Create, Update & Delete Deployment with the Dynamic Package

This example program demonstrates the fundamental operations for managing on Deployment resources, such as `Create`, `List`, `Update` and `Delete` using client-go's `dynamic` package.

Typed Vs. Dynamic

The code in this directory is based on a similar example that uses Kubernetes typed client sets. The typed client sets make it simple to communicate with the API server using pre-generated local API objects to achieve an RPC-like programming experience. Typed clients uses program compilations to enforce data safety and some validation. However, when using typed clients, programs are forced to be tightly coupled with the version and the types used.

The `dynamic` package on the other hand, uses a simple type, `unstructured.Unstructured`, to represent all object values from the API server. Type `Unstructured` uses a collection of nested `map[string]interface{}` values to create an internal structure that closely resemble the REST payload from the server.

The dynamic package defers all data bindings until runtime. This means programs that use the dynamic client will not get any of the benefits of type validation until the program is running. This may be a problem for certain types of applications that require strong data type check and validation.

Being loosely coupled, however, means that programs that uses the `dynamic` package do not require recompilation when the client API changes. The client program has more flexibility in handling updates to the API surface without knowing ahead of time what those changes are.

Running this example

Make sure you have a Kubernetes cluster and `kubectl` is configured:

```
kubectl get nodes
```

Compile this example on your workstation:

```
cd dynamic-create-update-delete-deployment
go build -o ./app
```

Now, run this application on your workstation with your local kubeconfig file:

```
./app
# or specify a kubeconfig file with flag
./app -kubeconfig=$HOME/.kube/config
```

Running this command will execute the following operations on your cluster:

1. **Create Deployment:** This will create a 2 replica Deployment. Verify with `kubectl get pods`.
2. **Update Deployment:** This will update the Deployment resource created in previous step by setting the replica count to 1 and changing the container image to `nginx:1.13`. You are encouraged to inspect the retry loop that handles conflicts. Verify the new replica count and container image with `kubectl describe deployment demo`.
3. **List Deployments:** This will retrieve Deployments in the `default` namespace and print their names and replica counts.
4. **Delete Deployment:** This will delete the Deployment object and its dependent ReplicaSet resource. Verify with `kubectl get deployments`.

Each step is separated by an interactive prompt. You must hit the Return key to proceed to the next step. You can use these prompts as a break to take time to run `kubectl` and inspect the result of the operations executed.

You should see an output like the following:

```
Creating deployment...
Created deployment "demo-deployment".
-> Press Return key to continue.

Updating deployment...
Updated deployment...
-> Press Return key to continue.

Listing deployments in namespace "default":
 * demo-deployment (1 replicas)
-> Press Return key to continue.

Deleting deployment...
Deleted deployment.
```

Cleanup

Successfully running this program will clean the created artifacts. If you terminate the program without completing, you can clean up the created deployment with:

```
kubectl delete deploy demo-deployment
```

Troubleshooting

If you are getting the following error, make sure Kubernetes version of your cluster is v1.13 or higher in `kubectl version`:

```
panic: the server could not find the requested resource
```