Red Hat build of Quarkus 3.2

Red Hat build of 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

Last Updated: 2024-05-10

Red Hat build of Quarkus 3.2 Red Hat build of 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

法律上の通知

概要

このガイドでは、Red Hat build of 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

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、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 RED HAT BUILD OF 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

アプリケーション開発者は、Red Hat build of Quarkus 3.2 を使用して、OpenShift Container Platform 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。Quarkus アプリケーションは、通常の Java アプリケーションとして (Java 仮想マシン上で) 実行することも、ネイティブ実行可能ファイルにコンパイルすることもできます。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、対応する Java アプリケーションよりもメモリーフットプリントが小さく、短時間で起動します。

このガイドでは、Red Hat build of Quarkus 3.2 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。Red Hat build of Quarkus スタートガイド で作成したアプリケーションが必要になります。

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て、シングルコマンドでネイティブ実行可能ファイルをビルドします。
- 生成されたネイティブ実行可能ファイルを使用して、カスタムコンテナーイメージを作成します。
- OpenShift Container Platform Docker ビルドストラテジーを使用して、コンテナーイメージを作成します。
- Quarkus ネイティブアプリケーションを OpenShift Container Platform にデプロイします。
- 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 17 がインストールされ、**JAVA_HOME** 環境変数が Java SDK の場所を指定するように設定されている。
  - Red Hat build of OpenJDK は、Red Hat カスタマーポータルにログインして ソフトウェアダウンロード ページからダウンロードできます。
- Open Container Initiative (OCI) と互換性のあるコンテナーランタイム (Podman、Docker など)。
- Quarkus Getting Started プロジェクトを完了している。
  - 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。

1.1. ネイティブ実行可能ファイルの作成

ネイティブバイナリーは、特定のオペレーティングシステムおよび CPU アーキテクチャー上で実行するために作成された実行可能ファイルです。

以下に、ネイティブ実行可能ファイルの例をいくつかリストします。

- Linux AMD 64 ビット用の ELF バイナリー
- Windows AMD 64 ビット用の EXE バイナリー
- ARM 64 ビット用の ELF バイナリー

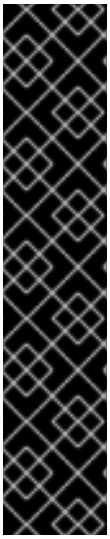
ネイティブ実行可能ファイルをビルドする場合の利点の 1 つは、アプリケーションと JVM を含む依存関係が 1 つのファイルにパッケージ化されることです。アプリケーションのネイティブ実行可能ファイルには、次の項目が含まれています。

- コンパイルされたアプリケーションコード。
- 必要な Java ライブラリー。
- アプリケーションの起動時間を短縮し、ディスクとメモリーのフットプリントを最小限に抑えるための Java 仮想マシン (JVM) の縮小バージョン。これは、アプリケーションコードとその依存関係に合わせて調整されています。

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生成する場合は、コンテナ内ビルドまたはローカルホストビルドのいずれかを選択できます。次の表は、使用可能なビルドオプションとその説明を示しています。

表1.1 ネイティブ実行可能ファイルの生成に使用できるビルドオプション

ビルドオプション	Requires	用途	結果	利点
コンテナ内ビルド - サポート対象	コンテナーランタイム (Podman や Docker など)	デフォルトの **registry.access.redhat.com/quarkus/mandrel-23-rhel8:23.0** ビルダーイメージ	ホストの CPU アーキテクチャーを使用する Linux 64 ビット実行可能ファイル	GraalVM はローカルで設定する必要がないため、CI パイプライン の実行効率が向上します。
ローカルホストビルド - アップストリームでのみサポート対象	GraalVM または Mandrel のローカルインストール	**quarkus.native.builder-image** プロパティーのデフォルトのローカルインストール	ビルドが実行されるマシンと同じオペレーティングシステムおよび CPU アーキテクチャーを持つ実行可能ファイル	Docker や Podman などのツールを使用できない、または使用しない開発者向けの代替手段。総じて、コンテナ内ビルドアプローチよりも高速です。



重要

- Red Hat build of Quarkus 3.2 は、Mandrel の製品化されたディストリビューションである、Java 17 ベースの Red Hat build of Quarkus ネイティブビルダー 使用した、ネイティブ Linux 実行可能ファイルのビルドのみをサポートします。他のイメージはコミュニティーで入手できますが、製品ではサポートされていないため、Red Hat のサポートを希望する実稼働ビルドには使用しないでください。
- ソースが Java 11 ベースで記述され、Java 12 - 17 の機能が使用されていないアプリケーションでも、Java 17 ベースの Mandrel 23.0 ベースイメージを使用してそのアプリケーションのネイティブ実行可能ファイルをコンパイルできます。
- Oracle GraalVM Community Edition (CE)、Mandrel Community Edition、またはその他の GraalVM ディストリビューションを使用したネイティブ実行可能ファイルのビルドは、Red Hat build of Quarkus ではサポートされていません。

1.1.1. コンテナ内ビルドを使用してネイティブ実行可能ファイルを生成する

ネイティブ実行可能ファイルを作成し、ネイティブイメージテストを実行するには、コンテナ内ビルド用に Red Hat build of Quarkus が提供する **native** プロファイルを使用します。

前提条件

- Podman または Docker がインストールされている。
- コンテナーは、8 GB 以上のメモリーにアクセスできる。

手順

1. Getting Started プロジェクトの **pom.xml** ファイルを開き、プロジェクトに **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>
    <activation>
      <property>
        <name>native</name>
      </property>
    </activation>
    <properties>
      <skipITs>false</skipITs>
      <quarkus.package.type>native</quarkus.package.type>
    </properties>
  </profile>
</profiles>
```

2. 次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。
   - Maven の使用:
     - Docker の場合:

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

- Podman の場合:

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

- Quarkus CLI を使用する:
  - Docker の場合:

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

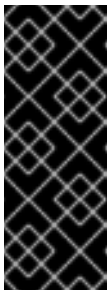
  - Podman の場合:

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

手順を実行した結果

これらのコマンドを実行すると、**target** ディレクトリーに ***-runner** バイナリーが作成されます。その場合、以下が適用されます。

- ***-runner** ファイルは、Quarkus が生成したビルド済みのネイティブバイナリーです。
- **target** ディレクトリーは、Maven アプリケーションをビルドするときに Maven が作成するディレクトリーです。



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して、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するには、以下のコマンドを入力します。

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

関連情報

- ネイティブ実行可能ファイルの設定プロパティー

1.1.2. ローカルホストビルドを使用してネイティブ実行可能ファイルを生成する

Docker または Podman を使用しない場合は、ネイティブ実行可能ファイルの作成と実行に Quarkus ローカルホストビルドオプションを使用します。

ローカルホストビルドアプローチを使用すると、コンテナーを使用した場合と比べて高速になるため、Linux オペレーティングシステムを使用するマシンに適しています。

重要

Red Hat build of Quarkus では、次に示す手順の実稼働環境での使用はサポートされません。この方法は、テストで、もしくは Docker や Podman が使用できない場合のバックアップアプローチとしてのみ使用してください。

前提条件

- Building a native executable ガイドに沿って正しく設定された、Mandrel または GraalVm のローカルインストール。
  - GraalVM インストールの場合は、**native-imageジ** もインストールされている必要があります。

手順

1. GraalVM または Mandrel の場合は、次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。
   - Maven の使用:

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

   - Quarkus CLI を使用する:

     ```
     quarkus build --native
     ```

     手順を実行した結果

     これらのコマンドを実行すると、**target** ディレクトリーに ***-runner** バイナリーが作成されます。その場合、以下が適用されます。
     - ***-runner** ファイルは、Quarkus が生成したビルド済みのネイティブバイナリーです。
     - **target** ディレクトリーは、Maven アプリケーションをビルドするときに Maven が作成するディレクトリーです。

注記

ネイティブ実行可能ファイルをビルドすると、**quarkus.profile** プロパティーで変更されない限り、**prod** プロファイルが有効になります。

2. ネイティブ実行可能ファイルを実行します。

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

関連情報

詳細は、Quarkus コミュニティーの "Building a native executable" ガイドで Producing a native executable セクションを参照してください。

1.2. 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。

- 手動でコンテナーを作成します。
- OpenShift Container Platform Docker ビルドを使用してコンテナーを作成します。

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して、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。

1.2.1. 手動でコンテナーを作成します。

このセクションでは、 Linux AMD64 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 ネイティブコンテナーを使用してネイティブイメージを生成すると、そのネイティブイメージは Linux AMD64 をターゲットとする実行可能ファイルを作成します。ホストオペレーティングシステムが Linux AMD64 ではない場合、バイナリーは直接実行できず、手動でコンテナーを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: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"]
```

注記

Universal Base Image (UBI)

次のリストは、Dockerfile での使用に適したイメージを示しています。

- Red Hat Universal Base Image 8 (UBI8)。このベースイメージは、コンテナー化されたすべてのアプリケーション、ミドルウェア、ユーティリティーのベースレイヤーになるように設計されています。

```
registry.access.redhat.com/ubi8/ubi:8.8
```

- Red Hat Universal Base Image 8 Minimal (UBI8-minimal)。microdnf をパッケージマネージャーとして使用する、最小化された UBI8 イメージ。

```
registry.access.redhat.com/ubi8/ubi-minimal:8.8
```

- すべての Red Hat Base イメージは、コンテナーイメージ カタログサイトで入手できます。

手順

1. 次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。
   - Docker:

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

   - Podman:

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

2. 次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。
   - 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. 次のいずれかの方法を使用して、コンテナーを実行します。
   - 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. OpenShift Docker ビルドを使用してコンテナーを作成する

OpenShift Container Platform 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、クラスター内のビルド設定を使用してコンテナーイメージを作成します。

前提条件

- OpenShift Container Platform クラスターとインストールされている **oc** ツールの最新バージョンにアクセスできる。**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 を参照してください。
- OpenShift Container Platform 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** ファイルに基づいて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 Container Platform にデプロイします。

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

6. サービスを公開するには、以下を実行します。

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

1.3. 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リストです。

プロパティー	説明	型	デフォルト

quarkus.native.debug.enabled	デバッグが有効でデバッグシンボルが生成される場合、そのシンボルは別の **.debug** ファイルに生成されます。	boolean	false
quarkus.native.resources.excludes	ネイティブイメージに追加するべきではないリソースパスに一致する glob のコンマ区切りリスト。	文字列のリスト	
quarkus.native.additional-build-args	ビルドプロセスにパスする追加の引数。	文字列のリスト	
quarkus.native.enable-http-url-handler	HTTP URL ハンドラーを有効にします。これにより、HTTP URL に対して **URL.openConnection()** を実行できるようになります。	boolean	**true**
quarkus.native.enable-https-url-handler	HTTPS URL ハンドラーを有効にします。これにより、HTTPS URL に対する **URL.openConnection()** が可能になります。	boolean	**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。	boolean	**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。	boolean	**false**
quarkus.native.graalvm-home	GraalVM ディストリビューションのパスが含まれます。	string	**${GRAALVM_HOME:}**
quarkus.native.java-home	JDK のパスが含まれます。	ファイル	**${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。	string	
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まで待機します。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。	boolean	**false**

quarkus.native.publish-debug-build-process-port	**debug-build-process** が **true** の場合、docker でビルドする際にデバッグポートを公開します。	boolean	**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。	boolean	**false**
quarkus.native.enable-isolates	メモリー管理を改善するために分離を有効にします。	boolean	**true**
quarkus.native.enable-fallback-images	ネイティブイメージが失敗した場合に、JVM ベースのフォールバックイメージを作成します。	boolean	**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が高速化されますが、を高速化できますが、キャッシュ無効化の問題により変更が失われる可能性があります。	boolean	**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。	boolean	**false**
quarkus.native.dump-proxies	検査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。	boolean	**false**
quarkus.native.container-build	コンテナーランタイムを使用するビルド。デフォルトで Docker が使用されます。	boolean	**false**
quarkus.native.builder-image	イメージをビルドするための Docker イメージ。	string	**registry.access.redhat.com/quarkus/mandrel-23-rhel8:23.0**
quarkus.native.container-runtime	イメージのビルド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。	string	
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。	文字列のリスト	
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。	boolean	**false**
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。	boolean	**true**

quarkus.native.enable-reports	呼び出しパスと、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。	boolean	**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。	boolean	**true**
quarkus.native.report-errors-at-runtime	ランタイム時にエラーを報告します。これにより、サポートされていない機能を使用すると、ランタイムでアプリケーションが失敗する可能性があります。	boolean	**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、パス区切り文字としてスラッシュ (/) 文字を使用します。glob の開始文字をスラッシュにしてはいけません。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 1 つに **bar/some.txt** ファイルが含まれ、**quarkus.native.resources.includes** が **foo/,bar//*.txt** に設定されている と、**src/main/resources/foo/selected.png** ファイルと **bar/some.txt** ファイルが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。詳細は、サポートされている glob 機能をリストした次の表を参照してください。	文字列のリスト	
quarkus.native.debug.enabled	デバッグを有効にし、別の **.debug** ファイルにデバッグシンボルを生成します。**quarkus.native.container-build** と使用する場合、Red Hat build of Quarkus は、ネイティブイメージからのデバッグ情報を分割する **objcopy** ユーティリティーをインストールする **binutils** パッケージを含む Red Hat Enterprise Linux または他の Linux ディストリビューションのみをサポートします。	boolean	**false**

サポートされている glob 機能

次の表は、サポートされる glob 機能とその説明をリスト化したものです。

Character	機能の説明
*	スラッシュ (/) を含まない、空の可能性もある文字列を照合します。
**	スラッシュ (/) を含むかもしれない、空の可能性もある文字列を照合します。
?	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]	括弧内に指定された 1 文字を照合しますが、スラッシュは照合しません。
[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}	コンマで区切られた代替のトークンを照合します。トークンには、ワイルドカード、ネストされた論理和指定子、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。

関連情報

- Red Hat build of Quarkus アプリケーションの設定

1.3.1. Red Hat build of Quarkus ネイティブコンパイレーションのメモリー消費を設定する

Red Hat build of 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、分析と最適化中に大量のメモリーが消費されます。**quarkus.native.native-image-xmx** 設定プロパティーを設定して、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** ファイルを使用します。

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

  - システムプロパティーの設定

```
mvn package -Dnative -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。Podman を使用するには、**-Dquarkus.native.container-runtime=podman** 引数を追加します。

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** を入力します。値は、2MB より大きい 1024 の倍数にする必要があります。メガバイトを示す **m** または **M**、ギガバイトを示す **g** または **G** の文字を追加します。

1.4. 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には、ネイティブモードでアプリケーションをテストします。**@QuarkusIntegrationTest** 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。

重要

次の例は、GraalVM または Mandrel のローカルインストールを使用してネイティブ実行可能ファイルをテストする方法を示しています。テストを実行する前に、次の点を考慮してください。

- ネイティブ実行可能ファイルを作成する で説明されているとおり、このシナリオは Red Hat build of Quarkus でサポートされていません。
- ここでテストするネイティブ実行可能ファイルは、ホストのオペレーティングシステムとアーキテクチャーに一致する必要があります。つまり、この手順は macOS やコンテナ内ビルドでは機能しません。

手順

1. **pom.xml** ファイルを開き、**build** セクションに次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>
```

- Maven Failsafe プラグイン (**maven-failsafe-plugin**) は結合テストを実行し、生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/GreetingResourceIT.java** ファイルを開き、次のコンテンツが含まれていることを確認します。

```
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) 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、Maven Failsafe プラグインで設定された **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。

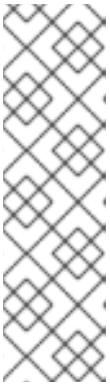
3. テストを実行します。

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

以下の例は、このコマンドの出力を示しています。

```
./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 ---
```

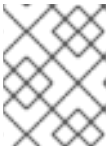


注記

Quarkus は、ネイティブイメージが起動するのを 60 秒間待機してから、自動的にネイティブテストに失敗します。この期間は、**quarkus.test.wait-time** システムプロパティーを設定することで変更できます。

以下のコマンドを使用して、待機時間を延長できます。この場合の **<duration>** は待機時間 (秒単位) です。

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



注記

- **quarkus.test.native-image-profile** プロパティーで変更されない限り、ネイティブテストはデフォルトで **prod** プロファイルを使用して実行されます。

1.4.1. 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

ネイティブ実行可能ファイルに対してテストを実行する場合、たとえばアプリケーションの HTTP エンドポイントと対話するなどの、ブラックボックステストのみを実行できます。

注記

ブラックボックス とは、ブラックボックステストなど、製品またはプログラムの隠された内部動作を指します。

テストはネイティブで実行されないため、JVM でテストを実行する場合と同様に、アプリケーションのコードにはリンクできません。したがって、ネイティブテストでは Bean を注入できません。

JVM とネイティブ実行の間でテストクラスを共有し、**@DisabledOnNativeImage** アノテーションを使用して特定のテストを除外して、その JVM 上でのみテストを実行できます。

1.4.2. 既存のネイティブ実行可能ファイルのテスト

Failsafe Maven プラグインを使用すると、既存の実行可能ビルドに対してテストを実行できます。ビルド後に、バイナリーで複数のテストセットを段階的に実行できます。

注記

Quarkus で生成したネイティブ実行可能ファイルをテストするには、利用可能な Maven コマンドを使用します。コマンドラインを使用してこのタスクを完了できる同等の Quarkus CLI コマンドはありません。

手順

- ビルド済みの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 プラグインを使用して、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>
```

1.5. 関連情報

- Red Hat build of Quarkus アプリケーションを OpenShift Container Platform にデプロイする
- Apache Maven を使用した Red Hat build of Quarkus アプリケーションの開発とコンパイル
- Quarkus community: Building a native executable
- Apache Maven Project
- UBI イメージページ
- **UBI-minimal** イメージページ
- **UBI-minimal** タグのリスト

改訂日時: 2024-05-10