# Example-1 of Helm Operator

Created by Sebastiano Panichella, last modified just a moment ago

## BEFORE STARTING:

**Pre-requisities**

- git
- docker version 17.03+.
- kubectl version v1.9.0+.
- ansible version v2.6.0+
- ansible-runner version v1.1.0+
- ansible-runner-http version v1.0.0+
- dep version v0.5.0+. (Optional if you aren't installing from source)
- go version v1.10+. (Optional if you aren't installing from source)
- Access to a kubernetes v.1.9.0+ cluster.
- Install also, to have a quick try of it, Minishift: https://github.com/minishift/minishift
- Set $GOPATH
- Install the Operator SDK CLI as explained in previous examples [LONG VERSIONs]
- Install Minishift: https://github.com/minishift/minishift

**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

## [LONG VERSION]

**Create and deploy an app-operator using the SDK CLI:**

```
# Create an app-operator project that defines the App CR.
```

### 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.

If you'd like to create your nginx-operator project to be cluster-scoped use the following `operator-sdk new` command instead:

```
$ mkdir -p $GOPATH/src/github.com/example-inc/
# Create a new app-operator-ansible project
$ cd $GOPATH/src/github.com/example-inc/
#$ operator-sdk new nginx-operator --api-version=example.com/v1alpha1 --kind=Nginx --type=helm
#or
$ operator-sdk new nginx-operator --cluster-scoped --api-version=example.com/v1alpha1 --kind=Nginx --type=helm
$ cd nginx-operator/
```

This creates the nginx-operator project specifically for watching the Nginx resource with APIVersion `example.com/v1apha1` and Kind `Nginx`.

```
#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".
```

### Customize the operator logic

For this example the nginx-operator will execute the following reconciliation logic for each `Nginx` Custom Resource (CR):

- Create a nginx Deployment if it doesn't exist
- Create a nginx Service if it doesn't exist
- Create a nginx Ingress if it is enabled and doesn't exist
- Ensure that the Deployment, Service, and optional Ingress match the desired configuration (e.g. replica count, image, service type, etc) as specified by the `Nginx` CR

## Watch the Memcached CR

By default, the nginx-operator watches `Nginx` resource events as shown in `watches.yaml` and executes Helm releases using the specified chart:

```
---
- version: v1alpha1
  group: example.com
  kind: Nginx
  chart: /opt/helm/helm-charts/nginx
```

## Reviewing the Nginx Helm Chart

When a Helm operator project is created, the SDK creates an example Helm chart that contains a set of templates for a simple Nginx release.

For this example, we have templates for deployment, service, and ingress resources, along with a NOTES.txt template, which Helm chart developers use to convey helpful information about a release.

If you aren't already familiar with Helm Charts, take a moment to review the Helm Chart developer documentation.

### Understanding the Nginx CR spec

Helm uses a concept called values to provide customizations to a Helm chart's defaults, which are defined in the Helm chart's `values.yaml` file.

Overriding these defaults is a simple as setting the desired values in the CR spec.

Let's use the number of replicas as an example.

First, inspecting `helm-charts/nginx/values.yaml`, we see that the chart has a value called `replicaCount` and it is set to 1 by default.

If we want to have ***2 nginx instances*** in our deployment, we would need to make sure our CR spec contained `replicaCount: 2`.

Update `deploy/crds/example_v1alpha1_nginx_cr.yaml` to look like the following:

```
apiVersion: example.com/v1alpha1
kind: Nginx
metadata:
  name: example-nginx
spec:
  replicaCount: 2
```

As you may have noticed, the Helm operator simply applies the entire spec as if it was the contents of a values file, just like "`helm install -f ./overrides.yaml`" works.

## Build and run the operator

Before running the operator, Kubernetes needs to know about the new custom resource definition the operator will be watching.

Deploy the CRD:

```
kubectl create -f deploy/crds/example_v1alpha1_nginx_crd.yaml
```

**Once this is done, there are two ways to run the operator:**

1. As a pod inside a Kubernetes cluster
2. As a go program outside the cluster using `operator-sdk #(we will not see this case for now.)`

### 1. Run as a pod inside a Kubernetes cluster

Running as a pod inside a Kubernetes cluster is preferred for production use.

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

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

```
$ sudo docker login
$ operator-sdk build  <docker id>/nginx-operator:v.0.0.1
#(e.g., operator-sdk build docker.io/spanichella/nginx-operator)
$ sed -i "" 's|REPLACE_IMAGE|docker.io/<docker id>/nginx-operator|g' deploy/operator.yaml
 (e.g., sed -i "" 's|REPLACE_IMAGE|docker.io/spanichella/nginx-operator|g' deploy/operator.yaml)
#If you created your operator using --cluster-scoped=true, update the service account namespace in the generated Clus
#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, otherwis
# push it to a registry with Docker:
$ docker push   <docker id>/nginx-operator:v.0.0.1
#(e.g., docker push docker.io/spanichella/nginx-operator)
```

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

```
$ kubectl create -f deploy/crds/example_v1alpha1_nginx_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-ansible  1          1          1             1            56s
```

## Deploy the Nginx custom resource

Apply the nginx CR that we modified earlier:

```
$ cat deploy/crds/example_v1alpha1_nginx_cr.yaml
$ kubectl apply -f deploy/crds/example_v1alpha1_nginx_cr.yamlminishift
```

No labels