## Create a Two tier Application using Kubernetes Deployment and Services

### **Creating the backend using a Deployment**

The backend is a simple hello greeter microservice. Here is the configuration file for the backend Deployment:

| [**hello.yaml**](https://raw.githubusercontent.com/kubernetes/kubernetes.github.io/master/docs/tasks/access-application-cluster/hello.yaml) |
| --- |
| **apiVersion: apps/v1beta1**  **kind: Deployment**  **metadata:**  **name: hello**  **spec:**  **replicas: 7**  **template:**  **metadata:**  **labels:**  **app: hello**  **tier: backend**  **track: stable**  **spec:**  **containers:**  **- name: hello**  **image: "gcr.io/google-samples/hello-go-gke:1.0"**  **ports:**  **- name: http**  **containerPort: 80** |

Create the backend Deployment:

**kubectl create -f https://k8s.io/docs/tasks/access-application-cluster/hello.yaml**

View information about the backend Deployment:

**kubectl describe deployment hello**

The output is similar to this:

**Name: hello**

**Namespace: default**

**CreationTimestamp: Mon, 24 Oct 2016 14:21:02 -0700**

**Labels: app=hello**

**tier=backend**

**track=stable**

**Annotations: deployment.kubernetes.io/revision=1**

**Selector: app=hello,tier=backend,track=stable**

**Replicas: 7 desired | 7 updated | 7 total | 7 available | 0 unavailable**

**StrategyType: RollingUpdate**

**MinReadySeconds: 0**

**RollingUpdateStrategy: 1 max unavailable, 1 max surge**

**Pod Template:**

**Labels: app=hello**

**tier=backend**

**track=stable**

**Containers:**

**hello:**

**Image: "gcr.io/google-samples/hello-go-gke:1.0"**

**Port: 80/TCP**

**Environment: <none>**

**Mounts: <none>**

**Volumes: <none>**

**Conditions:**

**Type Status Reason**

**---- ------ ------**

**Available True MinimumReplicasAvailable**

**Progressing True NewReplicaSetAvailable**

**OldReplicaSets: <none>**

**NewReplicaSet: hello-3621623197 (7/7 replicas created)**

**Events:**

**...**

### **Creating the backend Service object**

The key to connecting a frontend to a backend is the backend Service. A Service creates a persistent IP address and DNS name entry so that the backend microservice can always be reached. A Service uses selector labels to find the Pods that it routes traffic to.

First, explore the Service configuration file:

| [**hello-service.yaml**](https://raw.githubusercontent.com/kubernetes/kubernetes.github.io/master/docs/tasks/access-application-cluster/hello-service.yaml) |
| --- |
| **kind: Service**  **apiVersion: v1**  **metadata:**  **name: hello**  **spec:**  **selector:**  **app: hello**  **tier: backend**  **ports:**  **- protocol: TCP**  **port: 80**  **targetPort: http** |

In the configuration file, you can see that the Service routes traffic to Pods that have the labels **app: hello** and **tier: backend**.

Create the **hello** Service:

**kubectl create -f https://k8s.io/docs/tasks/access-application-cluster/hello-service.yaml**

At this point, you have a backend Deployment running, and you have a Service that can route traffic to it.

### **Creating the frontend**

Now that you have your backend, you can create a frontend that connects to the backend. The frontend connects to the backend worker Pods by using the DNS name given to the backend Service. The DNS name is “hello”, which is the value of the **name** field in the preceding Service configuration file.

The Pods in the frontend Deployment run an nginx image that is configured to find the hello backend Service. Here is the nginx configuration file:

| [**frontend/frontend.conf**](https://raw.githubusercontent.com/kubernetes/kubernetes.github.io/master/docs/tasks/access-application-cluster/frontend/frontend.conf) |
| --- |
| **upstream hello {**  **server hello;**  **}**  **server {**  **listen 80;**  **location / {**  **proxy\_pass http://hello;**  **}**  **}** |

Similar to the backend, the frontend has a Deployment and a Service. The configuration for the Service has **type: LoadBalancer**, which means that the Service uses the default load balancer of your cloud provider.

| [**frontend.yaml**](https://raw.githubusercontent.com/kubernetes/kubernetes.github.io/master/docs/tasks/access-application-cluster/frontend.yaml) |
| --- |
| **kind: Service**  **apiVersion: v1**  **metadata:**  **name: frontend**  **spec:**  **selector:**  **app: hello**  **tier: frontend**  **ports:**  **- protocol: "TCP"**  **port: 80**  **targetPort: 80**  **type: LoadBalancer**  **---**  **apiVersion: apps/v1beta1**  **kind: Deployment**  **metadata:**  **name: frontend**  **spec:**  **replicas: 1**  **template:**  **metadata:**  **labels:**  **app: hello**  **tier: frontend**  **track: stable**  **spec:**  **containers:**  **- name: nginx**  **image: "gcr.io/google-samples/hello-frontend:1.0"**  **lifecycle:**  **preStop:**  **exec:**  **command: ["/usr/sbin/nginx","-s","quit"]** |

Create the frontend Deployment and Service:

**kubectl create -f https://k8s.io/docs/tasks/access-application-cluster/frontend.yaml**

The output verifies that both resources were created:

**deployment "frontend" created**

**service "frontend" created**

**Note**: The nginx configuration is baked into the [container image](https://kubernetes.io/docs/tasks/access-application-cluster/frontend/Dockerfile). A better way to do this would be to use a [ConfigMap](https://kubernetes.io/docs/tasks/configure-pod-container/configmap/), so that you can change the configuration more easily.

### **Interact with the frontend Service**

Once you’ve created a Service of type LoadBalancer, you can use this command to find the external IP:

**kubectl get service frontend**

The external IP field may take some time to populate. If this is the case, the external IP is listed as **<pending>**.

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

**frontend 10.51.252.116 <pending> 80/TCP 10s**

Repeat the same command again until it shows an external IP address:

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

**frontend 10.51.252.116 XXX.XXX.XXX.XXX 80/TCP 1m**

### **Send traffic through the frontend**

The frontend and backends are now connected. You can hit the endpoint by using the curl command on the external IP of your frontend Service.

**curl http://<EXTERNAL-IP>**

The output shows the message generated by the backend:

**{"message":"Hello"}**