Quarkus & Kubernetes II

This cheat sheet covers more integrations that 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, kubernetes-client, health, kubernetes-config" \
  -Dpath="/hello"
```

Tip You can generate the project in https://code.quarkus.io/ and selecting kubernetes, kubernetes-client, health and kubernetes-config extensions.

KUBERNETES

Quarkus use the Dekorate project to generate Kubernetes resources.
Running ./mvnw package the Kubernetes resources are created at target/kubernetes/ directory.

HEALTH CHECKS

The generated Kubernetes resources are integrated with MicroProfile Health spec, registering liveness/readiness probes based on the health checks defined using the spec.
If the extension is present, a default liveness/readiness probes are registered at /health/live and /health/ready endpoints.
You can implement a custom liveness/readiness probes:

```
import io.smallrye.health.HealthStatus;
@ApplicationScoped
public class DatabaseHealthCheck {
   @Liveness
   HealthCheck isAlive() {
    return HealthStatus.up("successful-live");
   }
   @Readiness
   HealthCheck isReady() {
    return HealthStatus.state("successful-read", this::isServiceReady)
   }
   private boolean isServiceReady() {
      return true;
   }
}
```

Quarkus comes with three health check implementations for checking the service status.

SocketHealthCheck

It checks if the host is reachable using a socket.

UrlHealthCheck

It checks if the host is reachable using a HTTP URL connection.

InetAddressHealthCheck

It checks if host is reachable using InetAddress.isReachable method.

```
@Readiness
HealthCheck isGoogleReady() {
   return new UrlHealthCheck("https://www.google.com").name("Google-Check");
}
```

The following Quarkus extensions agroal (datasource), kafka, mongoDB, neo4j, artemis, kafka-streams and vault provide readiness health checks by default. They can be enabled/disabled by setting quarkus.<component>.health.enabled to true/false.

```
quarkus.kafka-streams.health.enabled=true
quarkus.mongodb.health.enabled=false
```

KUBERNETES CONFIGURATION

Kubernetes Config extension uses Kubernetes API Server to get `config-map`s and inject their key/value using MicroProfile Config spec.
You need to enable the extension and set name of the `config-map`s that contains the properties to inject:

```
quarkus.kubernetes-config.enabled=true
quarkus.kubernetes-config.config-maps=cmap1,cmap2
```

To inject cmap1 and cmap2 values, you need to set the key name in the @ConfigProperty annotation:

```
@ConfigProperty(name = "some.prop1")
String someProp1;
@ConfigProperty(name = "some.prop2")
String someProp2;
```

If the config key is a Quarkus configuration file, application.properties or application.yaml, the content of these files is parsed and each key/value of the configuration file can be injected as well.
List of Kubernetes Config parameters.

quarkus.kubernetes-config.enabled

The application will attempt to look up the configuration from the API server, defaults to false.

quarkus.kubernetes-config.fail-on-missing-config

The application will not start if any of the configured config sources cannot be located, defaults to true.

quarkus.kubernetes-config.config-maps

ConfigMaps to look for in the namespace that the Kubernetes Client has been configured for. Supports CSV format.

KUBERNETES CLIENT

Quarkus integrates with Fabric8 Kubernetes Client to access Kubernetes Server API.

```
@Inject
KubernetesClient client;
ServiceList myServices = client.services().list();
Service myservice = client.services()
      .inNamespace("default")
      .withName("myservice")
      .get();
```

Kubernetes Client can be configured programmatically:

```
@Dependent
public class KubernetesClientProducer {
   @Produces
   public KubernetesClient kubernetesClient() {
      Config config = new ConfigBuilder()
         .withMasterUrl("https://mymaster.com")
         .build();
      return new DefaultKubernetesClient(config);
   }
}
```

Or also in application.properties.
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