Red Hat build of Quarkus 2.13

Quarkus スタートガイド

Last Updated: 2024-04-24

法律上の通知

概要

このガイドでは、Apache Maven を使用して簡単な Quarkus アプリケーションを作成する方法を説明します。

目次

第1章 QUARKUS スタートガイド

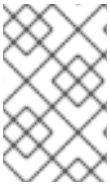
アプリケーション開発者は、Red Hat ビルドの Quarkus を使用して、OpenShift 環境で実行される Java で書かれたマイクロサービスベースのアプリケーションを作成できます。Quarkus アプリケーションは、Java 仮想マシン (JVM) 上で実行することも、ネイティブの実行可能ファイルにコンパイルすることもできます。ネイティブアプリケーションは、JVM アプリケーションよりもメモリー使用量が小さく、短時間で起動します。

Quarkus アプリケーションは、次のいずれかの方法で作成できます。

- Apache Maven と Quarkus Maven プラグインを使用する
- code.quarkus.redhat.com を使用する
- Quarkus コマンドラインインターフェイス (CLI) を使用する

Quarkus を使用するにあたり、まずは **hello** HTTP エンドポイントを公開する単純な Quarkus プロジェクトを作成、テスト、パッケージ化、実行してみましょう。依存性注入のデモンストレーションとして、**hello** HTTP エンドポイントは **greeting** Bean を使用します。

78_OpenShift_0420



注記

Getting Started の演習の完全な例については、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 **getting-started** ディレクトリーに移動します。

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

1.1. RED HAT BUILD OF QUARKUS について

Red Hat ビルドの Quarkus は、コンテナーおよび Red Hat OpenShift Container Platform と使用するために最適化された Kubernetes ネイティブ Java スタックです。Quarkus は、Eclipse MicroProfile、Eclipse Vert.x、Apache Camel、Apache Kafka、Hibernate ORM with Java Persistence API (JPA)、RESTEasy (JAX-RS)などの一般的な Java 標準、フレームワーク、およびライブラリーと連携するように設計されています。

開発者は、Java アプリケーションに必要な Java フレームワークを選択できます。これは、Java 仮想マシン (JVM) モードで実行することも、ネイティブモードでコンパイルして実行することもできます。Quarkus は、コンテナーファーストという手法で Java アプリケーションをビルドします。コンテナーファーストのアプローチにより、コンテナー化と、マイクロサービスと関数の効率的な実行が容易になります。このため、Quarkus アプリケーションのメモリーフットプリントは小さく、起動時間が短縮されます。

Quarkus はまた、統合設定、未設定のサービスの自動プロビジョニング、ライブコーディング、コード変更に関する即時フィードバックを提供する継続的テストなどの機能により、アプリケーション開発プロセスを最適化します。

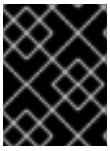
Quarkus コミュニティーバージョンと Red Hat ビルドの Quarkus の相違点は、Quarkus コミュニティーバージョン と Red Hat ビルド の Quarkus の相違点 を参照し てください。

1.2. 環境の準備

Quarkus の使用を開始する前に、環境を準備する必要があります。

手順

- システムで次のインストールが完了していることを確認します。
  - OpenJDK 11 または 17 をインストールし、**JAVA_HOME** 環境変数を設定して Java SDK の場所を指定している。
    - Red Hat build of OpenJDK をダウンロードするには、Red Hat カスタマーポータルにログインし、ソフトウェアダウンロード に移動します。
  - Apache Maven 3.8.x がインストールされている。x は 6 以降です。Maven は、Apache Maven Project の Web サイトから入手できます。
  - **オプション**: Quarkus コマンドラインインターフェイス (CLI) を使用する場合は、それがインストールされていることを確認してください。
    - Quarkus CLI のインストール方法については、Quarkus CLI のコミュニティー固有の情報を参照してください。



重要

Quarkus CLI は、開発モードのみを対象としています。Red Hat は、実稼働環境での Quarkus CLI の使用をサポートしません。

1.2.1. Quarkus BOM について

Red Hat build of Quarkus 2.2 以降、すべてのコア Quarkus エクステンションの依存関係バージョンは、**com.redhat.quarkus.platform:quarkus-bom** ファイルを使用して管理されます。

Bill of Materials (BOM) ファイルの目的は、プロジェクト内の Quarkus アーティファクトの依存関係バージョンを管理することです。これにより、プロジェクトで BOM を使用するときに、どの依存関係バージョンが連携するかを指定する必要がなくなります。代わりに、Quarkus BOM ファイルを **pom.xml** 設定ファイルにインポートできます。依存関係のバージョンは **<dependencyManagement>** セクションに含まれています。そのため、**pom.xml** ファイルの指定の BOM で管理される個別の Quarkus 依存関係のバージョンを記述する必要はありません。

Red Hat ビルドの Quarkus で利用可能なサポート対象のエクステンション固有の BOM に関する情報は、Red Hat build of Quarkus Component Details を参照してください。

アプリケーションで使用するプラットフォームメンバーのエクステンションのメンバー固有 BOM をインポートのみインポートする必要があります。したがって、モノリシックなシングル BOM と比較して、管理する依存関係が少なくなります。すべてのメンバー固有 BOM はユニバーサル Quarkus BOM のフラグメントであるため、競合を引き起こすことなくメンバー BOM を任意の順序でインポートできます。

1.2.2. Apache Maven および Quarkus について

Apache Maven は分散型構築自動化ツールで、Java アプリケーション開発でソフトウェアプロジェクトの作成、ビルド、管理に使用されます。Maven は Project Object Model (POM) ファイルと呼ばれる標準の設定ファイルを使用して、プロジェクトの定義や構築プロセスの管理を行います。POM ファイルには、モジュールとコンポーネントの依存関係、ビルド順序、結果として得られるプロジェクトのパッケージ化と出力のターゲットがXML ファイルを使用して記述されており、これによりプロジェクトが正しく均一にビルドされることが保証されます。

Maven リポジトリー

Maven リポジトリーには、Java ライブラリー、プラグイン、およびその他のビルドアーティファクトが格納されます。デフォルトのパブリックリポジトリーは Maven 2 Central Repository ですが、複数の開発チームの間で共通のアーティファクトを共有する目的で、社内のプライベートおよび内部リポジトリーとすることが可能です。また、サードパーティーのリポジトリーも利用できます。

Red Hat がホストする Maven リポジトリーを Quarkus プロジェクトで使用するか、Red Hat build of Quarkus Maven リポジトリーをダウンロードできます。

Maven プラグイン

Maven プラグインは、1 つ以上のタスクを実行する POM ファイルの定義済みの部分です。Red Hat build of Quarkus アプリケーションでは、次の Maven プラグインを使用します。

- **Quarkus Maven プラグイン(quarkus-maven-plugin)**: Maven による Quarkus プロジェクトの作成、アプリケーションを JAR ファイルにパッケージ化し、開発モードを提供できるようにします。

- **Maven Surefire プラグイン(maven-surefire-plugin)**: Quarkus が **テスト** プロファイルを有効にする場合、Maven Surefire プラグインはビルドライフサイクルの **テスト** フェーズで使用され、アプリケーションでユニットテストを実行します。プラグインは、テストレポートが含まれる

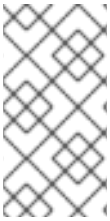
テキストファイルと XML ファイルを生成します。

関連情報

- Quarkus アプリケーションの設定

1.2.3. オンラインリポジトリーの Maven の settings.xml ファイルを設定する

Red Hat がホストする Quarkus リポジトリーを Quarkus Maven プロジェクトで使用するには、ユーザー用に **settings.xml** ファイルを設定します。リポジトリーマネージャーまたは共有サーバー上のリポジトリーで使用される Maven 設定により、プロジェクトの制御と管理が向上します。



注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。設定を特定のプロジェクトにのみ適用する場合は、**-s** オプションを使用して、プロジェクト固有の **settings.xml** ファイルへのパスを指定します。

手順

1. テキストエディターまたは統合開発環境 (IDE) で、Maven **$HOME/.m2/settings.xml** ファイルを開きます。

注記

$ **HOME/.m2/ ディレクトリーに settings.xml ファイルがない場合は、$ MAVEN_HOME/.m2 /conf/ ディレクトリーの settings.xml ファイルを $HOME/.m2/** ディレクトリーにコピーします。

2. 以下の行を Maven の **settings.xml** ファイルの **<profiles>** 要素に追加します。

```
<!-- 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. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

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

1.2.4. Quarkus Maven リポジトリーの設定

オンライン Maven リポジトリーを使用しない場合は、Quarkus Maven リポジトリーをダウンロードして設定できます。Quarkus Maven リポジトリーには、Java 開発者がアプリケーションの構築に使用する依存関係が複数含まれています。この手順では、**settings.xml** ファイルを編集して Quarkus Maven リポジトリーを設定する方法を説明します。

注記

Maven の **settings.xml** ファイルを変更してリポジトリーを設定する場合、変更はすべての Maven プロジェクトに適用されます。特定のプロジェクトに設定を適用する場合は、**-s** オプションを使用して、プロジェクト固有の **settings.xml** ファイルへのパスを指定します。

手順

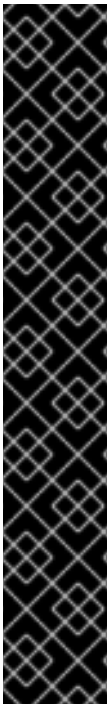
1. Red Hat カスタマーポータル にログインし、Software Downloads に移動し、Quarkus Maven リポジトリーの ZIP ファイルをダウンロードします。
2. ダウンロードしたアーカイブをデプロイメントします。
3. **$HOME/.m2/** ディレクトリーに移動し、テキストエディターまたは統合開発環境(IDE)で Maven の **settings.xml** ファイルを開きます。
4. ダウンロードした Quarkus Maven リポジトリーのパスを、**settings.xml** ファイルの **<profiles>** 要素に追加します。Quarkus Maven リポジトリーのパスの形式は、**file:// $PATH にする必要があります（例：file:///home/userX/<root-directory-of-the-downloaded-archive>/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>file:///path/to/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>file:///path/to/Quarkus/Maven/repository/</url>
    <releases>
     <enabled>true</enabled>
    </releases>
    <snapshots>
     <enabled>false</enabled>
    </snapshots>
   </pluginRepository>
  </pluginRepositories>
</profile>
```

5. 以下の行を **settings.xml** ファイルの **<activeProfiles>** 要素に追加し、ファイルを保存します。

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



重要

Maven リポジトリーに古いアーティファクトが含まれる場合は、プロジェクトをビルドまたはデプロイしたときに以下のいずれかの Maven エラーメッセージが表示されることがあります。

- **Missing artifact <project_name>**
- **[ERROR] Failed to execute goal on project <artifact_name>; Could not resolve dependencies for <project_name>**

ここでは、以下のようになります。

- & **lt;ARTIFACT_NAME&gt**; は、欠落しているアーティファクトの名前です。
- **<PROJECT_NAME** > - ビルドを試みているプロジェクトの名前。

この問題を解決するには、**$HOME/.m2/** repository ディレクトリーにあるローカルリポジトリーのキャッシュバージョンを削除して、最新の Maven アーティファクトを強制的にダウンロードします。

1.2.5. Maven プロジェクトを Red Hat build of Quarkus に再設定する

Quarkus コミュニティープロジェクトは、プロジェクト POM ファイルの Maven 設定を変更することで Red Hat build of Quarkus に移行できます。

前提条件

- **pom.xml** ファイルの Quarkus コミュニティーアーティファクトに依存する Maven でビルドされる Quarkus プロジェクトがあります。

手順

- プロジェクトの **pom.xml** ファイルの **<properties>** セクションで、次の値を変更します。
  - **<quarkus.platform.group-id>** プロパティーの値を **com.redhat.quarkus.platform** に変更します。

- < **quarkus.platform.version** > プロパティーの値を 2 **.13.9.SP2-redhatTRUSTED に変更します。**

  pom.xml

```
<project>
  ...
  <properties>
    ...
    <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
    <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
    <quarkus.platform.version>2.13.9.SP2-redhat-00003</quarkus.platform.version>
    ...
  </properties>
  ...
</project>
```

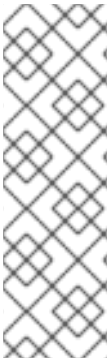
1.3. QUARKUS 開発者ツールの設定

Quarkus 開発者ツールを使用すると、次のようなタスクを完了できます。

- アプリケーション用の Maven プロジェクトの作成
- アプリケーションで使用するエクステンションの追加と設定
- OpenShift クラスターへのアプリケーションのデプロイ

1.3.1. Quarkus エクステンションレジストリークライアントの設定

エクステンションレジストリー **registry.quarkus.redhat.com** は、Red Hat が提供する Quarkus エクステンションをホストします。レジストリーをレジストリークライアント設定ファイルに追加することで、このレジストリー内のエクステンションにアクセスするように Quarkus 開発者ツールを設定できます。レジストリークライアント設定ファイルは、レジストリーリストが含まれる YAML ファイルです。



注記

- デフォルトの Quarkus レジストリーは **registry.quarkus.io** ですが、通常、これを設定する必要はありません。ただし、ユーザーがカスタムレジストリーリストを提供し、そのリストに **registry.quarkus.io** が含まれていない場合、**registry.quarkus.io** は有効になりません。
- 目的のレジストリーがレジストリーリストの最初に表示されていることを確認してください。Quarkus 開発者ツールがレジストリーを検索する場合、リストの先頭からレジストリーを検索します。

手順

1. エクステンションレジストリー設定が含まれる **config.yaml** ファイルを開きます。エクステンションレジストリーの初回設定時には、マシン上の **<user_home_directory_name>/.quarkus** ディレクトリーに **config.yaml** ファイルを作成する必要がある場合があります。
2. 新しいレジストリーを **config.yaml** ファイルに追加します。以下に例を示します。

config.yaml

```
registries:
  - registry.quarkus.redhat.com
  - registry.quarkus.io
```

1.4. GETTING STARTED プロジェクトの作成

getting-started プロジェクトを作成すると、単純な Quarkus アプリケーションを起動して実行できます。**getting-started** プロジェクトは、次のいずれかの方法で作成できます。

- Apache Maven と Quarkus Maven プラグインを使用する
- code.quarkus.redhat.com を使用して Quarkus Maven プロジェクトを生成する
- Quarkus コマンドラインインターフェイス (CLI) を使用する

前提条件

- 環境の準備が完了している。詳細は、環境の準備 を 参照してください。

手順

- 要件に応じて、**getting-started** プロジェクトの作成に使用する方法を選択します。

1.4.1. Apache Maven を使用して Getting Started プロジェクトを作成する

Apache Maven と Quarkus Maven プラグインを使用して、**getting-started** プロジェクトを作成できます。この **getting-started** プロジェクトを使用して、単純な Quarkus アプリケーションを起動して実行できます。

前提条件

- Apache Maven を使用するための環境を準備している。詳細は、環境の準備 を 参照してください。
- Quarkus Maven リポジトリーを設定している。Maven を使用して Quarkus アプリケーションを作成するには、Red Hat がホストする Quarkus リポジトリーを使用するか、Quarkus Maven リポジトリーをダウンロードして設定できます。
  - Red Hat がホストする Quarkus リポジトリーを使用する場合は、オンラインリポジトリーの Maven settings.xml ファイルの設定 を参照し てください。
  - Maven を使用して Quarkus アプリケーションを作成するように Quarkus Maven リポジトリーを設定する場合は、Quarkus Maven リポジトリーの設定 を 参照してください。

手順

1. Maven が OpenJDK 11 または 17 を使用していることを確認するには、Maven のバージョンが 3.8.x (x は 6 以降)で、**mvn** が PATH 環境変数からアクセスできることを確認するには、以下のコマンドを入力します。

```
mvn --version
```

2. 前のコマンドで OpenJDK 11 または 17 が返されない場合は、OpenJDK 11 または 17 へのパスを PATH 環境変数に追加し、前のコマンドを再度入力します。

3. プロジェクトをビルドするには、以下のコマンドのいずれか 1 つを実行します。

注記

Apple macOS および Microsoft Windows は開発者環境としてサポートされますが、実稼働環境ではサポートされません。

- Linux または Apple macOS を使用している場合は、以下のコマンドを入力します。

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.13.9.SP2-redhat-
00003:create \
    -DprojectGroupId=org.acme \
    -DprojectArtifactId=getting-started \
    -DplatformGroupId=com.redhat.quarkus.platform \
    -DplatformVersion=2.13.9.SP2-redhat-00003 \
    -DclassName="org.acme.quickstart.GreetingResource" \
    -Dpath="/hello"
cd getting-started
```

- Microsoft Windows のコマンドラインを使用している場合は、以下のコマンドを入力します。

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.13.9.SP2-redhat-
00003:create
-DprojectGroupId=org.acme -DprojectArtifactId=getting-started
-DplatformGroupId=com.redhat.quarkus.platform
-DplatformVersion=2.13.9.SP2-redhat-00003
-DclassName="org.acme.quickstart.GreetingResource"
-Dpath="/hello"
```

- Microsoft Windows Powershell を使用している場合は、以下のコマンドを入力します。

```
mvn com.redhat.quarkus.platform:quarkus-maven-plugin:2.13.9.SP2-redhat-
00003:create
"-DprojectGroupId=org.acme"
"-DprojectArtifactId=getting-started"
"-DplatformVersion=2.13.9.SP2-redhat-00003"
"-DplatformGroupId=com.redhat.quarkus.platform"
"-DclassName=org.acme.quickstart.GreetingResource"
"-Dpath=/hello"
```

これらのコマンドにより、**./getting-started** ディレクトリーに以下の要素が作成されます。

  - Maven プロジェクトディレクトリー構造
  - **/hello** で公開される **org.acme.quickstart.GreetingResource** リソース
  - ネイティブモードおよび JVM モードでアプリケーションをテストするための関連するユニットテスト
  - アプリケーションの起動後に **http://localhost:8080** でアクセス可能なランディングページ

- **src/main/docker** ディレクトリー内のサンプル Dockerfile
- アプリケーションの設定ファイル

4. ディレクトリー構造が作成されたら、テキストエディターで **pom.xml** ファイルを開き、ファイルの内容を確認します。

pom.xml

```
<project>
 ...
 <properties>
   ...
  <quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
  <quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
  <quarkus.platform.version>2.13.9.SP2-redhat-00003</quarkus.platform.version>
   ...
 </properties>
 ...
 <dependencyManagement>
   <dependencies>
    <dependency>
     <groupId>${quarkus.platform.group-id}</groupId>
     <artifactId>${quarkus.platform.artifact-id}</artifactId>
     <version>${quarkus.platform.version}</version>
     <type>pom</type>
     <scope>import</scope>
    </dependency>
   </dependencies>
 </dependencyManagement>
 ...
 <build>
   ...
   <plugins>
    ...
    <plugin>
     <groupId>${quarkus.platform.group-id}</groupId>
     <artifactId>quarkus-maven-plugin</artifactId>
     <version>${quarkus.platform.version}</version>
     <extensions>true</extensions>
     <executions>
      <execution>
       <goals>
        <goal>build</goal>
        <goal>generate-code</goal>
        <goal>generate-code-tests</goal>
       </goals>
      </execution>
     </executions>
    </plugin>
    ...
   </plugins>
   ...
 </build>
...
</project>
```

pom.xml ファイルの **<dependencyManagement>** セクションには、Quarkus BOM が格納されています。そのため、**pom.xml** ファイルに個別の Quarkus 依存関係のバージョンを記述する必要はありません。**pom.xml** ファイルでは、アプリケーションのパッケージ化を行う **quarkus-maven-plugin** プラグインを見つけることもできます。

5. **pom.xml** ファイル内の **quarkus-resteasy-reactive** 依存関係を確認します。この依存関係により、REST アプリケーションを開発できます。

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-resteasy-reactive</artifactId>
</dependency>
```

6. **src/main/java/org/acme/quickstart/GreetingResource.java** ファイルを確認します。

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
{CompanyName}
@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "Hello from RESTEasy Reactive";
    }
}
```

このファイルには、**/hello** エンドポイントに送信するリクエストへの応答として **hello** を返す単純な REST エンドポイントが含まれます。

注記

Quarkus では、JAX-RS の **Application** クラスはサポートされますが、必須ではありません。さらに、リクエストごとに 1 つのインスタンスが作成されるのではなく、**GreetingResource** クラスのインスタンスが 1 つだけ作成されます。これは、別の ***Scoped** アノテーション (例: **ApplicationScoped**、**RequestScoped** など) を使用して設定できます。

1.4.2. code.quarkus.redhat.com を使用して Getting Started プロジェクトを作成する

アプリケーション開発者は、code.quarkus.redhat.com を使用して Quarkus Maven プロジェクトを生成し、アプリケーションで使用するエクステンションを自動的に追加および設定できます。さらに、code.quarkus.redhat.com は、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ために必要な設定パラメーターを自動的に管理します。

次のアクティビティーを含む Quarkus Maven プロジェクトを生成できます。

- アプリケーションの基本情報の指定
- プロジェクトに追加するエクステンションの選択
- プロジェクトファイルのダウンロード可能なアーカイブの生成

- アプリケーションをコンパイルおよび起動するカスタムコマンドの使用

前提条件

- Web ブラウザーがある。
- で説明されているように、Apache Maven を使用するための環境を準備している。

環境の準備

- Quarkus Maven リポジトリーを設定している。Red Hat がホストする Quarkus リポジトリーを使用するか、Quarkus Maven リポジトリーをダウンロードして設定して、Maven で Quarkus アプリケーションを作成できます。
  - Red Hat がホストする Quarkus リポジトリーを使用するには、オンラインリポジトリーの Maven settings.xml ファイルの設定 を参照し てください。
  - Maven を使用して Quarkus アプリケーションを作成するように Quarkus Maven リポジトリーを設定するには、Quarkus Maven リポジトリーの設定 を 参照してください。
- **オプション**: Quarkus コマンドラインインターフェイス(CLI)をインストールしている。これは、開発モードで Quarkus を開始するために使用できる方法の 1 つです。

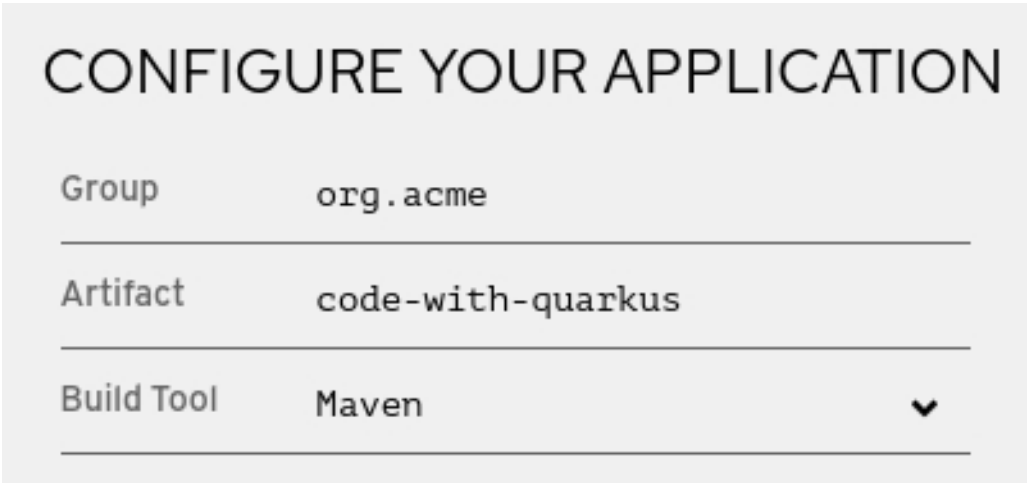
詳細は、Quarkus CLI のインストール を参照し てください。

注記

Quarkus CLI は、開発モードのみを対象としています。Red Hat は、実稼働環境での Quarkus CLI の使用をサポートしません。

手順

1. Web ブラウザーで、https://code.quarkus.redhat.com に移動します。
2. プロジェクトの基本情報を指定します。



   a. プロジェクトのグループ名を入力します。名前形式は、Java パッケージの命名規則に従います（例： **org.acme** ）。
   b. プロジェクトから生成された Maven アーティファクトに使用する名前を入力します（例： **code-with-quarkus** ）。

c. アプリケーションのコンパイルおよび起動に使用するビルドツールを選択します。選択したビルドツールにより、次の設定が決まります。

- 生成されたプロジェクトのディレクトリー構造。
- 生成したプロジェクトで使用される設定ファイルのフォーマット
- プロジェクトの生成後に code.quarkus.redhat.com に表示される、アプリケーションのコンパイルと起動のためのカスタムビルドスクリプトおよびコマンド

注記

Red Hat は、Quarkus Maven プロジェクトを作成する場合にのみ code.quarkus.redhat.com の使用をサポートします。

3. アプリケーションプロジェクトに関する追加情報を指定します。

a. アプリケーションの詳細を含むフィールドを表示するには、**More options** を選択します。

b. プロジェクトから生成されるアーティファクトに使用するバージョンを入力します。このフィールドのデフォルト値は **1.0.0-SNAPSHOT** です。semantic versioning の使用が推奨されます。ただし、別のタイプのバージョン管理を指定することもできます。

c. code.quarkus.redhat.com がスターターコードをプロジェクトに追加するかどうかを選択します。**CODE**でマークされたエクステンションをプロジェクトに追加する場合、このオプションを有効にして、プロジェクトの生成時にこれらのエクステンションのクラスおよびリソースファイルのサンプルを自動的に作成できます。ただし、サンプルコードを提供するエクステンションを追加しない場合、このオプションは生成されたプロジェクトには影響しません。

注記

code.quarkus.redhat.com リポジトリーは、Red Hat ビルドの Quarkus の最新リリースを自動的に使用します。プロジェクトの生成後に、**pom.xml** ファイルで BOM バージョンを手動で変更できます。

4. 使用するエクステンションを選択します。選択したエクステンションは、Quarkus アプリケーションの依存関係として含まれています。Quarkus プラットフォームは、これらのエクステンションが将来のバージョンと互換性があることも確認します。

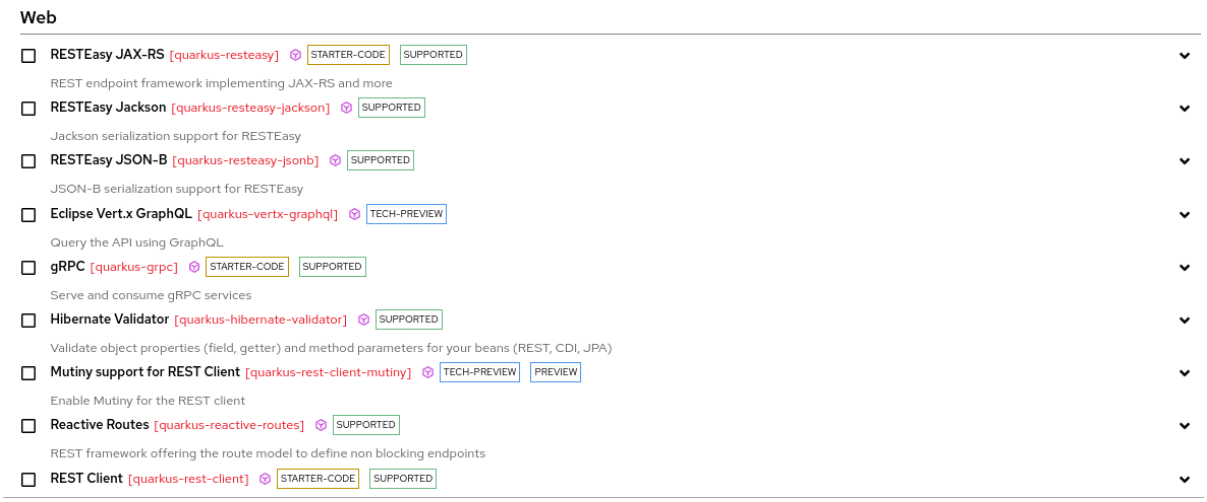
重要

RESTEasy エクステンションと **RESTEasy Reactive** エクステンションは、同じプロジェクト内で使用しないでください。

エクステンションの横にある quark アイコン ( ) は、そのエクステンションが Red Hat build

of Quarkus プラットフォームリリースの一部であることを示します。Red Hat は、同じプラットフォームの拡張機能を一緒にテストおよび検証しているため、使用することを推奨します。したがって、使用とアップグレードが容易であるからです。

このオプションを有効にすると、**STARTER-CODE** のマークが付いたエクステンションのスターターコードを自動的に生成できます。



5. 選択を確認するには、**Generate your application** を選択します。次の項目が表示されます。
   - 生成されたプロジェクトを含むアーカイブをダウンロードするためのリンク
   - アプリケーションをコンパイルして起動するために使用できるカスタムコマンド
6. 生成されたプロジェクトファイルを含むアーカイブをマシンに保存するには、**Download the ZIP** を選択します。
7. アーカイブの内容をデプロイメントします。
8. 展開したプロジェクトファイルが含まれるディレクトリーに移動します。

```
cd <directory_name>
```

9. 開発モードでアプリケーションをコンパイルおよび起動するには、以下のいずれかの方法を使用します。
   - Maven の使用:

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

   - Quarkus CLI を使用する:

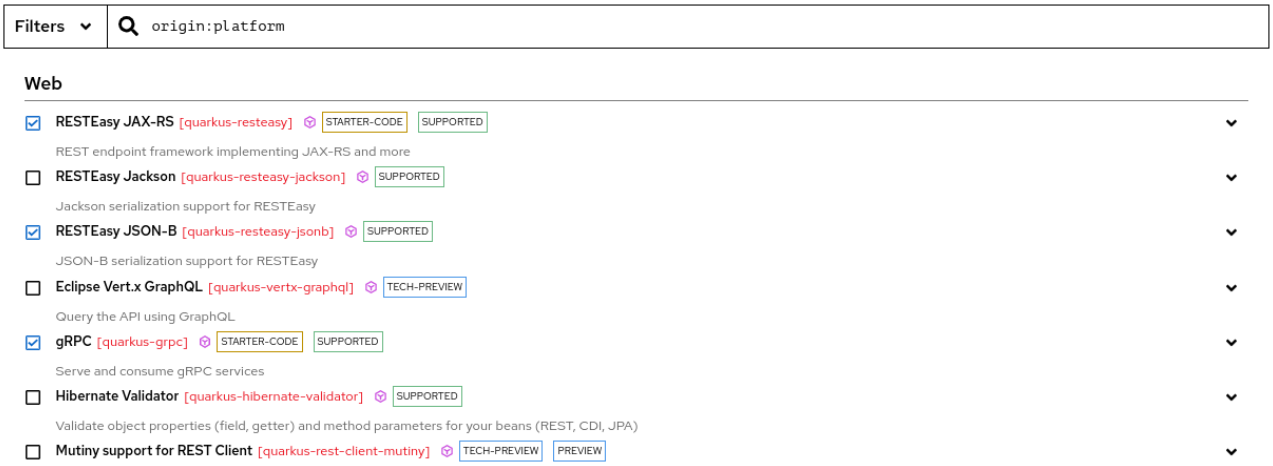
```
quarkus dev
```

関連情報

Quarkus エクステンションのサポートレベル

1.4.2.1. Quarkus エクステンションのサポートレベル

Red Hat は、code.quarkus.redhat.com で入手して Quarkus プロジェクトに追加できるエクステンションにたいして、さまざまなレベル のサポートを提供しています。各エクステンションの名前の横にあるラベルは、サポートレベルを示します。



注記

Red Hat は、ラベルのないエクステンションの実稼働環境での使用はサポートしません。

Quarkus は、Quarkus エクステンション向けに Quarkus によって提供される .Support レベル のサポートを提供します。

サポートレベル	説明
SUPPORTED	Red Hat は、実稼働環境のエンタープライズアプリケーションにおけるエクステンションの使用に対し、フルサポートを提供します。
TECH-PREVIEW	Red Hat は、テクノロジープレビュー機能のサポート範囲 に基づき、実稼働環境におけるエクステンションの使用に対して限定的なサポートを提供します。
DEV-SUPPORT	Red Hat は、実稼働環境におけるエクステンションの使用をサポートしませんが、Red Hat 開発者が新規アプリケーションの開発で使用するために提供するコア機能をサポートします。
DEPRECATED	Red Hat は、エクステンションを同じ機能を提供する最新のテクノロジーまたは実装に置き換える予定です。
STARTER-CODE	エクステンションのサンプルコードを自動生成できます。

各エクステンションの横にある矢印アイコン (⌄) をクリックすると、オーバーフローメニューを展開して、そのエクステンションの他のアクションにアクセスできます。以下に例を示します。

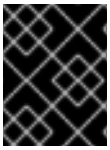
- コマンドラインで Quarkus Maven プラグインを使用して、既存のプロジェクトにエクステンションを追加します。
- プロジェクトの **pom.xml** ファイルにエクステンションを追加するために、XML の抜粋をコピーします。

- 各エクステンションの **groupId**、**artifactId**、**version** を取得します。
- エクステンションガイドを開きます。

1.4.3. Quarkus CLI を使用した Getting Started プロジェクトの作成

Quarkus コマンドラインインターフェイス (CLI) を使用して、**getting-started** プロジェクトを作成できます。

Quarkus CLI を使用して、プロジェクトの作成、エクステンションの管理、ビルドおよび開発コマンドの実行を行えます。



重要

Quarkus CLI は、開発モードのみを対象としています。Red Hat は、実稼働環境での Quarkus CLI の使用をサポートしません。

前提条件

- Quarkus CLI がインストールされている。詳細は、環境の準備 を 参照してください。
- Quarkus 開発者ツールを、エクステンションレジストリー内のエクステンションにアクセスするように設定している。詳細は、Quarkus エクステンションレジストリークライアントの設定 を参照してください。

手順

1. プロジェクトを生成するには、コマンドターミナルで次のコマンドを入力します。

```
quarkus create && cd code-with-quarkus
```

注記

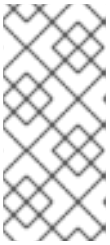
'app' サブコマンド (例: **quarkus create app)** を指定することもできます。ただし、'app' サブコマンドが指定されていない場合は暗黙的に指定されるため、必須ではありません。

このコマンドを使用すると、Quarkus プロジェクトが現在の作業ディレクトリーの 'code-with-quarkus' というフォルダーに作成されます。

2. デフォルトでは、**groupId** 属性、**artifactId** 属性、**version** 属性は次のデフォルト値に指定されます。
   - groupId='org.acme'
   - artifactId='code-with-quarkus'
   - version='1.0.0-SNAPSHOT'
     groupId、**artifactId**、および **version** 属性の値を変更するには、**quarkus create** コマンドを発行し、CLI で次の構文を指定します。

     groupId:artifactId:version

     たとえば **quarkus create app mygroupId:myartifactid:version** です。



注記

使用可能なすべての Quarkus コマンドに関する情報を表示するには、**help** パラメーターを指定します。

```
quarkus --help
```

3. テキストエディターで **src/main/java/org/acme/GreetingResource.java** ファイルを確認します。

```
package org.acme;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "Hello from RESTEasy Reactive";
    }
}
```

このファイルには、**/hello** エンドポイントに送信するリクエストへの応答として **hello** を返す単純な REST エンドポイントが含まれます。

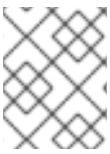
検証

1. 開発モードでアプリケーションをコンパイルして起動します。
2. Quarkus CLI から Getting Started プロジェクトをパッケージ化して実行します。

1.5. QUARKUS GETTING STARTED プロジェクトのコンパイルおよび起動

Quarkus Getting Started プロジェクトを作成したら、**Hello** アプリケーションをコンパイルし、**hello** エンドポイントが **hello** を返すことを確認できます。

この手順では、Quarkus の組み込み開発モードを使用しているため、アプリケーションの実行中にアプリケーションソースおよび設定を更新できます。変更は、実行中のアプリケーションに表示されます。



注記

Quarkus **Hello** アプリケーションのコンパイルに使用するコマンドは、マシンにインストールした開発者ツールにより異なります。

前提条件

- Quarkus Getting Started プロジェクトを作成している。

手順

1. プロジェクトディレクトリーに移動します。
2. Quarkus **Hello** アプリケーションを開発モードでコンパイルするには、使用する開発者ツールに応じて、以下のいずれかのメソッドを使用します。
   - Apache Maven を使用する場合は、次のコマンドを入力します。

     ```
     mvn quarkus:dev
     ```

   - Quarkus コマンドラインインターフェイス (CLI) を使用する場合は、次のコマンドを入力します。

     ```
     quarkus dev
     ```

   - Maven ラッパーを使用する場合は、次のコマンドを入力します。

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

     予想される出力

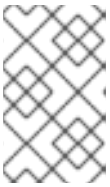
     次の抜粋は、予想される出力の例を示しています。

     ```
     INFO  [io.quarkus] (Quarkus Main Thread) Profile dev activated. Live Coding activated.
     INFO  [io.quarkus] (Quarkus Main Thread) Installed features: [cdi, resteasy, smallrye-context-propagation]
     ```

検証

- アプリケーションが提供するエンドポイントにリクエストを送信するには、新しいターミナルウィンドウに次のコマンドを入力します。

  ```
  curl -w "\n" http://localhost:8080/hello
  Hello from RESTEasy Reactive
  ```



注記

\n 属性は、コマンドの出力の前に新しい行を自動的に追加します。これにより、ターミナルが '%' 文字を出力したり、結果と次のシェルプロンプトの両方を同じ行に配置したりすることがなくなります。

1.6. QUARKUS ディペンデンシーインジェクション (依存性の注入) の使用

ディペンデンシーインジェクション (依存性の注入) により、クライアントによる消費とは完全に独立した方法で、サービスが使用されるようになります。クライアントの依存関係の作成がクライアントの動作から分離されるため、プログラム設計を疎結合にできます。

Red Hat build of Quarkus での依存性の注入は、Quarkus アーキテクチャーに合わせて調整された、コンテキストと依存性注入 (CDI) をベースとするビルドタイム指向の依存性注入ソリューションです。ArC は quarkus-resteasy の推移的な依存関係であり、 **quarkus-resteasy** はお客様のプロジェクトの依存関係であるため、ArC はすでにダウンロードされています。

前提条件

前提条件

- Quarkus Getting Started プロジェクトを作成している。

手順

1. アプリケーションを変更し、コンパニオン Bean を追加するには、以下の内容で **src/main/java/org/acme/quickstart/GreetingService.java** ファイルを作成します。

```
package org.acme.quickstart;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class GreetingService {

    public String greeting(String name) {
        return "hello " + name;
    }

}
```

2. **src/main/java/org/acme/quickstart/GreetingResource.java** を編集して **GreetingService** を注入し、これを使用して新しいエンドポイントを作成します。

```
package org.acme.quickstart;

import javax.inject.Inject;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import org.jboss.resteasy.annotations.jaxrs.PathParam;

@Path("/hello")
public class GreetingResource {

    @Inject
    GreetingService service;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    @Path("/greeting/{name}")
    public String greeting(@PathParam String name) {
        return service.greeting(name);
    }

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "hello";
    }
}
```

3. アプリケーションを停止した場合は、以下のコマンドを入力して再起動します。

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

4. エンドポイントが **hello quarkus** を返すことを確認するには、新しいターミナルウィンドウに以下のコマンドを入力します。

```
curl -w "\n" http://localhost:8080/hello/greeting/quarkus
hello quarkus
```

1.7. QUARKUS アプリケーションのテスト

Quarkus Getting Started プロジェクトをコンパイルした後、JUnit 5 フレームワークを使用してアプリケーションをテストすることで、それが期待通りに実行するか確認できます。

注記

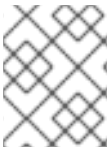
Quarkus アプリケーションの継続的なテストを有効にすることもできます。詳細は、継続的なテストの有効化と実行 を参照してください。

Quarkus プロジェクトは、**pom.xml** ファイルに次の 2 つのテスト依存関係を生成します。

- **quarkus-junit5**: JUnit 5 テストフレームワークを制御する **@QuarkusTest** アノテーションを提供するため、テストに必要です。
- **rest-assured**: **rest-assured** 依存関係は必須ではありませんが、HTTP エンドポイントをテストする便利な方法として使用できるため、統合されています。**rest-assured** 依存関係により正しい URL が自動的に設定されるため、設定は必要ありません。

サンプル pom.xml ファイル:

```
<dependency>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-junit5</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.rest-assured</groupId>
    <artifactId>rest-assured</artifactId>
    <scope>test</scope>
</dependency>
```



注記

これらのテストは REST-assured フレームワークを使用しますが、希望する場合は別のライブラリーを使用できます。

前提条件

- Quarkus Getting Started プロジェクトをコンパイルしている。詳細は、Quarkus Getting Started プロジェクトのコンパイルおよび起動 を参照してください。

手順

1. 生成された **pom.xml** ファイルを開き、コンテンツを確認します。

```
<plugin>
  <artifactId>maven-surefire-plugin</artifactId>
  <version>${surefire-plugin.version}</version>
  <configuration>
    <systemPropertyVariables>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
      <maven.home>${maven.home}</maven.home>
    </systemPropertyVariable>
  </configuration>
</plugin>
```

次のプロパティーの値に注意してください。

- アプリケーションがテストで必ず正しいログマネージャーを使用するように、**java.util.logging.manager** システムプロパティーが設定されます。
- **maven.home** プロパティーは、プロジェクトに適用するカスタム Maven 設定を保存できる **settings.xml** ファイルの場所を参照します。

2. **src/test/java/org/acme/quickstart/GreetingResourceTest.java** ファイルを以下の内容に一致するように編集します。

```
package org.acme.quickstart;

import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Test;

import java.util.UUID;

import static io.restassured.RestAssured.given;
import static org.hamcrest.CoreMatchers.is;

@QuarkusTest
public class GreetingResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
          .when().get("/hello")
          .then()
             .statusCode(200)
             .body(is("hello"));
    }

    @Test
    public void testGreetingEndpoint() {
        String uuid = UUID.randomUUID().toString();
        given()
          .pathParam("name", uuid)
          .when().get("/hello/greeting/{name}")
          .then()
            .statusCode(200)
            .body(is("hello " + uuid));
```

```
    }

}
```

注記

QuarkusTest ランナーを使用することで、テストを開始する前に JUnit にアプリケーションを起動するよう指示します。

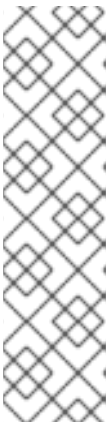
3. Maven からこれらのテストを実行するには、以下のコマンドを入力します。

```
./mvnw test
```

注記

IDE からテストを実行することもできます。その場合は、まずアプリケーションを停止してください。

デフォルトでは、テストはポート **8081** で実行されるため、実行中のアプリケーションと競合しません。Quarkus では、**RestAssured** 依存関係はこのポートを使用するように設定されています。



注記

別のクライアントを使用する場合は、**@TestHTTPResource** アノテーションを使用して、テスト済みアプリケーションの URL を **Test** クラスのフィールドに直接注入します。このフィールドには、**String**、**URL**、または **URI** の型を使用できます。**@TestHTTPResource** アノテーションにテストパスを入力することもできます。たとえば、**/myservlet** にマップされたサーブレットをテストするには、以下の行をテストに追加します。

```
@TestHTTPResource("/myservlet")
URL testUrl;
```

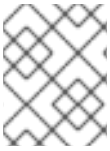
4. 必要な場合は、**quarkus.http.test-port** 設定プロパティーにテストポートを指定します。

1.8. 継続的テストの有効化と実行

Red Hat build of Quarkus を使用すると、アプリケーションの開発中にコードの変更を継続的にテストできます。Quarkus には、コードに変更を加えて保存した直後に実行できる継続的テスト機能があります。

継続的テストを実行すると、アプリケーションの起動後にテストが一時停止されます。アプリケーションの起動後、すぐにテストを再開できます。Quarkus アプリケーションは、実行するテストを決定します。これにより、変更されたコードに対してのみテストが実行されます。

Quarkus の継続的テスト機能は、デフォルトで有効になっています。**src/main/resources/application.properties** ファイルの **quarkus.test.continuous-testing** プロパティーを **disabled** に設定することで、継続的テストを無効にできます。



注記

以前に継続的テストを無効にしており、再度有効にしたい場合は、テストを開始する前に Quarkus アプリケーションを再起動する必要があります。

前提条件

- Quarkus Getting Started アプリケーション (またはその他のアプリケーション) をコンパイルしちえる。詳細は、Quarkus Getting Started プロジェクトのコンパイルおよび起動 を 参照してください。

手順

1. Quarkus アプリケーションを起動します。
   - code.quarkus.redhat.com または Quarkus CLI を使用して Getting Project プロジェクトを作成した場合、プロジェクトの生成時に Maven ラッパーが提供されています。プロジェクトディレクトリーから次のコマンドを入力します。

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

   - マシンにインストールされている Apache Maven を使用して Getting Project プロジェクトを作成した場合、次のコマンドを入力します。

     ```
     mvn quarkus:dev
     ```

   - 開発モードで継続的なテストを実行し、Quarkus CLI を使用している場合は、以下のコマンドを入力します。

     ```
     quarkus dev
     ```

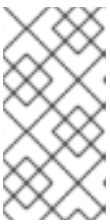
2. 生成された出力ログで、テストステータスの詳細を表示します。

注記

出力ログを表示するには、画面の一番下までスクロールする必要がある場合があります。

継続的テストが有効になっている場合、次のメッセージが表示されます。

```
Tests paused, press [r] to resume, [h] for more options>
```



注記

デフォルトでは、継続的テストが有効になっている場合、アプリケーションの起動後にテストが一時停止されます。テストの実行方法を制御するために使用できるキーボードコマンドについては、継続的なテストを制御するコマンド を参照してください。

3. テストの実行を開始するには、キーボードの `r` を押します。

4. 更新された出力ログを表示して、テストステータスとテスト結果の監視、およびテストの統計情報の確認を行い、フォローアップアクションのガイダンスを取得します。以下に例を示します。

```
Tests all passed, 2 tests were run, 0 were skipped. Tests took 1470ms.
Press [r] to re-run, [v] to view full results, [p] to pause, [h] for more options>
```

検証

1. コードを変更します。たとえば、テキストエディターで **src/main/java/GreetingsResource.java** ファイルを開きます。

2. "Hello world" を返すように "hello" エンドポイントを変更し、ファイルを保存します。

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class GreetingResource {

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        return "Hello world";
    }
}
```

3. Quarkus がすぐにテストを再実行して、変更後のコードをテストすることを確認します。

4. 出力ログを表示してテスト結果を確認します。この例では、テストでは変更された文字列に値 hello が含まれているかどうかを確認します。文字列が "Hello world" に変更されているため、テストは失敗します。

```
2021-05-11 14:21:34,338 ERROR [io.qua.test] (Test runner thread) Test
GreetingResourceTest#testHelloEndpoint() failed
: java.lang.AssertionError: 1 expectation failed.
Response body doesn't match expectation.
Expected: is "hello"
  Actual: Hello world
 at io.restassured.internal.ValidatableResponseImpl.body(ValidatableResponseImpl.groovy)
 at
org.acme.getting.started.GreetingResourceTest.testHelloEndpoint(GreetingResourceTest.java:
21)
--
Test run failed, 2 tests were run, 1 failed, 0 were skipped. Tests took 295ms
Press [r] to re-run, [v] to view full results, [p] to pause, [h] for more options>
```

5. 継続的テストを終了するには、キーボードの **Ctrl-C** または 'q' を押します。

注記

値を "hello" に戻すと、テストが自動的に再度実行されます。

1.8.1. 継続的テストの制御コマンド

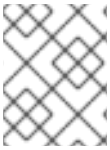
キーボードのホットキーコマンドを使用して、継続的テストのオプションを制御できます。コマンドの完全なリストを表示するには、キーボードの 'h' を押します。以下のオプションを設定できます。

コマンド	説明
r	すべてのテストを再実行します。
f	失敗したすべてのテストを再実行します。
b	'broken only' モードに切り替えます。他のテストもコード変更の影響を受ける場合でも、以前に失敗したテストのみが実行されます。このオプションは、多くのテストで使用されるコードを変更するが、失敗したテストのみを確認したい場合に便利です。
v	最後のテスト実行からのテスト失敗の詳細をコンソールに出力します。このオプションは、最後のテスト実行以降に大量のコンソール出力があった場合に便利です。
p	実行中のテストを一時的に停止します。これは、多くのコード変更を行っているが、変更が完了するまでテストのフィードバックを取得したくない場合に便利です。
q	継続的テストを終了します。
o	テスト出力をコンソールに出力します。これはデフォルトで無効にされます。テスト出力が無効になっている場合、出力はフィルタリングされて保存されますが、コンソールには表示されません。テスト出力は開発 UI で表示できます。
i	インストルメンテーションベースのリロードを切り替えます。このオプションを使用してもテストに直接的な影響はありませんが、ライブリロードが可能になります。これは、変更がクラスの構造に影響を与えない場合に再起動を回避するために役立ちます。
l	ライブリロードを切り替えます。このオプションを使用してもテストに直接的な影響はありませんが、ライブリロードのオンまたはオフへの切り替えが可能になります。

コマンド	説明
s	強制的に再起動します。このオプションを使用すると、変更されたファイルのスキャンと変更を含むライブリロードを強制できます。コードに変更がなく、ライブリロードが無効になっている場合でも、アプリケーションが再起動されることに注意してください。

1.9. QUARKUS GETTING STARTED アプリケーションのパッケージ化および実行

Quarkus Getting Started プロジェクトをコンパイルしたら、JAR ファイルでパッケージ化し、コマンドラインから実行できます。



注記

Quarkus Getting Started アプリケーションをパッケージ化して実行するために使用するコマンドは、マシンにインストールされている開発者ツールにより異なります。

前提条件

- Quarkus Getting Started プロジェクトをコンパイルしている。

手順

1. **getting-started** プロジェクトディレクトリーに移動します。
2. Quarkus Getting Started プロジェクトをパッケージ化するには、使用する開発者ツールに応じて、次のいずれかの方法を使用します。
   - Apache Maven を使用する場合は、次のコマンドを入力します。

     ```
     mvn package
     ```

   - Quarkus コマンドラインインターフェイス (CLI) を使用する場合は、次のコマンドを入力します。

     ```
     quarkus build
     ```

   - Maven ラッパーを使用する場合は、次のコマンドを入力します。

     ```
     ./mvnw package
     ```

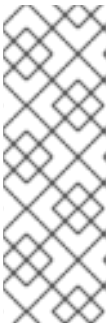
     このコマンドは、以下の JAR ファイルを **/target** ディレクトリーに生成します。

     - **getting-started-1.0-0-SNAPSHOT.jar**: プロジェクトのクラスおよびリソースが含まれます。これは、Maven ビルドで生成される通常のアーティファクトです。

- **quarkus-app/quarkus-run.jar**: 実行可能な JAR ファイルです。このファイルは uber-JAR ファイルではありません。依存関係は **target/quarkus-app/lib** ディレクトリーにコピーされます。

3. 以下のコマンドを入力してアプリケーションを起動します。

```
java -jar target/quarkus-app/quarkus-run.jar
```



注記

- アプリケーションを実行する前に、開発モード(CTRL+C を表示)を停止するか、ポートの競合が発生します。
- **quarkus-run.jar** ファイルの **MANIFEST.MF** ファイルの **Class-Path** エントリーには、**lib** ディレクトリーの JAR ファイルが明示的にリストされます。別の場所からアプリケーションをデプロイする場合は、**quarkus-run.jar** ファイルと **lib** ディレクトリーをコピーする必要があります。

1.10. JVM とネイティブビルドモード

次のセクションでは、Mandrel または GraalVM の **native-image** ツールを使用して、クラシック JVM アプリケーションおよびネイティブアプリケーションをコンパイルする方法を説明します。

1.10.1. アプリケーションをクラシック JVM アプリケーションとしてコンパイルする

アプリケーションを、JVM アプリケーションとしてコンパイルできます。このオプションは、**quarkus.package.type** 設定プロパティーに基づいており、次のいずれかのファイルを生成します。

- **fast-jar**: Quarkus およびデフォルトの設定オプション用に最適化された JAR ファイル。その結果、起動時間がわずかに短縮され、メモリー使用量がわずかに減少します。
- **legacy-jar**: 典型的な JAR ファイル。
- **uber-jar**: 単一のスタンドアロン JAR ファイル。
これらの JAR ファイルはすべてのオペレーティングシステムで動作し、ネイティブイメージよりもはるかに短時間でビルドされます。

1.10.2. アプリケーションをネイティブイメージにコンパイルする

アプリケーションをネイティブイメージにコンパイルできます。その場合、**quarkus.package.type** 設定プロパティーを **native** に設定します。

このプロパティーを使用すると、たとえば Windows の **.exe** ファイルなど、選択したオペレーティングシステム専用にコンパイルされた実行可能バイナリーファイルを作成できます。これらのファイルは JAVA JAR ファイルよりも起動時間が短く、RAM 消費量も少なくなりますが、コンパイルには数分かかります。さらに、プロファイルに基づく最適化が欠落しているため、ネイティブバイナリーを使用して達成できる最大スループットは、通常の JVM アプリケーションよりも低くなります。

- **Mandrel を使用する**
Mandrel は、GraalVM for Red Hat build of Quarkus 用に特化されたディストリビューションで、Linux コンテナー化環境をターゲットとするネイティブ実行可能ファイルをビルドする場合に推奨されるアプローチでもあります。Mandrel アプローチは、コンテナー化された環境にコンパイル出力を埋め込むのに最適ですが、Linux64 ビットのネイティブ実行可能ファイルしか提供されません。したがって、**.exe** などの結果はオプションではありません。

Mandrel ユーザーは、ネイティブ実行可能ファイルのビルドにコンテナーを使用することが推奨されます。

公式 Mandrel イメージを使用して、Docker または Podman のローカルインストールでアプリケーションをネイティブモードにコンパイルするには、次のプロパティーを指定して **mvn package** コマンドを入力します。

```
-Dquarkus.package.type=native
-Dquarkus.native.container-build=true
-Dquarkus.native.builder-image=quay.io/quarkus/ubi-quarkus-mandrel:{MandrelVersion}-{JDK-ver-other}
```

- Mandrel を使用してネイティブ実行可能ファイルをビルドする方法の詳細は、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 を参照してください。
- 利用可能な Mandrel イメージのリストは、Available Mandrel images で確認してください。

- **GraalVM を使用する**
  Mandrel は macOS および Windows プラットフォームをサポートしないため、Oracle GraalVM を使用してこれらのオペレーティングシステム上でネイティブ実行可能ファイルを構築できます。

  ベアメタル Linux または Windows ディストリビューション上で Oracle GraalVM を直接使用して、ネイティブ実行可能ファイルをビルドすることもできます。このプロセスの詳細は、Oracle GraalVM README およびリリースノートを参照してください。

  Oracle GraalVM を使用してネイティブ実行可能ファイルをビルドする方法は、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 を参照してください。

関連情報

- ネイティブ実行可能ファイルの構築、コンパイル、パッケージ化、およびデバッグに関する詳細は、ネイティブ実行可能ファイルの ビルド を 参照してください。
- Java アプリケーションをネイティブ実行可能ファイルとして実行しようとすると発生する可能性のある問題のトラブルシューティングに役立つ情報は、ネイティブアプリケーションを 作成するためのヒント を参照してください。

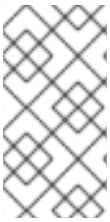
1.11. ネイティブモードでの QUARKUS GETTING STARTED アプリケーションのパッケージ化および実行

ネイティブモードでは、アプリケーションビルドからの出力は、圧縮またはアーカイブ JAR ファイルではなく、プラットフォームに依存するネイティブバイナリーファイルになります。ネイティブモードが JVM とどのように異なるかの詳細は、スタートガイド の JVM およびネイティブビルドモード の章を参照してください。

前提条件

- OpenJDK 11 または 17 がインストールされており、**JAVA_HOME** 環境変数を設定して Java SDK の場所を指定している。
- Apache Maven 3.8.x がインストールされている。x は 6 以降です。
- 作業用の C 開発環境 がある。

- Docker や Podman などの動作するコンテナーランタイムがある。
- **オプション**: Quarkus コマンドラインインターフェイス (CLI) を使用する場合は、それがインストールされていることを確認してください。
  - Quarkus CLI のインストール方法については、Quarkus CLI のコミュニティー固有の情報を参照してください。
- Quarkus Getting Started プロジェクト のクローンを作成してコンパイルしている。
- GraalVM のコミュニティーエディションまたはエンタープライズエディションをダウンロードし、インストールしている。
  - コミュニティーまたはエンタープライズエディションの GraalVM をダウンロードしてインストールするには、公式の GraalVM Getting Started ドキュメント を参照してください。
  - もしくは、sdkman、homebrew、scoop などの、プラットフォーム固有のインストールツールを使用します。



注記

GraalVM のコミュニティーエディションを使用して Getting Started ガイドのすべての手順を完了できますが、GraalVM のコミュニティーエディションは Red Hat build of Quarkus の実稼働環境ではサポートされません。詳細は、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 を参照してください。

手順

1. ランタイム環境を設定するために、**GRAALVM_HOME** 環境変数を GraalVM インストールディレクトリーに設定します。以下に例を示します。

```
export GRAALVM_HOME=$HOME/Development/graalvm/
```

   - macOS の場合、変数が **Home** サブディレクトリーを指すようにします。

```
export GRAALVM_HOME=$HOME/Development/graalvm/Contents/Home/
```

   - Windows の場合、コントロールパネルを使用して環境変数を設定します。

2. **native-image** ツールをインストールします。

```
${GRAALVM_HOME}/bin/gu install native-image
```

3. **JAVA_HOME** 環境変数を GraalVM インストールディレクトリーに設定します。

```
export JAVA_HOME=${GRAALVM_HOME}
```

4. GraalVM **bin** ディレクトリーをパスに追加します。

```
export PATH=${GRAALVM_HOME}/bin:$PATH
```

5. Getting Started プロジェクトフォルダーに移動します。

```
cd getting-started
```

6. 以下のいずれかの方法で、ネイティブイメージをコンパイルします。
   - Maven の使用:

     ```
     mvn clean package -Pnative
     ```

   - Quarkus CLI を使用する:

     ```
     quarkus build --native
     ```

検証

1. アプリケーションを起動します。

   ```
   ./target/getting-started-1.0.0-SNAPSHOT-runner
   ```

2. ログメッセージを観察し、**native** という単語が含まれていることを確認します。

   ```
   2022-03-04 09:51:51,505 INFO [io.quarkus] (main) getting-started 1.0.0-SNAPSHOT native
   (powered by Quarkus 2.7.3.Final) started in 0.043s. Listening on: http://0.0.0.0:8080
   ```

関連情報

- その他のヒントまたはトラブルシューティング情報は、Quarkus Building a native executable ガイドを参照してください。

1.12. 関連情報

- OpenShift Container Platform に Quarkus アプリケーションをデプロイする

改訂日時: 2024-04-23