CRD 介绍

本文主要介绍 CRD 资源对象的使用

介绍

前面我们讲解了很多关于 client-go 的实现，也介绍了如何使用 client-go 来创建一个控制器，但是我们前面介绍的都是 Kubernetes 中内置的资源对象，比如 Pod、Deployment 这些，而这些资源对象已经有了内置的控制器实现，那么我们还可以如何去使用控制器呢？那就需要去了解 CRD 这种资源对象了。

`Custom Resource Define` 简称 CRD，是 Kubernetes（v1.7+）为提高可扩展性，让开发者去自定义资源的一种方式。CRD 资源可以动态注册到集群中，注册完毕后，用户可以通过 kubectl 来创建访问这个自定义的资源对象，类似于操作 Pod 一样。不过需要注意的是 CRD 仅仅是资源的定义而已，需要一个对应的控制器去监听 CRD 的各种事件来添加自定义的业务逻辑，这个才是我们要重点学习的。

定义

如果说只是对 CRD 资源本身进行 CRUD 操作的话，不需要 Controller 也是可以实现的，相当于就是只有数据存入了 etcd 中，而没有对这个数据的相关操作而已。比如我们可以定义一个如下所示的 CRD 资源清单文件：

```
# crd-demo.yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  # name 必须匹配下面的spec字段：<plural>.<group>
  name: crontabs.stable.example.com
spec:
  # group 名用于 REST API 中的定义：/apis/<group>/<version>
  group: stable.example.com
  # 列出自定义资源的所有 API 版本
  versions:
  - name: v1beta1  # 版本名称，比如 v1、v2beta1 等等
    served: true  # 是否开启通过 REST APIs 访问 `/apis/<group>/<version>/...`
    storage: true # 必须将一个且只有一个版本标记为存储版本
    schema:  # 定义自定义对象的声明规范
      openAPIV3Schema:
        description: Define CronTab YAML Spec
        type: object
        properties:
          spec:
            type: object
            properties:
              cronSpec:
                type: string
              image:
                type: string
              replicas:
                type: integer
  # 定义作用范围：Namespaced（命名空间级别）或者 Cluster（整个集群）
  scope: Namespaced
  names:
    # kind 是 sigular 的一个驼峰形式定义，在资源清单中会使用
    kind: CronTab
    # plural 名字用于 REST API 中的定义：/apis/<group>/<version>/<plural>
    plural: crontabs
    # singular 名称用于 CLI 操作或显示的一个别名
    singular: crontab
    # shortNames 相当于缩写形式
    shortNames:
    - ct
```

> 需要注意的是 v1.16 版本以后已经 GA 了，使用的是 v1 版本，之前都是 v1beta1，定义规范有部分变化，所以要注意版本变化。

这个地方的定义和我们定义普通的资源对象比较类似，我们说我们可以随意定义一个自定义的资源对象，但是在创建资源的时候，肯定不是任由我们随意去编写 YAML 文件的，当我们把上面的 CRD 文件提交给 Kubernetes 之后，Kubernetes 会对我们提交的声明文件进行校验，从定义可以看出 CRD 是基于 OpenAPI v3 schem 进行规范的。当然这种校验只是对于字段的类型进行校验，比较初级，如果想要更加复杂的校验，这个时候就需要通过 Kubernetes 的 admission webhook 来实现了。关于校验的更多用法，可以前往官方文档查看。

同样现在我们可以直接使用 kubectl 来创建这个 CRD 资源清单：

```
$ kubectl apply -f crd-demo.yaml
customresourcedefinition.apiextensions.k8s.io/crontabs.stable.example.com created
```

这个时候我们可以查看到集群中已经有我们定义的这个 CRD 资源对象了：

```
$ kubectl get crd |grep example
crontabs.stable.example.com                     2019-12-19T02:37:54Z
```

这个时候一个新的 namespace 级别的 RESTful API 就会被创建：

```
/apis/stable/example.com/v1beta1/namespaces/*/crontabs/...
```

然后我们就可以使用这个 API 端点来创建和管理自定义的对象，这些对象的类型就是上面创建的 CRD 对象规范中的 `CronTab`。

现在在 Kubernetes 集群中我们就多了一种新的资源叫做 `crontabs.stable.example.com`，我们就可以使用它来定义一个 `CronTab` 资源对象了，这个自定义资源对象里面可以包含的字段我们在定义的时候通过 `schema` 进行了规范，比如现在我们来创建一个如下所示的资源清单：

```
# crd-crontab-demo.yaml
apiVersion: "stable.example.com/v1beta1"
kind: CronTab
metadata:
  name: my-new-cron-object
spec:
  cronSpec: "* * * * */5"
  image: my-awesome-cron-image
```

我们可以直接创建这个对象：

```
$ kubectl apply -f crd-crontab-demo.yaml
crontab.stable.example.com/my-new-cron-object created
```

然后我们就可以用 kubectl 来管理我们这里创建 CronTab 对象了，比如：

```
$ kubectl get ct  # 简写
NAME                 AGE
my-new-cron-object   42s
$ kubectl get crontab
NAME                 AGE
my-new-cron-object   88s
```

在使用 kubectl 的时候，资源名称是不区分大小写的，我们可以使用 CRD 中定义的单数或者复数形式以及任何简写。

我们也可以查看创建的这个对象的原始 YAML 数据：

```
$ kubectl get ct -o yaml
apiVersion: v1
items:
- apiVersion: stable.example.com/v1beta1
  kind: CronTab
  metadata:
    annotations:
      kubectl.kubernetes.io/last-applied-configuration: |
```

```
        {"apiVersion":"stable.example.com/v1beta1","kind":"CronTab","metadata":{"annotations":{},"name":"my-new-cron-object","namespac
    creationTimestamp: "2019-12-19T02:52:55Z"
    generation: 1
    name: my-new-cron-object
    namespace: default
    resourceVersion: "12342275"
    selfLink: /apis/stable.example.com/v1beta1/namespaces/default/crontabs/my-new-cron-object
    uid: dace308d-5f54-4232-9c7b-841adf6bab62
  spec:
    cronSpec: '* * * * */5'
    image: my-awesome-cron-image
kind: List
metadata:
  resourceVersion: ""
  selfLink: ""
```

我们可以看到它包含了上面我们定义的 `cronSpec` 和 `image` 字段。

就如上面我们说的，现在我们自定义的资源创建完成了，但是也只是单纯的把资源清单数据存入到了 etcd 中而已，并没有什么其他用处，因为我们没有定义一个对应的控制器来处理相关的业务逻辑，所以接下来我们需要来了解如何为 CRD 创建自定义的控制器。