Red Hat build of Quarkus 1.7

Deploying your Quarkus applications on Red Hat OpenShift Container Platform

Last Updated: 2021-04-19

Legal Notice

Abstract

This guide describes how to deploy a Quarkus application on Red Hat OpenShift Container Platform.

Table of Contents

PREFACE

As an application developer, you can deploy a Quarkus application on Red Hat OpenShift Container Platform using Apache Maven and the **quarkus-openshift** extension in a single build command or you can use the traditional source-to-image (S2I) method. The resulting image from either method is fully supported. See the Development Support Scope of Coverage page for the build process and tools that are covered under development support.

You can deploy a Quarkus application compiled to a native executable using OpenShift Serverless Knative Serving and scale services up or down. Scaling down services can improve memory capabilities.

Prerequisites

- OpenJDK 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 3.6.3 or higher is installed. Maven is available from the Apache Maven Project website.
- You have a Quarkus Maven project. For instructions on building a simple Quarkus application with Maven, see *Getting started with Quarkus*.

NOTE

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

- You have access to a Red Hat OpenShift Container Platform cluster and the latest version of the OpenShift CLI (oc) is installed. For information about installing oc, see the "Installing the CLI" section of the *Installing and configuring OpenShift Container Platform clusters* guide.

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. 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 startup times.

CHAPTER 2. RED HAT OPENSHIFT CONTAINER PLATFORM

Red Hat OpenShift Container Platform is a Kubernetes-based platform for developing and running containerized applications. It is designed to enable applications and the data centers that support them to expand from just a few systems and applications to thousands of systems that serve millions of clients.

OpenShift supports two workflows for building container images for applications: the source and the binary workflows. Both workflows are based on the source-to-image (S2I) feature and both workflows rely on S2I builds using the source workflow. The key difference is that the source workflow generates deployable artifacts of your application inside OpenShift, while the binary workflow generates these binary artifacts outside of OpenShift. Both of them build the application container image inside OpenShift. The binary workflow provides an application binary as the source for an OpenShift S2I build that generates a container image.

CHAPTER 3. USING THE QUARKUS OPENSHIFT EXTENSION TO DEPLOY QUARKUS APPLICATIONS ON OPENSHIFT

The traditional source-to-image (S2I) source workflow generates the deployable artifacts of your application inside OpenShift. The Quarkus OpenShift extension uses the S2I binary workflow to provide a more streamlined deployment process. Instead of building from the source, the extension uploads the JAR files from the local file system. As a result, the build process is up to ten times faster than the traditional S2I method. You can use the Quarkus OpenShift extension when developing locally as well as from a build server or continuous integration (CI) system to perform repeatable builds from source.

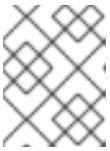
NOTE

Use the Quarkus OpenShift extension for development and testing purposes only. For production environments, consider using the traditional S2I method described in Chapter 4, Using S2I to deploy Quarkus applications on OpenShift.

Procedure

1. Change to the directory that contains your Quarkus Maven project.
2. To add the OpenShift extension to an existing project, enter the following command:

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



NOTE

You can include the **-Dextensions="openshift"** argument to add the Quarkus OpenShift extension when you create a new project.

When you add the OpenShift extension, the script adds the following dependency to the **pom.xml** file:

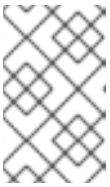
```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-openshift</artifactId>
</dependency>
```

3. If you are using an untrusted certificate, add the following line to the **src/main/resources/application.properties** file:

```
quarkus.kubernetes-client.trust-certs=true
```

4. To direct OpenShift to use the Open JDK 11 Red Hat Enterprise Linux 7 image, add the following line to the **application.properties** file:

```
quarkus.s2i.base-jvm-image=registry.access.redhat.com/ubi8/openjdk-11
```



NOTE

If you are deploying on IBM Z infrastructure, add **quarkus.s2i.base-jvm-image=registry.access.redhat.com/openj9/openj9-11-rhel8** to the **application.properties** file.

5. To create an OpenShift route, add the following line to the **application.properties** file:

```
quarkus.openshift.expose=true
```

6. Save the changes to the **application.properties** file.

7. Log in to the OpenShift CLI (oc):

```
oc login
```

8. To create a new OpenShift project, enter the following command where **<project_name>** is the name of your new project:

```
oc new-project <project_name>
```

9. To deploy your project to OpenShift, enter the following command in your Quarkus Maven project directory:

```
./mvnw clean package -Dquarkus.kubernetes.deploy=true
```

10. To display the names and routes of all deployed applications in the OpenShift project, enter the following command:

```
oc get route
```

11. To view the full URL to the application, where **<application_name>** is the name of an application deployed in your OpenShift project, enter the following command:

```
export URL="http://$(oc get route <application_name> -o jsonpath='{.spec.host}')"
echo "Application URL: $URL"
curl $URL/hello
```

12. To create an HTTP request on the route's **hello** endpoint, enter the following command:

```
curl $URL/hello
```

CHAPTER 4. USING S2I TO DEPLOY QUARKUS APPLICATIONS ON OPENSHIFT

The traditional source-to-image (S2I) method is widely used as the preferred method for deploying applications on Red Hat OpenShift Container Platform. With S2I, you must provide the source code to the build container either through a Git repository or by uploading the source at build time. Use this method to deploy your Quarkus applications in production environments.

Prerequisites

- You have a Quarkus Maven project hosted in a Git repository.

Procedure

1. Change to the directory that contains your Quarkus Maven project.
2. Create a hidden directory called **.s2i** at the same level as the **pom.xml** file.
3. Create a file called **environment** in the **.s2i** directory and add the following content:

```
ARTIFACT_COPY_ARGS=-p -r lib/ *-runner.jar
```

4. Commit and push your changes to the remote Git repository.
5. Log in to the OpenShift CLI (oc):

```
oc login
```

6. To create a new OpenShift project, enter the following command where **<project_name>** is the name of your new project:

```
oc new-project <project_name>
```

7. To import the supported OpenShift image, enter the following command:

```
oc import-image --confirm ubi8/openjdk-11 --from=registry.access.redhat.com/ubi8/openjdk-11
```

NOTE

If you are deploying on IBM Z infrastructure, enter **oc import-image --confirm openj9/openj9-11-rhel8 --from=registry.redhat.io/openj9/openj9-11-rhel8**.

For information about this image, see the Red Hat OpenJ9 11 Java Applications on RHEL8 page.

8. To build the project in OpenShift, enter the following command where **<git_path>** is the path to the Git repository that hosts your Quarkus project and **<project_name>** is the OpenShift project that you created.

```
oc new-app ubi8/openjdk-11 <git_path> --name=<project_name>
```

NOTE

If you are deploying on IBM Z infrastructure, enter **oc new-app openj9/openj9-11-rhel8 <git_path> --name=<project_name>**.

This command builds the project, creates the application, and deploys the OpenShift service.

9. To create an OpenShift route, enter the following command:

```
oc expose service/<project_name>
```

10. To view the new route, enter the following command where <application_name> is the name of an application deployed in your OpenShift project:

```
export URL="http://$(oc get route <application_name> -o jsonpath='{.spec.host}}')"
echo "Application URL: $URL"
```

11. To create an HTTP request on the route's **hello** endpoint, enter the following command:

```
curl $URL/hello
```

12. To use your application, enter the URL returned in the preceding command in a web browser.

13. To deploy an updated version of the project, push any updates to the Git repository then enter the following command:

```
oc start-build <project_name>
```

14. Refresh your browser page after the build completes to see the changes.

CHAPTER 5. DEPLOYING A QUARKUS APPLICATION COMPILED TO A NATIVE EXECUTABLE AS A OPENSHIFT SERVERLESS SERVICE

As an application developer, you can deploy a Quarkus application compiled to a native executable on Red Hat OpenShift Container Platform using OpenShift Serverless Knative Serving.

By using OpenShift Serverless Knative Serving, you can scale services up and down depending on the load size. Scaling down services that are currently not requested improves memory capabilities.

NOTE

You can run Quarkus as a native executable or as a Java application using OpenJDK. For native executables, use the Red Hat UBI 8 minimal image. For OpenJDK, use the Red Hat 8 UBI Java image.

5.1. DEPLOYING A CONTAINER IMAGE FOR A QUARKUS NATIVE APPLICATION IN A CONTINUOUS INTEGRATION AS A SERVERLESS APPLICATION

You can separate the native build, container build, and deployment steps when deploying a native serverless application. The following procedure demonstrates how to deploy a container image for a Quarkus native application in a continuous integration (CI) as a serverless application.

Prerequisites

- OpenShift Serverless operator is installed.
- OpenShift Knative Serving is installed.
- For native compilation, a container environment like Podman or Docker is required.
- The **kn** CLI tool is installed.

Procedure

1. Change to the directory that contains your Quarkus project.
2. Build a Linux executable using one of the following methods:
   a. For Docker use:

```
./mvnw package -Pnative -Dquarkus.native.container-build=true -
Dquarkus.native.builder-image=registry.access.redhat.com/quarkus/mandrel-20-
rhel8:20.3
```

   b. For Podman use:

```
./mvnw package -Pnative -Dquarkus.native.container-build=true -
Dquarkus.native.container-runtime=podman -Dquarkus.native.builder-
image=registry.access.redhat.com/quarkus/mandrel-20-rhel8:20.3
```

3. Open the **src/main/docker/Dockerfile.native** file and set the **<image_name>** and **<version>** parameters:

   a. For Docker use:

   ```
   docker build -f src/main/docker/Dockerfile.native -t <image_name>:<version> .
   ```

   b. For Podman use:

   ```
   podman build -f src/main/docker/Dockerfile.native -t <image_name>:<version>.
   ```

4. Push the container to a repository that your CI environment and your OpenShift environment can access, where **<registry>** is your registry URL:

   a. For Docker use:

   ```
   docker tag <image_name>:<version> <registry>/<image_name>:<version>
   docker push <registry>/<image_name>:<version>
   ```

   b. For Podman use:

   ```
   podman tag <image_name>:<version> <registry>/<image_name>:<version>
   podman push <registry>/<image_name>:<version>
   ```

5. Log in to the OpenShift CLI (oc):

   ```
   oc login
   ```

6. To create a new OpenShift project, enter the following command where **<project_name>** is the name of your new project:

   ```
   oc new-project <project_name>
   ```

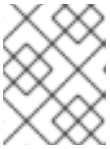
7. To deploy your container as a serverless application using the OpenShift Serverless CLI (kn), enter the following command where **<service_name>** is the name for your service:

   ```
   kn service create <service_name> --image REPOSITORY/<image_name>:<version>
   ```

8. To verify that the service is ready, enter the following command.

   ```
   kn service list <service_name>
   ```

   The output in the column called "READY" reads **true** if the service is ready.



NOTE

The **kn service** command returns **true** when the necessary components are created, not when the image is pulled down and ready.

CHAPTER 6. ADDITIONAL RESOURCES

- For information about creating Quarkus applications with Maven, see *Developing and compiling your Quarkus applications with Apache Maven*.
- For more information on how to compile the Quarkus applications to native executables and native executable testing, see *Compiling your Quarkus applications to native executables*

Revised on 2021-04-19 12:03:20 UTC