# Red Hat build of Quarkus 1.7

## Quarkus アプリケーションのネイティブ実行可能ファイルへのコンパイル

Last Updated: 2021-04-28

# Red Hat build of Quarkus 1.7 Quarkus アプリケーションのネイティブ実行可能ファイルへのコンパイル

## 法律上の通知



## 概要

本ガイドでは、Quarkus Getting Started プロジェクトをネイティブ実行可能ファイルにコンパイルする方法と、ネイティブ実行可能ファイルを設定してテストする方法を説明します。

# 目次

# 前書き

アプリケーション開発者は、Red Hat ビルドの Quarkus を使用して、OpenShift 環境およびサーバーレス環境で実行される Java で書かれたマイクロサービスを作成できます。ネイティブ実行可能ファイルにコンパイルされたアプリケーションは、メモリーのフットプリントが小さく、起動時間は高速です。

本ガイドでは、Quarkus Getting Started プロジェクトをネイティブ実行可能ファイルにコンパイルする方法と、ネイティブ実行可能ファイルを設定してテストする方法を説明します。**『Quarkus スタートガイド』** で作成したアプリケーションが必要です。

**Red Hat ビルドの Quarkus を使用したネイティブ実行可能ファイルのビルドでは、以下について説明します。**

- Podman または Docker などのコンテナーランタイムを使用した単一コマンドでのネイティブ実行可能ファイルのビルド
- 作成されたネイティブ実行可能ファイルを使用したカスタムコンテナーイメージの作成
- OpenShift Docker ビルドストラテジーを使用したコンテナーイメージの作成
- Quarkus ネイティブアプリケーションの OpenShift へのデプロイ
- ネイティブ実行可能ファイルの設定
- ネイティブ実行可能ファイルのテスト

**前提条件**

- OpenJDK (JDK) 11 がインストールされ、**JAVA_HOME** 環境変数が Java SDK の場所を指定していること。
  - Red Hat ビルドの Open JDK は、Red Hat カスタマーポータルの Software Downloads ページから入手可能です (ログインが必要です)。
- OCI (Open Container Initiative) と互換性のあるコンテナーランタイム (Podman または Docker など)。
- Quarkus Getting Started プロジェクトを完了していること。
  - Quarkus Getting Started プロジェクトのビルド方法は、**『Quarkus スタートガイド』** を参照してください。
  - あるいは、Quarkus quickstart archive をダウンロードするか、**Quarkus Quickstarts** Git リポジトリーをクローンしてください。プロジェクトのサンプルは、**getting-started** ディレクトリーにあります。

## 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、弊社 の CTO、Chris Wright のメッセージ を参照してください。

# 第1章 ネイティブ実行可能ファイルの作成

Podman または Docker などのコンテナーランタイムを使用して、Quarkus アプリケーションからネイティブ実行可能ファイルを作成することができます。Quarkus は、ビルダーイメージを使用してバイナリー実行可能ファイルを作成します。これは、Red Hat Universal Base Images RHEL8-UBI および RHEL8-UBI minimal と共に使用することができます。Red Hat ビルドの Quarkus 1.7 は、**quarkus.native.builder-image** プロパティーのデフォルトとして **registry.access.redhat.com/quarkus/mandrel-20-rhel8:20.3** を使用します。

お使いのアプリケーションのネイティブ実行可能ファイルには、アプリケーションコード、必須ライブラリー、Java API、および仮想マシン (VM) の縮小版が含まれます。縮小された仮想マシンベースは、アプリケーションの起動時間を高速化し、ディスクのフットプリントを小さくします。

**手順**

1. Getting Started プロジェクトの**pom.xml** ファイルを開き、**native** プロファイルが含まれていることを確認します。

```
<profiles>
  <profile>
    <id>native</id>
    <properties>
      <quarkus.package.type>native</quarkus.package.type>
    </properties>
  </profile>
</profiles>
```

**注記**

Quarkus **native** プロファイルを使用すると、ネイティブ実行可能ファイルおよびネイティブイメージテストの両方を実行することができます。

2. 以下のいずれかの方法を使用して、ネイティブ実行可能ファイルをビルドします。

   a. Docker を使用してネイティブ実行可能ファイルをビルドします。

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

   b. Podman を使用してネイティブ実行可能ファイルをビルドします。

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

   これらのコマンドは、**target** ディレクトリーに **getting-started-*-runner** バイナリーを作成します。

**重要**

Quarkus アプリケーションをネイティブ実行可能ファイルにコンパイルすると、分析および最適化の際にメモリーを大量に消費します。**quarkus.native.native-image-xmx** 設定プロパティーを設定することで、ネイティブコンパイル時に使用されるメモリーの量を制限することができます。メモリー制限を低く設定すると、ビルド時間が長くなる可能性があります。

3. ネイティブ実行可能ファイルを実行します。

```
./target/getting-started-*-runner
```

ネイティブ実行可能ファイルをビルドする場合、**prod** プロファイルが有効化され、Quarkus ネイティブテストは、**prod** プロファイルを使用して実行されます。これは、**quarkus.test.native-image-profile** プロパティーを使用して変更することができます。

# 第2章 カスタムコンテナーイメージの作成

以下のいずれかの方法を使用して、Quarkus アプリケーションからコンテナーイメージを作成できます。

- 手動でのコンテナーの作成
- OpenShift Docker ビルドを使用したコンテナーの作成

**重要**

Quarkus アプリケーションをネイティブ実行可能ファイルにコンパイルすると、分析および最適化の際にメモリーを大量に消費します。**quarkus.native.native-image-xmx** 設定プロパティーを設定することで、ネイティブコンパイル時に使用されるメモリーの量を制限することができます。メモリー制限を低く設定すると、ビルド時間が長くなる可能性があります。

## 2.1. 手動でのコンテナーの作成

本セクションでは、Linux X86_64 向けにアプリケーションを使用してコンテナーイメージを手動で作成する方法を説明します。Quarkus Native コンテナーを使用してネイティブイメージを作成する場合、Linux X86_64 オペレーティングシステムをターゲットとする実行可能ファイルを作成します。お使いのホストオペレーティングシステムが別のものである場合は、バイナリーを直接実行することはできないので、コンテナーを手動で作成する必要があります。

Quarkus Getting Started プロジェクトには、以下の内容と共に **src/main/docker** ディレクトリーに **Dockerfile.native** が含まれます。

```
FROM registry.access.redhat.com/ubi8/ubi-minimal
WORKDIR /work/
COPY target/*-runner /work/application
RUN chmod 775 /work
EXPOSE 8080
CMD ["./application", "-Dquarkus.http.host=0.0.0.0"]
```

**UNIVERSAL BASE IMAGE (UBI)**

**Dockerfiles** は、ベースイメージとして UBI を使用します。このベースイメージは、コンテナーで機能するように設計されています。**Dockerfiles** は、ベースイメージの minimal バージョン を使用して、作成されたイメージのサイズを縮小します。

**手順**

1. 以下のいずれかの方法を使用して、ネイティブ Linux 実行可能ファイルをビルドします。
   a. Docker を使用してネイティブ実行可能ファイルをビルドします。

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

   b. Podman を使用してネイティブ実行可能ファイルをビルドします。

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

2. 以下のいずれかの方法を使用して、コンテナーイメージをビルドします。

   a. Docker を使用してコンテナーイメージをビルドします。

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

   b. Podman を使用してコンテナーイメージをビルドします。

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

3. コンテナーを実行します。

   a. Docker を使用してコンテナーを実行します。

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

   b. Podman を使用してコンテナーを実行します。

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

Red Hat OpenShift Container Platform での Quarkus Maven アプリケーションのデプロイに関する詳細は、**『Red Hat OpenShift Container Platform での Quarkus アプリケーションのデプロイ』**を参照してください。

## 2.2. OPENSHIFT DOCKER ビルドを使用したコンテナーの作成

OpenShift Docker ビルドストラテジーを使用して、Quarkus アプリケーションのコンテナーイメージを作成できます。このストラテジーは、クラスターでビルド設定を使用してコンテナーを作成します。

**前提条件**

- Red Hat OpenShift Container Platform クラスターにアクセスでき、最新バージョンの OpenShift CLI (oc) がインストールされていること。oc のインストールに関する詳細は、**『OpenShift Container Platform クラスターのインストールおよび設定』**ガイドの「CLI のインストール」のセクションを参照してください。
- OpenShift API エンドポイントの URL。

**手順**

1. OpenShift CLI にログインします。

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

2. OpenShift に新規プロジェクトを作成します。

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

3. **src/main/docker/Dockerfile.native** ファイルをベースにビルド設定を作成します。

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

4. プロジェクトをビルドします。

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

5. プロジェクトを OpenShift にデプロイします。

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

# 第3章 ネイティブ実行可能ファイルの設定プロパティー

設定プロパティーは、ネイティブ実行可能ファイルの生成方法を定義します。**application.properties** ファイルを使用して、Quarkus アプリケーションを設定できます。

## 設定プロパティー

以下の表は、ネイティブ実行可能ファイルの生成方法を定義するよう設定できる設定プロパティーの一覧です。

| プロパティー | 説明 | タイプ | デフォルト |
|---|---|---|---|
| **quarkus.native.additional-build-args** | ビルドプロセスにパスする追加の引数。 | 文字列の一覧 | |
| **quarkus.native.enable-http-url-handler** | HTTP URL ハンドラーの有効化。これにより、HTTP URL に URL.openConnection() が可能となります。 | ブール値 | **true** |
| **quarkus.native.enable-https-url-handler** | HTTPS URL ハンドラーの有効化。これにより、HTTPS URL に URL.openConnection() が可能となります。 | ブール値 | **false** |
| **quarkus.native.enable-all-security-services** | ネイティブイメージにすべてのセキュリティーサービスを追加します。 | ブール値 | **false** |
| **quarkus.native.add-all-charsets** | ネイティブイメージにすべてのキャラクターセットを追加します。これにより、イメージサイズが大きくなります。 | ブール値 | **false** |
| **quarkus.native.graalvm-home** | Graal ディストリビューションのパスが含まれます。 | 文字列 | **${GRAALVM_HOME:}** |
| **quarkus.native.java-home** | JDK のパスが含まれます。 | ファイル | **${java.home}** |
| **quarkus.native.native-image-xmx** | ネイティブイメージを生成するために使用する Java の最大ヒープサイズ。 | 文字列 | |
| **quarkus.native.debug-build-process** | ネイティブイメージのビルドを実行する前に、デバッガーがビルドプロセスにアタッチするのを待ちます。GraalVM インターナルの知識のあるユーザーにとって、これは高度なオプションになります。 | ブール値 | **false** |

| プロパティー | 説明 | タイプ | デフォルト |
|---|---|---|---|
| **quarkus.native.publish-debug-build-process-port** | Docker を使用したビルド中にデバッグポートをパブリッシュし、debug-build-process が true の場合。 | ブール値 | **true** |
| **quarkus.native.cleanup-server** | ネイティブイメージサーバーを再起動します。 | ブール値 | **false** |
| **quarkus.native.enable-isolates** | メモリー管理を向上させるために分離を有効化します。 | ブール値 | **true** |
| **quarkus.native.enable-fallback-images** | ネイティブイメージが失敗した場合は、JVM ベースのフォールバックイメージを作成します。 | ブール値 | **false** |
| **quarkus.native.enable-server** | ネイティブイメージサーバーを使用します。これにより、コンパイルを高速化することは可能ですが、キャッシュの無効化問題により、ドロップが変更される可能性があります。 | ブール値 | **false** |
| **quarkus.native.auto-service-loader-registration** | すべての META-INF/services エントリーを自動的に登録します。 | ブール値 | **false** |
| **quarkus.native.dump-proxies** | インスペクション用にすべてのプロキシーのバイトコードをダンプします。 | ブール値 | **false** |
| **quarkus.native.container-build** | コンテナーランタイムを使用したビルド。デフォルトで Docker が使用されます。 | ブール値 | **false** |
| **quarkus.native.builder-image** | イメージをビルドするための Docker イメージ。 | 文字列 | **registry.access.redhat.com/quarkus/mandrel-20-rhel8:20.3** |
| **quarkus.native.container-runtime** | イメージをビルドするために使用されるコンテナーランタイム。たとえば、Docker などがあります。 | 文字列 | |
| **quarkus.native.container-runtime-options** | コンテナーランタイムにパスするオプション。 | 文字列の一覧 | |
| **quarkus.native.enable-vm-inspection** | イメージの VM イントロスペクションを有効化します。 | ブール値 | **false** |

| プロパティー | 説明 | タイプ | デフォルト |
|---|---|---|---|
| **quarkus.native.full-stack-traces** | イメージのフルスタックトレースを有効化します。 | ブール値 | **true** |
| **quarkus.native.enable-reports** | コールパスおよび含まれるパッケージ/クラス/メソッドのレポートを生成します。 | ブール値 | **false** |
| **quarkus.native.report-exception-stack-traces** | フルスタックトレースを使用して例外を報告します。 | ブール値 | **true** |
| **quarkus.native.report-errors-at-runtime** | ランタイム時にエラーを報告します。サポート対象外の機能を使用している場合は、お使いのアプリケーションがランタイム時に失敗する可能性があります。 | ブール値 | **false** |
| **quarkus.native.resources.includes** | ネイティブイメージに追加する必要のあるリソースパスに一致する glob のコンマ区切りリスト。すべてのプラットフォームで、スラッシュ (/) をパスセパレーターとして使用します。glob はスラッシュで開始してはいけません。たとえば、ソースツリーに **src/main/resources/ignored.png** と **src/main/resources/foo/selected.png** があり、依存関係 JAR のいずれかに **bar/some.txt** ファイルが含まれる場合、quarkus.native.resources.includes = foo/**,bar/**/*.txt の設定では **src/main/resources/foo/selected.png** ファイルおよび **bar/some.txt** ファイルはネイティブイメージに含まれますが、**src/main/resources/ignored.png** は含まれません。サポートされる glob 機能の詳細は、「サポートされる glob 機能とその説明」を参照してください。 | 文字列の一覧 | |
| **quarkus.native.debug.enabled** | デバッグを有効化し、別の .debug ファイルにデバッグシンボルを生成します。 | ブール値 | **false** |

### サポートされる glob 機能とその説明

以下の表は、サポートされる glob 機能とその説明を一覧表示しています。

| 文字 | 機能の説明 |
| --- | --- |
| * | スラッシュ (/) を含まない空の可能性のある文字列と一致します。 |
| ** | スラッシュ (/) を含むかもしれない空の可能性のある文字列と一致します。 |
| **?** | 1 つの文字と一致しますが、スラッシュとは一致しません。 |
| **[abc]** | ブラケットで指定した範囲の文字の 1 つと一致しますが、スラッシュとは一致しません。 |
| **[a-z]** | ブラケットで指定した範囲の文字の 1 つと一致しますが、スラッシュとは一致しません。 |
| **[!abc]** | ブラケットで指定していない文字の 1 つと一致しますが、スラッシュとは一致しません。 |
| **[a-z]** | ブラケットで指定した範囲外の文字の 1 つと一致しますが、スラッシュとは一致しません。 |
| **{one,two,three}** | コンマ区切り文字とトークンが交互に連続する文字列で、あらゆるトークンと一致します。トークンには、ワイルドカード、ネストされたオルタネーション、および範囲が含まれます。 |
| \ | エスケープ文字。エスケープには、**application.properties** パーサー、MicroProfile Config リストコンバーター、および Glob パーサーの 3 つのレベルがあります。3 つのレベルはすべて、バックスラッシュをエスケープ文字として使用します。 |

**その他のリソース**

- 『Quarkus アプリケーションの設定』

## 3.1. QUARKUS ネイティブコンパイルのメモリー消費の設定

Quarkus アプリケーションをネイティブ実行可能ファイルにコンパイルすると、分析および最適化の際にメモリーを大量に消費します。**quarkus.native.native-image-xmx** 設定プロパティーを設定することで、ネイティブコンパイル時に使用されるメモリーの量を制限することができます。メモリー制限を低く設定すると、ビルド時間が長くなる可能性があります。

**手順**

- 以下のいずれかの方法を使用して **quarkus.native.native-image-xmx** プロパティーに値を設定し、ネイティブイメージのビルドタイム中のメモリー消費を制限します。
  - **application.properties** ファイルの使用

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

- システムプロパティーの設定

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

このコマンドは、Docker を使用してネイティブ実行可能ファイルをビルドします。**-Dquarkus.native.container-runtime=podman** 引数を追加して Podman を使用します。

**注記**

たとえば、メモリー制限を 6 GB に設定するには、**quarkus.native.native-image-xmx=6g** と入力します。値は、2MB より大きい 1024 の倍数でなければなりません。メガバイトを示す m または M の文字を追加するか、ギガバイトを示す g または G の文字を追加します。

# 第4章 ネイティブ実行可能ファイルのテスト

ネイティブ実行可能ファイルの機能をテストするために、ネイティブモードで実行するアプリケーションをテストします。**@NativeImageTest** アノテーションを使用して、ネイティブ実行可能ファイルをビルドし、http エンドポイントに対してテストを実行します。

**手順**

1. **pom.xml** ファイルを開き、**native** プロファイルに以下の要素が含まれていることを確認します。

```
<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>
```

**failsafe-maven-plugin** はインテグレーションテストを実行し、作成されたネイティブ実行可能ファイルの場所を示します。

2. **src/test/java/org/acme/quickstart/NativeGreetingResourceIT.java** ファイルを開き、以下の内容が含まれていることを確認します。

```
package org.acme.quickstart;


import io.quarkus.test.junit.NativeImageTest;

@NativeImageTest (1)
public class NativeGreetingResourceIT extends GreetingResourceTest { (2)

    // Run the same tests

}
```

(1) テストの前に、ネイティブファイルからアプリケーションを開始する別のテストランナーを使用します。実行可能ファイルは、**Failsafe Maven Plugin** に設定された **native.image.path** システムプロパティーを使用して取得されます。

2 この例は、**GreetingResourceTest** を拡張しますが、新しいテストを作成することも可能です。

3. テストを実行します。

```
./mvnw verify -Pnative
```

以下の例は、このコマンドの出力を示しています。

```
./mvnw verify -Pnative
...
[getting-started-1.0-SNAPSHOT-runner:18820]     universe:     587.26 ms
[getting-started-1.0-SNAPSHOT-runner:18820]      (parse):   2,247.59 ms
[getting-started-1.0-SNAPSHOT-runner:18820]     (inline):   1,985.70 ms
[getting-started-1.0-SNAPSHOT-runner:18820]    (compile):  14,922.77 ms
[getting-started-1.0-SNAPSHOT-runner:18820]     compile:  20,361.28 ms
[getting-started-1.0-SNAPSHOT-runner:18820]       image:   2,228.30 ms
[getting-started-1.0-SNAPSHOT-runner:18820]       write:     364.35 ms
[getting-started-1.0-SNAPSHOT-runner:18820]     [total]:  52,777.76 ms
[INFO]
[INFO] --- maven-failsafe-plugin:2.22.1:integration-test (default) @ getting-started ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.acme.quickstart.NativeGreetingResourceIT
Executing [/data/home/gsmet/git/quarkus-quickstarts/getting-started/target/getting-started-
1.0-SNAPSHOT-runner, -Dquarkus.http.port=8081, -Dtest.url=http://localhost:8081, -
Dquarkus.log.file.path=build/quarkus.log]
2019-04-15 11:33:20,348 INFO  [io.quarkus] (main) Quarkus 999-SNAPSHOT started in
0.002s. Listening on: http://[::]:8081
2019-04-15 11:33:20,348 INFO  [io.quarkus] (main) Installed features: [cdi, resteasy]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.387 s - in
org.acme.quickstart.NativeGreetingResourceIT
...
```

**注記**

Quarkus は、ネイティブイメージの開始まで 60 秒間待機し、その後ネイティブテストに自動的に失敗します。この待機時間は、**quarkus.test.native-image-wait-time** システムプロパティーを使用して変更可能です。

以下のコマンドを使用して、待機時間を延長することができます。**<duration>** は、秒単位の待機時間になります。

```
./mvnw verify -Pnative -Dquarkus.test.native-image-wait-time=<duration>
```

## 4.1. ネイティブ実行可能ファイルとして実行する場合のテストを除外する

ネイティブアプリケーションに対してテストを実行する場合、HTTP エンドポイントと対話することしかできません。テストはネイティブでは実行されないため、お使いのアプリケーションのコードに対して、JVM での実行でリンクできる場合と同じようにリンクすることはできません。

JVM とネイティブ実行可能ファイルとの間でテストクラスを共有することができ、**@DisabledOnNativeImage** アノテーションを使用して特定のテストを除外して JVM でのみ実行することができます。

## 4.2. 既存のネイティブ実行可能ファイルのテスト

既存の実行可能ファイルのビルドに対してテストすることができます。これにより、バイナリーのビルド後にバイナリーで複数のテストのセットを段階的に実行することができます。

**手順**

- すでにビルドされたネイティブ実行可能ファイルに対してテストを実行します。

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

  このコマンドは、**Failsafe Maven Plugin** を使用して、既存のネイティブイメージに対してテストを実行します。

- あるいは、以下のコマンドを使用してネイティブ実行可能ファイルへのパスを指定することもできます。**<path>** は、ネイティブイメージパスになります。

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

# 第5章 その他のリソース

- 『Quarkus アプリケーションのテスト』
- 『Red Hat OpenShift Container Platform での Quarkus アプリケーションのデプロイ』
- 『Apache Maven を使用した Quarkus アプリケーションの開発およびコンパイル』
- Apache Maven Project
- UBI イメージページ
- **UBI-minimal** イメージページ
- **UBI-minimal** タグの一覧

**改訂日時:** 2021-04-27 03:44:13 UTC