



Red Hat build of Quarkus 1.3

Apache Maven を使用した Quarkus アプリケーションの作成

Red Hat build of Quarkus 1.3 Apache Maven を使用した Quarkus アプリケーションの作成

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[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

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

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] 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[®] 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 フレームワークを使用して 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ーの MAVEN の SETTINGS.XML ファイルの設定	5
2.2. QUARKUS MAVEN リポジトリーのダウンロードおよび設定	6
第3章 コマンドラインでの QUARKUS プロジェクトの作成	9
第4章 POM.XML ファイルの設定による QUARKUS プロジェクトの作成	12
第5章 JAVA コンパイラーの設定	13
第6章 QUARKUS アプリケーションを使用した JAVA エクステンションのインストールおよび管理	14
第7章 QUARKUS プロジェクトの IDE へのインポート	15
第8章 QUARKUS プロジェクトの出力の設定	17
第9章 QUARKUS アプリケーションのテスト	18
第10章 QUARKUS アプリケーションのビルドクラスパスツリーのロギング	19
第11章 QUARKUS 開発モードの使用	20
第12章 QUARKUS プロジェクトのデバッグ	21
第13章 その他のリソース	22

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 プラグインを使用して Quarkus プロジェクトを作成する方法を説明します。

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。

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>

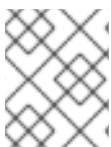
```

- 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 コマンドラインでの QUARKUS プロジェクトの作成

コマンドラインで Quarkus Maven プラグインを使用し、コマンドラインで属性および値を指定するか、インタラクティブモードでプラグインを使用して、Quarkus プロジェクトを作成できます。作成されるプロジェクトには以下の要素が含まれます。

- Maven の構造
- 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

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. Quarkus Maven プラグインを使用して新規プロジェクトを作成するには、以下のいずれかの方法を使用します。
 - 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=PROJECT_GROUP_ID \
  -DprojectArtifactId=PROJECT_ARTIFACT_ID \
  -DplatformGroupId=com.redhat.quarkus \
  -DplatformArtifactId=quarkus-universe-bom \
  -DplatformVersion=1.3.4.Final-redhat-00004 \
  -DclassName="CLASSNAME"
```

このコマンドで、以下の値を置き換えます。

- **PROJECT_GROUP_ID**: プロジェクトの一意識別子。
 - **PROJECT_ARTIFACT_ID**: プロジェクトおよびプロジェクトディレクトリーの名前。
 - **CLASSNAME**: 生成されたリソースの完全修飾名 (例: **org.acme.quarkus.sample.HelloResource**)。
- インタラクティブモードでプロジェクトを作成します。

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create
```

プロンプトが表示されたら、必要な属性値を入力します。



注記

以下のコマンドを入力し、プロジェクト属性のデフォルト値を使用してプロジェクトを作成することもできます。

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create -B
```

以下の表は、**create** コマンドで定義できる属性を一覧表示しています。

属性	デフォルト値	説明
projectGroupId	org.acme.sample	プロジェクトの一意識別子。
projectArtifactId	なし	プロジェクトおよびプロジェクトディレクトリーの名前。 projectArtifactId を指定しないと、Maven プラグインはインタラクティブモードを起動します。ディレクトリーがすでに存在する場合、生成は失敗します。
projectVersion	1.0-SNAPSHOT	プロジェクトのバージョン。
platformGroupId	io.quarkus	プラットフォームのグループ ID。既存のプラットフォームのデフォルト値はすべて、 io.quarkus です。ただし、デフォルト値は変更することができます。
platformArtifactId	quarkus-universe-bom	プラットフォーム BOM のアーティファクト ID。ローカルにビルドされた Quarkus を使用するには、 quarkus-universe-bom を pom.xml ファイルに追加します。
platformVersion	プラットフォームの最新バージョン	プロジェクトに使用するプラットフォームのバージョン。バージョン範囲を指定でき、Maven プラグインは最新バージョンを使用します。

属性	デフォルト値	説明
className	なし	生成されたリソースの完全修飾名。アプリケーションの作成後、REST エンドポイントは以下の URL で公開されます。 http://localhost:8080/\$path デフォルトの path を使用する場合、URL は http://localhost:8080/hello になります。
path	/hello	リソースパス (className を設定した場合のみ)。
extensions	[]	プロジェクトに追加するエクステンションのコンマ区切りリスト。



注記

デフォルトでは、Quarkus Maven プラグインは最新の **quarkus-universe-bom** ファイルを使用します。この BOM は、アプリケーションからエクステンションを参照し、依存関係バージョンを合わせるためにエクステンションを集約します。オフラインの場合、Quarkus Maven プラグインは、ローカルで利用可能な **quarkus-universe-bom** の最新バージョンを使用します。Maven が **quarkus-universe-bom** バージョン 2.0 以前を見つければ、**quarkus-universe-bom** をベースにしたプラットフォームを使用します。

第4章 POM.XML ファイルの設定による QUARKUS プロジェクトの作成

Maven POM XML ファイルを設定して Quarkus プロジェクトを作成できます。

手順

1. テキストエディターで **pom.xml** ファイルを開きます。
2. Quarkus GAV (Group、Artifact、Version) を追加し、**quarkus-universe-bom** ファイルを使用して、Quarkus の異なる依存関係のバージョンを省略します。

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>${quarkus.platform.group-id}</groupId>
      <artifactId>${quarkus.platform.artifact-id}</artifactId>
      <version>${quarkus-plugin.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

3. Quarkus Maven プラグインを追加します。

```
<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>
  </plugins>
</build>
```


第5章 JAVA コンパイラーの設定

デフォルトでは、Quarkus Maven プラグインはコンパイラーフラグを **maven-compiler-plugin** プラグインから **javac** コマンドに渡します。

開発モードで使用されるコンパイラーフラグをカスタマイズするには、**configuration** セクションを **plugin** ブロックに追加し、**compilerArgs** プロパティを設定します。**source**、**target**、**jvmArgs** を設定することもできます。たとえば、**--enable-preview** を JVM と **javac** の両方に渡すには、以下の行を追加します。

```
<plugin>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-maven-plugin</artifactId>
  <version>${quarkus-plugin.version}</version>

  <configuration>
    <source>${maven.compiler.source}</source>
    <target>${maven.compiler.target}</target>
    <compilerArgs>
      <arg>--enable-preview</arg>
    </compilerArgs>
    <jvmArgs>--enable-preview</jvmArgs>
  </configuration>

  ...
</plugin>
```

第6章 QUARKUS アプリケーションを使用した JAVA エクステンションのインストールおよび管理

Java エクステンションを使用して、アプリケーションの機能を拡張し、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。この手順では、エクステンションを検索して Quarkus プロジェクトに追加する方法を説明します。

前提条件

- Quarkus Maven プロジェクトがあること。

手順

1. Quarkus プロジェクトディレクトリーに移動します。
2. 利用可能なエクステンションを一覧表示するには、以下のコマンドを入力します。

```
./mvnw quarkus:list-extensions
```

3. プロジェクトにエクステンションを追加するには、以下のコマンドを入力します。ここで、**EXTENSION** は、追加するエクステンションの Group、Artifact、Version (GAV) に置き換えます。

```
./mvnw quarkus:add-extension -Dextensions="EXTENSION"
```

たとえば、Agroal エクステンションを追加するには、以下のコマンドを入力します。

```
./mvnw quarkus:add-extension -Dextensions="io.quarkus:quarkus-agroal"
```

4. 特定のエクステンションを検索するには、**-Dextensions=** の後にエクステンション名または名前の一部を入力します。以下の例では、名前に **jdbc**、**agroal**、および **non-existent** のテキストが含まれるエクステンションを検索します。

```
./mvnw quarkus:add-extension -Dextensions=jdbc,agroal,non-existent
```

このコマンドは、以下の結果を返します。

```
Multiple extensions matching 'jdbc'
* io.quarkus:quarkus-jdbc-h2
* io.quarkus:quarkus-jdbc-mariadb
* io.quarkus:quarkus-jdbc-postgresql
Be more specific e.g using the exact name or the full gav.
Adding extension io.quarkus:quarkus-agroal
Cannot find a dependency matching 'non-existent', maybe a typo?
[...]
```

5. 特定のテキスト文字列が返すすべてのエクステンションをインストールするには、**-Dextensions=** の後にエクステンション名または名前的一部分を入力します。以下の例では、**hibernate-** で始まるすべてのエクステンションを検索し、インストールします。

```
./mvnw quarkus:add-extension -Dextensions="hibernate-*
```

第7章 QUARKUS プロジェクトの IDE へのインポート

テキストエディターで Quarkus プロジェクトを開発することは可能ですが、統合開発環境 (IDE) を使用した方がプロジェクトの作業がしやすいかもしれません。以下の手順では、Quarkus プロジェクトを特定の IDE にインポートする方法を説明します。

前提条件

- Quarkus Maven プロジェクトがあること。

手順

以下のいずれかのセクションの手順を実行します。

CodeReady Studio または Eclipse

1. CodeReady Studio または Eclipse で、**File** → **Import** とクリックします。
2. **Maven** → **Existing Maven Project** と選択します。
3. 次の画面で、プロジェクトのルートロケーションを選択します。見つかったモジュールの一覧が表示されます。
4. 生成されたプロジェクトを選択し、**Finish** をクリックします。
5. アプリケーションを起動するには、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。

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

IntelliJ

1. IntelliJ で、以下のタスクのいずれかを実行します。
 - **File** → **New** → **Project From Existing Sources** と選択します。
 - **Welcome** ページで **Import project** を選択します。
2. プロジェクトの root ディレクトリーを選択します。
3. **Import project from external model** を選択してから、**Maven** を選択します。
4. オプションを確認して **Next** をクリックします。
5. **Finish** をクリックします。
6. アプリケーションを起動するには、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。

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

Apache NetBeans

1. **File** → **Open Project** と選択します。

2. プロジェクトの **root** ディレクトリーを選択します。
3. **Open Project** クリックします。
4. アプリケーションを起動するには、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。

```
┃ ./mvnw compile quarkus:dev
```

Visual Studio Code

1. Java Extension Pack をインストールします。
2. Visual Studio Code でプロジェクトディレクトリーを開きます。プロジェクトは Maven プロジェクトとしてロードされます。

第8章 QUARKUS プロジェクトの出力の設定

アプリケーションをビルドする前に、**application.properties** ファイルのアプリケーションプロパティのデフォルト値を変更することで、ビルドコマンドの出力を制御できます。

前提条件

- Quarkus Maven プロジェクトがあること。

手順

1. テキストエディターで **application.properties** ファイルを開きます。
2. 変更するプロパティの値を編集し、ファイルを保存します。
以下の表は、変更可能なプロパティをまとめたものです。

プロパティ	説明	タイプ	デフォルト
quarkus.package.main-class	アプリケーションのエントリーポイント。ほとんどの場合、この値は変更する必要があります。	string	io.quarkus.runner.GeneratedMain
quarkus.package.type	要求された出力タイプ。	string	jar
quarkus.package.uber-jar	Java ランナーを uber-JAR としてパックすべきかどうか。	boolean	false
quarkus.package.manifest.add-implementation-entries	実装情報をランナー JAR ファイルの MANIFEST.MF ファイルに含めるかどうか。	boolean	true
quarkus.package.user-configured-ignored-entries	出力アーティファクトにコピーしてはならないファイル。	string (list)	
quarkus.package.runner-suffix	ランナー JAR ファイルに適用される接尾辞。	string	-runner
quarkus.package.output-directory	アプリケーションビルドの出力フォルダー。これは、ビルドシステムのターゲットディレクトリーと相対的に解決されます。	string	
quarkus.package.output-name	最終的なアーティファクトの名前。	string	

第9章 QUARKUS アプリケーションのテスト

デフォルトでは、Quarkus アプリケーションをテストする場合、Maven は **test** 設定プロファイルを使用します。ただし、Maven Surefire プラグインを使用して、テスト用のカスタム設定プロファイルを作成することができます。

前提条件

- Apache Maven を使用して作成した Quarkus プロジェクトがあること。

手順

- 以下の例を編集し、テスト要件を満たします。**PROFILE_NAME** は、テストプロファイルの名前に置き換えます。

```
<project>
[...]
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${SureFirePluginVersion}</version>
      <configuration>
        <systemPropertyVariables>
          <quarkus.test.profile>PROFILE_NAME</quarkus.test.profile>
          <buildDirectory>${project.build.directory}</buildDirectory>
          [...]
        </systemPropertyVariables>
      </configuration>
    </plugin>
  </plugins>
</build>
[...]
```

第10章 QUARKUS アプリケーションのビルドクラスパスツリーのロギング

Quarkus ビルドプロセスでは、アプリケーションで使用するエクステンションのデプロイメント依存関係が、元のアプリケーションクラスパスに追加されます。ビルドクラスパスに含まれる依存関係およびバージョンを確認することができます。**quarkus-bootstrap** Maven プラグインには、アプリケーションのビルド依存関係ツリーを表示する **build-tree** ゴールが含まれます。

前提条件

- Quarkus Maven アプリケーションがあること。

手順

1. プラグイン設定を **pom.xml** ファイルに追加します。

```
<project>
  [...]
  <plugin>
    <groupId>io.quarkus</groupId>
    <artifactId>quarkus-bootstrap-maven-plugin</artifactId>
    <version>${quarkus-plugin.version}</version>
  </plugin>
  [...]
</project>
```

2. アプリケーションのビルド依存関係ツリーを一覧表示するには、以下のコマンドを入力します。

```
./mvnw quarkus-bootstrap:build-tree
```

3. このコマンドの出力は、以下のようになります。

```
[INFO] --- quarkus-bootstrap-maven-plugin:1.3:build-tree (default-cli) @ getting-started ---
[INFO] org.acme:getting-started:jar:1.0-SNAPSHOT
[INFO] └─ io.quarkus:quarkus-resteasy-deployment:jar:1.3 (compile)
[INFO] │   └─ io.quarkus:quarkus-resteasy-server-common-deployment:jar:1.3 (compile)
[INFO] │       └─ io.quarkus:quarkus-core-deployment:jar:1.3 (compile)
[INFO] │           └─ commons-beanutils:commons-beanutils:jar:1.9.3 (compile)
[INFO] │               └─ commons-logging:commons-logging:jar:1.2 (compile)
[INFO] │                   └─ commons-collections:commons-collections:jar:3.2.2 (compile)
...

```



注記

mvn dependency:tree コマンドは、アプリケーションのランタイム依存関係のみを表示します。

第11章 QUARKUS 開発モードの使用

開発モードはバックグラウンドコンパイルによるホットデプロイメントを可能にします。つまり、Java ファイルまたはリソースファイルを変更してブラウザーを更新すると、変更が自動的に反映されます。これは、設定プロパティファイルなどのリソースファイルでも同じく反映されます。

前提条件

- Quarkus Maven アプリケーションがあること。

手順

1. 開発モードで Quarkus を起動するには、Quarkus アプリケーションの **pom.xml** ファイルが含まれるディレクトリで以下のコマンドを入力します。

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

2. アプリケーションに変更を加え、ファイルを保存します。
3. ブラウザーを更新して、ワークスペースのスキャンをトリガーします。
変更が検出されると、Java ファイルが再コンパイルされ、アプリケーションが再デプロイされます。その後、要求は再デプロイされたアプリケーションによって処理されます。コンパイルまたはデプロイメントに問題がある場合には、エラーページが表示されます。

開発モードでは、デバッガーがアクティベートされ、ポート **5005** をリッスンします。

4. 任意手順: アプリケーションの実行前にデバッガーが割り当てられるのを待つには、**-Dsuspend** を追加します。

```
./mvnw quarkus:dev -Dsuspend
```

5. 任意手順: デバッガーが実行されないようにするには、**-Ddebug=false** を追加します。

```
./mvnw quarkus:dev -Ddebug=false
```


第12章 QUARKUS プロジェクトのデバッグ

Quarkus が開発モードで起動すると、デバッグはデフォルトで有効になります。デバッガーは、JVM を一時停止せずにポート **5005** でリスンします。

前提条件

- Quarkus Maven プロジェクトがあること。

手順

デバッグを制御するには、以下のいずれかの方法を使用します。

システムプロパティを使用したデバッガーの制御

1. 以下の **debug** システムプロパティの値の1つを変更します。ここで、**PORT** はデバッガーがリスンするポートです。
 - **false**: JVM はデバッグモードを無効にして開始します。
 - **true**: JVM はデバッグモードで開始され、ポート **5005** でリスンしています。
 - **client**: JVM はクライアントモードで起動され、**localhost:5005** への接続を試みます。
 - **PORT**: JVM はデバッグモードで開始され、**PORT** をリスンしています。
2. **suspend** システムプロパティの値を変更します。このプロパティは、Quarkus がデバッグモードで開始する際に使用されます。
 - **y** または **true**: デバッグモードの JVM の起動が一時停止します。
 - **n** または **false**: デバッグモードの JVM は一時停止せずに起動します。

コマンドラインからのデバッガーの制御

1. JVM を使用してデバッグモードで Quarkus アプリケーションを起動するには、以下のコマンドを入力します。

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

2. **localhost:5005** にデバッガーを割り当てます。

第13章 その他のリソース

- 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。