Red Hat build of Quarkus 1.3

Testing your Quarkus applications
Abstract

This guide describes how you can test your Quarkus Getting Started application.
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PREFACE</td>
<td>3</td>
</tr>
<tr>
<td>CHAPTER 1. VERIFY TEST DEPENDENCIES</td>
<td>4</td>
</tr>
<tr>
<td>CHAPTER 2. SPECIFYING THE TEST PORT</td>
<td>6</td>
</tr>
<tr>
<td>CHAPTER 3. INJECTING A URL INTO A TEST</td>
<td>7</td>
</tr>
<tr>
<td>CHAPTER 4. INJECTION OF CDI BEANS INTO TESTS</td>
<td>9</td>
</tr>
<tr>
<td>CHAPTER 5. APPLYING INTERCEPTORS TO TESTS</td>
<td>10</td>
</tr>
<tr>
<td>CHAPTER 6. MOCKING CDI BEANS</td>
<td>12</td>
</tr>
<tr>
<td>CHAPTER 7. ADDITIONAL RESOURCES</td>
<td>15</td>
</tr>
</tbody>
</table>
As an application developer, you can use Red Hat build of Quarkus to create microservices-based applications written in Java that run in serverless and OpenShift environments. These applications have small memory footprints and fast start-up times.

This guide shows you how to use Apache Maven to test the Quarkus Getting Started project in JVM mode and how to inject resources into your tests. You will expand the test that you created in Getting started with Quarkus.

Prerequisites

- OpenJDK (JDK) 11 is installed and the JAVA_HOME environment variable specifies the location of the Java SDK. Red Hat build of Open JDK is available from the Software Downloads page in the Red Hat Customer Portal (login required).

- Apache Maven 3.6.2 or higher is installed. Maven is available from the Apache Maven Project website.

- A completed Quarkus Getting Started project is available.
  - For instructions on building the Quarkus Getting Started project, see Getting started with Quarkus.
  - For a completed example of a Quarkus Maven project to use in this tutorial, download the Quarkus quickstart archive or clone the Quarkus Quickstarts Git repository. The example is in the getting-started directory.
CHAPTER 1. VERIFY TEST DEPENDENCIES

For this tutorial, you must have a completed Quarkus Getting Started project and the project pom.xml file must include the quarkus-junit5 and rest-assured dependencies. These dependencies will be present if you completed the Quarkus Getting Started exercise or if you downloaded the completed example.

- The quarkus-junit5 dependency is required for testing because it provides the @QuarkusTest annotation that controls the testing framework.

- The rest-assured dependency is not required but you can use it as a convenient way to test HTTP endpoints.

NOTE
Quarkus provides integration that automatically sets the correct URL, so no configuration is required.

Procedure

1. Open the Getting Started project pom.xml file.

2. Verify that the following dependencies are in the file and add them if necessary:

```xml
<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>
```

3. Verify that your pom.xml file includes the maven-surefire-plugin. Because this tutorial uses the JUnit 5 framework, the version of the maven-surefire-plugin must be set because the default version does not support JUnit 5:

```xml
<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>
```

4. Set the java.util.logging.manager system property to use the correct log manager for test.

5. Verify that the GreetingResourceTest.java file contains the following content and add it if necessary:
To run the test, enter the following command:

```
./mvnw clean verify
```

You can also run the test directly from your IDE.

**NOTE**

This test uses HTTP to directly test the REST endpoint. When the test is triggered, the application will start before the test runs.
CHAPTER 2. SPECIFYING THE TEST PORT

By default, Quarkus tests run on port 8081 to avoid conflict with the running application. This allows you to run tests while the application is running in parallel.

Procedure

- To specify the port used when you are testing your project, configure the `quarkus.http.test-port` property in the project `application.properties` file, where `<PORT>` is the port that you want to test on:

  
quarkus.http.test-port=<PORT>

**NOTE**

Quarkus provides RestAssured integration that updates the default port used by RestAssured before the tests are run, so no additional configuration is required.
CHAPTER 3. INJECTING A URL INTO A TEST

If you want to use a different client, use the Quarkus `@TestHTTPResource` annotation to directly inject the URL of the application to be tested 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. In this exercise, you will write a simple test that loads static resources.

Procedure

1. Create the `src/main/resources/META-INF/resources/index.html` file with the following content:

   ```html
   <html>
   <head>
     <title>Testing Guide</title>
   </head>
   <body>
     Information about testing
   </body>
   </html>
   ```

2. Create the `StaticContentTest.java` file with the following content to test that `index.html` is being served correctly:

   ```java
   package org.acme.quickstart;

   import java.io.ByteArrayOutputStream;
   import java.io.IOException;
   import java.io.InputStream;
   import java.net.URL;
   import java.nio.charset.StandardCharsets;
   import org.junit.jupiter.api.Assertions;
   import org.junit.jupiter.api.Test;
   import io.quarkus.test.common.http.TestHTTPResource;
   import io.quarkus.test.junit.QuarkusTest;

   @QuarkusTest
   public class StaticContentTest {

     @TestHTTPResource("index.html")
     URL url;

     @Test
     public void testIndexHtml() throws Exception {
       try (InputStream in = url.openStream()) {
         String contents = readStream(in);
         Assertions.assertTrue(contents.contains("<title>Testing Guide</title>"));
       }
     }

     private static String readStream(InputStream in) throws IOException {
       byte[] data = new byte[1024];
       int r;
     }
   }
   ```
The `@TestHTTPResource` annotation enables you to directly inject the URL of the Quarkus instance. The value of the annotation is the path component of the URL.
CHAPTER 4. INJECTION OF CDI BEANS INTO TESTS

You can perform unit testing and test CDI beans directly. Quarkus enables you to inject CDI beans into your tests through the `@Inject` annotation. In fact, tests in Quarkus are full CDI beans so you can use the complete CDI functionality.

**NOTE**

It is not possible to use injection with native tests.

**Procedure**

- Create the `GreetingServiceTest.java` file with the following content:

```java
package org.acme.quickstart;

import javax.inject.Inject;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
import io.quarkus.test.junit.QuarkusTest;

@QuarkusTest
public class GreetingServiceTest {

    @Inject
    GreetingService service;

    @Test
    public void testGreetingService() {
        Assertions.assertEquals("hello Quarkus", service.greeting("Quarkus"));
    }
}
```

The `GreetingService` bean will be injected into the test.
CHAPTER 5. APPLYING INTERCEPTORS TO TESTS

Quarkus tests are full CDI beans, so you can apply CDI interceptors as you would normally. For example, if you want a test method to run within the context of a transaction, you can apply the @Transactional annotation to the method. You can also create your own test stereotypes.

Procedure

1. Add the quarkus-narayana-jta dependency to your pom.xml file:

   ```xml
   <dependency>
   <groupId>io.quarkus</groupId>
   <artifactId>quarkus-narayana-jta</artifactId>
   </dependency>
   ```

2. Make sure the TransactionalQuarkusTest.java includes the following import statements:

   ```java
   package org.acme.quickstart;
   import java.lang.annotation.ElementType;
   import java.lang.annotation.Retention;
   import java.lang.annotation.RetentionPolicy;
   import java.lang.annotation.Target;
   import javax.enterprise.inject.Stereotype;
   import javax.transaction.Transactional;
   import io.quarkus.test.junit.QuarkusTest;
   ```

3. Create the @TransactionalQuarkusTest annotation:

   ```java
   @QuarkusTest
   @Stereotype
   @Transactional
   @Retention(RetentionPolicy.RUNTIME)
   @Target(ElementType.TYPE)
   public @interface TransactionalQuarkusTest {
   }
   ```

4. Apply this annotation to a test class where it will behave as if you applied both the @QuarkusTest and @Transactional annotations:

   ```java
   @TransactionalQuarkusTest
   public class TestStereotypeTestCase {

   @Inject
   UserTransaction userTransaction;

   @Test
   public void testUserTransaction() throws Exception {
       Assertions.assertEquals(Status.STATUS_ACTIVE, userTransaction.getStatus());
   }
   }
   ```
This is a simple test that evaluates the greeting service directly without using HTTP.
CHAPTER 6. MOCKING CDI BEANS

Quarkus allows you to mock certain CDI beans for specific tests.

You can mock an object using one of the following methods:

- Override the bean you that you want to mock with a class in the src/test/java directory, and put the @Alternative and @Priority(1) annotations on the bean.

- Use the io.quarkus.test.Mock stereotype annotation. The @Mock annotation contains the @Alternative, @Priority(1) and @Dependent annotations.

The following procedure shows how to mock an external service using the @Alternative annotation.

Procedure

1. Create the ExternalService in the src/main/java directory similar to the following example:

```java
package org.acme.quickstart;

import javax.enterprise.context.ApplicationScoped;

@ApplicationScoped
public class ExternalService {
    public String service() {
        return "external";
    }
}
```

2. Create a class UsesExternalService that uses ExternalService in the src/main/java directory:

```java
package org.acme.quickstart;

import javax.enterprise.context.ApplicationScoped;
import javax.inject.Inject;

@ApplicationScoped
public class UsesExternalService {
    @Inject
    ExternalService externalService;

    public String doSomething() {
        return externalService.service();
    }
}
```

3. Create a test in the src/test/java directory similar to the following example:

```java
package org.acme.quickstart;

import javax.inject.Inject;
```
4. Create the `MockExternalService` in the `src/test/java` that uses the `@Alternative` annotation:

```java
package org.acme.quickstart;

import javax.annotation.Priority;
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Alternative;

@Alternative
@Priority(1)
@ApplicationScoped
public class MockExternalService extends ExternalService {

    @Override
    public String service() {
        return "mock";
    }
}
```

The `MockExternalService` is injected wherever the `ExternalService` is being used. In this example, `MockExternalService` will be used in `UsesExternalService`.  

---

1. The `MockExternalService` is injected wherever the `ExternalService` is being used. In this example, `MockExternalService` will be used in `UsesExternalService`.  

---

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

@QuarkusTest
class UsesExternalServiceTest {

    @Inject
    UsesExternalService usesExternalService;

    @Test
    public void testDoSomething() {
        Assertions.assertEquals("external", usesExternalService.doSomething());
    }
}
```
NOTE

You can use the @Mock annotation instead of the @Alternative,@Priority(1) and @Dependent annotations.

The following example shows how to create MockExternalService class that uses the @Mock annotation:

```java
import javax.enterprise.context.ApplicationScoped;
import io.quarkus.test.Mock;

@Mock
@ApplicationScoped
public class MockExternalService extends ExternalService {
    @Override
    public String service() {
        return "mock";
    }
}
```

5. Change the asserted string from "external" to "mock" in the test:

```java
package org.acme.quickstart;
import javax.inject.Inject;
import io.quarkus.test.junit.QuarkusTest;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

@QuarkusTest
class UsesExternalServiceTest {
    @Inject
    UsesExternalService usesExternalService;

    @Test
    public void testDoSomething() {
        Assertions.assertEquals("mock", usesExternalService.doSomething());
    }
}
```
CHAPTER 7. ADDITIONAL RESOURCES

- For information about creating Quarkus applications with Maven, see *Developing and compiling your Quarkus applications with Apache Maven*.

- 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 20:09:53 UTC*