Writing a Kubernetes Operator in Java

This cheat sheet covers how to create a Kubernetes Operator in Java using Quarkus.

```
mvn "io.quarkus:quarkus-maven-plugin:1.4.0.Final:create" \
      -DprojectGroupId="org.acme" \
      -DprojectArtifactId="pizza-operator" \
      -DprojectVersion="1.0-SNAPSHOT" \
      -Dextensions="kubernetes, kubernetes-client" \
```

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

DEFINING THE CRD

First, you need to create a CRD de ning the custom resource:

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
      name: pizzas.mykubernetes.acme.org
      labels:
            app: pizzamaker
            mylabel: stuff
spec:
      group: mykubernetes.acme.org
      scope: Namespaced
      version: v1beta2
      names:
            kind: Pizza
            listKind: PizzaList
            plural: pizzas
            singular: pizza
            shortNames:
            - pz
```

An example of a pizza resource:

```
apiVersion: mykubernetes.acme.org/v1beta2
kind: Pizza
metadata:
      name: alexmeats
spec:
      toppings:
      - mozzarella
      - pepperoni
      - sausage
      - bacon
      sauce: extra
```

DEFINING THE JAVA CODE

Parsing of the pizza resource

You need to create a parser for reading the content of pizza resource.

```
@JsonDeserialize
public class PizzaResourceSpec {
      @JsonProperty("toppings")
      private List<String> toppings = new ArrayList<>();
      @JsonProperty("sauce")
      private String sauce;
      // getters/setters
}
@JsonDeserialize
public class PizzaResourceStatus {}
@JsonDeserialize
public class PizzaResource extends CustomResource {
      private PizzaResourceSpec spec;
      private PizzaResourceStatus status;
      // getters/setters
}
@JsonSerialize
public class PizzaResourceList extends
CustomResourceList<PizzaResource> {}

public class PizzaResourceDoneable extends
      CustomResourceDoneable<PizzaResource> {
public PizzaResourceDoneable(PizzaResource resource,
      Function<PizzaResource, PizzaResource>
      function)
      { super(resource, function);}
}
```

Registering the CRD in Kubernetes Client

```
public class KubernetesClientProducer {

@Produces
@Singleton
@Named("namespace")
String findMyCurrentNamespace() throws
IOException {
      return new
      String(Files.readAllBytes(Paths.get("/
      var/run/secrets/kubernetes.io/
      serviceaccount/namespace")));
}
@Produces
@Singleton
KubernetesClient
makeDefaultClient(@Named("namespace") String
      namespace) {
return new
DefaultKubernetesClient().inNamespace(namespace);
}
```

```
@Produces
@Singleton
MixedOperation<PizzaResource, PizzaResourceList,
PizzaResourceDoneable, Resource<PizzaResource,
PizzaResourceDoneable>>

makeCustomHelloResourceClient(KubernetesClient
defaultClient) {

KubernetesDeserializer.registerCustomKind("mykuberne
      tes.acme.org/v1beta2", "Pizza",
      PizzaResource.class);
CustomResourceDefinition crd =
      defaultClient.customResourceDefinitions().
      list().getItems().stream().findFirst()
            .orElseThrow(RuntimeException::new);
return defaultClient.customResources(crd,
      PizzaResource.class, PizzaResourceList.class,
            PizzaResourceDoneable.class);
    }
}
```

Implement the Operator

Operator is the logic that is executed when the custom resource (pizza) is applied. In this case, a pod is instantiated with pizza-maker image.

```
public class PizzaResourceWatcher {

@Inject
KubernetesClient defaultClient;

@Inject
MixedOperation<PizzaResource, PizzaResourceList,
PizzaResourceDoneable, Resource<PizzaResource,
PizzaResourceDoneable>> crClient;

void onStartup(@Observes StartupEvent event) {

   crClient.watch(new Watcher<PizzaResource>() {
      @Override
      public void eventReceived(Action action,
      PizzaResource resource) {
         if (action == Action.ADDED) {
            final String app = resource.getMetadata()
            .getName();
            final String sauce = resource.getSpec()
            .getSauce();
            final List<Stri ng> toppings =
            resource.getSpec().getToppings();

            final Map<String, String> labels = new
            HashMap<>();labels.put("app", app);

            final ObjectMetaBuilder objectMetaBuilder =
            new ObjectMetaBuilder().withName(app + "-
            pod")
              .withNamespace(resource.getMetadata()
              .getNamespace()).withLabels(labels);

            final ContainerBuilder containerBuilder =
            new ContainerBuilder().withName("pizza-
            maker")
              .withImage("quay.io/lordofthejars/
              pizza-maker:1.0.0").withCommand("/work/
              application")
              .withArgs("--sauce=" + sauce, "--
              toppings=" + String.join(",",toppings));

            final PodSpecBuilder podSpecBuilder = new
            PodSpecBuilder().withContainers
            (containerBuilder.build())
              .withRestartPolicy("Never");

            final PodBuilder podBuilder = new
            PodBuilder().withMetadata
            (objectMetaBuilder.build())
              .withSpec(podSpecBuilder.build());

            final Pod pod = podBuilder.build();
            defaultClient.resource(pod)
            .createOrReplace();
          }
        }

        @Override
        public void onClose(KubernetesClientException e)
        {
        }
     });
   }
}
```

Deploy Operator

You need to package and create a container with all the operator code and deploy it to the cluster.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: quarkus-operator-example
rules:
- apiGroups:
  - ''
  resources:
  - pods
  verbs:
  - get
  - list
  - watch
  - create
  - update
  - delete
  - patch
- apiGroups:
  - apiextensions.k8s.io
  resources:
  - customresourcedefinitions
  verbs:
  - list
```

```
    - watch
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: quarkus-operator-example
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: quarkus-operator-example
subjects:
- kind: ServiceAccount
  name: quarkus-operator-example
  namespace: default
roleRef:
  kind: ClusterRole
  name: quarkus-operator-example
  apiGroup: rbac.authorization.k8s.io
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: quarkus-operator-example
spec:
  selector:
    matchLabels:
      app: quarkus-operator-example
  replicas: 1
  template:
    metadata:
      labels:
        app: quarkus-operator-example
    spec:
      serviceAccountName: quarkus-operator-example
      containers:
      - image: quay.io/lordofthejars/pizza-operator:1.0.0
        name: quarkus-operator-example
        imagePullPolicy: IfNotPresent
```

Run the `kubectl apply -f pizza-crd.yaml` command to register the CRD in the cluster. Run the `kubectl apply -f deploy.yaml` command to register the operator.

Running the example

Apply the custom resource by running: `kubectl apply -f meat-pizza.yaml` and check the output of `kubectl get pods` command.

Author Alex Soto
Java Champion, Working at Red Hat