Custom Resource Example

Note: CustomResourceDefinition is the successor of the deprecated ThirdPartyResource.

This particular example demonstrates how to generate a client for CustomResources using `k8s.io/code-generator`. The clientset can be generated using the `./hack/update-codegen.sh` script.

The `update-codegen` script will automatically generate the following files and directories:

- `pkg/apis/cr/v1/zz_generated.deepcopy.go`
- `pkg/client/`

The following code-generators are used:

- `deepcopy-gen` - creates a method `func (t* T) DeepCopy() *T` for each type T
- `client-gen` - creates typed clientsets for CustomResource APIGroups
- `informer-gen` - creates informers for CustomResources which offer an event based interface to react on changes of CustomResources on the server
- `lister-gen` - creates listers for CustomResources which offer a read-only caching layer for GET and LIST requests.

Changes should not be made to these files manually, and when creating your own controller based off of this implementation you should not copy these files and instead run the `update-codegen` script to generate your own.

Please see `k8s.io/sample-controller` for an example controller for CustomResources using the generated client.

Use Cases

CustomResourceDefinitions can be used to implement custom resource types for your Kubernetes cluster. These act like most other Resources in Kubernetes, and may be `kubectl apply`'d, etc.

Some example use cases:

- Provisioning/Management of external datastores/databases (eg. CloudSQL/RDS instances)
- Higher level abstractions around Kubernetes primitives (eg. a single Resource to define an etcd cluster, backed by a Service and a ReplicationController)

Defining types

Each instance of your custom resource has an attached Spec, which should be defined via a `struct{}` to provide data format validation. In practice, this Spec is arbitrary key-value data that specifies the configuration/behavior of your Resource.

For example, if you were implementing a custom resource for a Database, you might provide a DatabaseSpec like the following:

```go
type DatabaseSpec struct {
    Databases []string `json:"databases"`
    Users     []User   `json:"users"`
    Version   string   `json:"version"`
}
```

```go
type User struct {
    Name     string `json:"name"`
    Password string `json:"password"`
}
```