How to Generate Scaffold CRDs

Learn how to generate a scaffold CRD.

We'll cover the following

Scaffold CRDs

With CRDs, we can easily extend Kubernetes APIs by declaring objects in the YAML or JSON format.

Now, let's take a look at the CRD and schema in more detail, so we can learn how to make a scaffold CRD with our own definitions.

CRD schema

The schema of CRDs is shown below:

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name must be in the form: <plural>.<group>
  name: <name>
spec:
  group: <group name>
  conversion: #optional
    # Specifies how custom resources are converted between versions
    # can be None or Webhook
    strategy: None
  names: # Specify the resource and kind names for the custom resource
    categories: # optional
    # List of categories this custom resource belongs to (e.g. 'all')
    - <mycategory>
    kind: <Uppercase name>
    listKind: <Uppercase list kind, defaulted to be kindList>
    plural: <lowercase plural name>
    shortNames: # optional
    # List of strings as short names
    - <alias1>
    singular: <lowercase singular name, defaulted to be lowercase kind>
  scope: Namespaced  # Namespaced or cluster scope
  versions: # List of all API versions
  - name: v1alpha1
    schema: # Optional
```

```
27        openAPIV3Schema: # OpenAPI v3 schema to use for validation and pruning
28          description: HelmChart is the Schema for the helm chart
29          properties: # Describe all the fields
30            ...
```

CRD schema

Defining a simple API can be quite easy because we've only got a few fields. However, creating OpenAPI schema manually can be very tedious, because we have to we declare all the fields, including their types, descriptions, validation patterns, etc. Normally, this CRD is only used by the `kube-apiserver`; we have a separate file, such as `types.go`, to declare the custom resource struct. So, things get difficult because we need ensure that two separate files are always matching. This is extremely difficult to do for complicated and volatile APIs.

Luckily, we do have code generation to help make this much easier. Let's take a look.

Create a scaffold API

Below is the development environment in which we can create a scaffold project for CRD. Click the "Run" button to initialize it:

foo_types.go

Search in directory...

/

foo_types.go

```
1  package v1beta1
2
3  import (
4          metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
5  )
6
7  // EDIT THIS FILE!  THIS IS SCAFFOLDING FOR YOU TO OWN!
8  // NOTE: json tags are required.  Any new fields you add must have
9
10 // FooSpec defines the desired state of Foo
11 type FooSpec struct {
12         // INSERT ADDITIONAL SPEC FIELDS - desired state of cluste
13         // Important: Run "make" to regenerate code after modifyi
14
15         // Foo is an example field of Foo. Edit foo_types.go to re
16         Foo string `json:"foo,omitempty"`
17 }
18
19 // FooStatus defines the observed state of Foo
20 type FooStatus struct {
21         // INSERT ADDITIONAL STATUS FIELD - define observed state
22         // Important: Run "make" to regenerate code after modifyi
23 }
24
25 //+kubebuilder:object:root=true
26 //+kubebuilder:subresource:status
27
28 // Foo is the Schema for the foos API
29 type Foo struct {
30         metav1.TypeMeta   `json:",inline"`
31         metav1.ObjectMeta `json:"metadata,omitempty"`
```

Create a project

First, we create a new folder and initialize the project. We use `educative.io` as the domain name. Run the following commands in the terminal above:

```
mkdir -p ./projects/pwk
cd ./projects/pwk
kubebuilder init --domain educative.io --repo educative.io/pwk
```

Create a project

The output will be as follows:

```
Writing kustomize manifests for you to edit...
Writing scaffold for you to edit...
Get controller runtime:
$ go get sigs.k8s.io/controller-runtime@v0.12.2
go: downloading sigs.k8s.io/controller-runtime v0.12.2
go: downloading k8s.io/apimachinery v0.24.2
go: downloading github.com/gogo/protobuf v1.3.2
go: downloading github.com/google/gofuzz v1.1.0
go: downloading github.com/go-logr/logr v1.2.0
go: downloading k8s.io/client-go v0.24.2
go: downloading k8s.io/klog/v2 v2.60.1
go: downloading k8s.io/utils v0.0.0-20220210201930-3a6ce19ff2f9
go: downloading github.com/prometheus/client_golang v1.12.1
go: downloading gopkg.in/inf.v0 v0.9.1
go: downloading sigs.k8s.io/structured-merge-diff/v4 v4.2.1
go: downloading golang.org/x/net v0.0.0-20220127200216-cd36cc0744dd
go: downloading github.com/evanphx/json-patch v4.12.0+incompatible
go: downloading golang.org/x/time v0.0.0-20220210224613-90d013bbcef8
go: downloading gomodules.xyz/jsonpatch/v2 v2.2.0
go: downloading k8s.io/api v0.24.2
go: downloading k8s.io/apiextensions-apiserver v0.24.2
go: downloading github.com/imdario/mergo v0.3.12
go: downloading github.com/spf13/pflag v1.0.5
go: downloading golang.org/x/term v0.0.0-20210927222741-03fcf44c2211
go: downloading k8s.io/component-base v0.24.2
go: downloading github.com/prometheus/client_model v0.2.0
go: downloading github.com/prometheus/common v0.32.1
go: downloading github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da
go: downloading sigs.k8s.io/json v0.0.0-20211208200746-9f7c6b3444d2
go: downloading github.com/json-iterator/go v1.1.12
go: downloading gopkg.in/yaml.v2 v2.4.0
```

The output of creating a project

Create an API

Next, we create our scaffold API by running the following commands in the terminal above:

```
kubebuilder create api --group apps --version v1beta1 --kind Foo --controller=false --make --resource
```

Create the scaffold API

The API is in version `v1beta1` of group `apps`. The output will be as follows:

```
Writing kustomize manifests for you to edit...
Writing scaffold for you to edit...
api/v1beta1/foo_types.go
Update dependencies:
$ go mod tidy
Running make:
$ make generate
mkdir -p /usercode/projects/pwk/bin
test -s /usercode/projects/pwk/bin/controller-gen || GOBIN=/usercode/projects/pwk/bin go install sigs.k8
go: downloading sigs.k8s.io/controller-tools v0.9.2
go: downloading github.com/spf13/cobra v1.4.0
go: downloading github.com/gobuffalo/flect v0.2.5
go: downloading k8s.io/apiextensions-apiserver v0.24.0
go: downloading k8s.io/apimachinery v0.24.0
go: downloading golang.org/x/tools v0.1.10-0.20220218145154-897bd77cd717
go: downloading github.com/fatih/color v1.12.0
go: downloading k8s.io/api v0.24.0
go: downloading github.com/mattn/go-colorable v0.1.8
go: downloading github.com/mattn/go-isatty v0.0.12
go: downloading golang.org/x/mod v0.6.0-dev.0.20220106191415-9b9b3d81d5e3
/usercode/projects/pwk/bin/controller-gen object:headerFile="hack/boilerplate.go.txt" paths="./..."
Next: implement your new API and generate the manifests (e.g. CRDs,CRs) with:
$ make manifests
```

The output of creating the scaffold API

Now, we can see the `Foo` struct in `api/v1beta1/foo_types.go`, and we can insert more fields there. After we finish our new API, let's run the following commands in the terminal above to automatically generate the manifests (e.g. CRDs, CRs, etc). Let's see the magic!

```
make manifests
```

Command to help generate the manifests

The CRD manifest is already generated at `config/crd/bases/apps.educative.io_foos.yaml`. Below is the scaffold project view, where we can see lots of template files that would help us easily extend Kubernetes APIs with CRDs.

```
|-- Dockerfile
|-- Makefile
|-- PROJECT
|-- README.md
|-- api
|   `-- v1beta1
|       |-- foo_types.go
|       |-- groupversion_info.go
|       `-- zz_generated.deepcopy.go
|-- bin
|   `-- controller-gen
|-- config
|   |-- crd
|   |   |-- kustomization.yaml
|   |   |-- kustomizeconfig.yaml
|   |   `-- patches
|   |       |-- cainjection_in_foos.yaml
```

```
|   |       `-- webhook_in_foos.yaml
|   |-- default
|   |   |-- kustomization.yaml
|   |   |-- manager_auth_proxy_patch.yaml
|   |   `-- manager_config_patch.yaml
|   |-- manager
|   |   |-- controller_manager_config.yaml
|   |   |-- kustomization.yaml
|   |   `-- manager.yaml
|   |-- prometheus
|   |   |-- kustomization.yaml
|   |   `-- monitor.yaml
|   |-- rbac
|   |   |-- auth_proxy_client_clusterrole.yaml
```

The architecture of the scaffold project

Test it out

Now, we can install the CRDs to our cluster by running the following command in the terminal above:

```
make install
```

Install the CRD

We can check and verify the CRD by running the command below:

```
kubectl get crds
```

List all the installed CRDs

The output will be as follows:

```
NAME                    CREATED AT
foos.apps.educative.io   2022-10-13T09:37:31Z
```

Output of listing CRDs

Great! We've successfully installed our own CRD into the cluster. Now let's create custom resources of kind `Foo`.

```
kubectl apply -f config/samples/
```

Apply sample custom resources

We can check the result with the command below.

```
$ kubectl get foo -A
```

List all the custom resources

The output will be as follows:

```
NAMESPACE   NAME         AGE
default     foo-sample   14s
```

All done!

Back

How to Use Kubernetes CRDs

Next

Quiz on CRD