Operator SDK

» Go-based Operator (Pod Set)

Let’s begin my creating a new project called `myproject` :

```
oc new-project myproject
```

Let’s now create a new directory in our `$GOPATH/src/` directory:

```
mkdir -p $GOPATH/src/github.com/redhat/
```

Navigate to the directory:

```
cd $GOPATH/src/github.com/redhat/
```

Create a new Go-based Operator SDK project for the PodSet:

```
operator-sdk new podset-operator --type=go --skip-git-init
```

Navigate to the project root:

```
cd podset-operator
```

Add a new Custom Resource Definition(CRD) API called PodSet, with APIVersion `app.example.com/v1alpha1` and Kind `PodSet` :

```
operator-sdk add api --api-version=app.example.com/v1alpha1 --kind=PodSet
```

This will scaffold the PodSet resource API under `pkg/apis/cache/v1alpha1/...` .

The Operator-SDK automatically creates the following manifests for you under the `/deploy` directory.

- Custom Resource Definition
- Custom Resource
- Service Account

- Role
- RoleBinding
- Deployment

Inspect the Custom Resource Definition manifest:

```
cat deploy/crds/app_v1alpha1_podset_crd.yaml
```

Modify the spec and status of the `PodSet` Custom Resource(CR) at `go/src/github.com/redhat/podset-operator/pkg/apis/app/v1alpha1/podset_types.go` :

```
package v1alpha1

import (
    metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
)

// EDIT THIS FILE!  THIS IS SCAFFOLDING FOR YOU TO OWN!
// NOTE: json tags are required.  Any new fields you add must have json tags for the fields to be serialized.

// PodSetSpec defines the desired state of PodSet
type PodSetSpec struct {
    Replicas int32 `json:"replicas"`
}

// PodSetStatus defines the observed state of PodSet
type PodSetStatus struct {
    PodNames []string `json:"podNames"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// PodSet is the Schema for the podsets API
// +k8s:openapi-gen=true
type PodSet struct {
    metav1.TypeMeta   `json:",inline"`
    metav1.ObjectMeta `json:"metadata,omitempty"`

    Spec   PodSetSpec   `json:"spec,omitempty"`
    Status PodSetStatus `json:"status,omitempty"`
}

// +k8s:deepcopy-gen:interfaces=k8s.io/apimachinery/pkg/runtime.Object

// PodSetList contains a list of PodSet
type PodSetList struct {
    metav1.TypeMeta `json:",inline"`
    metav1.ListMeta `json:"metadata,omitempty"`
    Items           []PodSet `json:"items"`
}

func init() {
    SchemeBuilder.Register(&PodSet{}, &PodSetList{})
}
```

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

```
operator-sdk generate k8s
```

We can also automatically update the CRD with OpenAPI v3 schema details based off the newly updated `*_types.go` file:

```
operator-sdk generate openapi
```

Observe the CRD now reflects the `spec.replicas` and `status.podNames` OpenAPI v3 schema validation in the spec:

```
cat deploy/crds/app_v1alpha1_podset_crd.yaml
```

Deploy your PodSet Custom Resource Definition to the live OpenShift Cluster:

```
oc create -f deploy/crds/app_v1alpha1_podset_crd.yaml
```

Confirm the CRD was successfully created:

```
oc get crd
```

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

```
operator-sdk add controller --api-version=app.example.com/v1alpha1 --kind=PodSet
```

This will scaffold a new Controller implementation under `go/src/github.com/redhat/podset-operator/pkg/controller/podset/podset_controller.go` .

Modify the PodSet controller logic at `go/src/github.com/redhat/podset-operator/pkg/controller/podset/podset_controller.go` :

```
vim go/src/github.com/redhat/podset-operator/pkg/controller/podset/podset_controller.go
```

Now we can test our logic by running our Operator outside the cluster via our `kubeconfig` credentials:

```
operator-sdk up local --namespace myproject
```

In a new terminal, inspect the Custom Resource manifest:

```
cd $GOPATH/src/github.com/redhat/podset-operator/
cat deploy/crds/app_v1alpha1_podset_cr.yaml
```

Ensure your `kind: PodSet` Custom Resource (CR) is updated with `spec.replicas` :

```
apiVersion: app.example.com/v1alpha1
kind: PodSet
metadata:
  name: example-podset
spec:
  replicas: 3
```

Deploy your PodSet Custom Resource to the live OpenShift Cluster:

```
oc create -f deploy/crds/app_v1alpha1_podset_cr.yaml
```

Verify the PodSet operator has created 3 pods:

```
oc get pods
```

Verify that status shows the name of the pods currently owned by the PodSet:

```
oc get podset example-podset -o yaml
```

Increase the number of replicas owned by the PodSet:

```
oc patch podset example-podset --type='json' -p '[{"op": "replace", "path": "/spec/replicas", "value":5}]'
```