Red Hat Integration 2021.Q4

Getting Started with Camel Extensions for Quarkus

Getting Started with Camel Extensions for Quarkus

Last Updated: 2022-03-08

Red Hat Integration 2021.Q4 Getting Started with Camel Extensions for Quarkus

Getting Started with Camel Extensions for Quarkus

Legal Notice

Abstract

This guide introduces Camel Extensions for Quarkus and explains the various ways to create and deploy an application using Camel Extensions for Quarkus.

Table of Contents

PREFACE

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright's message .

CHAPTER 1. GETTING STARTED WITH CAMEL EXTENSIONS FOR QUARKUS

This guide introduces Camel Extensions for Quarkus, the various ways to create a project and how to get started building an application using Camel Extensions for Quarkus:

- Section 1.1, “Camel Extensions for Quarkus overview”
- Section 1.2, “Tooling”
- Section 1.3, “Building your first project with Camel Extensions for Quarkus”

NOTE

Red Hat provides Maven repositories that host the content we ship with our products. These repositories are available to download from the software downloads page.

For Camel Extensions for Quarkus the following repositories are required:

- rhi-camel-extensions-for-quarkus

Installation of Camel Extensions for Quarkus in offline mode is not supported in this release.

For information on using the Apache Maven repository for Camel Quarkus, see Chapter 2.2. “Downloading and configuring the Quarkus Maven repository” in the *Developing and compiling your Quarkus applications with Apache Maven* guide

1.1. CAMEL EXTENSIONS FOR QUARKUS OVERVIEW

Camel Extensions for Quarkus brings the integration capabilities of Apache Camel and its vast component library to the Quarkus runtime.

The benefits of using Camel Extensions for Quarkus include the following:

- Enables users to take advantage of the performance benefits, developer joy and the container first ethos which Quarkus provides.
- Provides Quarkus extensions for many of the Apache Camel components.
- Takes advantage of the many performance improvements made in Camel 3, which results in a lower memory footprint, less reliance on reflection and faster startup times.
- You can define Camel routes using the Java DSL.

1.2. TOOLING

IMPORTANT

Red Hat does not provide support for these developer tools.

1.2.1. IDE plugins

Quarkus has plugins for most of the popular development IDEs which provide Quarkus language support, code/configuration completion, project creation wizards and much more. The plugins are available at each respective IDE marketplace.

- Eclipse plugin
- IntelliJ plugin
- VSCode plugin

Check the plugin documentation to discover how to create projects for your preferred IDE.

1.2.2. Camel content assist

The following plugins provide support for content assist when editing Camel routes and **application.properties**:

- VS Code Language support for Camel - a part of the Camel extension pack
- Eclipse Desktop Language Support for Camel - a part of Jboss Tools and CodeReady Studio
- Apache Camel IDEA plugin (not always up to date)
- Users of other IDEs supporting Language Server Protocol may choose to install and configure Camel Language Server manually

1.3. BUILDING YOUR FIRST PROJECT WITH CAMEL EXTENSIONS FOR QUARKUS

1.3.1. Overview

You can use code.quarkus.redhat.com to generate a Quarkus Maven project which automatically adds and configures the extensions that you want to use in your application.

This section walks you through the process of creating a Quarkus Maven project with Camel Extensions for Quarkus including:

- Creating the skeleton application using code.quarkus.redhat.com
- Adding a simple Camel route
- Exploring the application code
- Compiling the application in development mode
- Testing the application

1.3.2. Generating the skeleton application

Projects can be bootstrapped and generated at code.quarkus.redhat.com. The Camel Extensions for Quarkus extensions are located under the 'Integration' heading.

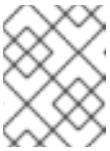
Use the 'search' field to find the extensions that you require.

Select the component extensions that you want to work with and click the 'Generate your application' button to download a basic skeleton project. There is also the option to push the project directly to GitHub.

For more information about using **code.quarkus.redhat.com** to generate Quarkus Maven projects, see Creating a Quarkus Maven project using code.quarkus.redhat.com in the *Getting Started with Quarkus* guide.

Procedure

1. Using the *code.quarkus.redhat.com* website, select the following extensions for this example:
   - **camel-quarkus-rest**
   - **camel-quarkus-jackson**



NOTE

You should not compile the application as stated in the final step of the above procedure as you will perform that task as part of this guide.

2. Navigate to the directory where you extracted the generated project files from the previous step:

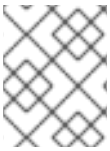
```
$ cd <directory_name>
```

1.3.3. Explore the application code

The application has two compile dependencies which are managed within the **com.redhat.quarkus.platform:quarkus-camel-bom** that is imported in **<dependencyManagement>**.:

pom.xml

```
<quarkus.platform.artifact-id>quarkus-bom</quarkus.platform.artifact-id>
<quarkus.platform.group-id>com.redhat.quarkus.platform</quarkus.platform.group-id>
<quarkus.platform.version>
    <!-- The latest 2.2.x version from
https://maven.repository.redhat.com/ga/com/redhat/quarkus/platform/quarkus-bom -->
</quarkus.platform.version>
...
<dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>${quarkus.platform.artifact-id}</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
<dependency>
    <groupId>${quarkus.platform.group-id}</groupId>
    <artifactId>quarkus-camel-bom</artifactId>
    <version>${quarkus.platform.version}</version>
    <type>pom</type>
    <scope>import</scope>
</dependency>
```



NOTE

For more information about BOM dependency management, see Developing Applications with Camel Extensions for Quarkus

The application is configured by properties defined within **src/main/resources/application.properties**, for example, the **camel.context.name** can be set there.

1.3.4. Adding a simple Camel route

Procedure

1. Create a file named **Routes.java** in the **src/main/java/org/acme/** subfolder.
2. Add a Camel Rest route as shown in the following code snippet:

Routes.java

```
package org.acme;

import java.util.Arrays;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.CopyOnWriteArrayList;

import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.model.rest.RestBindingMode;

import io.quarkus.runtime.annotations.RegisterForReflection;

public class Routes extends RouteBuilder {
    private final List<Fruit> fruits = new CopyOnWriteArrayList<>(Arrays.asList(new
Fruit("Apple")));

    @Override
    public void configure() throws Exception {
        restConfiguration().bindingMode(RestBindingMode.json);

        rest("/fruits")
                .get()
                .route()
                .setBody(e -> fruits)
                .endRest()

                .post()
                .type(Fruit.class)
                .route()
                .process().body(Fruit.class, (Fruit f) -> fruits.add(f))
                .endRest();

    }

    @RegisterForReflection // Let Quarkus register this class for reflection during the native
build
    public static class Fruit {
```

```
        private String name;

        public Fruit() {
        }

        public Fruit(String name) {
            this.name = name;
        }

        public String getName() {
            return name;
        }

        public void setName(String name) {
            this.name = name;
        }

        @Override
        public int hashCode() {
            return Objects.hash(name);
        }

        @Override
        public boolean equals(Object obj) {
            if (this == obj)
                return true;
            if (obj == null)
                return false;
            if (getClass() != obj.getClass())
                return false;
            Fruit other = (Fruit) obj;
            return Objects.equals(name, other.name);
        }

    }

}
```

1.3.5. Development mode

```
$ mvn clean compile quarkus:dev
```

This command compiles the project, starts your application and lets the Quarkus tooling watch for changes in your workspace. Any modifications in your project will automatically take effect in the running application.

Check the application in the browser, for example, http://localhost:8080/fruits for the **rest-json** example

If you change the application code, for example, change 'Apple' to 'Orange', your application will be automatically updated. To see the changes applied, simply refresh your browser.

Please refer to Quarkus documentation for more details about the development mode.

1.3.6. Testing

1.3.6.1. JVM mode

To test the Camel Rest route that we have created in JVM mode, you can add a test class as follows:

Procedure

1. Create a file named **RoutesTest.java** in the **src/test/java/org/acme/** subfolder.
2. Add the **RoutesTest** class as shown in the following code snippet:

RoutesTest.java

```
package org.acme;

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

import static io.restassured.RestAssured.given;
import org.hamcrest.Matchers;

@QuarkusTest
public class RoutesTest {

    @Test
    public void testFruitsEndpoint() {

        /* Assert the initial fruit is there */
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body(
                        "$.size()", Matchers.is(1),
                        "name", Matchers.contains("Orange"));

        /* Add a new fruit */
        given()
                .body("{\"name\": \"Pear\"}")
                .header("Content-Type", "application/json")
                .when()
                .post("/fruits")
                .then()
                .statusCode(200);

        /* Assert that pear was added */
        given()
                .when().get("/fruits")
                .then()
                .statusCode(200)
                .body(
                        "$.size()", Matchers.is(2),
                        "name", Matchers.contains("Orange", "Pear"));
    }

}
```

The JVM mode tests are run by **maven-surefire-plugin** in the **test** Maven phase:

```
$ mvn clean test
```

1.3.6.2. Native mode

To test the Camel Rest route that we have created in Native mode, you can add a test class as follows:

Procedure

1. Create a file named **NativeRoutesIT.java** in the **src/test/java/org/acme/** subfolder.
2. Add the **NativeRoutesIT** class as shown in the following code snippet:

NativeRoutesIT.java

```
package org.acme;

import io.quarkus.test.junit.NativeImageTest;

@NativeImageTest
public class NativeRoutesIT extends RoutesTest {

    // Execute the same tests but in native mode.
}
```

The native mode tests are verified by **maven-failsafe-plugin** in the **verify** phase. Pass the **native** property to activate the profile that runs them:

```
$ mvn clean verify -Pnative
```

1.3.7. Package and run the application

1.3.7.1. JVM mode

mvn package prepares a thin **jar** for running on a stock JVM:

```
$ mvn clean package
$ ls -lh target/quarkus-app
...
-rw-r--r--. 1 user user 238K Oct 11 18:55  quarkus-run.jar
...
```

You can run it as follows:

```
$ java -jar target/quarkus-app/quarkus-run.jar
...
[io.quarkus] (main) Quarkus started in 1.163s. Listening on: http://[::]:8080
```

Notice the boot time around a second.

The thin **jar** contains just the application code. To run it, the dependencies in **target/quarkus-app/lib** are required too.

1.3.7.2. Native mode

NOTE

See Producing a native executable in the *Compiling your Quarkus applications to native executables* guide, for additional information about preparing a native executable.

To prepare a native executable, run the following command:

```
$ mvn clean package -Pnative
$ ls -lh target
...
-rwxr-xr-x. 1 user user  46M Oct 11 18:57  code-with-quarkus-1.0.0-SNAPSHOT-runner
...
```

Note that the **runner** in the listing above has no **.jar** extension and has the **x** (executable) permission set. Thus it can be run directly:

```
$ ./target/*-runner
...
[io.quarkus] (main) Quarkus started in 0.013s. Listening on: http://[::]:8080
...
```

Note that the application started in just 13 milliseconds. To see how it handles memory efficiently, enter the following command:

```
$ ps -o rss,command -p $(pgrep code-with)
  RSS COMMAND
65852 ./target/code-with-quarkus-1.0.0-SNAPSHOT-runner
```

In the above example, the application uses just 65 MB of memory.

TIP

Quarkus Native executable guide contains more details including steps for creating a container image.

CHAPTER 2. DEPLOYING QUARKUS APPLICATIONS

You can deploy your Quarkus application on OpenShift by using any of the the following build strategies:

- Docker build
- S2I Binary
- Source S2I

For more details about each of these build strategies, see *Chapter 1. OpenShift build strategies and Quarkus* of the *Deploying your Quarkus applications to OpenShift* guide.

NOTE

The OpenShift Docker build strategy is the preferred build strategy that supports Quarkus applications targeted for JVM as well as Quarkus applications compiled to native executables. You can configure the deployment strategy using the **quarkus.openshift.build-strategy** property.

CHAPTER 3. EXAMPLES

The quickstart examples listed in the following table can be cloned or downloaded from the Camel Quarkus Examples Git repository.

Number of Examples: 2

Example	Description
File consumer with Bindy and FTP	Shows how to consume CSV files, marshal & unmarshal the data and send it onwards via FTP
Kafka example	Shows how to produce and consume messages in a Kafka topic, using Strimzi Operator

3.1. GETTING STARTED WITH THE FILE CONSUMER QUICKSTART EXAMPLE

You can download or clone the quickstarts from the Camel Quarkus Examples Git repository. The example is in the **file-bindy-ftp** directory.

Extract the contents of the zip file or clone the repository to a local folder, for example a new folder named **quickstarts**.

You can run this example in development mode on your local machine from the command line. Using development mode, you can iterate quickly on integrations in development and get fast feedback on your code. Please refer to the Development mode section of the Camel Quarkus User guide for more details.

NOTE

If you need to configure container resource limits or enable the Quarkus Kubernetes client to trust self signed certificates, you can find these configuration options in the **src/main/resources/application.properties** file.

Prerequisites

- You have **cluster admin** access to the OpenShift cluster.
- You have access to an SFTP server and you have set the server properties (which are prefixed by **ftp**) in the application properties configuration file: **src/main/resources/application.properties**.

Procedure

1. Use Maven to build the example application in development mode:

```
$ cd quickstarts/file-bindy-ftp
$ mvn clean compile quarkus:dev
```

The application triggers the timer component every 10 seconds, generates some random "books" data and creates a CSV file in a temporary directory with 100 entries. The following message is displayed in the console:

```
[route1] (Camel (camel-1) thread #3 - timer://generateBooks) Generating randomized books CSV data
```

Next, the CSV file is read by a file consumer and Bindy is used to marshal the individual data rows into Book objects:

```
[route2] (Camel (camel-1) thread #1 - file:///tmp/books) Reading books CSV data from 89A0EE24CB03A69-0000000000000000
```

Next the collection of Book objects is split into individual items and is aggregated based on the genre property:

```
[route3] (Camel (camel-1) thread #0 - AggregateTimeoutChecker) Processed 34 books for genre 'Action'
[route3] (Camel (camel-1) thread #0 - AggregateTimeoutChecker) Processed 31 books for genre 'Crime'
[route3] (Camel (camel-1) thread #0 - AggregateTimeoutChecker) Processed 35 books for genre 'Horror'
```

Finally, the aggregated book collections are unmarshalled back to CSV format and uploaded to the test FTP server.

```
[route4] (Camel (camel-1) thread #2 - seda://processed) Uploaded books-Action-89A0EE24CB03A69-0000000000000069.csv
[route4] (Camel (camel-1) thread #2 - seda://processed) Uploaded books-Crime-89A0EE24CB03A69-0000000000000069.csv
[route4] (Camel (camel-1) thread #2 - seda://processed) Uploaded books-Horror-89A0EE24CB03A69-0000000000000069.csv
```

2. To run the application in JVM mode, enter the following commands:

```
$ mvn clean package -DskipTests
$ java -jar target/*-runner.jar
```

3. You can build and deploy the example application to OpenShift, by entering the following command:

```
$ mvn clean package -DskipTests -Dquarkus.kubernetes.deploy=true
```

4. Check that the pods are running:

```
$oc get pods

NAME                                      READY   STATUS      RESTARTS   AGE
camel-quarkus-examples-file-bindy-ftp-1-d72mb   1/1     Running     0          5m15s
ssh-server-deployment-5f6f685658-jtr9n          1/1     Running     0          5m28s
```

5. Optional: Enter the following command to monitor the application log:

```
oc logs -f camel-quarkus-examples-file-bindy-ftp-5d48f4d85c-sjl8k
```

Additional resources

- Developing Applications with Camel Extensions for Quarkus
- Camel Quarkus User guide