Red Hat build of Quarkus 3.2

Compiling your Red Hat build of Quarkus applications to native executables

Last Updated: 2024-04-04

Red Hat build of Quarkus 3.2 Compiling your Red Hat build of Quarkus applications to native executables

Abstract

This guide shows you how to compile the Red Hat build of Quarkus Getting Started project into a native executable and how to configure and test the native executable.

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE 3

CHAPTER 1. COMPILING YOUR RED HAT BUILD OF QUARKUS APPLICATIONS TO NATIVE EXECUTABLES . 4

1.1. PRODUCING A NATIVE EXECUTABLE 4

1.1.1. Producing a native executable by using an in-container build 6

1.1.2. Producing a native executable by using a local-host build 7

1.2. CREATING A CUSTOM CONTAINER IMAGE 8

1.2.1. Creating a container manually 9

1.2.2. Creating a container by using the OpenShift Docker build 10

1.3. NATIVE EXECUTABLE CONFIGURATION PROPERTIES 11

1.3.1. Configuring memory consumption for Red Hat build of Quarkus native compilation 15

1.4. TESTING THE NATIVE EXECUTABLE 15

1.4.1. Excluding tests when running as a native executable 18

1.4.2. Testing an existing native executable 19

1.5. ADDITIONAL RESOURCES 19

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. COMPILING YOUR RED HAT BUILD OF QUARKUS APPLICATIONS TO NATIVE EXECUTABLES

As an application developer, you can use Red Hat build of Quarkus 3.2 to create microservices written in Java that run on OpenShift Container Platform and serverless environments. Quarkus applications can run as regular Java applications (on top of a Java Virtual Machine), or be compiled into native executables. Applications compiled to native executables have a smaller memory footprint and faster startup times than their Java counterpart.

This guide shows you how to compile the Red Hat build of Quarkus 3.2 Getting Started project into a native executable and how to configure and test the native executable. You will need the application that you created earlier in Getting started with Red Hat build of Quarkus .

Building a native executable with Red Hat build of Quarkus covers:

- Building a native executable with a single command by using a container runtime such as Podman or Docker
- Creating a custom container image using the produced native executable
- Creating a container image using the OpenShift Container Platform Docker build strategy
- Deploying the Quarkus native application to OpenShift Container Platform
- Configuring the native executable
- Testing the native executable

Prerequisites

- Have OpenJDK 17 installed and the **JAVA_HOME** environment variable set to specify the location of the Java SDK.
  - Log in to the Red Hat Customer Portal to download Red Hat build of OpenJDK from the Software Downloads page.
- An Open Container Initiative (OCI) compatible container runtime, such as Podman or Docker.
- A completed Quarkus Getting Started project.
  - To learn how to build the Quarkus Getting Started project, see Getting started with Quarkus.
  - Alternatively, you can download the Quarkus quickstart archive or clone the **Quarkus Quickstarts** Git repository. The sample project is in the **getting-started** directory.

1.1. 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.

1.1.1. 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-xmx** 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

1.1.2. 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.

1.2. CREATING A CUSTOM CONTAINER IMAGE

You can create a container image from your Quarkus application using one of the following methods:

- Creating a container manually

- Creating a container by using the OpenShift Container Platform Docker build

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-xmx** configuration property. Setting low memory limits might increase the build time.

1.2.1. 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:

- 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
     ```

1.2.2. Creating a container by using the OpenShift Docker build

You can create a container image for your Quarkus application by using the OpenShift Container Platform Docker build strategy. This strategy creates a container image by using a build configuration in the cluster.

Prerequisites

- You have access to an OpenShift Container Platform cluster and the latest version of the **oc** tool installed. For information about installing **oc**, see *Installing the CLI* in the Installing and configuring OpenShift Container Platform clusters guide.
- A URL for the OpenShift Container Platform API endpoint.

Procedure

1. Log in to the OpenShift CLI:

   ```
   oc login -u <username_url>
   ```

2. Create a new project in OpenShift:

   ```
   oc new-project <project_name>
   ```

3. Create a build config based on the **src/main/docker/Dockerfile.native** file:

```
cat src/main/docker/Dockerfile.native | oc new-build --name <build_name> --strategy=docker
--dockerfile -
```

4. Build the project:

```
oc start-build <build_name> --from-dir .
```

5. Deploy the project to OpenShift Container Platform:

```
oc new-app <build_name>
```

6. Expose the services:

```
oc expose svc/<build_name>
```

1.3. NATIVE EXECUTABLE CONFIGURATION PROPERTIES

Configuration properties define how the native executable is generated. You can configure your Quarkus application using the **application.properties** file.

Configuration properties

The following table lists the configuration properties that you can set to define how the native executable is generated:

Property	Description	Type	Default
quarkus.native.debug.enabled	If debug is enabled and debug symbols are generated, the symbols are generated in a separate **.debug** file.	boolean	false
quarkus.native.resources.excludes	A comma-separated list of globs to match resource paths that should not be added to the native image.	list of strings	
quarkus.native.additional-build-args	Additional arguments to pass to the build process.	list of strings	
quarkus.native.enable-http-url-handler	Enables HTTP URL handler. This allows you to do **URL.openConnection()** for HTTP URLs.	boolean	**true**
quarkus.native.enable-https-url-handler	Enables HTTPS URL handler. This allows you to do **URL.openConnection()** for HTTPS URLs.	boolean	**false**

quarkus.native.enable-all-security-services	Adds all security services to the native image.	boolean	**false**
quarkus.native.add-all-charsets	Adds all character sets to the native image. This increases the image size.	boolean	**false**
quarkus.native.graalvm-home	Contains the path of the GraalVM distribution.	string	**${GRAALVM_HOME:}**
quarkus.native.java-home	Contains the path of the JDK.	File	**${java.home}**
quarkus.native.native-image-xmx	The maximum Java heap used to generate the native image.	string	
quarkus.native.debug-build-process	Waits for a debugger to attach to the build process before running the native image build. This is an advanced option for those familiar with GraalVM internals.	boolean	**false**
quarkus.native.publish-debug-build-process-port	Publishes the debug port when building with docker if **debug-build-process** is **true**.	boolean	**true**
quarkus.native.cleanup-server	Restarts the native image server.	boolean	**false**
quarkus.native.enable-isolates	Enables isolates to improve memory management.	boolean	**true**
quarkus.native.enable-fallback-images	Creates a JVM-based fallback image if the native image fails.	boolean	**false**
quarkus.native.enable-server	Uses the native image server. This can speed up compilation but can result in lost changes due to cache invalidation issues.	boolean	**false**
quarkus.native.auto-service-loader-registration	Automatically registers all **META-INF/services** entries.	boolean	**false**
quarkus.native.dump-proxies	Dumps the bytecode of all proxies for inspection.	boolean	**false**
quarkus.native.container-build	Builds that use a container runtime. Docker is used by default.	boolean	**false**

quarkus.native.builder-image	The docker image to build the image.	string	**registry.access.redhat.com/quarkus/mandrel-23-rhel8:23.0**
quarkus.native.container-runtime	The container runtime used to build the image. For example, Docker.	string	
quarkus.native.container-runtime-options	Options to pass to the container runtime.	list of strings	
quarkus.native.enable-vm-inspection	Enables VM introspection in the image.	boolean	**false**
quarkus.native.full-stack-traces	Enables full stack traces in the image.	boolean	**true**
quarkus.native.enable-reports	Generates reports on call paths and included packages, classes, or methods.	boolean	**false**
quarkus.native.report-exception-stack-traces	Reports exceptions with a full stack trace.	boolean	**true**
quarkus.native.report-errors-at-runtime	Reports errors at runtime. This might cause your application to fail at runtime if you use unsupported features.	boolean	**false**

quarkus.native.resources.includes	A comma-separated list of globs to match resource paths that should be added to the native image. Use a slash (/) character as a path separator on all platforms. Globs must not start with a slash. For example, if you have **src/main/resources/ignored.png** and **src/main/resources/foo/selected.png** in your source tree and one of your dependency JARs contains a **bar/some.txt** file, with **quarkus.native.resources.includes** set to **foo/,bar//*.txt**, the files **src/main/resources/foo/selected.png** and **bar/some.txt** will be included in the native image, while **src/main/resources/ignored.png** will not be included. For more information, see the following table, which lists the supported glob features.	list of strings	
quarkus.native.debug.enabled	Enables debugging and generates debug symbols in a separate **.debug** file. When used with **quarkus.native.container-build**, Red Hat build of Quarkus only supports Red Hat Enterprise Linux or other Linux distributions as they contain the **binutils** package that installs the **objcopy** utility that splits the debug info from the native image.	boolean	**false**

Supported glob features

The following table lists the supported glob features and descriptions:

Character	Feature description
*	Matches a possibly-empty sequence of characters that does not contain slash (/).
**	Matches a possibly-empty sequence of characters that might contain slash (/).
?	Matches one character, but not slash.
[abc]	Matches one character specified in the bracket, but not slash.

[a-z]	Matches one character from the range specified in the bracket, but not slash.
[!abc]	Matches one character not specified in the bracket; does not match slash.
[!a-z]	Matches one character outside the range specified in the bracket; does not match slash.
{one,two,three}	Matches any of the alternating tokens separated by commas; the tokens can contain wildcards, nested alternations, and ranges.
\	The escape character. There are three levels of escaping: **application.properties** parser, MicroProfile Config list converter, and Glob parser. All three levels use the backslash as the escape character.

Additional resources

- Configuring your Red Hat build of Quarkus applications

1.3.1. Configuring memory consumption for Red Hat build of Quarkus native compilation

Compiling a Red Hat build of 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-xmx** configuration property. Setting low memory limits might increase the build time.

Procedure

- Use one of the following methods to set a value for the **quarkus.native.native-image-xmx** property to limit the memory consumption during the native image build time:
  - Using the **application.properties** file:

    ```
    quarkus.native.native-image-xmx=<maximum_memory>
    ```

  - Setting system properties:

    ```
    mvn package -Dnative -Dquarkus.native.container-build=true -Dquarkus.native.native-image-xmx=<maximum_memory>
    ```

    This command builds the native executable with Docker. To use Podman, add the **-Dquarkus.native.container-runtime=podman** argument.

NOTE

For example, to set the memory limit to 6 GB, enter **quarkus.native.native-image-xmx=6g**. The value must be a multiple of 1024 and greater than 2MB. Append the letter **m** or **M** to indicate megabytes, or **g** or **G** to indicate gigabytes.

1.4. 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>

<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
      63 types,    68 fields, and    55 methods registered for JNI access
       4 native libraries: dl, pthread, rt, z
[3/8] Building universe...                                               (5.9s @ 1.31GB)
[4/8] Parsing methods...      [**]                                        (3.7s @ 2.08GB)
[5/8] Inlining methods...     [***]                                       (2.0s @ 1.92GB)
[6/8] Compiling methods...    [******]                                     (34.4s @
3.25GB)
[7/8] Layouting methods...    [[7/8] Layouting methods...    [**]
(4.1s @ 1.78GB)
[8/8] Creating image...       [**]                                        (4.5s @ 2.31GB)
  20.93MB (48.43%) for code area:    33,233 compilation units
  21.95MB (50.80%) for image heap:  285,664 objects and 8 resources
 337.06kB ( 0.76%) for other data
  43.20MB in total

....
```

```
[INFO]
[INFO] --- maven-failsafe-plugin:3.0.0-M7:integration-test (default) @ getting-started ---
[INFO] Using auto detected provider
org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.acme.GreetingResourceIT
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2023-08-28 14:04:52,681 INFO  [io.quarkus] (main) getting-started 1.0.0-SNAPSHOT native
(powered by Red Hat build of Quarkus 3.2.9.Final) started in 0.038s. Listening on:
http://0.0.0.0:8081
2023-08-28 14:04:52,682 INFO  [io.quarkus] (main) Profile prod activated.
2023-08-28 14:04:52,682 INFO  [io.quarkus] (main) Installed features: [cdi, resteasy-reactive,
smallrye-context-propagation, vertx]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.696 s - in
org.acme.GreetingResourceIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-failsafe-plugin:3.0.0-M7:verify (default) @ getting-started ---
```

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.

1.4.1. Excluding tests when running as a native executable

When you run tests against your native executable, you can only run black-box testing, for example, interacting with the HTTP endpoints of your application.

NOTE

Black box refers to the hidden internal workings of a product or program, such as in black-box testing.

Because tests do not run natively, you cannot link against your application's code like you do when running tests on the JVM. Therefore, in your native tests, you cannot inject beans.

You can share your test class between your JVM and native executions and exclude certain tests using the **@DisabledOnNativeImage** annotation to run tests only on the JVM.

1.4.2. Testing an existing native executable

By using the **Failsafe** Maven plugin, you can test against the existing executable build. You can run multiple sets of tests in stages on the binary after it is built.

NOTE

To test the native executable that you produced with Quarkus, use the available Maven commands. There are no equivalent Quarkus CLI commands to complete this task by using the command line.

Procedure

- Run a test against a native executable that is already built:

  ```
  ./mvnw test-compile failsafe:integration-test
  ```

  This command runs the test against the existing native image by using the **Failsafe** Maven plugin.

- Alternatively, you can specify the path to the native executable with the following command where **<path>** is the native image path:

  ```
  ./mvnw test-compile failsafe:integration-test -Dnative.image.path=<path>
  ```

1.5. ADDITIONAL RESOURCES

- Deploying your Red Hat build of Quarkus applications to OpenShift Container Platform
- Developing and compiling your Red Hat build of Quarkus applications with Apache Maven
- Quarkus community: Building a native executable
- Apache Maven Project
- The UBI Image Page
- The *UBI-minimal* Image Page
- The List of *UBI-minimal* Tags

Revised on 2024-04-04 11:41:47 UTC