

Quarkus - Openshift extension

This guide covers generating and deploying Openshift resources based on sane default and user supplied configuration.

Prerequisites

To complete this guide, you need:

- roughly 5 minutes
- an IDE
- JDK 1.8+ installed with `JAVA_HOME` configured appropriately
- Apache Maven 3.5.3+
- access to an Openshift or cluster (Minishift is a viable options)

Creating the Maven project

First, we need a new project that contains the Openshift extension. This can be done using the following command:

```
mvn io.quarkus:quarkus-maven-plugin:1.3.0.CR1:create \
  -DprojectId=org.acme \
  -DprojectId=openshift-quickstart \
  -DclassName="org.acme.rest.GreetingResource" \
  -Dpath="/greeting" \
  -Dextensions="openshift"

cd openshift-quickstart
```

Openshift

Quarkus offers the ability to automatically generate Openshift resources based on sane default and user supplied configuration. The Openshift extension is actually a wrapper extension that brings together the [kubernetes](#) and [container-image-s2i](#) extensions with sensible defaults so that it's easier for the user to get started with Quarkus on Openshift.

When we added the `openshift` extension to the command line invocation above, the following dependency was added to the `pom.xml`

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

By adding this dependency, we now have the ability to configure the Openshift resource generation and application using the usual `application.properties` approach that Quarkus provides. The configuration items that are available can be found in: `io.quarkus.kubernetes.deployment.OpenshiftConfig` class. Furthermore, the items provided by `io.quarkus.deployment.ApplicationConfig` affect the Openshift resources.

Building

Building is handled by the `container-image-s2i` extension. To trigger a build:

```
mvn clean package -Dquarkus.container-image.build=true
```

The command above will trigger an s2i binary build.

Deploying

To trigger a deployment:

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

The command above will trigger a container image build and will apply the generated Openshift resources, right after. The generated resources are using Openshift's `DeploymentConfig` that is configured to automatically trigger a redeployment when a change in the `ImageStream` is noticed. In other words, any container image build after the initial deployment will automatically trigger redeployment, without the need to delete, update or re-apply the generated resources.

Customizing

All available customization options are available in the [Openshift configuration options](#).

Some examples are provided in the sections below:

Exposing Routes

To expose a `Route` for the Quarkus application:

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

Tip: You don't necessarily need to add this property in the `application.properties`. You can pass it as a command line argument:

```
mvn clean package -Dquarkus.openshift.expose=true
```

The same applies to all properties listed below.

Labels

To add a label in the generated resources:

```
quarkus.openshift.labels.foo=bar
```

Annotations

To add an annotation in the generated resources:

```
quarkus.openshift.annotations.foo=bar
```

Environment variables

To add an annotation in the generated resources:

```
quarkus.openshift.env-vars.my-env-var.value=foobar
```

The command above will add `MY_ENV_VAR=foobar` as an environment variable. Please note that the key `my-env-var` will be converted to uppercase and dashes will be replaced by underscores resulting in `MY_ENV_VAR`.

You may also noticed that in contrast to labels, and annotations for environment variables you don't just use a key=value approach. That is because for environment variables there are additional options rather than just value.

Environment variables from Secret

To add all key value pairs of a `Secret` as environment variables:

```
quarkus.openshift.env-vars.my-env-var.secret=my-secret
```

Environment variables from ConfigMap

To add all key value pairs of a `ConfigMap` as environment variables:

```
quarkus.openshift.env-vars.my-env-var.configmap=my-secret
```

Mounting volumes

The Openshift extension allows the user to configure both volumes and mounts for the application.

Any volume can be mounted with a simple configuration:

```
quarkus.openshift.mounts.my-volume.path=/where/to/mount
```

This will add a mount to my pod for volume `my-volume` to path `/where/to/mount`

The volumes themselves can be configured as shown in the sections below:

Secret volumes

```
quarkus.openshift.secret-volumes.my-volume.secret-name=my-secret
```

ConfigMap volumes

```
quarkus.openshift.config-map-volumes.my-volume.config-map-name=my-secret
```