Red Hat build of Quarkus 1.3

Getting started with Quarkus

Last Updated: 2020-12-08

Legal Notice

Abstract

This guide describes how to create a simple Quarkus application with Apache Maven.

Table of Contents

PREFACE

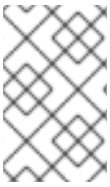
As an application developer, you can use Red Hat build of Quarkus to create microservices-based applications written in Java that run in OpenShift and serverless environments. Applications compiled to native executables have small memory footprints and fast startup times.

This guide shows you how to use Apache Maven to create, test, package, and run a simple Quarkus project that exposes a **hello** HTTP endpoint. To demonstrate dependency injection, this endpoint uses a **greeting** bean.

78_OpenShift_0420

Prerequisites

- OpenJDK (JDK) 11 is installed and the **JAVA_HOME** environment variable specifies the location of the Java SDK.
    - Log in to the Red Hat Customer Portal to download Red Hat build of Open JDK from the Software Downloads page.
- Apache Maven 3.6.2 or higher is installed. Maven is available from the Apache Maven Project website.



NOTE

For a completed example of the getting started exercise, download the Quarkus quickstart archive or clone the **Quarkus Quickstarts** Git repository. The Getting Started example is in the **getting-started** directory.

CHAPTER 1. RED HAT BUILD OF QUARKUS

Red Hat build of Quarkus is a Kubernetes-native Java stack that is optimized for use with containers and Red Hat OpenShift Container Platform. Quarkus is designed to work with popular Java standards, frameworks, and libraries such as Eclipse MicroProfile, Apache Kafka, RESTEasy (JAX-RS), Hibernate ORM (JPA), Spring, Infinispan, and Apache Camel.

The Quarkus dependency injection solution is based on CDI (contexts and dependency injection) and includes an extension framework to expand functionality and to configure, boot, and integrate a framework into your application.

Quarkus provides a container-first approach to building Java applications. This approach makes it much easier to build microservices-based applications written in Java as well as enabling those applications to invoke functions running on serverless computing frameworks. For this reason, Quarkus applications have small memory footprints and fast start-up times.

CHAPTER 2. APACHE MAVEN AND QUARKUS

Apache Maven is a distributed build automation tool used in Java application development to create, manage, and build software projects. Maven uses standard configuration files called Project Object Model (POM) files to define projects and manage the build process. POM files describe the module and component dependencies, build order, and targets for the resulting project packaging and output using an XML file. This ensures that the project is built in a correct and uniform manner.

Maven repositories

A Maven repository stores Java libraries, plug-ins, and other build artifacts. The default public repository is the Maven 2 Central Repository, but repositories can be private and internal within a company to share common artifacts among development teams. Repositories are also available from third-parties.

You can use the online Maven repository with your Quarkus projects or you can download the Red Hat build of Quarkus Maven repository.

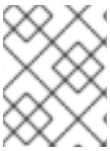
Maven plug-ins

Maven plug-ins are defined parts of a POM file that achieve one or more goals. Quarkus applications use the following Maven plug-ins:

- Quarkus Maven plug-in (**quarkus-maven-plugin**): Enables Maven to create Quarkus projects, supports the generation of uber-JAR files, and provides a development mode.
- Maven Surefire plug-in (**maven-surefire-plugin**): Used during the test phase of the build lifecycle to execute unit tests on your application. The plug-in generates text and XML files that contain the test reports.

2.1. CONFIGURING THE MAVEN SETTINGS.XML FILE FOR THE ONLINE REPOSITORY

You can use the online Quarkus repository with your Quarkus Maven project by configuring your user **settings.xml** file. This is the recommended approach. Maven settings used with a repository manager or repository on a shared server provide better control and manageability of projects.



NOTE

When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Procedure

1. Open the Maven **~/.m2/settings.xml** file in a text editor or integrated development environment (IDE).

NOTE

If there is not a **settings.xml** file in the **~/.m2/** directory, copy the **settings.xml** file from the **$MAVEN_HOME/.m2/conf/** directory into the **~/.m2/** directory.

2. Add the following lines to the **<profiles>** element of the **settings.xml** file:

```
<!-- 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. Add the following lines to the **<activeProfiles>** element of the **settings.xml** file and save the file.

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

2.2. DOWNLOADING AND CONFIGURING THE QUARKUS MAVEN REPOSITORY

If you do not want to use the online Maven repository, you can download and configure the Quarkus Maven repository to create a Quarkus application with Maven. The Quarkus Maven repository contains many of the requirements that Java developers typically use to build their applications. This procedure describes how to edit the **settings.xml** file to configure the Quarkus Maven repository.

NOTE

When you configure the repository by modifying the Maven **settings.xml** file, the changes apply to all of your Maven projects.

Procedure

1. Download the Quarkus Maven repository ZIP file from the Software Downloads page of the Red Hat Customer Portal (login required).

2. Expand the downloaded archive.

3. Change directory to the **~/.m2/** directory and open the Maven **settings.xml** file in a text editor or integrated development environment (IDE).

4. Add the following lines to the **<profiles>** element of the **settings.xml** file, where ***QUARKUS_MAVEN_REPOSITORY*** is the path of the Quarkus Maven repository that you downloaded. The format of ***QUARKUS_MAVEN_REPOSITORY*** must be **file://$PATH**, for example **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. Add the following lines to the **<activeProfiles>** element of the **settings.xml** file and save the file.

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

IMPORTANT

If your Maven repository contains outdated artifacts, you might encounter one of the following Maven error messages when you build or deploy your project, where ***ARTIFACT_NAME*** is the name of a missing artifact and ***PROJECT_NAME*** is the name of the project you are trying to build:

- **Missing artifact *PROJECT_NAME***
- **[ERROR] Failed to execute goal on project *ARTIFACT_NAME*; Could not resolve dependencies for *PROJECT_NAME***

To resolve the issue, delete the cached version of your local repository located in the ~/**.m2**/**repository** directory to force a download of the latest Maven artifacts.

CHAPTER 3. CREATING THE GETTING STARTED PROJECT

The **getting-started** project lets you get up and running with a simple Quarkus application using Apache Maven and the Quarkus Maven plug-in.

Procedure

1. In a command terminal, enter the following command to verify that Maven is using JDK 11 and that the Maven version is 3.6.2 or higher:

```
mvn --version
```

2. If the preceding command does not return JDK 11, add the path to JDK 11 to the PATH environment variable and enter the preceding command again.

3. To generate the project, enter one of the following commands:

NOTE

Apple macOS and Microsoft Windows are not supported production environments.

- If you are using Linux or Apple macOS, enter the following command:

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

- If you are using the Microsoft Windows command line, enter the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create -
DprojectGroupId=org.acme -DprojectArtifactId=getting-started -
DplatformGroupId=com.redhat.quarkus -DplatformVersion=1.3.4.Final-redhat-00004 -
DclassName="org.acme.quickstart.GreetingResource" -Dpath="/hello"
```

- If you are using the Microsoft Windows Powershell, enter the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.4.Final-redhat-00004:create "-
DprojectGroupId=org.acme" "-DprojectArtifactId=getting-started" "-
DplatformGroupId=com.redhat.quarkus" "-
DclassName=org.acme.quickstart.GreetingResource" "-Dpath=/hello"
```

These commands create the following elements in the **./getting-started** directory:

  - The Maven structure
  - An **org.acme.quickstart.GreetingResource** resource exposed on **/hello**

- An associated unit test
- A landing page that is accessible on **http://localhost:8080** after you start the application
- Example **Dockerfile** file in **src/main/docker**
- The application configuration file

4. After the directory structure is created, open the **pom.xml** file in a text editor and examine the contents of the file:

```
<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>
```

The Quarkus BOM is imported into the **pom.xml** file. Therefore, you do not need to list the

versions of individual Quarkus dependencies in the **pom.xml** file. In addition, you can see the **quarkus-maven-plugin** plug-in that is responsible for packaging the application and providing the development mode.

5. Review the **quarkus-resteasy** dependency in the **pom.xml** file. This dependency enables you to develop REST applications:

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

6. Review the **src/main/java/org/acme/quickstart/GreetingResource.java** file:

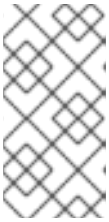
```
package org.acme.quickstart;

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";
    }
}
```

This file contains the Java API for RESTful Web Services (JAX-RS) and a very simple REST endpoint that returns **hello** to requests on **/hello**.



NOTE

With Quarkus, the **Application** class for JAX-RS is supported but not required. In addition, only one instance of the **GreetingResource** class is created and not one per request. You can configure this instance by using the different ***Scoped** annotations, for example **ApplicationScoped**, **RequestScoped**, and so forth.

CHAPTER 4. COMPILING THE QUARKUS GETTING STARTED PROJECT

After you have created the Quarkus Getting Started project, you can compile the Hello application and verify that the **hello** endpoint returns **hello**.

This example uses the Quarkus built-in development mode. In development mode, you can update the application sources and configurations while your application is running. Your changes will appear in the running application.

Prerequisites

- You have created the Quarkus Getting Started project.

Procedure

1. To compile the Quarkus Hello application in development mode, enter the following command from the project directory:

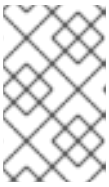
```
./mvnw compile quarkus:dev
```

   The following example shows the output of this command:

```
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. To request the provided endpoint, enter the following command in a new terminal window:

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



NOTE

This example uses the **"\n"** attribute to automatically add a new line before the output of the command. This prevents your terminal from printing a '%' character or putting both the result and the next command prompt on the same line.

3. To stop the application, press **CTRL+C**.

CHAPTER 5. USING QUARKUS DEPENDENCY INJECTION

Dependency injection enables a service to be used in a way that is completely independent of any client consumption. It separates the creation of client dependencies from the client's behavior, which enables program designs to be loosely coupled.

Dependency injection in Red Hat build of Quarkus is based on Quarkus ArC which is a CDI-based build-time oriented dependency injection solution tailored for Quarkus architecture. Because ArC is a transitive dependency of **quarkus-resteasy**, and **quarkus-resteasy** is a dependency of your project, ArC will already be downloaded.

Prerequisites

- You have created the Quarkus Getting Started project.

Procedure

1. To modify the application and add a companion bean, create the **src/main/java/org/acme/quickstart/GreetingService.java** file with the following content:

```
package org.acme.quickstart;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class GreetingService {

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

}
```

2. Edit the **src/main/java/org/acme/quickstart/GreetingResource.java** to inject the **GreetingService** and create a new endpoint using it:

```
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. If you stopped the application, enter the following command to restart it:

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

4. To verify that the endpoint returns **hello quarkus**, enter the following command in a new terminal window:

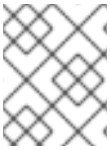
```
curl -w "\n" http://localhost:8080/hello/greeting/quarkus
hello quarkus
```

CHAPTER 6. TESTING YOUR QUARKUS APPLICATION WITH JUNIT

After you compile your Quarkus Getting Started project, test your application with the JUnit 5 framework to ensure that it runs as expected. There are two test dependencies in the Quarkus project generated **pom.xml** file:

```
<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>
```

The **quarkus-junit5** dependency is required for testing because it provides the **@QuarkusTest** annotation that controls the JUnit 5 testing framework. The **rest-assured** dependency is not required but because it provides a convenient way to test HTTP endpoints, it is integrated as well. It automatically sets the correct URL so no configuration is required.



NOTE

These tests use the REST-assured framework, but you can use a different library if you prefer.

Prerequisites

- You have compiled the Quarkus Getting Started project.

Procedure

1. Set the version of the Surefire Maven plug-in:

```
<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>
```

   The default Surefire Maven plugin version does not support JUnit 5.

2. Set the **java.util.logging** system property to make sure tests use the correct log manager.

3. Edit the **src/test/java/org/acme/quickstart/GreetingResourceTest.java** file to match the following content:

```
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));
    }

}
```

NOTE

By using the **QuarkusTest** runner, you instruct JUnit to start the application before starting the tests.

4. To run these tests from Maven, enter the following command:

```
./mvnw test
```

NOTE

You can also run the test from your IDE. If you do this, make sure to stop the application first.

By default, tests run on port **8081** so they do not conflict with the running application. In Quarkus, the **RestAssured** dependency is configured to use this port. If you want to use a different client, use the **@TestHTTPResource** annotation to directly inject the URL of the

tested application into a field on the **Test** class. This field can be of the type **string**, **URL** or **URI**. You can also provide the test path in this annotation. For example, to test a servlet mapped to /**myservlet**, add the following lines to your test:

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

5. If necessary, specify the test port in the **quarkus.http.test-port** configuration property.

NOTE

Quarkus also creates a system property called **test.url** that is set to the base test URL for situations where you cannot use injection.

CHAPTER 7. PACKAGING AND RUNNING THE QUARKUS GETTING STARTED APPLICATION

After you compile your Quarkus Getting Started project, you can package it in a JAR file and run it from the command line.

Prerequisites

- You have compiled the Quarkus Getting Started project.

Procedure

1. To package your Quarkus Getting Started project, enter the following command in the **root** directory:

```
./mvnw package
```

   This command produces the following JAR files in the **/target** directory:

   - **getting-started-1.0-SNAPSHOT.jar**: Contains the classes and resources of the projects. This is the regular artifact produced by the Maven build.
   - **getting-started-1.0-SNAPSHOT-runner.jar**: Is an executable JAR file. Be aware that this file is not an uber-JAR file because the dependencies are copied into the **target/lib** directory.

2. If development mode is running, press **CTRL+C** to stop development mode. If you do not do this, you will have a port conflict.

3. To run the application, enter the following command:

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

NOTE

The **Class-Path** entry of the **MANIFEST.MF** file from the **runner** JAR file explicitly lists the JAR files from the **lib** directory. If you want to deploy your application from another location, you must copy the **runner** JAR file as well as the **lib** directory.

CHAPTER 8. ADDITIONAL RESOURCES

- For more information about creating Quarkus applications with Maven, see *Developing and compiling your Quarkus applications with Apache Maven*.
- For more information on how to configure Quarkus applications see Configuring your Quarkus applications.
- For information about deploying Quarkus Maven applications on Red Hat OpenShift Container Platform, see *Deploying your Quarkus applications on Red Hat OpenShift Container Platform* .
- For more information about the Maven Surefire plug-in, see the Apache Maven Project website.
- For information about the JUnit 5 testing framework, see the JUnit 5 website.
- For information about REST-assured, see the REST-assured website.

Revised on 2020-12-08 20:09:08 UTC