a8s Applicant Homework: Write and Deploy a Custom Kubernetes Controller

The goal of this homework is to use the Operator SDK to write a small Kubernetes Custom Controller in Go and then deploy it on a Kubernetes cluster.

To complete the homework you can freely use any resource you want. That includes Google, StackOverflow, any mailing list/chat (e.g. Slack) that you think could help you, etc... . However, you MUST do the homework on your own without any assitance from friends, relatives etc... .

As part of the homework, you'll have to write code and other artifacts. Everything you produce should be committed in this repo.

Step 1 (Optional): Familiarize yourself with the Concepts and Tools

This step has no deliverable and is here only to help you. Feel free to skip it.

To start with, you should get an understanding of what a Kubernetes controller is and the basics of writing one with the Operator SDK and Go. Follow some resources, both theoretical and practical, that should help you. Obviously, you can also do your own research and consult resources that are not listed here if that helps you.

- Kubernetes API Objects
- Kubernetes Controllers
- Kubernetes Operators
- Operator SDK installation
- Operator SDK tutorial

Step 2: Define a Custom Resource and a Controller that Logs the Spec of the Custom Resource

Use the Operator SDK to:

1. Create a new custom API type called "Dummy".
2. Create a new custom controller for resources of kind "Dummy".

"Dummy" resources should have no `status` field and a `spec` field that has only one string subfield called "message". Follows an example:

```yaml
apiVersion: interview.com/v1alpha1
kind: Dummy
metadata:
  name: dummy1
  namespace: default
spec:
  message: "I'm just a dummy"
```

The custom controller must process each Dummy API object simply by logging its name, namespace and the value of `spec.message`.

Step 3: Give Dummies a Status and Make the Custom Controller Write to it

Extend the Dummy custom API type by giving to it a `status` field which contains only a string subfield called `specEcho`.

The custom controller should not only log each Dummy name, namespace and `spec.message`, it should now also copy the value of `spec.message` into `status.specEcho`.

For example, after a Dummy has been processed by the custom controller it should look like:

```yaml
apiVersion: interview.com/v1alpha1
kind: Dummy
metadata:
  name: dummy1
  namespace: default
spec:
  message: "I'm just a dummy"
status:
  specEcho: "I'm just a dummy"
```

Step 4: Associate a Pod to each Dummy API object

Extend the custom controller to also make it create a Pod for each Dummy API object that is created. When a Dummy is deleted its Pod should also cease to exist (either immediately or shortly afterwards). The Pod should run nginx.

Futhermore, the Dummy custom API type should be extended by giving to it a status field that tracks the status of the Pod associated to the Dummy.

For example, a Dummy should initially start as:

```yaml
apiVersion: interview.com/v1alpha1
kind: Dummy
metadata:
  name: dummy1
  namespace: default
spec:
  message: "I'm just a dummy"
status:
  specEcho: "I'm just a dummy"
  podStatus: "Pending"
```

But once its Pod is up and running it should become:

```yaml
apiVersion: interview.com/v1alpha1
kind: Dummy
metadata:
  name: dummy1
  namespace: default
spec:
  message: "I'm just a dummy"
status:
  specEcho: "I'm just a dummy"
  podStatus: "Running"
```

Step 5: Run the Custom Controller on Kubernetes

Deploy the custom controller on a Kubernetes cluster (for example, one created with Minikube) and test it. Also write a markdown file for the reviewer of your homework with instructions on how to do

the same. Notice that you will need to push the container image of your custom controller to a publicly accessible container registry (for example on Docker Hub), so that the reviewer can test your solution.

We organized the homework in incremental steps to help you do one thing at a time, but you're not constrained to follow the steps we described: you can do the homework in any order/way you like. In fact, we only care about the final result rather than the result after each intermediate step.

Good luck!