GripDev

Code, Apps and Thoughts @lawrencegripper

CODING, **HOW TO**

K8s Operator with dynamic CRDs using controller runtime (no structs)

JULY 20, 2020 | LAWRENCEGRIPPER | CONTROLLER-RUNTIME, DYNAMIC, KUBEBUILDER, KUBERNETES | LEAVE A COMMENT

Warning: This is a bit of a brain dump.

I'm working on a project at the moment which dynamically creates a set of CRDs in Kubernetes and an operator to manage them based off a schema which is provided by the user **at runtime**.

When the code is being built it doesn't know the schema/shape of the CRDs. This means the standard approach used in Kubebuilder with `controller-gen` isn't going to work. (https://book.kubebuilder.io/reference/generating-crd.html)

Now, for those that haven't played with Kubebuilder it's gives you a few super useful things to build a K8s operator in Go:

1. Controller-gen creates all the structs, templated controllers and keeps all those type files in sync for you. So you change a CRDs Struct and the CRD Yaml spec is updated etc. These are all **build time** tools so we can't use em.
2. A nice abstraction around how to interact with K8s as a controller – The controller-runtime. (https://github.com/kubernetes-sigs/controller-runtime) As the name suggests we can use this one at runtime.

So while we can't use the build time `controller-gen` we can still use all the goodness of the `controller-runtime`. In theory.

This is where the fun came in, there aren't any docs on interacting with a dynamic/unstructured object type using the controller runtime so I did a bit of playing around.

(Code samples for illustration – if you want end2end running example skip to the bottom).

To get started on this journey we need a helping hand. Kuberentes has an API for working which objects which don't have a Golang struct defined. This is how we can start: Lets check out the go docs for `unstructured`..

`"k8s.io/apimachinery/pkg/apis/meta/v1/unstructured"` (https://godoc.org/k8s.io/apimachinery/pkg/apis/meta/v1/unstructured)

Ok so this gives us some nice ways to work with a CRD which doesn't have a struct defined.

To use this meaningfully we're going to have to tell it the type it represents – In K8s this means telling it it's `Group`, `Version` and `Kind`. These are all wrapped up nicely in the `schema.GroupVersionKind` struct. Lets look at the docs:

`"k8s.io/apimachinery/pkg/runtime/schema"` (https://godoc.org/k8s.io/apimachinery/pkg/runtime/schema#GroupVersionKind).

Great so hooking these two up together we can create an `Unstructured` instance that represents a CRD, like so!

```
groupVersionKind := schema.GroupVersionKind{
    Group:   "azurerm.tfb.local",
    Version: "valpha1",
    Kind: "resource-group",
}

crdForKind := unstructured.Unstructured{}
crdForKind.SetGroupVersionKind(groupVersionKind)
```

Cool, so what can we do from here? Well the controller runtime uses the `runtime.object` interface for all it's interactions and guess what we have now? Yup a `runtime.Object`.. wrapper method to make things obvious

```
func runtimeObjFromGVK(r schema.GroupVersionKind) runtime.Object {
    obj := &unstructured.Unstructured{}
    obj.SetGroupVersionKind(r)
    return obj
}
```

Well now we can create an instance of the controller for our `unstructured` CRD.

```
    setupLog.Info("Enabling controller for resource", "kind", gv.GroupVersionKind.Kind)
    client := mgr.GetClient()
    err = ctrl.NewControllerManagedBy(mgr).
        For(runtimeObjFromGVK(gv.GroupVersionKind)).
        Complete(&controller{
            Client:       client,
            tfReconciler: NewTerraformReconciler(provider, client),
            scheme:       mgr.GetScheme(),
            gvk:          &groupVersionKind,
        })
    if err != nil {
        setupLog.Error(err, "unable to create controller", "kind", gv.GroupVersionKind.Kind)
        os.Exit(1)
    }
}
```

Notice that I'm passing the `GroupVersionKind` into the controller struct – this will be useful when we come to make changes to a CRD we're handling.

In the same way that you can use the `r.Client` on the controller in Kubebuilder you can now use it with the `unstructured` resource. We use the `gvk` again here to set the type so that the `client` knows how to work with it.

```
func (r *controller) Reconcile(req ctrl.Request) (ctrl.Result, error) {
    log := recLog.WithValues("generic reconciler", req.NamespacedName)
    log.V(1).Info("reconciling runtimeobj")
    ctx := context.Background()

    _ = ctx

    // Note: Create an unstructued type for the Client to use and set the Kind
    // so that it can GET/UPDATE/DELETE etc
    resource := &unstructured.Unstructured{}
    resource.SetGroupVersionKind(*r.gvk)

    // Get the CRD content and update `resource` var with it
    err := r.Client.Get(ctx, req.NamespacedName, resource)
    if err != nil {
        log.Error(err, "Failed getting resource", "req", req.NamespacedName)
    }

    // Make some change to the resource here...
    // just an example - not very useful in reality
    resource.SetAnnotations(map[string]string{
        "anotation": "Change",
    })

    // Update the CRD with that chnage
    err = r.Client.Update(ctx, resource)
    if err != nil {
        log.Error(err, "Failed saving resource")
        panic("Error updating CRD instance (Add)") // TODO handle retries
    }

    return ctrl.Result{}, nil
}
```

Now you might be thinking – wow isn't it going to be painful working without the strongly typed CRD structs?

Yes it's harder **but there are some helper methods in the `unstructured` api** which make things much easier. For example, the following let you easily retrieve or set a string which is nested inside it.

```
unstructured.NestedString(resource.Object, "status", "_tfoperator",
"tfState")

unstructured.SetNestedField(resource.Object, string(serializedState),
"status", "_tfoperator", "tfState")
```

Here is the end result hooking up the controller runtime to a set of dynamically created and managed CRDS. (https://github.com/lawrencegripper/tfoperatorbridge/blob/4d0d3c9bb944d49575c80afd09d936aa6feafba8/controller.go)It's very much a work in progress and I'd love feedback if there are easier ways to tackle this or things that I've got wrong.