# Example 2:

## Description of the Go operator:

"The *Operator monitors and handles (add remove duplicates of) Pods running in Kubernetes*". With the SDK, developers create custom resource definitions (CRDs) which are then, using an appropriate Dockerfile, turned into spawnable container images and, using the appropriate YAML description, registered with the platform. In the following, we introduce a sample annotation listener operator which triggers an action any time a developer updates an annotation. As a prerequisite, we show what annotations can be used for in cloud-native applications hosted on Kubernetes.

## BEFORE STARTING:

**git clone https://github.com/appuio/operator-sdk-examples**

**Other Prerequisites**

- dep version v0.5.0+.
- git
- go version v1.10+.
- docker version 17.03+.
- kubectl version v1.11.0+.

- Access to a kubernetes v.1.11.0+ cluster.

- and Install also, to have a quick try of it, Minishift: https://github.com/minishift/minishift

- Set $GOPATH

**Note:** This guide uses minikube version v0.25.0+ as the local kubernetes cluster and quay.io for the public registry.

*For this first example of operator we provide a short and fast way to deploy it:*

**[SHORT VERSION]:** few lines and you will deploy the operator

**[LONG VERSION]:** full steps to deploy the operator

## [SHORT VERSION]: few steps to deploy the operator

**Description:**

**(Operator based on operator-sdk APIs that monitor and handle (add remove duplicates of) Pods running in Kubernetes is available at the following link: operator-handling-pods-replicates.**

**Clone the operator from https://github.com/appuio/operator-sdk-examples**

```
$ git clone https://github.com/appuio/operator-sdk-examples
#MOVE THE FOLDER memcached-operator-long under "the GOPATH"
$ cd memcached-operator-long
```

**# Push the app-operator directly from DOCKER:**

### Build the memcached-operator image and push it to a registry with Docker:

```
$ sudo docker login
$ operator-sdk build  <docker id>/memcached-operator-long:v.0.0.1
#(e.g., operator-sdk build docker.io/spanichella/memcached-operator-long)
$ sed -i "" 's|REPLACE_IMAGE|docker.io/<docker id>/memcached-operator-long|g' deploy/operator.yaml.
 (e.g., sed -i "" 's|REPLACE_IMAGE|docker.io/spanichella/memcached-operator-long|g' deploy/operator.yaml)
#If you created your operator using --cluster-scoped=true, update the service account namespace in the
generated ClusterRoleBinding to match where you are deploying your operator.
#check namespaces with
$ kubectl get namespaces
#(or with $ oc get project)
#then set the correct namespace (in my case it was "blogpost-project")
$ export OPERATOR_NAMESPACE=blogpost-project
$ sed -i "" 's|REPLACE_NAMESPACE|blogpost-project|g' deploy/role_binding.yaml
#check if the previous line worked correctly, otherwise set it manually...
# push it to a registry with Docker:
$ docker push   <docker id>/memcached-operator-long:v.0.0.1
#(e.g., docker push docker.io/spanichella/memcached-operator-long)
```

### "Before running the operator, the CRD must be registered with the Kubernetes apiserver":

```
$ kubectl create -f deploy/crds/cache_v1alpha1_memcached_crd.yaml
 OR
$ oc create -f deploy/crds/cache_v1alpha1_memcached_crd.yaml
```

**Setup RBAC and deploy the memcached-operator:**

**(instead of "kubectl" you can also use "*oc*" command instead), thus, to register the CRD:**

```
$ kubectl create -f deploy/service_account.yaml
$ kubectl create -f deploy/role.yaml
$ kubectl create -f deploy/role_binding.yaml
$ kubectl create -f deploy/operator.yaml
```

### Verify that the memcached-operator is up and running:

```
$ kubectl get deployment
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
memcached-operator-long 1 1 1 1 1m
$ kubectl get pods
$ kubectl get memcached/example-memcached -o yaml
$ cat deploy/crds/cache_v1alpha1_memcached_cr.yaml
#update the number of desired pods (e.g., change "size" to 3 or 4) in deploy/crds/cache_v1alpha1_memcached_cr.
yaml, then execute:
$ kubectl apply -f deploy/crds/cache_v1alpha1_memcached_cr.yaml
$ kubectl get deployment
$ kubectl get pods
```

**# Cleanup**

```
$ kubectl delete -f deploy/crds/cache_v1alpha1_memcached_cr.yaml
$ kubectl delete -f deploy/crds/cache_v1alpha1_memcached_crd.yaml
$ kubectl delete -f deploy/operator.yaml
$ kubectl delete -f deploy/role.yaml
$ kubectl delete -f deploy/role_binding.yaml
$ kubectl delete -f deploy/service_account.yaml
$ oc delete project  blogpost-project
```

**Execution time**

**When accessing to kubernetes resources the time to remove/add pods is instantaneous, usually few seconds, since the annotation is all triggered by the change in the yaml file. Clearly, when the logic is more complex and the change is triggered by a more advanced monitoring system, then the response time of the operators could be slower, which requires possible investigations.**

## [END SHORT VERSION]

## [LONG VERSION]: full steps to deploy the operator

**This operator version has also some unclear part in the documentation, we have opened, and fixed an issue here: https://github.com/operator-framework/operator-sdk/issues/927**

**Quick Start**

### Operator scope

**A namespace-scoped operator (the default) watches and manages resources in a single namespace, whereas a cluster-scoped operator watches and manages resources cluster-wide. Namespace-scoped operators are preferred because of their flexibility. They enable decoupled upgrades, namespace isolation for failures and monitoring, and differing API definitions. However, there are use cases where a cluster-scoped operator may make sense. For example, the cert-manager operator is often deployed with cluster-scoped permissions and watches so that it can manage issuing certificates for an entire cluster.**

**First, checkout and install the operator-sdk CLI:**

```
$ mkdir -p $GOPATH/src/github.com/example-inc/
$ cd $GOPATH/src/github.com/example-inc/
$ operator-sdk new memcached-operator-long --cluster-scoped
$ cd memcached-operator-long
```

```
#Using --cluster-scoped will scaffold the new operator with the following modifications:
 - deploy/operator.yaml - Set WATCH_NAMESPACE="" instead of setting it to the pod's namespace

 - deploy/role.yaml - Use ClusterRole instead of Role

 - deploy/role_binding.yaml:

 - Use ClusterRoleBinding instead of RoleBinding
 - Set the subject namespace to REPLACE_NAMESPACE. "This must be changed to the namespace in which the operator
is deployed".
```

**Add a new Custom Resource Definition**

**Add a new Custom Resource Definition(CRD) API called Memcached, with APIVersion cache.example.com/v1apha1 and Kind Memcached.**

```
# This will scaffold the Memcached resource API under pkg/apis/cache/v1alpha1/....
$ operator-sdk add api --api-version=cache.example.com/v1alpha1 --kind=Memcached
```

## Define the spec and status

**Modify the spec and status of the `Memcached` Custom Resource(CR) at `pkg/apis/cache/v1alpha1/memcached_types.go`:**

```
type MemcachedSpec struct {
        // Size is the size of the memcached deployment
        Size int32 `json:"size"`
}
type MemcachedStatus struct {
        // Nodes are the names of the memcached pods
        Nodes []string `json:"nodes"`
}
```

**After modifying the `*_types.go` file always run the following command to update the generated code for that resource type:**

```
$ operator-sdk generate k8s
```

**Add a new Controller to the project that will watch and reconcile the Memcached resource:**

```
$ operator-sdk add controller --api-version=cache.example.com/v1alpha1 --kind=Memcached
```

**This will scaffold a new Controller implementation under pkg/controller/memcached/....**

**For this example replace the generated Controller file pkg/controller/memcached/memcached_controller.go with the example memcached_controller.go implementation.**

**The example Controller executes the following reconciliation logic for each Memcached CR:**

- **Create a memcached Deployment if it doesn't exist**
- **Ensure that the Deployment size is the same as specified by the Memcached CR spec**
- **Update the Memcached CR status using the status writer with the names of the memcached pods**

## [ For a guide on Reconcilers, Clients, and interacting with resource Events, see the Client API doc. ]

### There are two ways to run the operator:

**1) As a Deployment inside a Kubernetes cluster**
**2) As Go program outside a cluster**

**We will discuss just the case 1).**

## Go program outside a cluster

**1. Run as a Deployment inside the cluster**

### Build the memcached-operator image and push it to a registry with Docker:

```
$ sudo docker login
$ operator-sdk build  <docker id>/memcached-operator-long:v.0.0.1
#(e.g., operator-sdk build docker.io/spanichella/memcached-operator-long)
$ sed -i "" 's|REPLACE_IMAGE|docker.io/<docker id>/memcached-operator-long|g' deploy/operator.yaml.
 (e.g., sed -i "" 's|REPLACE_IMAGE|docker.io/spanichella/memcached-operator-long|g' deploy/operator.yaml)
#If you created your operator using --cluster-scoped=true, update the service account namespace in the
generated ClusterRoleBinding to match where you are deploying your operator.
#check namespaces with
$ kubectl get namespaces
#(or with $ oc get project)
#then set the correct namespace (in my case it was "blogpost-project")
$ export OPERATOR_NAMESPACE=blogpost-project
$ sed -i "" 's|REPLACE_NAMESPACE|blogpost-project|g' deploy/role_binding.yaml #check if it worked correctly,
otherwise set it manually...
# push it to a registry with Docker:
$ docker push   <docker id>/memcached-operator-long:v.0.0.1
#(e.g., docker push docker.io/spanichella/memcached-operator-long)
```

### "Before running the operator, the CRD must be registered with the Kubernetes apiserver":

```
$ kubectl create -f deploy/crds/cache_v1alpha1_memcached_crd.yaml
 OR
$ oc create -f deploy/crds/cache_v1alpha1_memcached_crd.yaml
```

**Setup RBAC and deploy the memcached-operator:**

**(instead of "kubectl" you can also use "*oc*" command instead), thus, to register the CRD:**

```
$ kubectl create -f deploy/service_account.yaml
$ kubectl create -f deploy/role.yaml
$ kubectl create -f deploy/role_binding.yaml
$ kubectl create -f deploy/operator.yaml
```

### Verify that the memcached-operator is up and running:

```
$ kubectl get deployment
NAME DESIRED CURRENT UP-TO-DATE AVAILABLE AGE
memcached-operator 1 1 1 1 1m
$ kubectl get pods
$ kubectl get memcached/example-memcached -o yaml
$ cat deploy/crds/cache_v1alpha1_memcached_cr.yaml
#update the number of desired pods (e.g., change "size" to 3 or 4)
$ kubectl apply -f deploy/crds/cache_v1alpha1_memcached_cr.yaml
$ kubectl get deployment
$ kubectl get pods
```

**# Cleanup**

```
$ kubectl delete -f deploy/crds/cache_v1alpha1_memcached_cr.yaml
$ kubectl delete -f deploy/crds/cache_v1alpha1_memcached_crd.yaml
$ kubectl delete -f deploy/operator.yaml
$ kubectl delete -f deploy/role.yaml
$ kubectl delete -f deploy/role_binding.yaml
$ kubectl delete -f deploy/service_account.yaml
$ oc delete project  blogpost-project
```

## [END LONG VERSION]