KubeCon

North America 2017

Prototyping with CRDs

Tomáš Smetana, Engineering Manager, Red Hat

Background

- “Kubernetes is a large project” – cpt. Obvious
  - Well estabilished processes
    - Reviews
    - Feature proposals
    - Community meetings
    - …
    - Stability!
- How do I get my new feature in?
  - Even when it’s obviously so cool as persistent volume snapshots

Volume Snapshots story

- Simple idea: let’s take snapshot of a PersistentVolume in k8s
  - Present the idea to the community
  - Create proposal and have it reviewed
  - Write the code and have it reviewed
  - Have the code merged
  - \o/
- … nope
  - Use the plan “B”

Plan B

- Start implementing the feature outside of Kubernetes
  - Figure out the details
  - Get user feedback
  - Make changes as needed
  - Merge into the main tree when ready, update proposal as needed

Custom Resources

- Custom Resources
  - Basic building block for easy Kuberentes extensions
  - API objects
  - Dynamically added/registered
- Experiment outside of Kubernetes
  - Example: kubernetes-incubator on github

Custom Resource Definitions

- Built-in API: “register” custom objects in the API server
  - Custom objects behave just like the default ones
  - Could be handled by external controllers

```go
const (
    VolumeSnapshotResourcePlural = "volumesnapshots"
    GroupName = "volumesnapshot.external-storage.k8s.io"
)

type VolumeSnapshot struct {
    metav1.TypeMeta `json:",inline"`
    Metadata        metav1.ObjectMeta `json:"metadata"`
    Spec VolumeSnapshotSpec `json:"spec" protobuf:"bytes,2,opt,name=spec"`
    Status VolumeSnapshotStatus `json:"status" protobuf:"bytes,3,opt,name=status"`
}
```

```
apiextensionsv1beta1.CustomResourceDefinition{
    ObjectMeta: metav1.ObjectMeta{
        Name: crdv1.VolumeSnapshotResourcePlural + "." + crdv1.GroupName,
    },
    Spec: apiextensionsv1beta1.CustomResourceDefinitionSpec{
        Group: crdv1.GroupName,
        Version: schema.GroupVersion{Group: GroupName, Version: "v1"}
        Scope: apiextensionsv1beta1.NamespaceScoped,
        Names: apiextensionsv1beta1.CustomResourceDefinitionNames{
            Plural: crdv1.VolumeSnapshotResourcePlural,
            Kind: reflect.TypeOf(crdv1.VolumeSnapshot{}).Name(),
        },
    },
}
```

Controller

- Custom external controller
  - The objects themselves can’t do much
  - Controller talks to API server and makes use of the new objects (watches for updates)
  - Takes care about registering the CRDs

```
// Create the CRD on the API server using kubernetes.Interface
clientset.ApiextensionsV1beta1().CustomResourceDefinitions().Create(crd)
…
wait.Poll(100*time.Millisecond, 60*time.Second, func() (bool, error) {
        _, err := snapshotClient.Get().
            Resource(crdv1.VolumeSnapshotDataResourcePlural).DoRaw()
        if err == nil {
            return true, nil
        }
        if apierrors.IsNotFound(err) {
            return false, nil
        }
        return false, err
})
```

```
…
func InstallHandlers(client *rest.RESTClient, scheme *runtime.Scheme, … ) {
    sc := &snapshotController{
        snapshotClient: client,
        snapshotScheme: scheme,
    }
    …
    source := kcache.NewListWatchFromClient(
            sc.snapshotClient,
            crdv1.VolumeSnapshotResourcePlural,
            apiv1.NamespaceAll,
            fields.Everything())
    …
    …
    sc.snapshotStore, sc.snapshotController = kcache.NewInformer(
        source,
        // The object type.
        &crdv1.VolumeSnapshot{},
        // Every resyncPeriod, all resources will retrigger events.
        time.Minute*60,
        // The custom resource event handlers.
        kcache.ResourceEventHandlerFuncs{
            AddFunc:    sc.onSnapshotAdd,
            UpdateFunc: sc.onSnapshotUpdate,
            DeleteFunc: sc.onSnapshotDelete,
    })
    …
}
```

Advantages

- Not bound to the Kubernetes development cycle
- Quicker changes
- Deeper changes
- Might make in-tree acceptance easier

Disadvantages

- Might need more API calls – performance penalties
- More work for the users/admins (deployments, RBAC, …)
- Dependencies might be more complicated to manage
- Less visible for potential users and contributors

Conclusion

- Easy way of extending Kubernetes features
- Suitable for experimenting
- … or features not generic enough for the main Kubernetes tree

The End

- Questions?