



Red Hat build of Quarkus 1.11

Apache Maven を使用した Quarkus アプリケーションの開発およびコンパイル

Red Hat build of Quarkus 1.11 Apache Maven を使用した Quarkus アプリケーションの開発およびコンパイル

法律上の通知

Copyright © 2023 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

このガイドでは、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

はじめに	3
RED HAT ドキュメントへのフィードバック (英語のみ)	4
多様性を受け入れるオープンソースの強化	5
第1章 RED HAT ビルドの QUARKUS	6
第2章 APACHE MAVEN および QUARKUS	7
2.1. オンラインリポジトリの MAVEN の SETTINGS.XML ファイルの設定	7
2.2. QUARKUS MAVEN リポジトリのダウンロードおよび設定	8
第3章 コマンドラインでの QUARKUS プロジェクトの作成	10
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	13
第5章 CODE.QUARKUS.REDHAT.COM を使用した QUARKUS MAVEN プロジェクトの作成	15
第6章 JAVA コンパイラーの設定	19
第7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	20
第8章 QUARKUS プロジェクトの IDE へのインポート	21
第9章 QUARKUS プロジェクトの出力の設定	23
第10章 QUARKUS アプリケーションのテスト	24
第11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	25
第12章 ネイティブ実行可能ファイルの作成	26
12.1. 手動でのコンテナの作成	27
第13章 ネイティブ実行可能ファイルのテスト	29
第14章 QUARKUS 開発モードの使用	31
第15章 QUARKUS プロジェクトのデバッグ	32
第16章 関連情報	33

はじめに

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

本ガイドでは、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。
 - Red Hat ビルドの Open JDK は、Red Hat カスタマーポータルの [Software Downloads](#) ページからダウンロードできます (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。
 - Maven は [Apache Maven Project](#) の Web サイトからダウンロードできます。

RED HAT ドキュメントへのフィードバック (英語のみ)

弊社の技術的な内容についてのフィードバックに感謝します。ご意見をお聞かせください。コメントの追加、Insights の提供、誤字の修正、および質問を行う必要がある場合は、ドキュメントで直接行うこともできます。



注記

Red Hat アカウントがあり、カスタマーポータルにログインしている必要があります。

カスタマーポータルからドキュメントのフィードバックを送信するには、以下の手順を実施します。

1. **Multi-page HTML** 形式を選択します。
2. ドキュメントの右上にある **Feedback** ボタンをクリックします。
3. フィードバックを提供するテキストのセクションを強調表示します。
4. ハイライトされたテキストの横にある **Add Feedback** ダイアログをクリックします。
5. ページの右側のテキストボックスにフィードバックを入力し、**Submit** をクリックします。

フィードバックを送信すると、自動的に問題の追跡が作成されます。**Submit** をクリックすると表示されるリンクを開き、問題の監視を開始するか、さらにコメントを追加します。

貴重なフィードバックにご協力いただきありがとうございます。

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 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ため、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わ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構築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のターゲットを記述し、XML ファイルを使用して出力します。これにより、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、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、Red Hat ビルドの Quarkus Maven リポジトリをダウンロードすることができます。

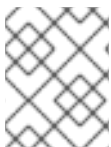
Maven プラグイン

Maven プラグインは、POM ファイルの定義済みの部分で1つ以上のゴールを達成します。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して、uber-JAR ファイルの生成をサポートし、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 プロジェクトで使うことができます。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性を向上させます。

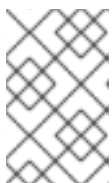


注記

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には、`$MAVEN_HOME/.m2/conf/` ディレクトリから `~/.m2/` ディレクトリに **settings.xml** ファイルをコピーします。

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>
```

- 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>
```

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構築に使用する要件の多くが含まれています。この手順では、Maven の **settings.xml** ファイルを編集し、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

- Quarkus Maven リポジトリの ZIP ファイルは、Red Hat カスタマーポータルの [Software Downloads](#) ページからダウンロードできます (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。
- ~/**.m2**/ ディレクトリに移動し、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 リポジトリのパスの形式は、**file://\$PATH** でなければなりません (例: **file:///home/userX/rh-quarkus-1.11.7.GA-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>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で、**<project_name>** は構築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 JVM およびネイティブモードの **Dockerfile** ファイルの例
- アプリケーションの設定ファイル

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:1.11.7.Final-redhat-00009:create \
  -DprojectId=<project_group_id> \
  -DprojectArtifactId=<project_artifact_id> \
  -DplatformGroupId=com.redhat.quarkus \
  -DplatformArtifactId=quarkus-universe-bom \
  -DplatformVersion=1.11.7.Final-redhat-00009 \
  -DclassName="<classname>"
```

このコマンドでは、以下の値を置き換えてください。

- **<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:1.11.7.Final-redhat-00009: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:1.11.7.Final-redhat-00009: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. 以下が含まれる設定プロパティを追加します。
 - Quarkus Maven プラグインのバージョン
 - Quarkus BOM の **groupId**、**artifactId**、および **version**
 - Maven Surefire プラグインのバージョン

```
<properties>
  <quarkus-plugin.version>1.11.7.Final-redhat-00009</quarkus-plugin.version>
  <quarkus.platform.artifact-id>quarkus-universe-bom</quarkus.platform.artifact-id>
  <quarkus.platform.group-id>com.redhat.quarkus</quarkus.platform.group-id>
  <quarkus.platform.version>1.11.7.Final-redhat-00009</quarkus.platform.version>
  <surefire-plugin.version>3.0.0-M5</surefire-plugin.version>
</properties>
```

3. 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.platform.version}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

4. 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.platform.version}</version>
      <executions>
        <execution>
          <goals>
            <goal>build</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

```

        </executions>
      </plugin>
    </plugins>
  </build>

```

5. 任意手順: ネイティブアプリケーションをビルドするには、Maven Surefire および Maven Failsafe プラグインを含む特定のネイティブプロファイルを追加し、**native** パッケージタイプを有効にします。

```

<profiles>
  <profile>
    <id>native</id>
    <properties>
      <quarkus.package.type>native</quarkus.package.type>
    </properties>
    <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-failsafe-plugin</artifactId>
          <version>${surefire-plugin.version}</version>
          <executions>
            <execution>
              <goals>
                <goal>integration-test</goal>
                <goal>verify</goal>
              </goals>
              <configuration>
                <systemProperties>
<native.image.path>${project.build.directory}/${project.build.finalName}-
runner</native.image.path>
                </systemProperties>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>

```

名前に **IT** が含まれるテストは、**@NativeImageTest** アノテーションが付けられ、ネイティブ実行可能ファイルに対して実行されます。

第5章 CODE.QUARKUS.REDHAT.COM を使用した QUARKUS MAVEN プロジェクトの作成

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。

本セクションでは、以下を含む Quarkus Maven プロジェクトを生成するプロセスについて説明します。

- 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機能拡張の選択
- プロジェクトファイルでダウンロード可能なアーカイブの生成
- アプリケーションのコンパイルおよび起動のカスタムコマンドの使用

前提条件

- Web ブラウザーがある。

手順

1. Web ブラウザーを使用して <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。名前の形式は、Java パッケージ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。選択するビルドツールにより、以下が決定されます。
 - 生成されたプロジェクトのディレクトリ構造。
 - 生成されたプロジェクトで使用する設定ファイルの形式。
 - アプリケーションをコンパイルおよび起動するためのカスタムビルドスクリプトおよびコマンド (プロジェクトの生成後に code.quarkus.redhat.com が表示)。



注記


Red Hat は、code.quarkus.redhat.com を使用して Quarkus Maven プロジェクトのみを作成する際にサポートを提供します。Red Hat では、Gradle プロジェクトの生成はサポートしていません。

Configure your application details

Group	org.acme
Artifact	code-with-quarkus
Build Tool	Maven ▼

CONFIGURE MORE OPTIONS

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。

- Configure more options** を選択し、追加のアプリケーションの詳細が含まれるフィールドを表示します。
- プロジェクトから生成されたアーティファクトで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が、必要に応じて、別のタイプのバージョンを使用できます。
- code.quarkus.redhat.com がサンプル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。エクステンションの一覧から  アイコンでマーク付け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は、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ファイル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。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。

Configure your application details


Group	org.acme	Version	1.0.0-SNAPSHOT
Artifact	code-with-quarkus	✈ Example Code	Yes, Please ▼
Build Tool	Maven ▼	CLOSE	



注記

code.quarkus.redhat.com 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。

- エクステンションの一覧から、アプリケーションで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、それらのバージョンは互換性を確保するために Quarkus プラットフォームによって管理されます。

このオプションを有効にすると、 アイコンでマーク付けされたエクステンションのサンプルコードを自動的に生成できます。

Pick your extensions

Q RESTEasy, Hibernate ORM, Web...

Selected Extensions

REST Client **SUPPORTED**

YAML Configuration **SUPPORTED** ✈

Web

<input type="checkbox"/> RESTEasy JAX-RS SUPPORTED ✈	REST endpoint framework implementing JAX-RS and more	⋮
<input type="checkbox"/> RESTEasy Jackson SUPPORTED ✈	Jackson serialization support for RESTEasy	⋮
<input type="checkbox"/> RESTEasy JSON-B SUPPORTED	JSON-B serialization support for RESTEasy	⋮
<input type="checkbox"/> Eclipse Vert.x GraphQL TECH-PREVIEW	Query the API using GraphQL	⋮
<input type="checkbox"/> Hibernate Validator SUPPORTED	Validate object properties (field, getter) and method parameters for y...	⋮
<input type="checkbox"/> Mutiny support for REST Client TECH-PREVIEW	Enable Mutiny for the REST client	⋮
<input checked="" type="checkbox"/> REST Client SUPPORTED	Call REST services	⋮
<input type="checkbox"/> REST Client JAXB SUPPORTED	Enable XML serialization for the REST Client	⋮

Red Hat は、一覧にある個別のエクステンションに対して **さまざまなレベル** のサポートを提供します。レベルは、各エクステンション名の横にあるラベルで示されています。

- **SUPPORTED** エクステンション: Red Hat は、実稼働環境でのエンタープライズアプリケーションでの使用に対して完全にサポートしています。
- **TECH-PREVIEW** 拡張: 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、限定的に、実稼働環境でのサポートを提供します。
- **DEV-SUPPORT** エクステンション: Red Hat は、実稼働環境での使用をサポートしていません。ただし、新規アプリケーションの開発での使用に対しては、Red Hat 開発者がこれらのコア機能をサポートしています。
- Red Hat は、ラベル付けされていないエクステンションの実稼働環境での使用をサポートしていません。
- **DEPRECATED** エクステンション: 同じ機能を提供する新しいテクノロジーまたは実装に置き換えられる予定です。
各エクステンションの横にあるオーバーフローメニュー (⋮) を展開して、以下を実行する追加オプションにアクセスできます。
 - エクステンションを既存プロジェクトに追加する (コマンドラインで Quarkus Maven プラグインを使用)。
 - プロジェクトの **pom.xml** ファイルへエクステンションを追加するための XML スニペットをコピーする。
 - 各エクステンションの **groupId**、**artifactId**、および **version** を取得する。
 - エクステンションガイドを開く。

Web

<input checked="" type="checkbox"/>	RESTEasy JAX-RS	SUPPORTED	▶	REST endpoint framework implementing JAX-RS and more	⋮
<input type="checkbox"/>	RESTEasy Jackson	SUPPORTED	▶	Jackson serialization support for RESTEasy	⋮
<input type="checkbox"/>	RESTEasy JSON-B	SUPPORTED		JSON-B serialization support for RESTEasy	⋮
<input type="checkbox"/>	Eclipse Vert.x GraphQL	TECH-PREVIEW		Query the API using GraphQL	⋮
<input type="checkbox"/>	Hibernate Validator	SUPPORTED		Validate object properties (field, getter) and method parameters for y...	⋮
<input type="checkbox"/>	Mutiny support for REST Client	TECH-PREVIEW		Enable Mutiny for the REST client	⋮
<input checked="" type="checkbox"/>	REST Client	SUPPORTED		Call REST services	⋮
<input type="checkbox"/>	REST Client JAXB	SUPPORTED		Enable XML se	⋮
<input checked="" type="checkbox"/>	REST Client JSON-B	SUPPORTED		Enable JSON-t	⋮
<input type="checkbox"/>	REST Client Jackson	SUPPORTED		Enable Jackson	⋮
<input type="checkbox"/>	REST resources for Hibernate ORM with P...	TECH-PREVIEW		Generate JAX-	⋮
<input type="checkbox"/>	REST resources for MongoDB with Panache			Generate JAX-	⋮
<input type="checkbox"/>	RESTEasy JAXB	SUPPORTED		XML serializati	⋮
<input type="checkbox"/>	RESTEasy Multipart	SUPPORTED		Multipart support for RESTEasy	⋮

- Copy the command to add it with Maven
- Copy the command to add it with Gradle
- Copy the extension pom.xml snippet
- Copy the extension GAV
- Open Extension Guide

5. **Generate your application** を選択して選択内容を確認し、生成されたプロジェクトを含むアーカイブのダウンロードリンクのオーバーレイ画面を表示します。オーバーレイ画面には、アプリケーションのコンパイルおよび起動に使用できるカスタムコマンドも表示されます。
6. **Download the ZIP** を選択して、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します。
7. アーカイブの内容を展開します。

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して起動します。

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

第6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。

手順

- 開発モードで使用す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** プロパティーを設定します。**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7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。

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) です。

```
./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-exist-ent** 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-exist-ent
```

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-exist-ent', 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8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。

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9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。

手順

- テキストエディターで **application.properties** ファイルを開きます。
- 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	型	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。	文字列	io.quarkus.runner.GeneratedMain
quarkus.package.type	要求された出力タイプ。	文字列	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。	ブール値	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。	ブール値	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。	文字列 (リスト)	
quarkus.package.runner-suffix	ランナー JAR ファイルに適用される接尾辞。	文字列	-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。	文字列	
quarkus.package.output-name	最終的なアーティファクトの名前。	文字列	

第10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。

手順

- テスト要件を満たすには、以下の例を編集します。ここで、**<profile_name>** は、テストプロファイルの名前になります。

```
<project>
[...]
```

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

```
</project>
```



注記

ネイティブモードでは、カスタムのテスト設定プロファイルは使用できません。ネイティブテストは常に、**prod** プロファイルを使用して実行されます。

第11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。

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:1.11:build-tree (default-cli) @ getting-started ---
[INFO] org.acme:getting-started:jar:1.0-SNAPSHOT
[INFO] └─ io.quarkus:quarkus-resteasy-deployment:jar:1.11 (compile)
[INFO] │   └─ io.quarkus:quarkus-resteasy-server-common-deployment:jar:1.11 (compile)
[INFO] │       └─ io.quarkus:quarkus-core-deployment:jar:1.11 (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12章 ネイティブ実行可能ファイルの作成

Podman または Docker などのコンテナランタイムを使用して、Quarkus アプリケーションからネイティブ実行可能ファイルを作成できます。Quarkus は、ビルダーイメージを使用してバイナリー実行可能ファイルを作成します。これは、Red Hat Universal Base Images RHEL8-UBI および RHEL8-UBI minimal と共に使用できます。Red Hat ビルドの Quarkus 1.11 は、**quarkus.native.builder-image** プロパティのデフォルトとして **registry.access.redhat.com/quarkus/mandrel-20-rhel8:20.3** を使用します。

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

手順

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

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



注記

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

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

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

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

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

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

これらのコマンドにより、**target** ディレクトリーに **getting-started-*-runner** バイナリーが作成されます。



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。詳細は、[ネイティブ実行可能ファイルの設定プロパティー](#) を参照してください。

3. ネイティブ実行可能ファイルを実行します。

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

ネイティブ実行可能ファイルをビルドすると、**prod** プロファイルが有効になり、**prod** プロファイルを使用して Quarkus ネイティブテストが実行されます。これは、**quarkus.test.native-image-profile** プロパティーを使用して変更できます。

12.1. 手動でのコンテナの作成

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

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

```
FROM registry.access.redhat.com/ubi8/ubi-minimal:8.3
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

CMD ["/application", "-Dquarkus.http.host=0.0.0.0"]
```



UNIVERSAL BASE IMAGE (UBI)

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

手順

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

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

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

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

2. 以下のいずれかの方法を使用して、コンテナイメージをビルドします。

- a. Docker を使用してコンテナイメージをビルドします。

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

- b. Podman を使用してコンテナイメージをビルドします。

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

3. コンテナを実行します。

- a. Docker を使用してコンテナを実行します。

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

- b. Podman を使用してコンテナを実行します。

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


第13章 ネイティブ実行可能ファイルのテスト

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

手順

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

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-failsafe-plugin</artifactId>
  <version>${surefire-plugin.version}</version>
  <executions>
    <execution>
      <goals>
        <goal>integration-test</goal>
        <goal>verify</goal>
      </goals>
      <configuration>
        <systemPropertyVariables>
          <native.image.path>${project.build.directory}/${project.build.finalName}-runner</native.image.path>
        </systemPropertyVariables>
      </configuration>
    </execution>
  </executions>
</plugin>

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

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

2. **src/test/java/org/acme/quickstart/NativeGreetingResourceIT.java** ファイルを開き、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.quickstart;

import io.quarkus.test.junit.NativeImageTest;

@NativeImageTest ❶
public class NativeGreetingResourceIT extends GreetingResourceTest { ❷

    // Run the same tests

}
```

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

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

3. テストを実行します。

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

以下の例は、このコマンドの出力を示しています。

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

注記

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

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

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

第14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。

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ー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ようにするには、**-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15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。

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ー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ーの値の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し、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ーの値を変更します。このプロパティーは、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

- 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
```

特定のホストドメインのデバッガーの有効化

開発モードでは、**localhost** で実行するアプリケーションに対してのみ、デバッガーが有効になります。他のホストで手動でデバッグを有効にする必要があります。

- 特定のホストでのデバッグを有効にするには、**debugHost** オプションを使用します。**<host-ip-address>** は、デバッグを有効化するホストの IP アドレスに置き換える必要があります。

```
./mvnw compile quarkus:dev -DdebugHost=<host-ip-address>
```

すべてのホストでデバッグを有効にするには、**<host-ip-address>** を **0.0.0** に置き換えます。

```
./mvnw compile quarkus:dev -DdebugHost=0.0.0.0
```

第16章 関連情報

- [Quarkus アプリケーションの OpenShift へのデプロイ](#)
- [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](#)
- [Quarkus アプリケーションのテスト](#)
- [Apache Maven Project](#)

改訂日時: 2023-05-16