Deploy multi-tier application

This lab shows you how to build, deploy and manage a simple, multi-tier web application using Kubernetes.

We will be deploying the guestbook demo application which is made up of Redis master, Redis slave, and guestbook frontend. After successfully deploying we will update the application and then rollback to the previous version.

Start up Redis Master

The guestbook application uses Redis to store its data. It writes data to a Redis master instance and reads data from multiple Redis slave instances.

Creating the Redis Master Deployment

The manifest file, included below, specifies a Deployment controller that runs a single replica Redis master Pod.

Apply the Redis Master deployment file

```
kubectl apply -f manifests/redis-master-deployment.yaml
```

Verify the Redis master is running

```
kubectl get pods
```

You should see something like:

```
NAME                            READY     STATUS    RESTARTS   AGE
redis-master-585798d8ff-s9qmr   1/1       Running   0          44s
```

Now let's check the logs

```
kubectl logs -f <POD NAME>
```

If everything looks good continue

Create the Redis Master Service

The guestbook applications needs to communicate to the Redis master to write its data. You need to apply a Service to proxy the traffic to the Redis master Pod. A Service defines a policy to access the Pods.

Apply the Service

```
kubectl apply -f manifests/redis-master-service.yaml
```

This manifest file creates a Service named redis-master with a set of labels that match the labels previously defined, so the Service routes network traffic to the Redis master Pod.

Confirm service is running

```
kubectl get svc
```

You should see running service

```
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.96.0.1      <none>        443/TCP    34m
redis-master   ClusterIP   10.107.62.78   <none>        6379/TCP   56s
```

Start up the Redis Slaves

Although the Redis master is a single pod, you can make it highly available to meet traffic demands by adding replica Redis slaves.

Create Redis Slave Deployment

Deployments scale based off of the configurations set in the manifest file. In this case, the Deployment object specifies two replicas.
If there are not any replicas running, this Deployment would start the two replicas on your container cluster. Conversely, if there are more than two replicas are running, it would scale down until two replicas are running.

Apply the Redis slave deployment

```
kubectl apply -f manifests/redis-slave-deployment.yaml
```

Confirm it's running successfully.

```
kubectl get pods
```

You should now see the following

```
NAME                            READY     STATUS    RESTARTS   AGE
redis-master-585798d8ff-s9qmr   1/1       Running   0          6m
redis-slave-865486c9df-bf68k    1/1       Running   0          8s
redis-slave-865486c9df-btg6h    1/1       Running   0          8s
```

Create Redis Slave service

The guestbook application needs to communicate to Redis slaves to read data. To make the Redis slaves discoverable, you need to set up a Service. A Service provides transparent load balancing to a set of Pods.

Apply Redis Slave Service

```
kubectl apply -f manifests/redis-slave-service.yaml
```

Confirm services are running

```
kubectl get services
```

You should see:

```
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.96.0.1      <none>        443/TCP    38m
redis-master   ClusterIP   10.107.62.78   <none>        6379/TCP   5m
redis-slave    ClusterIP   10.98.54.128   <none>        6379/TCP   35s
```

Setup and Expose the Guestbook Frontend

The guestbook application has a web frontend serving the HTTP requests written in PHP. It is configured to connect to the `redis-master` Service for write requests and the `redis-slave` service for Read requests.

Create the Guestbook Frontend Deployment

Apply the YAML file using the `--record` flag.
NOTE: We are using the `--record` flag to keep a history of the deployment, which enables us to rollback.

```
kubectl apply --record -f manifests/frontend-deployment.yaml
```

Now let's verify they are running

```
kubectl get pods -l app=guestbook -l tier=frontend
```

You should see something like this

```
NAME                        READY     STATUS    RESTARTS   AGE
frontend-67f65745c-jwhdw    1/1       Running   0          27s
frontend-67f65745c-lxpxj    1/1       Running   0          27s
frontend-67f65745c-tsq9k    1/1       Running   0          27s
```

Create the Frontend Service

The `redis-slave` and `redis-master` Services you applied are only accessible within the container cluster because the default type for a Service is `ClusterIP`. ClusterIP provides a single IP address for the set of Pods the Service is pointing to. This IP address is accessible only within the cluster.

If you want guests to be able to access your guestbook, you must configure the frontend Service to be externally visible, so a client can request the Service from outside the container cluster.

Apply the Frontend Service

```
kubectl apply -f manifests/frontend-service.yaml
```

Confirm the service is running

```
kubectl get services
```

You should see something like this

```
NAME           TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)        AGE
frontend       NodePort    10.107.73.47   <none>        80:31495/TCP   34s
kubernetes     ClusterIP   10.96.0.1      <none>        443/TCP        44m
redis-master   ClusterIP   10.107.62.78   <none>        6379/TCP       11m
redis-slave    ClusterIP   10.98.54.128   <none>        6379/TCP       6m
```

Viewing the Frontend Service

To load the front end in a browser visit your Master servers IP and use the port from previous command.

In the example above we can see that `frontend` Service is running on `NodePort` 31495 so I would visit the following in a web browser

`http://<masterIP>:31495`

Scale Web Frontend

Scaling up or down is easy because your servers are defined as a Service that uses a Deployment controller.

Run the following command to scale up the number of frontend Pods:

```
kubectl scale deployment frontend --replicas=5
```

Now verify the Pods increased to specified number of replicas

```
kubectl get pods -l app=guestbook -l tier=frontend
```

To scale back down run

```
kubectl scale deployment frontend --replicas=2
```

Now check to see if Pods are being destroyed

```
kubectl get pods -l app=guestbook -l tier=frontend
```

Update frontend image

Confirm the version of the image you are using

```
kubectl describe deployment frontend |grep Image
```

You should see `v4`

```
Image:      gcr.io/google-samples/gb-frontend:v4
```

Now we are going to update our YAML file

```
vim manifests/frontend-deployment.yaml
```

Replace `v4` with `v5` so it looks like below:

```
..snip
- name: php-redis
        image: gcr.io/google-samples/gb-frontend:v5
```

Now save the file and deploy the new version

```
kubectl apply --record -f  manifests/frontend-deployment.yaml
```

Run the following to see that the Pods are being updated

```
kubectl get pods -l tier=frontend
```

You should see some Pods being terminated and new Pods being created

```
NAME                              READY     STATUS              RESTARTS   AGE
frontend-56d4ff456b-jpdhk         0/1       ContainerCreating   0          0s
frontend-56d4ff456b-pv2m2         1/1       Running             0          9s
frontend-56d4ff456b-rbz5p         1/1       Running             0          19s
frontend-56f7975f44-fgxk8         1/1       Running             0          7m
frontend-56f7975f44-j76lw         1/1       Terminating         0          7m
redis-master-6b464554c8-jdxhk     1/1       Running             0          11m
redis-slave-b58dc4644-crbfs       1/1       Running             0          10m
redis-slave-b58dc4644-htwkm       1/1       Running             0          10m
```

Great! Now you can confirm it updated to `v5`

```
kubectl describe deployment frontend | grep Image
```

Now that you're successfully running `v5` update the YAML file to `v6` and deploy it. Do not forget to use `--record`

After the update has completed confirm it is running `v6`

```
kubectl describe deployment frontend | grep Image
```

Rollback deployment

Now let's say that something went wrong during our update, and we need to rollback to a

previous version of our application.

As long as we used the `--record` option when deploying this is easy.

Run the following to check the rollout history

```
kubectl rollout history deployment frontend
```

```
REVISION  CHANGE-CAUSE
1         kubectl apply --record=true --filename=manifests/frontend-
deployment.yaml
2         kubectl apply --record=true --filename=manifests/frontend-
deployment.yaml
3         kubectl apply --record=true --filename=manifests/frontend-
deployment.yaml
```

To see the changes made for each revision we can run the following, replacing `--revision` with the one you want to know more about

```
kubectl rollout history deployment frontend --revision=2
```

Now to rollback to our previous revision we can run:

```
kubectl rollout undo deployment frontend
```

If we needed to choose a version previous to our last we can specify it:

```
kubectl rollout undo deployment frontend --to-revision=1
```

What does the rollout history look like now?

```
kubectl rollout history deployment frontend
```

Remember when you rolled back the previous version it changed the order of deployment revisions.

Use the following command to see details about each deployment revision, replacing `<number>` with the actual revision number.

```
kubectl rollout history deployment/frontend --revision=<number>
```

Cleanup

To clean up everything run

```
kubectl delete deployment -l app=redis
kubectl delete service -l app=redis
kubectl delete deployment -l app=guestbook
kubectl delete service -l app=guestbook
```

Confirm everything was deleted

```
kubectl get pods
```

Lab Complete