**This tutorial shows you how to build and deploy a simple, multi-tier web application using Kubernetes and Docker.**

This example consists of the following components:

- A single-instance Redis master to store guestbook entries
- Multiple replicated Redis instances to serve reads
- Multiple web frontend instances

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.

**Login to your AWS Instance and make a dir /home/<your-user-name>/guestbook-application**

$ cd /home/<your-username>
$ mkdir application
$ cd guestbook-application
$ git clone https://github.com/LovesCloud/k8s_guestbook_app.git

Before proceeding please edit **(vim)** all the below yaml files so that you can identify your deployments and services once they are provisioned.

| | |
|---|---|
| **r**edis-mater-deployment.yaml | **- change name under metadata to redis-master-deployment<your-name>** |
| redis-master-service.yaml | **- change name under metadata to redis-master-service-<your-name>** |
| redis-slave-deployment.yaml | **- change name under metadata to redis-slave-deployment-<your-name>** |
| redis-slave-service.yaml | **- change name under metadata to redis-slave-service<your-name>** |
| frontend-deployment.yaml | **- change name under metadata to frontend-deployment-<your-name>** |
| frontend-service.yaml | **- change name under metadata to frontend-service-<your-name>** |

# Start up the Redis Master

## Creating the Redis Master Deployment

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

1. Cd to the directory you downloaded the manifest files.
2. Apply the Redis Master Deployment from the `redis-master-deployment.yaml` file:

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

3. Query the list of Pods to verify that the Redis Master Pod is running:

   ```
   kubectl get pods
   ```

**The response should be similar to this:**

```
NAME                            READY     STATUS    RESTARTS   AGE
redis-master-1068406935-3lswp   1/1       Running   0          28s
```

4. Run the following command to view the logs from the Redis Master Pod:

   ```
   kubectl logs -f POD-NAME
   ```

## Creating 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.

1. Apply the Redis Master Service from the following `redis-master-service.yaml` file:

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

2. Query the list of Services to verify that the Redis Master Service is running: `kubectl get service`

3. The response should be similar to this:

```
NAME           TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
kubernetes     ClusterIP   10.0.0.1      <none>        443/TCP    1m
redis-master   ClusterIP   10.0.0.151    <none>        6379/TCP   8s
```

# 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.

## Creating the 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.

1. Apply the Redis Slave Deployment from the `redis-slave-deployment.yaml` file

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

2. Query the list of Pods to verify that the Redis Slave Pods are running:

   ```
   kubectl get pods
   ```

   The response should be similar to this:

   ```
   NAME                            READY     STATUS              RESTARTS   AGE
   redis-master-1068406935-3lswp   1/1       Running             0          1m
   redis-slave-2005841000-fpvqc    0/1       ContainerCreating   0          6s
   redis-slave-2005841000-phfv9    0/1       ContainerCreating   0          6s
   ```

## Creating the 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

1. **Apply the Redis Slave Service from the following `redis-slave-service.yaml` file:**

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

2. **Query the list of Services to verify that the Redis slave service is running:**

   ```
   kubectl get services
   ```

   **The response should be similar to this:**

   ```
   NAME           TYPE        CLUSTER-IP    EXTERNAL-IP   PORT(S)    AGE
   kubernetes     ClusterIP   10.0.0.1      <none>        443/TCP    2m
   redis-master   ClusterIP   10.0.0.151    <none>        6379/TCP   1m
   redis-slave    ClusterIP   10.0.0.223    <none>        6379/TCP   6s
   ```

# Set up 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.

## Creating the Guestbook Frontend Deployment

1. Apply the frontend Deployment from the **`frontend-deployment.yaml`** file:

   ```
   kubectl apply -f frontend-deployment.yaml
   ```

2. Query the list of Pods to verify that the three frontend replicas are running:

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

3. The response should be similar to this:

   ```
   NAME                        READY     STATUS    RESTARTS   AGE
   frontend-3823415956-dsvc5   1/1       Running   0          54s
   frontend-3823415956-k22zn   1/1       Running   0          54s
   frontend-3823415956-w9gbt   1/1       Running   0          54s
   ```

## Creating 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.

1. Apply the frontend Service from the `frontend-service.yaml` file:

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

2. Query the list of Services to verify that the frontend Service is running:

   ```
   kubectl get services
   ```

3. The response should be similar to this:

   ```
   NAME           TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)          AGE
   frontend     LoadBalancer   10.0.0.112    <none>        80:31323/TCP     6s
   kubernetes     ClusterIP    10.0.0.1      <none>        443/TCP          4m
   redis-master   ClusterIP    10.0.0.151    <none>        6379/TCP         2m
   redis-slave    ClusterIP    10.0.0.223    <none>        6379/TCP         1m
   ```

## Viewing the Frontend Service via `Load Balancer`

If you deployed the `frontend-service.yaml` manifest with type: `LoadBalancer` you need to find the IP address to view your Guestbook.

1. Run the following command to get the IP address for the frontend Service.

   ```
   kubectl get service frontend-<your-name>
   ```

2. The response should be similar to this:

```
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP                PORT(S)        AGE
frontend-arshad    LoadBalancer   100.68.164.181   a898.elb.amazonaws.com   80:31128/TCP   19s
```

3. Copy the external IP address, and load the page in your browser to view your guestbook.

# Scale the Web Frontend

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

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

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

2. Query the list of Pods to verify the number of frontend Pods running:

```
kubectl get pods
```

   The response should look similar to this:

```
NAME                             READY     STATUS    RESTARTS   AGE
frontend-3823415956-70qj5        1/1       Running   0          5s
frontend-3823415956-dsvc5        1/1       Running   0          54m
frontend-3823415956-k22zn        1/1       Running   0          54m
frontend-3823415956-w9gbt        1/1       Running   0          54m
frontend-3823415956-x2pld        1/1       Running   0          5s
redis-master-1068406935-3lswp    1/1       Running   0          56m
redis-slave-2005841000-fpvqc     1/1       Running   0          55m
redis-slave-2005841000-phfv9     1/1       Running   0          55m
```

3. Run the following command to **scale down** the number of frontend Pods:

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

4. Query the list of Pods to verify the number of frontend Pods running:

   ```
   kubectl get pods
   ```

   The response should look similar to this:

   ```
   NAME                             READY     STATUS    RESTARTS   AGE
   frontend-3823415956-k22zn        1/1       Running   0          1h
   frontend-3823415956-w9gbt        1/1       Running   0          1h
   redis-master-1068406935-3lswp    1/1       Running   0          1h
   redis-slave-2005841000-fpvqc     1/1       Running   0          1h
   redis-slave-2005841000-phfv9     1/1       Running   0          1h
   ```

# Cleaning up

Deleting the Deployments and Services also deletes any running Pods. Use labels to delete multiple resources with one command.

1. Run the following commands to delete all Pods, Deployments, and Services.

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

2. The responses should be:

   ```
   deployment.apps "redis-master" deleted
   deployment.apps "redis-slave" deleted
   service "redis-master" deleted
   service "redis-slave" deleted
   deployment.apps "frontend" deleted
   service "frontend" deleted
   ```

3. Query the list of Pods to verify that no Pods are running:

   ```
   kubectl get pods
   ```

4. The response should be this:

   ```
   No resources found.
   ```