# Red Hat build of Quarkus 3.8

## 将红帽构建的 Quarkus 应用程序编译到原生可执行文件

Last Updated: 2024-06-22

# Red Hat build of Quarkus 3.8 将红帽构建的 Quarkus 应用程序编译到原生可执行文件

## 法律通告



## 摘要

本指南介绍了如何将红帽构建的 Quarkus 入门项目编译到原生可执行文件中，以及如何配置和测试原生可执行文件。

# 目录

# 提供有关红帽构建的 QUARKUS 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

**流程**

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

# 使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中有问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

# 第 1 章 将红帽构建的 QUARKUS 应用程序编译到原生可执行文件

作为应用程序开发人员，您可以使用红帽构建的 Quarkus 3.8 创建在 OpenShift Container Platform 和无服务器环境中运行的 Java 编写的微服务。Quarkus 应用程序可以作为常规 Java 应用程序（在 Java 虚拟机之上）运行，或者编译到原生可执行文件中。编译到原生可执行文件的应用程序具有比 Java 对应部分小的内存占用和更快的启动时间。

本指南介绍了如何将红帽构建的 Quarkus 3.8 Getting Started 项目编译到原生可执行文件中，以及如何配置和测试原生可执行文件。您需要您之前在 红帽构建的 Quarkus 中创建 的应用程序。

**使用 Red Hat build of Quarkus 构建原生可执行文件涵盖了：**

- 使用 Podman 或 Docker 等容器运行时（如 Podman 或 Docker）通过单个命令构建原生可执行文件
- 使用生成的原生可执行文件创建自定义容器镜像
- 使用 OpenShift Container Platform Docker 构建策略创建容器镜像
- 将 Quarkus 原生应用程序部署到 OpenShift Container Platform
- 配置原生可执行文件
- 测试原生可执行文件

**先决条件**

- 设置 **JAVA_HOME** 环境变量，以指定 Java SDK 的位置。
  - 登录到红帽客户门户网站，从 Software Downloads 页面下载红帽构建的 OpenJDK。
- 一个兼容开放容器项目(OCI)的容器运行时，如 Podman 或 Docker。
- 已完成的 Quarkus 入门项目。
  - 要了解如何构建 Quarkus Getting Started 项目，请参阅开始使用 Red Hat build of Quarkus。
  - 或者，您可以下载 Quarkus Quickstarts 归档或克隆 Quarkus Quickstarts Git 存储库。示例项目处于 **getting-started** 目录中。

## 1.1. 生成原生可执行文件

原生二进制文件是创建在特定操作系统和 CPU 架构上运行的可执行文件。

以下列表概述了原生可执行文件的一些示例：

- Linux AMD 64 位的 ELF 二进制文件
- Windows AMD 64 位的 EXE 二进制文件
- ARM 64 位的 ELF 二进制文件

**注意**

红帽构建的 Quarkus 仅支持 Linux AMD 64 位的 ELF 二进制文件。

构建原生可执行文件时，您的应用程序和依赖项（包括 JVM）被打包到一个文件中。应用程序的原生可执行文件包含以下项目：

- 编译的应用程序代码
- 所需的 Java 库
- 减少了虚拟机(VM)的版本，用于改进应用程序启动时间和最小磁盘和内存占用量，这也是为应用程序代码及其依赖项量身定制的

要从 Quarkus 应用生成原生可执行文件，您可以选择容器内构建或 local-host 构建。下表解释您可以使用的不同构建选项：

**表 1.1. 构建生成原生可执行文件的选项**

| 构建选项 | Requires | 使用 | 结果 | 优点 |
| --- | --- | --- | --- | --- |
| in-container build - Supported | 容器运行时，如 Podman 或 Docker | 默认 **registry.access.redhat.com/quarkus/mandrel-for-jdk-21-rhel8:23.1** 构建器镜像 | 使用主机的 CPU 架构进行 Linux 64 位可执行文件 | GraalVM 不需要在本地设置，从而使 CI 管道 更有效地运行 |
| local-host build - 仅支持上游 | GraalVM 或 Mandrel 的本地安装 | 其本地安装作为 **quarkus.native.builder-image** 属性的默认设置 | 具有与执行构建的机器相同的操作系统和 CPU 架构的可执行文件 | 不允许或不想使用 Docker 或 Podman 等工具的开发人员的替代方案。总体而言，它比容器内构建方法更快。 |

**重要**

- 红帽构建的 Quarkus 3.8 仅支持使用基于 Java 21 的 红帽构建的 Quarkus 原生构建器镜像构建原生 Linux 可执行文件，该镜像是 Mandrel 的产品化分发。虽然其他镜像在社区中可用，但产品不支持它们，因此您不应该将其用于您希望红帽提供支持的生产构建。
- 其源基于 17 （不使用 Java 18 - 21 的功能）编写的应用程序仍然可以使用基于 Java 21 的 Mandrel 23.1 基础镜像编译应用程序的原生可执行文件。
- 使用红帽构建的 Quarkus 不支持使用 Oracle GraalVM 社区版(CE)、Mael 社区版本或任何其他 GraalVM 发行版构建原生可执行文件。

### 1.1.1. 使用 in-container 构建生成原生可执行文件

要创建原生可执行文件并运行原生镜像测试，请使用由红帽构建的 Quarkus 提供的 **原生** 配置集进行容器内构建。

**先决条件**

- podman 或 Docker 已安装。

- 容器可以访问至少 8GB 内存。

**流程**

1. 打开 Getting Started project **pom.xml** 文件，并验证项目是否包含 **native** 配置集：

```
<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. 使用以下方法之一构建原生可执行文件：

- 使用 Maven：
  - 对于 Docker：

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

  - 对于 Podman：

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

- 使用 Quarkus CLI：
  - 对于 Docker：

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

  - 对于 Podman：

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

**步骤结果**

这些命令在目标目录中创建一个 ***-runner** 二进制文件，其中适用以下内容：

  - ***-runner 文件是由 Quarkus 生成的构建原生二进制文件。**

- 目标目录 是一个目录，Maven 会在构建 Maven 应用程序时创建该目录。

**重要**

将 Quarkus 应用程序编译到原生可执行文件会在分析和优化过程中消耗大量内存。您可以通过设置 quarkus.native.native-image-xmx 配置属性来限制原生编译过程中使用的内存量。设置低内存限值可能会增加构建时间。

3. 要运行原生可执行文件，请输入以下命令：

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

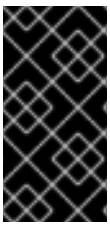
**其他资源**

- 原生可执行配置属性

### 1.1.2. 使用 local-host 构建生成原生可执行文件

如果您不使用 Docker 或 Podman，请使用 Quarkus local-host build 选项来创建和运行原生可执行文件。

使用本地主机构建方法比使用容器更快，并适用于使用 Linux 操作系统的机器。



**重要**

红帽构建的 Quarkus 不支持在生产环境中使用以下步骤。只有在 Docker 或 Podman 不可用时，才使用这个方法测试或作为备份方法。

**先决条件**

- Mandrel 或 GraalVm 的本地安装，根据 构建原生可执行文件 指南进行了正确配置。
  - 另外，对于 GraalVM 安装，还必须安装 native-image。

**流程**

1. 对于 GraalVM 或 Mandrel，使用以下方法之一构建原生可执行文件：

   - 使用 Maven：

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

   - 使用 Quarkus CLI：

   ```
   quarkus build --native
   ```

   步骤结果

   这些命令在目标目录中创建一个 *-runner 二进制文件，其中适用以下内容：

   - *-runner 文件是 Quarkus 生成的内置原生二进制文件。
   - 目标目录 是一个目录，Maven 会在构建 Maven 应用程序时创建该目录。

   注意

   构建原生可执行文件时，会启用 prod 配置集，除非在 quarkus.profile 属性中修改了。

2. 运行原生可执行文件：

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

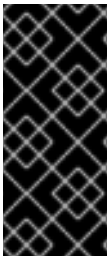
其他资源

如需更多信息，请参阅 Quarkus "Building a native executable" 指南中的 Producing a native executable 部分。

## 1.2. 创建自定义容器镜像

您可以使用以下方法之一从 Quarkus 应用程序创建容器镜像：

- 手动创建容器
- 使用 OpenShift Container Platform Docker 构建创建容器



重要

将 Quarkus 应用程序编译到原生可执行文件会在分析和优化过程中消耗大量内存。您可以通过设置 quarkus.native.native-image-xmx 配置属性来限制原生编译过程中使用的内存量。设置低内存限值可能会增加构建时间。

### 1.2.1. 手动创建容器

您可以使用应用程序为 Linux AMD64 手动创建容器镜像。当您使用 Quarkus Native 容器生成原生镜像时，原生镜像会创建一个以 Linux AMD64 为目标的可执行文件。如果您的主机操作系统与 Linux AMD64 不同，则无法直接运行二进制文件，您需要手动创建容器。

您的 Quarkus Getting Started 项目在 src/main/docker 目录中包含一个 Dockerfile.native，其内容如下：

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.9
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"]
```

**注意**

**通用基础镜像(UBI)**

**以下列表显示了可用于 Dockerfile 的合适镜像。**

- **Red Hat Universal Base Image 8 (UBI8)。此基础镜像旨在设计并设计成为所有容器化应用程序、中间件和实用程序的基础层。**

```
registry.access.redhat.com/ubi8/ubi:8.9
```

- **Red Hat Universal Base Image 8 Minimal (UBI8-minimal)。使用 microdnf 作为软件包管理器的精简版 UBI8 镜像。**

```
registry.access.redhat.com/ubi8/ubi-minimal:8.9
```

- **所有红帽基础镜像都可在容器镜像目录站点中找到。https://catalog.redhat.com/software/containers/search?q=UBI&p=1**

**流程**

1. **使用以下方法之一构建原生 Linux 可执行文件：**

- **docker:**

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

- **Podman:**

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

2. **使用以下方法之一构建容器镜像：**

- **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. 使用以下方法之一运行容器：

- 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. 使用 OpenShift Docker 构建创建容器

您可以使用 OpenShift Container Platform Docker 构建策略为 Quarkus 应用程序创建容器镜像。此策略使用集群中的构建配置创建容器镜像。

**先决条件**

- 您可以访问 OpenShift Container Platform 集群并安装 oc 工具的最新版本。有关安装 oc 的详情，请参考 安装和配置 OpenShift Container Platform 集群指南中的安装 *CLI*。
- OpenShift Container Platform API 端点的 URL。

**流程**

1. 登录到 OpenShift CLI：

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

2. 在 OpenShift 中创建一个新项目：

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

3. 根据 src/main/docker/Dockerfile.native 文件创建构建配置：

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

4. 构建项目：

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

5. 将项目部署到 OpenShift Container Platform：

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

6. 公开服务：

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

### 1.3. 原生可执行配置属性

配置属性定义如何生成原生可执行文件。您可以使用 application.properties 文件配置 Quarkus 应用程序。

#### 配置属性

下表列出了您可以设置的配置属性来定义如何生成原生可执行文件：

| 属性 | 描述 | 类型 | default |
|---|---|---|---|
| **quarkus.native.debug.enabled** | 如果启用了 debug 并生成调试符号，则会在单独的 **.debug** 文件中生成符号。 | 布尔值 | false |
| **quarkus.native.resources.excludes** | 以逗号分隔的 glob 列表，以匹配不应添加到原生镜像的资源路径。 | 字符串列表 | |

| | | | |
|---|---|---|---|
| **quarkus.native.additional-build-args** | 传递给构建过程的额外参数。 | 字符串列表 | |
| **quarkus.native.enable-http-url-handler** | 启用 HTTP URL 处理程序，您可以在其中为 HTTP URL 进行 **URL.openConnection （）**。 | 布尔值 | **true** |
| **quarkus.native.enable-https-url-handler** | 启用 HTTPS URL 处理程序，您可以在其中为 HTTPS URL 进行 **URL.openConnection （）**。 | 布尔值 | **false** |
| **quarkus.native.enable-all-security-services** | 将所有安全服务添加到原生镜像。 | 布尔值 | **false** |
| **quarkus.native.add-all-charsets** | 将所有字符集添加到原生镜像。这会增加镜像大小。 | 布尔值 | **false** |
| **quarkus.native.graalvm-home** | 包含 GraalVM 分发的路径。 | string | **${GRAALVM_HOME:}** |
| **quarkus.native.java-home** | 包含 JDK 的路径。 | file | **${java.home}** |
| **quarkus.native.native-image-xmx** | 用于生成原生镜像的最大 Java 堆。 | string | |
| **quarkus.native.debug-build-process** | 在运行原生镜像构建前，等待调试器附加到构建过程。对于熟悉 GraalVM 内部的用户，这是一个高级选项。 | 布尔值 | **false** |
| **quarkus.native.publish-debug-build-process-port** | 如果 **debug-build-process** 为 **true**，则使用 docker 构建时发布调试端口。 | 布尔值 | **true** |
| **quarkus.native.cleanup-server** | 重启原生镜像服务器。 | 布尔值 | **false** |
| **quarkus.native.enable-isolates** | 启用隔离以提高内存管理。 | 布尔值 | **true** |
| **quarkus.native.enable-fallback-images** | 如果原生镜像失败，则创建基于 JVM 的回退镜像。 | 布尔值 | **false** |
| **quarkus.native.enable-server** | 使用原生镜像服务器。这可以加快编译速度，但可能会导致由于缓存无效问题而丢失更改。 | 布尔值 | **false** |
| **quarkus.native.auto-service-loader-registration** | 自动注册所有 **META-INF/services** 条目。 | 布尔值 | **false** |

| | | | |
|---|---|---|---|
| **quarkus.native.dump-proxies** | 转储所有代理的字节代码以进行检查。 | 布尔值 | **false** |
| **quarkus.native.container-build** | 使用容器运行时的构建。默认使用 Docker。 | 布尔值 | **false** |
| **quarkus.native.builder-image** | 构建镜像的 docker 镜像。 | string | **registry.access.redhat.com/quarkus/mandrel-for-jdk-21-rhel8:23.1** |
| **quarkus.native.container-runtime** | 用于构建镜像的容器运行时。例如，Docker。 | string | |
| **quarkus.native.container-runtime-options** | 传递给容器运行时的选项。 | 字符串列表 | |
| **quarkus.native.enable-vm-inspection** | 在镜像中启用虚拟机内省。 | 布尔值 | **false** |
| **quarkus.native.full-stack-traces** | 在镜像中启用完整的堆栈跟踪。 | 布尔值 | **true** |
| **quarkus.native.enable-reports** | 生成关于调用路径和包含的软件包、类或方法的报告。 | 布尔值 | **false** |
| **quarkus.native.report-exception-stack-traces** | 报告完整堆栈追踪的异常。 | 布尔值 | **true** |
| **quarkus.native.report-errors-at-runtime** | 在运行时报告错误。如果您使用不支持的功能，这可能会导致应用程序在运行时失败。 | 布尔值 | **false** |

| | | | |
|---|---|---|---|
| **quarkus.native.resources.includes** | 以逗号分隔的 glob 列表，以匹配应添加到原生镜像的资源路径。在所有平台上使用斜杠(/)字符作为路径分隔符。globs 不得以斜杠开头。例如，如果您在源树中有 **src/main/resources/ignored.png** 和 **src/main/resources/foo/selected.png**，并且您的依赖项 JAR 包含一个 bar/some.txt 文件，quarkus.native.resources.includes 设置为 foo/ ,bar/ job，而您的依赖项 JAR 包含一个 **bar/some.txt** 文件，其中 **quarkus.native.resources.includes** 设置为 **foo/,bar/.txt**, 文件 **src/main/resources/foo/selected.png** 和 **bar/some.txt** 将包含在原生镜像中，而 **src/main/resources/ignored.png** 不会被包含。如需更多信息，请参阅下表，它列出了支持的 glob 功能。 | 字符串列表 | |
| **quarkus.native.debug.enabled** | 启用调试并在单独的 **.debug** 文件中生成调试符号。与 **quarkus.native.container-build** 一起使用时，Red Hat build of Quarkus 只支持 Red Hat Enterprise Linux 或其他 Linux 发行版本，因为它们包含从原生镜像分割调试信息的 **binutils** 软件包。 | 布尔值 | **false** |

**在构建配置期间，如果要包含在项目中共享通用模式或位置的一组文件或资源，您可以使用 glob 模式。**

**例如，如果您有一个包含多个配置文件的目录，您可以使用 glob 模式包含该目录中的所有文件。**

**例如：**

```
quarkus.native.resources.includes = my/config/files/*
```

以下示例显示了以逗号分隔的 glob 列表，以匹配要添加到原生镜像的资源路径。这些模式会导致将 classpath 上找到的任何 .png 镜像添加到原生镜像，并在文件夹栏下以 .txt 结尾的所有文件，即使嵌套在子目录下：

```
quarkus.native.resources.includes = **/*.png,bar/**/*.txt
```

**支持的 glob 功能**

下表列出了支持的 glob 功能和描述：

| 字符 | 功能描述 |
| --- | --- |
| * | 匹配不包含斜杠(/)的可能为空的字符序列。 |
| ** | 匹配可能包含斜杠(/)的可能空字符序列。 |
| ? | 匹配一个字符，但不匹配斜杠。 |
| [abc] | 匹配 bracket 中指定的一个字符，但不匹配斜杠。 |
| [a-z] | 匹配 bracket 中指定的范围中的一个字符，但不匹配斜杠。 |
| [!abc] | 匹配括号中未指定的字符； 不匹配斜杠。 |
| [!a-z] | 匹配 bracket 中指定的范围之外的一个字符； 不匹配斜杠。 |
| {one,two,three} | 匹配以逗号分开的任何 alternating 令牌；令牌可以包含通配符、嵌套更改和范围。 |
| \ | 转义字符。有三个级别的转义程序： **application.properties** parser、MicroProfile Config list converter 和 Glob parser。所有三个级别都使用反斜杠作为转义字符。 |

**其他资源**

- 配置红帽构建的 Quarkus 应用程序

### 1.3.1. 为红帽构建的 Quarkus 原生编译配置内存消耗

将红帽构建的 Quarkus 应用程序编译到原生可执行文件会在分析和优化过程中消耗大量内存。您可以通过设置 quarkus.native.native-image-xmx 配置属性来限制原生编译过程中使用的内存量。设置低内存限值可能会增加构建时间。

**流程**

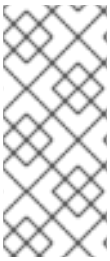
- 使用以下方法之一为 quarkus.native.native-image-xmx 属性设置值，以限制原生镜像构建期间的内存消耗：
  - 使用 application.properties 文件：

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

  - 设置系统属性：

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

此命令使用 Docker 构建原生可执行文件。要使用 Podman，请添加 -Dquarkus.native.container-runtime=podman 参数。

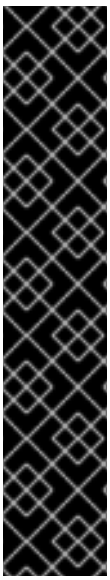


注意

例如，要将内存限制设置为 8 GB，请输入 quarkus.native.native-image-xmx=8g。该值必须是 1024 的倍数，大于 2MB。附加字母 m 或 M 表示兆字节，或 G 或 G 表示 GB。

### 1.4. 测试原生可执行文件

以原生模式测试应用，以测试原生可执行文件的功能。使用 @QuarkusIntegrationTest 注释来构建原生可执行文件，并根据 HTTP 端点运行测试。



重要

以下示例演示了如何使用本地安装 GraalVM 或 Mandrel 测试原生可执行文件。开始之前，请考虑以下点：

- Red Hat build of Quarkus 不支持这种情况，如 Producing a native executable 所述。
- 您在此处测试的原生可执行文件必须与主机的操作系统和架构匹配。因此，这个过程不适用于 macOS 或 in-container 构建。

**流程**

1. 打开 pom.xml 文件，并验证 build 部分是否具有以下元素：

```
<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>
```

- Maven Failsafe 插件(maven-failsafe-plugin)运行集成测试，并指示生成的原生可执行文件的位置。

2. 打开 src/test/java/org/acme/GreetingResourceIT.java 文件，并验证该文件是否包含以下内容：

```
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) 在测试之前，使用另一个测试运行程序从原生文件启动应用。可执行文件通过使用 Maven Failsafe 插件中配置的 native.image.path 系统属性来检索。

2

本例扩展了 GreetingResourceTest，但您也可以创建新的测试。

3. 运行测试：

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

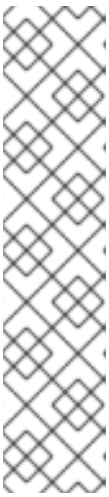
以下示例显示了这个命令的输出：

```
./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-23.1.0.0-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 {ProductLongName} 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
__  ____  __  _____   ___  __ ____  ______
 --/ __ \/ / / / _ | / _ \/ //_/ / / / __/
 -/ /_/ / /_/ / __ |/ , _/ ,< / /_/ /\ \
--\___\_\____/_/ |_/_/|_/_/|_|\____/___/
2024-02-21 14:04:52,681 INFO  [io.quarkus] (main) getting-started 1.0.0-SNAPSHOT
native (powered by Red Hat build of Quarkus 3.8.0.Final) started in 0.038s. Listening
on: http://0.0.0.0:8081
2024-02-21 14:04:52,682 INFO  [io.quarkus] (main) Profile prod activated.
2024-02-21 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 ---
```



**注意**

Quarkus 在自动失败原生测试前等待 60 秒启动原生镜像。您可以通过配置 quarkus.test.wait-time 系统属性来更改此持续时间。

您可以使用以下命令扩展等待时间，其中 < *duration*> 是等待时间（以秒为单位）：

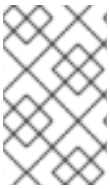
```
./mvnw verify -Dnative -Dquarkus.test.wait-time=<duration>
```

**注意**

- 默认情况下，使用 prod 配置集运行原生测试，除非在 quarkus.test.native-image-profile 属性中进行了修改。

### 1.4.1. 当作为原生可执行文件运行时排除测试

当您针对原生可执行文件运行测试时，您只能运行黑框测试，例如与应用的 HTTP 端点交互。



注意

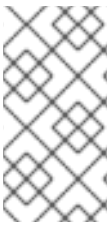
*黑色框* 指的是产品或程序的隐藏的内部工作，如黑色测试。

由于测试没有原生运行，所以无法在 JVM 上运行测试时链接到您的应用代码。因此，在您的原生测试中，您无法注入 Bean。

您可以在 JVM 和原生执行间共享测试类，并通过使用 @DisabledOnIntegrationTest 注释来仅在 JVM 上运行测试，从而排除某些测试。

### 1.4.2. 测试现有的原生可执行文件

通过使用 Failsafe Maven 插件，您可以针对现有的可执行构建进行测试。您可以在二进制代码构建后，以阶段方式运行多组测试。



注意

要测试您使用 Quarkus 生成的原生可执行文件，请使用可用的 Maven 命令。使用命令行没有等同的 Quarkus CLI 命令来完成此任务。

流程

- 针对已构建的原生可执行文件运行测试：

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

此命令使用 Failsafe Maven 插件针对现有的原生镜像运行测试。

- 另外，您可以使用以下命令指定原生可执行文件的路径，其中 < path> 是原生镜像路径：

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

## 1.5. 其他资源

- 将红帽构建的 Quarkus 应用程序部署到 OpenShift Container Platform
- 使用 Apache Maven 开发并编译您的红帽构建的 Quarkus 应用程序
- Quarkus 社区：构建原生可执行文件
- Apache Maven 项目
- Red Hat Universal Base Image 8 Minimal
- *UBI-minimal* 标签列表

*更新于 2024-05-10*