Quarkus & Kubernetes I

This cheat sheet covers the integrations you can find in the form of extensions between Quarkus and Kubernetes.

CREATING THE PROJECT

```
mvn "io.quarkus:quarkus-maven-plugin:1.4.0.Final:create" \
  -DprojectGroupId="org.acme" \
  -DprojectArtifactId="greeting" \
  -DprojectVersion="1.0-SNAPSHOT" \
  -DclassName="org.acme.GreetingResource" \
  -Dextensions="kubernetes, jib" \
  -Dpath="/hello"
```

Tip You can generate the project in https://code.quarkus.io/ and selecting kubernetes and jib extensions.

NATIVE EXECUTABLE SUPPORT

You can build a native image by using GraalVM. But since Kubernetes works with containers, you need to create the native executable inside a container. Quarkus allows you to do that by running the following command:

```
./mvnw package -Pnative -Dquarkus.native.container-build=true
```

Or using podman:

```
./mvnw package -Pnative -Dquarkus.native.container-runtime=podman -Dquarkus.native.container-build=true
```

CONTAINER IMAGE CREATION

Quarkus comes with default Dockerfiles to build the container. They are found in src/main/docker.

Dockerfile.jvm
: It can be used to create a container containing the generated Java files (runner JAR + lib folder).

Dockerfile.native
: It can be used to create a container containing the generated native executable file.

You can use Docker to create the container image: `docker build -f src/main/docker/Dockerfile.native -t quarkus/getting-started .` or you can leverage to Quarkus the creation and release of the container images. Several extensions are provided to make it so.

Standard properties that can be set as Java system properties or in the src/main/resources/application.properties.

quarkus.container-image.group
: The group/repository of the image, defaults to ${user.name}.

quarkus.container-image.name
: The name of the image, defaults to the application name.

quarkus.container-image.tag
: The tag of the image, defaults to the application version.

quarkus.container-image.registry
: The registry to use for pushing, defaults to docker.io.

quarkus.container-image.username
: The registry username.

quarkus.container-image.password
: The registry password.

quarkus.container-image.insecure
: Flag to allow insecure registries, defaults to false.

quarkus.container-image.build
: Flag to set if the image should be built, defaults to false.

quarkus.container-image.push
: Flag to set if the image should be pushed, defaults to false.

Specific builders:

Jib

You can use Jib to build the container image. Jib builds Docker and OCI images for Java applications in a dockerless fashion.

```
./mvnw quarkus:add-extensions -Dextensions="jib"
```

Specific properties for the Jib extension are:

quarkus.container-image-jib.base-jvm-image
: The base image to use for the Jib build, defaults to fabric8/java-alpine-openjdk8-jre.

quarkus.container-image-jib.base-native-image
: The base image to use for the native build, defaults to registry.access.redhat.com/ubi8/ubi-minimal.

quarkus.container-image-jib.jvm-arguments
: The arguments to pass to Java, defaults to -Dquarkus.http.host=0.0.0.0,-Djava.util.logging.manager=org.jboss.logmanager.LogManager.

quarkus.container-image-jib.native-arguments
: The arguments to pass to the native application, defaults to -Dquarkus.http.host=0.0.0.0.

quarkus.container-image-jib.environment-variables
: Map of environment variables.

Docker

You can use the Docker extension to build the container image using Docker CLI.

```
./mvnw quarkus:add-extensions -Dextensions="docker"
```

Specific properties for the Docker extension are:

quarkus.container-image-docker.dockerfile-jvm-path
: Path to the JVM Dockerfile, defaults to ${project.root}/src/main/docker/Dockerfile.jvm.

quarkus.container-image-docker.dockerfile-native-path
: Path to the native Dockerfile, defaults to ${project.root}/src/main/docker/Dockerfile.native.

S2I

You can use the S2I to build the container image.

```
./mvnw quarkus:add-extensions -Dextensions="s2i"
```

Specific properties for the S2I extension are:

quarkus.container-image-s2i.base-jvm-image
: The base image to use for the s2i build, defaults to fabric8/java-alpine-openjdk8-jre.

quarkus.container-image-s2i.base-native-image
: The base image to use for the native build, defaults to registry.access.redhat.com/ubi8/ubi-minimal.

KUBERNETES

Quarkus use the Dekorate project to generate Kubernetes resources.
Running ./mvnw package the Kubernetes resources are created at target/kubernetes/ directory.
You can choose the target deployment type by setting the quarkus.kubernetes.deployment-target property. Possible values are kubernetes, openshift and knative. The default target is kubernetes.
You can customize the generated resource by setting specific properties in application.properties. Full list of configurable elements are:
https://quarkus.io/guides/kubernetes#configuration-options
src/main/resources/application.properties

```
quarkus.kubernetes.replicas=3
quarkus.kubernetes.readiness-probe.period-seconds=45
quarkus.kubernetes.mounts.github-token.path=/deployment/github
quarkus.kubernetes.mounts.github-token.read-only=true
quarkus.kubernetes.secret-volumes.github-token.volume-name=github-token
quarkus.kubernetes.secret-volumes.github-token.secret-name=greeting-security
quarkus.kubernetes.secret-volumes.github-token.default-mode=420
quarkus.kubernetes.config-map-volumes.github-token.config-map-name=my-secret
quarkus.kubernetes.labels.foo=bar
quarkus.kubernetes.annotations.foo=bar
quarkus.kubernetes.expose=true
```

Moreover, the generated resources are integrated with MicroProfile Health spec, registering liveness/readiness probes based on the health checks defined using the spec.
To deploy the generated resources automatically, you need to set quarkus.container.deploy flag to true.

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

Setting this flag to true, makes the build and push flags from the container-image set to true too.
Kubernetes extension uses the Kubernetes Client to deploy resources. By default, Kubernetes Client reads connection properties from the ~/.kube/config folder but you can set them too by using some of the kubernetes-client properties:

quarkus.kubernetes-client.trust-certs

Trust self-signed certificates, defaults to false.

quarkus.kubernetes-client.master-url

URL of Kubernetes API server.

quarkus.kubernetes-client.namespace

Default namespace.

quarkus.kubernetes-client.ca-cert-file

CA certificate data.

quarkus.kubernetes-client.client-cert-file

Client certificate file.

quarkus.kubernetes-client.client-cert-data

Client certificate data.

quarkus.kubernetes-client.client-key-data

Client key data.

quarkus.kubernetes-client.client-key-algorithm

Client key algorithm.

quarkus.kubernetes-client.username

Username.

quarkus.kubernetes-client.password

Password.

Author Alex Soto
Java Champion, Working at Red Hat