KUBERNETES CUSTOM RESOURCE DEFINITION (CRDS) EXPLAINED

In this post, we are going to talk about Custom Resource Definitions (CRDs). We will go over:

- What CRDs are
- How they are used
- How to create them
- A few tips and tricks

To follow along, I assume you have prior knowledge of Kubernetes, kubectl, and have minikube running.

(This article is part of our Kubernetes Guide. Use the right-hand menu to navigate.)

CRD concepts in K8S

To begin to understand what CRD is, we must go over a couple of concepts in Kubernetes:

- A **resource** is an endpoint in k8s API that allow you to store an API object of any kind.
- A **custom resource** allows you to create your own API objects and define your own kind just like Pod, Deployment, ReplicaSet, etc.

Custom Resource allows you to extend Kubernetes capabilities by adding any kind of API object useful for your application. Custom Resource Definition is what you use to define a Custom Resource. This is a powerful way to extend Kubernetes capabilities beyond the default installation.

How to create a CRD

The manifest below shows an example CRD *crd.yaml*

```
apiVersion: apiextensions.k8s.io/v1beta1
kind: CustomResourceDefinition
metadata:
  name: appconfigs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
  scope: Namespaced
  names:
    plural: appconfigs
    singular: appconfig
    kind: AppConfig
    shortNames:
    - ac
```

Let's explain what the CRD above will create:

- The first two lines defines what the apiversion is and we are saying we want to create a custom resource definition.
- The metadata field helps us define what the name of the resource is. In our case appconfigs (plural).
- Spec group help us define what the group name will be.
- Spec version helps us define the version. We are saying it is v1, and we want this version to be our storage version.
- As you can see, we can define a version of our CRD and only one version can be a storage version at a time, so keep that in mind. We then made sure that this CRD is a namespaced and not cluster wide. This allow us to create the CRD for either just a specific namespace or for the whole cluster.
- Next, we defined what the singular and plural name of our CRD will be.
- Lastly we defined the kind name and the short name. We can then create it with "*kubectl create -f crd.yaml*". The new namespaced RESTful API endpoint for our CRD will be found at /apis/stable.example.com/v1/namespaces/*/appconfigs/

So we are probably wondering ok this is good but how is this useful? To use, the CRD above, we create a manifest using the kind we created with the CRD. For example we have *my-kind.yaml*

```
apiVersion: "stable.example.com/v1"
kind: AppConfig
metadata:
  name: demo-appconfig
spec:
  uri: "some uri"
```

```
Command: "some command"
image: my-image
```

As you can see, we are using the kind we defined in our CRD, then we defined the fields we want our kind object to have. Here we want to define, the uri, command and image for our app. Now if we run "*kubectl create -f my-kind.yaml*" our application can consume the data we created with the manifest above. To see what is going on we can run "*kubectl get ac -o yaml*", we can see detailed info on what we just created. Notice how we are using the short name (ac) we defined in our CRD.

How to delete a CRD

To delete the CRD and resources we created, simply run kubectl delete just like with any other resources. It is important to know that the above CRD is just data which can be stored and retrieved therefore, it doesn't give us a fully declarative API. The way to make this resource fully declarative is to add a custom controller, whose job is to make sure that the current state and the desired state are always in sync. An example is something like a replication controller.

CRDs expand Kubernetes

CRD is a way to extend kubernetes allowing us to create a custom resource of our choice and making it declarative with the help of a custom controller.

Additional resources

For more on Kubernetes, explore these resources:

- Kubernetes Guide, with 20+ articles and tutorials
- BMC DevOps Blog
- Bring Kubernetes to the Serverless Party
- How eBay is Reinventing Their IT with Kubernetes & Replatforming Program