



## Red Hat build of Quarkus 1.3

Red Hat ビルドの Quarkus のスタートガイド





## 法律上の通知

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本ガイドでは、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

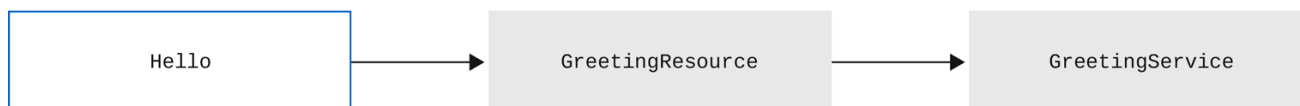
前書き .....	3
第1章 RED HAT ビルドの QUARKUS .....	4
第2章 APACHE MAVEN および QUARKUS .....	5
2.1. オンラインリポジトリの MAVEN の SETTINGS.XML ファイルの設定 .....	5
2.2. QUARKUS MAVEN リポジトリのダウンロードおよび設定 .....	6
第3章 GETTING STARTED プロジェクトの作成 .....	9
第4章 QUARKUS GETTING STARTED プロジェクトのコンパイル .....	12
第5章 QUARKUS ディペンデンシーインジェクション (依存性の注入) の使用 .....	13
第6章 JUNIT を使用した QUARKUS アプリケーションのテスト .....	15
第7章 QUARKUS GETTING STARTED アプリケーションのパッケージ化および実行 .....	18
第8章 その他のリソース .....	19



## 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、サーバーレス環境および 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。これらのアプリケーションのメモリーフットプリントは小さくなり、起動時間は高速化されます。

本ガイドでは、Apache Maven を使用して **hello** HTTP エンドポイントを公開する簡単な Quarkus プロジェクトを作成、テスト、パッケージ化、および実行する方法を説明します。ディペンデシーインジェクション (依存性の注入) を実証するために、このエンドポイントは **greeting** Bean を使用します。



78\_OpenShift\_0420

### 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)。
- Apache Maven 3.6.2 以降がインストールされていること。Maven は [Apache Maven Project](#) の Web サイトから入手できます。



### 注記

Getting Started の演習の完全なサンプルについては、[Quarkus quickstart archive](#) をダウンロードするか、**Quarkus Quickstarts** Git リポジトリをクローンしてください。この例は **getting-started** ディレクトリーにあります。

## 第1章 RED HAT ビルドの QUARKUS

Red Hat ビルドの Quarkus は、コンテナおよび 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、Apache Kafka、RESTEasy (JAX-RS)、Hibernate ORM (JPA)、Spring、Infinispan、Apache Camel 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。

Quarkus のディペンデンシーインジェクション (依存性の注入) ソリューションは、CDI (コンテキストとディペンデンシーインジェクション) をベースとし、エクステンションフレームワークを備えているので、機能の拡張、およびフレームワークの設定、起動、アプリケーションへの統合が可能です。

Quarkus は、コンテナ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。この手法により、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のビルドが大幅に容易になるほか、これらのアプリケーションがサーバーレスコンピューティングフレームワークで実行している関数を呼び出すことができるようになります。これにより、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なり、起動時間は高速化されます。



## 第2章 APACHE MAVEN および QUARKUS

Apache Maven は、ソフトウェアプロジェクトの作成、管理、ビルドを行う Java アプリケーションの開発で使用される分散型ビルド自動化ツールです。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ビルドプロセスの管理を行います。POM ファイルは、モジュールおよびコンポーネントの依存関係、ビルドの順番、結果となるプロジェクトパッケージングのターゲットを定義し、XML ファイルを使用して出力します。この結果、プロジェクトが適切かつ統一された状態でビルドされるようになります。

### Maven リポジトリ

Maven リポジトリには、Java ライブラリー、プラグイン、その他のビルドアーティファクトが格納されています。デフォルトのパブリックリポジトリは 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を使用することができます。また、サードパーティーのリポジトリも利用できます。

Quarkus プロジェクトでオンライン Maven リポジトリを使用するか、または Red Hat ビルドの Quarkus の Maven リポジトリをダウンロードすることができます。

### Maven プラグイン

Maven プラグインは、1つ以上のゴールを達成する POM ファイルの定義された部分です。Quarkus アプリケーションは以下の 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を実現、uber-JAR ファイルの生成をサポート、そして開発モードを提供します。
- Maven Surefire プラグイン (**maven-surefire-plugin**): 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。

## 2.1. オンラインリポジトリの MAVEN の SETTINGS.XML ファイルの設定

ユーザーの **settings.xml** ファイルを設定して、オンライン Quarkus リポジトリを Quarkus Maven プロジェクトで使用することができます。これは推奨される手法です。リポジトリマネージャーまたは共有サーバー上のリポジトリと使用する Maven 設定は、プロジェクトの制御および管理を行いやすくします。



#### 注記

Maven の **settings.xml** ファイルを変更してリポジトリを設定する場合、変更はすべての Maven プロジェクトに適用されます。

### 手順

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



#### 注記

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

2. 以下の行を **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>

```

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

```

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

```

## 2.2. QUARKUS MAVEN リポジトリのダウンロードおよび設定

オンライン Maven リポジトリを使用しない場合は、Quarkus Maven リポジトリをダウンロードして設定し、Maven を使用して Quarkus アプリケーションを作成できます。Quarkus Maven リポジトリには、Java 開発者がアプリケーションのビルドに使用する必要があるものが数多く含まれています。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。

### 手順

- Red Hat カスタマーポータル [Software Downloads](#) ページから、Quarkus Maven リポジトリの ZIP ファイルをダウンロードします (ログインが必要です)。
- ダウンロードしたアーカイブを展開します。

3. ~/.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. 以下の行を **settings.xml** ファイルの **<profiles>** 要素に追加します。ここで、**QUARKUS\_MAVEN\_REPOSITORY** はダウンロードした Quarkus Maven リポジトリーのパスになります。**QUARKUS\_MAVEN\_REPOSITORY** の形式は、**file://\$PATH** でなければなりません (例: **file:///home/userX/rh-quarkus-1.3.4.SP1-maven-repository/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>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>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>** 要素に追加して、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。ここで、**ARTIFACT\_NAME** は不明なアーティファクトの名前、**PROJECT\_NAME** はビルドを試みているプロジェクトの名前になります。

- **Missing artifact PROJECT\_NAME**
- **[ERROR] Failed to execute goal on project ARTIFACT\_NAME; Could not resolve dependencies for PROJECT\_NAME**

この問題を解決するには、`~/.m2/repository` ディレクトリにあるローカルリポジトリのキャッシュバージョンを削除し、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。

## 第3章 GETTING STARTED プロジェクトの作成

**getting-started** プロジェクトでは、Apache Maven および Quarkus Maven プラグインを使用して、簡単な Quarkus アプリケーションを使い始めることができます。

### 手順

1. コマンドターミナルで以下のコマンドを入力し、Maven が JDK 11 を使用していること、そして Maven のバージョンが 3.6.2 以上であることを確認します。

```
mvn --version
```

2. 上記のコマンドで JDK 11 が返されない場合は、JDK 11 へのパスを PATH 環境変数に追加し、上記のコマンドを再度入力します。
3. プロジェクトを生成するには、以下のコマンドのいずれかを入力します。



### 注記

Apple macOS および Microsoft Windows は、本番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 io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create \
  -DprojectId=org.acme \
  -DprojectArtifactId=getting-started \
  -DplatformGroupId=com.redhat.quarkus \
  -DplatformVersion=1.3.4.Final-redhat-00004 \
  -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 io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create -
  DprojectId=org.acme -DprojectArtifactId=getting-started -
  DplatformGroupId=com.redhat.quarkus -DplatformVersion=1.3.4.Final-redhat-00004 -
  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 io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create "-
  DprojectId=org.acme" "-DprojectArtifactId=getting-started" "-
  DplatformGroupId=com.redhat.quarkus" "-
  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 の構造
- **/hello** で公開される **org.acme.quickstart.GreetingResource** リソース

- 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** の **Dockerfile** ファイルの例
  - アプリケーション設定ファイル
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。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.redhat.quarkus</groupId>
      <artifactId>quarkus-universe-bom</artifactId>
      <version>${quarkus-plugin.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>

<build>
  <plugins>
    <plugin>
      <groupId>io.quarkus</groupId>
      <artifactId>quarkus-maven-plugin</artifactId>
      <version>${quarkus-plugin.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    <plugin>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${compiler-plugin.version}</version>
    </plugin>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemProperties>

<java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
      </systemProperties>
    </configuration>
    </plugin>
  </plugins>
</build>
```

Quarkus BOM は **pom.xml** ファイルにインポートされます。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。さらに、アプリケーションをパッケージ化し、開発モードを提供する **quarkus-maven-plugin** プラグインがあることを確認できます。

5. **pom.xml** ファイルで、**quarkus-resteasy** の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</artifactId>
</dependency>
```

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

```
package org.acme.quickstart;

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";
    }
}
```

このファイルには、Java API for RESTful Web Services (JAX-RS) のほか、**/hello** への要求に対して **hello** を返す非常に簡単な 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。このインスタンスは、別の **\*Scoped** アノテーションを使用して設定できます。たとえば、**ApplicationScoped**、**RequestScoped** などを使用できます。

## 第4章 QUARKUS GETTING STARTED プロジェクトのコンパイル

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。

この例では、Quarkus の組み込み開発モードを使用しています。開発モードでは、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。変更が実行中のアプリケーションに反映されます。

### 前提条件

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

### 手順

1. 開発モードで Quarkus Hello アプリケーションをコンパイルするには、プロジェクトディレクトリから以下のコマンドを入力します。

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

以下の例は、このコマンドの出力を示しています。

```
2020-04-02 10:53:44,263 INFO [io.quarkus] (main) getting-started 1.0-SNAPSHOT
(powered by Quarkus 1.3.4.Final-redhat-00004) started in 0.946s. Listening on:
http://0.0.0.0:8080
2020-04-02 10:53:44,267 INFO [io.quarkus] (main) Profile dev activated. Live Coding
activated.
2020-04-02 10:53:44,267 INFO [io.quarkus] (main) Installed features: [cdi, resteasy]
```

2. 提供されたエンドポイントを要求するには、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
hello
```



### 注記

この例では、`"\n"` 属性を使用して、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、ターミナルで `'%'` 文字が出力されたり、出力結果と次のコマンドプロンプトが同じ行に表示されたりすることを防ぎます。

3. アプリケーションを停止するには、**CTRL+C** を押します。



## 第5章 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。クライアントの動作からクライアントの依存関係の作成を分離させるので、プログラム設計の結合度を弱めることができます。

Red Hat ビルドの Quarkus におけるディペンデンシーインジェクション (依存性の注入) は、Quarkus Arc をベースとしています。Quarkus Arc とは、Quarkus アーキテクチャーに合わせた CDI ベースのビルドタイム指向のディペンデンシーインジェクションソリューションのことです。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

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

### 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** をインジェクトし、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には、以下のコマンドを入力して再起動します。

```
./mvnw compile 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
```

## 第6章 JUNIT を使用した 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ら、アプリケーションを JUnit 5 フレームワークでテストし、想定どおりに実行されることを確認します。Quarkus プロジェクトが生成した **pom.xml** ファイルには、テスト用の依存関係が2つあります。

```
<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>
```

**quarkus-junit5** 依存関係は、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** 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を提供するため、統合されています。正しい URL を自動的に設定するので、何も設定する必要はありません。



### 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こと。

### 手順

1. Surefire Maven プラグインのバージョンを設定します。

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

    <java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
    </systemProperties>
  </configuration>
</plugin>
```

デフォルトの Surefire Maven プラグインバージョンは、JUnit 5 をサポートしません。

2. **java.util.logging** システムプロパティを設定して、テストが適切なログマネージャーを使用するようにします。
3. **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。

4. 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、最初に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。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インジェクトしま

す。このフィールドには、タイプ **string**、**URL**、または **URI** を使用できます。このアノテーションにテストパスを指定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;
```

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



### 注記

Quarkus は、インジェクションを使用できない状況のベーステスト URL に設定される **test.url** という名前のシステムプロパティも作成します。

## 第7章 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。

### 前提条件

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

### 手順

1. Quarkus Getting Started プロジェクトをパッケージ化するには、**root** ディレクトリーで以下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-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。
  - **getting-started-1.0-SNAPSHOT-runner.jar**: 実行可能な JAR ファイルです。依存関係は **target/lib** ディレクトリーにコピーされるため、このファイルは uber-JAR ファイルではない点に注意してください。
2. 開発モードを実行している場合は、**CTRL+C** を押して開発モードを停止します。停止しないと、ポートの競合が発生します。
  3. アプリケーションを実行するには、以下のコマンドを入力します。

```
java -jar target/getting-started-1.0-SNAPSHOT-runner.jar
```



### 注記

**runner** JAR ファイルからの **MANIFEST.MF** ファイルの **Class-Path** エントリーは、**lib** ディレクトリーからの JAR ファイルを明示的に記述します。別の場所からアプリケーションをデプロイする場合は、**runner** 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。

## 第8章 その他のリソース

- Maven を使用した Quarkus アプリケーションの作成に関する詳細は、[『Apache Maven を使用した Quarkus アプリケーションの作成』](#) を参照してください。
- 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。
- Maven Surefire プラグインの詳細は、[Apache Maven Project](#) の Web サイトを参照してください。
- JUnit 5 テストフレームワークの詳細は、[JUnit 5](#) の Web サイトを参照してください。
- REST-assured の詳細は、[REST-assured](#) の Web サイトを参照してください。

改訂日時: 2020-11-18 01:18:15 UTC