**Deploying PHP Guestbook application with MongoDB**

This tutorial shows you how to build and deploy a simple *(not production ready)*, multi-tier web application using Kubernetes and [Docker](https://www.docker.com/). This example consists of the following components:

* A single-instance [MongoDB](https://www.mongodb.com/) to store guestbook entries
* Multiple web frontend instances

Objectives

* Start up a Mongo database.
* 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/tasks/tools/" \l "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/)

Your Kubernetes server must be at or later than version v1.14. To check the version, enter kubectl version.

Start up the Mongo Database

The guestbook application uses MongoDB to store its data.

Creating the Mongo Deployment

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

[application/guestbook/mongo-deployment.yaml](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/mongo-deployment.yaml)

**apiVersion**: apps/v1

**kind**: Deployment

**metadata**:

**name**: mongo

**labels**:

**app.kubernetes.io/name**: mongo

**app.kubernetes.io/component**: backend

**spec**:

**selector**:

**matchLabels**:

**app.kubernetes.io/name**: mongo

**app.kubernetes.io/component**: backend

**replicas**: 1

**template**:

**metadata**:

**labels**:

**app.kubernetes.io/name**: mongo

**app.kubernetes.io/component**: backend

**spec**:

**containers**:

- **name**: mongo

**image**: mongo:4.2

**args**:

- --bind\_ip

- 0.0.0.0

**resources**:

**requests**:

**cpu**: 100m

**memory**: 100Mi

**ports**:

- **containerPort**: 27017

1. Launch a terminal window in the directory you downloaded the manifest files.
2. Apply the MongoDB Deployment from the mongo-deployment.yaml file:
3. kubectl apply -f https://k8s.io/examples/application/guestbook/mongo-deployment.yaml
4. Query the list of Pods to verify that the MongoDB Pod is running:
5. kubectl get pods

The response should be similar to this:

NAME READY STATUS RESTARTS AGE

mongo-5cfd459dd4-lrcjb 1/1 Running 0 28s

1. Run the following command to view the logs from the MongoDB Deployment:
2. kubectl logs -f deployment/mongo

Creating the MongoDB Service

The guestbook application needs to communicate to the MongoDB to write its data. You need to apply a [Service](https://kubernetes.io/docs/concepts/services-networking/service/) to proxy the traffic to the MongoDB Pod. A Service defines a policy to access the Pods.

[application/guestbook/mongo-service.yaml](https://raw.githubusercontent.com/kubernetes/website/master/content/en/examples/application/guestbook/mongo-service.yaml)

**apiVersion**: v1

**kind**: Service

**metadata**:

**name**: mongo

**labels**:

**app.kubernetes.io/name**: mongo

**app.kubernetes.io/component**: backend

**spec**:

**ports**:

- **port**: 27017

**targetPort**: 27017

**selector**:

**app.kubernetes.io/name**: mongo

**app.kubernetes.io/component**: backend

1. Apply the MongoDB Service from the following mongo-service.yaml file:
2. kubectl apply -f https://k8s.io/examples/application/guestbook/mongo-service.yaml
3. Query the list of Services to verify that the MongoDB Service is running:
4. 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

mongo ClusterIP 10.0.0.151 <none> 27017/TCP 8s

**Note:** This manifest file creates a Service named mongo with a set of labels that match the labels previously defined, so the Service routes network traffic to the MongoDB Pod.

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 mongo Service to store Guestbook entries.

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

**kind**: Deployment

**metadata**:

**name**: frontend

**labels**:

**app.kubernetes.io/name**: guestbook

**app.kubernetes.io/component**: frontend

**spec**:

**selector**:

**matchLabels**:

**app.kubernetes.io/name**: guestbook

**app.kubernetes.io/component**: frontend

**replicas**: 3

**template**:

**metadata**:

**labels**:

**app.kubernetes.io/name**: guestbook

**app.kubernetes.io/component**: frontend

**spec**:

**containers**:

- **name**: guestbook

**image**: paulczar/gb-frontend:v5

*# image: gcr.io/google-samples/gb-frontend:v4*

**resources**:

**requests**:

**cpu**: 100m

**memory**: 100Mi

**env**:

- **name**: GET\_HOSTS\_FROM

**value**: dns

**ports**:

- **containerPort**: 80

1. Apply the frontend Deployment from the frontend-deployment.yaml file:
2. kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-deployment.yaml
3. Query the list of Pods to verify that the three frontend replicas are running:
4. kubectl get pods -l app.kubernetes.io/name=guestbook -l app.kubernetes.io/component=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 mongo Services you applied is only accessible within the Kubernetes cluster because the default type for a Service is [ClusterIP](https://kubernetes.io/docs/concepts/services-networking/service/" \l "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 Kubernetes cluster. However a Kubernetes user you can use kubectl port-forward to access the service even though it uses a ClusterIP.

**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, 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.kubernetes.io/name**: guestbook

**app.kubernetes.io/component**: frontend

**spec**:

*# 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.kubernetes.io/name**: guestbook

**app.kubernetes.io/component**: frontend

1. Apply the frontend Service from the frontend-service.yaml file:
2. kubectl apply -f https://k8s.io/examples/application/guestbook/frontend-service.yaml
3. Query the list of Services to verify that the frontend Service is running:
4. 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/TCP 6s

kubernetes ClusterIP 10.0.0.1 <none> 443/TCP 4m

mongo ClusterIP 10.0.0.151 <none> 6379/TCP 2m

Viewing the Frontend Service via kubectl port-forward

1. Run the following command to forward port 8080 on your local machine to port 80 on the service.
2. kubectl port-forward svc/frontend 8080:80

The response should be similar to this:

Forwarding from 127.0.0.1:8080 -> 80

Forwarding from [::1]:8080 -> 80

1. load the page [http://localhost:8080](http://localhost:8080/) 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.
2. kubectl get service frontend

The response should be similar to this:

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

frontend LoadBalancer 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

You can scale up or down as needed 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:
2. kubectl scale deployment frontend --replicas=5
3. Query the list of Pods to verify the number of frontend Pods running:
4. 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

mongo-1068406935-3lswp 1/1 Running 0 56m

1. Run the following command to scale down the number of frontend Pods:
2. kubectl scale deployment frontend --replicas=2
3. Query the list of Pods to verify the number of frontend Pods running:
4. 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

mongo-1068406935-3lswp 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.kubernetes.io/name=mongo
3. kubectl delete service -l app.kubernetes.io/name=mongo
4. kubectl delete deployment -l app.kubernetes.io/name=guestbook
5. kubectl delete service -l app.kubernetes.io/name=guestbook

The responses should be:

deployment.apps "mongo" deleted

service "mongo" deleted

deployment.apps "frontend" deleted

service "frontend" deleted

1. Query the list of Pods to verify that no Pods are running:
2. kubectl get pods

The response should be this:

No resources found.

What's next

* Complete the [Kubernetes Basics](https://kubernetes.io/docs/tutorials/kubernetes-basics/) Interactive Tutorials
* Use Kubernetes to create a blog using [Persistent Volumes for MySQL and Wordpress](https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume/#visit-your-new-wordpress-blog)
* Read more about [connecting applications](https://kubernetes.io/docs/concepts/services-networking/connect-applications-service/)
* Read more about [Managing Resources](https://kubernetes.io/docs/concepts/cluster-administration/manage-deployment/#using-labels-effectively)