Helm Operator

» Helm Operator

Navigate to your redhat directory within your $GOPATH/src:

```
cd $GOPATH/src/redhat
```

Use the CLI to create a new Helm-based Cockroachdb Operator project:

```
operator-sdk new cockroachdb-operator --type=helm --helm-chart cockroachdb --helm-chart-repo https://kubernetes-charts.storage.googlea
pis.com
cd cockroachdb-operator
```

Observe the `watches.yaml` file:

```
cat watches.yaml
```

Create a symlink that targets the Cockroachdb Helm chart in the cockroachdb-operator project directory:

```
mkdir -p /opt/helm/helm-charts
ln -s /root/tutorial/go/src/github.com/redhat/cockroachdb-operator/helm-charts/cockroachdb/ /opt/helm/helm-charts/
```

Apply the CockroachDB Custom Resource Definition to the cluster:

```
oc apply -f deploy/crds/charts_v1alpha1_cockroachdb_crd.yaml
```

Now we can run our Helm-based Operator from outside the cluster via our kubeconfig credentials. Running the command will block the current session so you can continue interacting with the OpenShift cluster by opening a new terminal window.

```
operator-sdk up local --namespace myproject
```

Open a new terminal window and navigate to the `cockroachdb-operator` top-level directory:

```
cd $GOPATH/src/github.com/redhat/cockroachdb-operator
```

Update the CockroachDB Custom Resource at `go/src/github.com/redhat/cockroachdb-operator/deploy/crds/charts_v1alpha1_cockroachdb_cr.yaml` with the following:

```
kind: Cockroachdb
metadata:
  name: example
spec:
  Replicas: 1
  Storage: "1Gi"
  StorageClass: local-storage
```

After updating the CockroachDB Custom Resource with our desired spec, apply it to the cluster:

```
oc apply -f deploy/crds/charts_v1alpha1_cockroachdb_cr.yaml
```

Confirm that the Custom Resource was created:

```
oc get cockroachdb
```

It may take some time for the environment to pull down the CockroachDB container image. Confirm that the Stateful Set was created:

```
oc get statefulset
```

Confirm that the Stateful Set's pod is currently running:

```
oc get pods -l chart=cockroachdb-2.1.1
```

Confirm that the CockroachDB "internal" and "public" ClusterIP Service were created:

```
oc get services
```

Verify that you can access the CockroachDB Web UI by first exposing the CockroachDB Service as a publicly accessible OpenShift Route:

```
COCKROACHDB_PUBLIC_SERVICE=`oc get svc -o jsonpath={$.items[1].metadata.name}`
oc expose --port=http svc $COCKROACHDB_PUBLIC_SERVICE
```

Fetch the OpenShift Route URL and copy/paste it into your browser:

```
COCKROACHDB_UI_URL=`oc get route -o jsonpath={$.items[0].spec.host}`
echo $COCKROACHDB_UI_URL
```

Let's talk to the CockroachDB cluster by connecting to the service from within the cluster. CockroachDB is PostgreSQL wire protocol compatible so there's a wide variety of supported clients. For the sake of example, we'll open up a SQL shell using CockroachDB's built-in shell and play around with it a bit.

```
oc run -it --rm cockroach-client --image=cockroachdb/cockroach --restart=Never --command -- ./cockroach sql --insecure --host $COCKROA
CHDB_PUBLIC_SERVICE
```

Once you see the SQL prompt, run the following to show the default databases:

```
SHOW DATABASES;
```

Create a new database called bank and populate a table with arbitrary values:

```
CREATE DATABASE bank;
CREATE TABLE bank.accounts (id INT PRIMARY KEY, balance DECIMAL);
INSERT INTO bank.accounts VALUES (1234, 10000.50);
```

Verify the table and values were successfully created:

```
SELECT * FROM bank.accounts;
```

Exit the SQL prompt:

```
\q
```

Let's now update the CockroachDB example Custom Resource and increase the desired number of replicas to 3:

```
oc patch cockroachdb example --type='json' -p '[{"op": "replace", "path": "/spec/Replicas", "value":3}]'
```

Verify that the CockroachDB Stateful Set is creating two additional pods:

```
oc get pods -l chart=cockroachdb-2.1.1
```

The CockroachDB UI should now reflect these additional nodes as well.

If any CockroachDB member fails it gets restarted or recreated automatically by the Kubernetes infrastructure, and will rejoin the cluster automatically when it comes back up. You can test this scenario by killing any of the pods:

```
oc delete pods -l chart=cockroachdb-2.1.1
```

Watch the pods respawn:

```
oc get pods -l chart=cockroachdb-2.1.1
```

Confirm that the contents of the database still persist by connecting to the database cluster:

```
COCKROACHDB_PUBLIC_SERVICE=`oc get svc -o jsonpath={$.items[1].metadata.name}`
oc run -it --rm cockroach-client --image=cockroachdb/cockroach --restart=Never --command -- ./cockroach sql --insecure --host $COCKROA
CHDB_PUBLIC_SERVICE
```

Once you see the SQL prompt, run the following to confirm the database contents are still present:

```
SELECT * FROM bank.accounts;
```

Exit the SQL prompt:

```
\q
```

Delete the CockroachDB cluster and all associated resources by deleting the example Custom Resource:

```
oc delete cockroachdb example
```

Verify that the Stateful Set, pods, and services are removed:

```
oc get statefulset
oc get pods
oc get services
```