

Deploy multi-tier application

This lab shows you how to build and deploy 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.

Download files

Log into Kubernetes master server and download the required files.

```
wget https://kubernetes.io/docs/tutorials/stateless-application/guestbook/redis-master-deployment.yaml
wget https://kubernetes.io/docs/tutorials/stateless-application/guestbook/redis-master-service.yaml
wget https://kubernetes.io/docs/tutorials/stateless-application/guestbook/redis-slave-deployment.yaml
wget https://kubernetes.io/docs/tutorials/stateless-application/guestbook/redis-slave-service.yaml
wget https://kubernetes.io/docs/tutorials/stateless-application/guestbook/frontend-deployment.yaml
wget https://kubernetes.io/docs/tutorials/stateless-application/guestbook/frontend-service.yaml
```

Start up Redis Master

The guestbook application uses Redis to store its data. It writes its 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 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 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 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 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

```
kubectl apply -f 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-lpxpj	1/1	Running	0	27s
frontend-67f65745c-ts9k	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 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
```

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
```