_REPOSITORY</url>
      <releases>
        <enabled>true</enabled>
      </releases>
      <snapshots>
        <enabled>>false</enabled>
      </snapshots>
    </pluginRepository>
  </pluginRepositories>
</profile>
```

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

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



IMPORTANT

If your Maven repository contains outdated artifacts, you might encounter one of the following Maven error messages when you build or deploy your project, where ***ARTIFACT_NAME*** is the name of a missing artifact and ***PROJECT_NAME*** is the name of the project you are trying to build:

- **Missing artifact *PROJECT_NAME***
- **[ERROR] Failed to execute goal on project *ARTIFACT_NAME*; Could not resolve dependencies for *PROJECT_NAME***

To resolve the issue, delete the cached version of your local repository located in the **`~/.m2/repository`** directory to force a download of the latest Maven artifacts.

CHAPTER 3. CREATING A QUARKUS PROJECT ON THE COMMAND LINE

You can use the Quarkus Maven plug-in on the command line to create a Quarkus project by providing attributes and values on the command line or by using the plug-in in interactive mode. The resulting project will contain 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** file in **src/main/docker**
- The application configuration file

Procedure

1. In a command terminal, enter the following command to verify that Maven is using JDK 11 and that the Maven version is 3.6.2 or higher:

```
mvn --version
```

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

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create \
  -DprojectId=PROJECT_GROUP_ID \
  -DprojectId=PROJECT_ARTIFACT_ID \
  -DplatformGroupId=com.redhat.quarkus \
  -DplatformArtifactId=quarkus-universe-bom \
  -DplatformVersion=1.3.4.Final-redhat-00004 \
  -DclassName="CLASSNAME"
```

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
- ***CLASSNAME***: The fully qualified name of the generated resource, for example **org.acme.quarkus.sample.HelloResource**
- Create the project in interactive mode:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create
```

When prompted, enter the required attribute values.

**NOTE**

Alternatively, you can create your project using the default values for the project attributes by entering the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create -B
```

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

| Attribute | Default Value | Description |
|---------------------------|-----------------------------|---|
| projectGroupId | org.acme.sample | A unique identifier of your project. |
| projectArtifactId | none | The name of your project and your project directory. If you do not specify the projectArtifactId , the Maven plug-in starts the interactive mode. If the directory already exists, the generation fails. |
| projectVersion | 1.0-SNAPSHOT | The version of your project. |
| platformGroupId | io.quarkus | The group id of your platform. All the existing platforms are provided by io.quarkus . However, you can change the default value. |
| platformArtifactId | quarkus-universe-bom | The artifact id of your platform BOM. To use the locally built Quarkus add quarkus-universe-bom to your pom.xml file. |
| platformVersion | The latest platform version | The version of the platform you want to use for your project. You can provide a version range and the Maven plug-in uses the latest version. |

| Attribute | Default Value | Description |
|-------------------|---------------|---|
| className | None | The fully qualified name of the generated resource. After the application is created, the REST endpoint is exposed at the following URL: http://localhost:8080/\$path If you use the default path , the URL is http://localhost:8080/hello . |
| path | /hello | The resource path, only if you set the className . |
| extensions | [] | The list of extensions you want to add to your project separated by comma. |



NOTE

By default, the Quarkus Maven plug-in uses the latest **quarkus-universe-bom** file. This BOM aggregates extensions so you can reference them from your applications to align the dependency versions. If you are offline, the Quarkus Maven plug-in uses the latest locally available version of the **quarkus-universe-bom**. If Maven finds the **quarkus-universe-bom** version 2.0 or earlier, it will use the platform based on the **quarkus-universe-bom**.

CHAPTER 4. CREATING A 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 Quarkus GAV (group, artifact, version) and use the **`quarkus-universe-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-plugin.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

3. Add the Quarkus Maven plug-in:

```
<build>
  <plugins>
    <plugin>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus-plugin.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```


CHAPTER 5. CONFIGURING THE JAVA COMPILER

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

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 **--enable-preview** to both the JVM and **javac** add the following lines:

```
<plugin>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-maven-plugin</artifactId>
  <version>${quarkus-plugin.version}</version>

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

  ...
</plugin>
```

CHAPTER 6. INSTALLING AND MANAGING JAVA EXTENSIONS WITH QUARKUS APPLICATIONS

You can use Java extensions to expand the functionality of your application and to 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 a Quarkus Maven project.

Procedure

1. Navigate to your Quarkus project directory.
2. To list the available extensions, enter the following command:

```
./mvnw quarkus:list-extensions
```

3. To add an extension to your project, enter the following command where **EXTENSION** is the group, artifact, 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"
```

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 **jdbc**, **agroal**, and **non-exist-ent** in the name:

```
./mvnw quarkus:add-extension -Dextensions=jdbc,agroal,non-exist-ent
```

This command returns the following result:

```
Multiple extensions matching 'jdbc'  
* io.quarkus:quarkus-jdbc-h2  
* io.quarkus:quarkus-jdbc-mariadb  
* io.quarkus:quarkus-jdbc-postgresql  
Be more specific e.g using the exact name or the full gav.  
Adding extension io.quarkus:quarkus-agroal  
Cannot find a dependency matching 'non-exist-ent', maybe a typo?  
[...]
```

5. To install all extensions that a specific text string returns, enter the extension name or partial name after **-Dextensions=**. The following example searches for and installs all extensions that begin with **hibernate-**:

```
./mvnw quarkus:add-extension -Dextensions="hibernate-**"
```

CHAPTER 7. IMPORTING YOUR QUARKUS PROJECT INTO AN IDE

Although it is possible to develop your Quarkus project in a text editor, you might find it easier to use an integrated development environment (IDE) to work on your project. The following instructions show you how to import your Quarkus project into specific IDEs.

Prerequisites

- You have a Quarkus Maven project.

Procedure

Complete the steps in one of the following sections:

CodeReady Studio or Eclipse

1. In CodeReady Studio or Eclipse, click **File** → **Import**.
2. Select **Maven** → **Existing Maven Project**
3. On the next screen, select the root location of the project. A list of the found modules appears.
4. Select the generated project and click **Finish**.
5. To start your application, enter the following command in a new terminal window:

```
./mvnw compile 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 **Finish**.
6. To start your application, enter the following command in a new terminal window:

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

Apache NetBeans

1. Select **File** → **Open Project**.
2. Select the project **root** directory.

3. Click **Open Project**.
4. To start your application, enter the following command in a new terminal window:

```
┃ ./mvnw compile quarkus:dev
```

Visual Studio Code

1. Install the Java Extension Pack.
2. In Visual Studio Code, open your project directory. The project loads as a Maven project.

CHAPTER 8. CONFIGURING THE QUARKUS PROJECT OUTPUT

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

Prerequisites

- You have a Quarkus Maven project.

Procedure

- Open the **application.properties** file in a text editor.
- Edit the values of properties that you want to change and save the file.
The following table list the properties that you can change:

| Property | Description | Type | Default |
|--|--|---------------|--|
| quarkus.package.main-class | The entry point of the application. In most cases, you should change this value. | string | io.quarkus.runner.GeneratedMain |
| quarkus.package.type | The requested output type. | string | jar |
| quarkus.package.uber-jar | Whether or not the Java runner should be packed as an uber-JAR. | boolean | false |
| quarkus.package.manifest.add-implementation-entries | Whether or not the implementation information should be included in the runner JAR file's MANIFEST.MF file. | boolean | true |
| quarkus.package.user-configured-ignored-entries | Files that should 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 | |

CHAPTER 9. TESTING YOUR QUARKUS APPLICATION

By default, when you test your Quarkus application, Maven uses the **test** configuration profile. However, you can create a custom configuration profile for your tests using the Maven Surefire plug-in.

Prerequisites

- You have a Quarkus project created with Apache Maven.

Procedure

- Edit the following example to meet your testing requirements, where **PROFILE_NAME** is a name for your test profile:

```
<project>
[...]
```

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${SureFirePluginVersion}</version>
      <configuration>
        <systemPropertyVariables>
          <quarkus.test.profile>PROFILE_NAME</quarkus.test.profile>
          <buildDirectory>${project.build.directory}</buildDirectory>
          [...]
        </systemPropertyVariables>
      </configuration>
    </plugin>
  </plugins>
</build>
[...]
```

```
</project>
```

CHAPTER 10. LOGGING THE 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 plug-in includes the **build-tree** goal which displays the build dependency tree for the application.

Prerequisites

- You have a Quarkus Maven application.

Procedure

1. Add the plug-in configuration to the **pom.xml** file:

```
<project>
  [...]
  <plugin>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-bootstrap-maven-plugin</artifactId>
    <version>${quarkus-plugin.version}</version>
  </plugin>
  [...]
</project>
```

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

```
./mvnw quarkus-bootstrap:build-tree
```

3. The output of this command should be similar to the following example:

```
[INFO] --- quarkus-bootstrap-maven-plugin:1.3:build-tree (default-cli) @ getting-started ---
[INFO] org.acme:getting-started:jar:1.0-SNAPSHOT
[INFO] └─ io.quarkus:quarkus-resteasy-deployment:jar:1.3 (compile)
[INFO] │   └─ io.quarkus:quarkus-resteasy-server-common-deployment:jar:1.3 (compile)
[INFO] │       └─ io.quarkus:quarkus-core-deployment:jar:1.3 (compile)
[INFO] │           └─ commons-beanutils:commons-beanutils:jar:1.9.3 (compile)
[INFO] │               └─ commons-logging:commons-logging:jar:1.2 (compile)
[INFO] │                   └─ commons-collections:commons-collections:jar:3.2.2 (compile)
...

```



NOTE

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

CHAPTER 11. USING QUARKUS DEVELOPMENT MODE

Development mode enables hot deployment with background compilation, which means that when you modify your Java files or your resource files and then refresh your browser, the changes automatically take effect. This also works for resource files such as the configuration property file.

Prerequisites

- You have a Quarkus Maven application.

Procedure

1. To start Quarkus in development mode, enter the following command in the directory that contains your Quarkus application **pom.xml** file:

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

2. Make changes to your application and save the files.
3. 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**.

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

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

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

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


CHAPTER 12. DEBUGGING YOUR QUARKUS PROJECT

When Quarkus starts in development mode, debugging is enabled by default. The debugger listens on port **5005** without suspending the JVM.

Prerequisites

- You have a Quarkus Maven project.

Procedure

Use one of the following methods to control debugging:

Control the debugger through system properties

1. 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**.
2. Change the value of the **suspend** system property. This property is used when Quarkus starts in debug mode.
 - **y** or **true**: The debug mode JVM launch suspends.
 - **n** or **false**: The debug mode JVM starts without suspending.

Control the debugger from the command line

1. To start your Quarkus application in debug mode with JVM, enter the following command:

```
./mvnw compile quarkus:dev -Ddebug
```

2. Attach a debugger to **localhost:5005**.

CHAPTER 13. ADDITIONAL RESOURCES

- For information about deploying Quarkus Maven applications on Red Hat OpenShift Container Platform, see [Deploying your Quarkus applications on Red Hat OpenShift Container Platform](#) .
- For more information about the Maven Surefire plug-in, see the [Apache Maven Project](#) website.