**Deploying PHP Guestbook application with Redis**

This tutorial shows you how to build and deploy a simple, multi-tier web application using Kubernetes and [Docker](https://www.docker.com/). This example consists of the following components:

* A single-instance [Redis](https://redis.io/) master to store guestbook entries
* Multiple [replicated Redis](https://redis.io/topics/replication) instances to serve reads
* Multiple web frontend instances
* [Objectives](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#objectives)
* [Before you begin](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#before-you-begin)
* [Start up the Redis Master](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#start-up-the-redis-master)
* [Start up the Redis Slaves](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#start-up-the-redis-slaves)
* [Set up and Expose the Guestbook Frontend](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#set-up-and-expose-the-guestbook-frontend)
* [Scale the Web Frontend](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#scale-the-web-frontend)
* [Cleaning up](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#cleaning-up)
* [What's next](https://kubernetes.io/docs/tutorials/stateless-application/guestbook/#what-s-next)

**Objectives**

* Start up a Redis master.
* Start up Redis slaves.
* Start up the guestbook frontend.
* Expose and view the Frontend Service.
* Clean up.

**Before you begin**

You need to have a Kubernetes cluster, and the kubectl command-line tool must be configured to communicate with your cluster. If you do not already have a cluster, you can create one by using [Minikube](https://kubernetes.io/docs/setup/minikube), or you can use one of these Kubernetes playgrounds:

* [Katacoda](https://www.katacoda.com/courses/kubernetes/playground)
* [Play with Kubernetes](http://labs.play-with-k8s.com/)

To check the version, enter kubectl version.

**Start up the 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.

| [**application/guestbook/redis-master-deployment.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/redis-master-deployment.yaml) |
| --- |
| apiVersion: apps/v1 *# for versions before 1.9.0 use apps/v1beta2*  kind: Deployment  metadata:  name: redis-master  labels:  app: redis  spec:  selector:  matchLabels:  app: redis  role: master  tier: backend  replicas: 1  template:  metadata:  labels:  app: redis  role: master  tier: backend  spec:  containers:  - name: master  image: k8s.gcr.io/redis:e2e *# or just image: redis*  resources:  requests:  cpu: 100m  memory: 100Mi  ports:  - containerPort: 6379 |

1. Launch a terminal window in the directory you downloaded the manifest files.
2. Apply the Redis Master Deployment from the redis-master-deployment.yaml file:

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-master-deployment.yaml

1. 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

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

kubectl logs -f POD-NAME

**Note:** Replace POD-NAME with the name of your Pod.

**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](https://kubernetes.io/docs/concepts/services-networking/service/) to proxy the traffic to the Redis master Pod. A Service defines a policy to access the Pods.

| [**application/guestbook/redis-master-service.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/redis-master-service.yaml) |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: redis-master  labels:  app: redis  role: master  tier: backend  spec:  ports:  - port: 6379  targetPort: 6379  selector:  app: redis  role: master  tier: backend |

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

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-master-service.yaml

1. Query the list of Services to verify that the Redis Master Service is running:

kubectl get service

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

**Note:** 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.

**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.

| [**application/guestbook/redis-slave-deployment.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/redis-slave-deployment.yaml) |
| --- |
| apiVersion: apps/v1 *# for versions before 1.9.0 use apps/v1beta2*  kind: Deployment  metadata:  name: redis-slave  labels:  app: redis  spec:  selector:  matchLabels:  app: redis  role: slave  tier: backend  replicas: 2  template:  metadata:  labels:  app: redis  role: slave  tier: backend  spec:  containers:  - name: slave  image: gcr.io/google\_samples/gb-redisslave:v1  resources:  requests:  cpu: 100m  memory: 100Mi  env:  - name: GET\_HOSTS\_FROM  value: dns  *# Using `GET\_HOSTS\_FROM=dns` requires your cluster to*  *# provide a dns service. As of Kubernetes 1.3, DNS is a built-in*  *# service launched automatically. However, if the cluster you are using*  *# does not have a built-in DNS service, you can instead*  *# access an environment variable to find the master*  *# service's host. To do so, comment out the 'value: dns' line above, and*  *# uncomment the line below:*  *# value: env*  ports:  - containerPort: 6379 |

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

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-slave-deployment.yaml

1. 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.

| [**application/guestbook/redis-slave-service.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/redis-slave-service.yaml) |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: redis-slave  labels:  app: redis  role: slave  tier: backend  spec:  ports:  - port: 6379  selector:  app: redis  role: slave  tier: backend |

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

kubectl apply -f https://k8s.io/examples/application/guestbook/redis-slave-service.yaml

1. 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**

| [**application/guestbook/frontend-deployment.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/frontend-deployment.yaml) |
| --- |
| apiVersion: apps/v1 *# for versions before 1.9.0 use apps/v1beta2*  kind: Deployment  metadata:  name: frontend  labels:  app: guestbook  spec:  selector:  matchLabels:  app: guestbook  tier: frontend  replicas: 3  template:  metadata:  labels:  app: guestbook  tier: frontend  spec:  containers:  - name: php-redis  image: gcr.io/google-samples/gb-frontend:v4  resources:  requests:  cpu: 100m  memory: 100Mi  env:  - name: GET\_HOSTS\_FROM  value: dns  *# Using `GET\_HOSTS\_FROM=dns` requires your cluster to*  *# provide a dns service. As of Kubernetes 1.3, DNS is a built-in*  *# service launched automatically. However, if the cluster you are using*  *# does not have a built-in DNS service, you can instead*  *# access an environment variable to find the master*  *# service's host. To do so, comment out the 'value: dns' line above, and*  *# uncomment the line below:*  *# value: env*  ports:  - containerPort: 80 |

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

kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-deployment.yaml

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

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

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](https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services---service-types). 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. Minikube can only expose Services through NodePort.

**Note:** Some cloud providers, like Google Compute Engine or Google Kubernetes Engine, support external load balancers. If your cloud provider supports load balancers and you want to use it, simply delete or comment out type: NodePort, and uncomment type: LoadBalancer.

| [**application/guestbook/frontend-service.yaml**](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/frontend-service.yaml) |
| --- |
| apiVersion: v1  kind: Service  metadata:  name: frontend  labels:  app: guestbook  tier: frontend  spec:  *# comment or delete the following line if you want to use a LoadBalancer*  type: NodePort  *# if your cluster supports it, uncomment the following to automatically create*  *# an external load-balanced IP for the frontend service.*  *# type: LoadBalancer*  ports:  - port: 80  selector:  app: guestbook  tier: frontend |

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

kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-service.yaml

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

kubectl get services

The response should be similar to this:

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

frontend ClusterIP 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 NodePort**

If you deployed this application to Minikube or a local cluster, 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.

minikube service frontend --url

The response should be similar to this:

http://192.168.99.100:31323

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

**Viewing the Frontend Service via LoadBalancer**

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

The response should be similar to this:

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

frontend ClusterIP 10.51.242.136 109.197.92.229 80:32372/TCP 1m

1. 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

1. 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

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

kubectl scale deployment frontend --replicas=2

1. 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.
2. kubectl delete deployment -l app=redis
3. kubectl delete service -l app=redis
4. kubectl delete deployment -l app=guestbook

kubectl delete service -l app=guestbook

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

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

kubectl get pods

The response should be this:

No resources found.