Certified Tech Developer
The Ultimate Degree

Infraestructura III

- Actividad grupal
- Nivel de complejidad: medio 🔥🔥

Consigna

Crear e identificar los recursos y los objetos de Kubernetes.

Pasos:

1. Descargar los manifiestos:

   # git clone https://github.com/damiancolaneri/dh-k8s.git

2. Ingresamos a la carpeta de la clase 10

   # cd dh-k8s/clase10

3. Creamos un namespace para nuestra app

   # kubectl create namespace app

4. Hacemos el deployment del frontend dentro del namespace creado

   # kubectl apply -f frontend-deployment.yaml -n app

5. Hacemos el deployment del backend el cual es una DB redis dentro del namespace creado

   # kubectl apply -f redis-master-deployment.yaml -n app

6. Hacemos el deployment de una DB redis slave dentro del namespace creado

   # kubectl apply -f redis-slave-deployment.yaml -n app

7. Veamos nuestros recursos creados:

#kubectl get deployment

```
→  clase10 kubectl get deployment
No resources found in default namespace.
→  clase10
```

No hay nada? 👀Por defecto si no especificamos un namespace nos mostrará el contenido de namespace default, nuestra app la desplegamos en el namespace “app”, nunca debemos olvidar especificar a qué namespace apuntamos. Veamos la forma correcta:

kubectl get deployment -n app

```
→  clase10 kubectl get deployment -n app
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
guestbook      3/3     3            3           32m
redis-master   1/1     1            1           31m
redis-slave    2/2     2            2           31m
→  clase10
```

Como vemos, cada uno de esos deployment esta formado 1 o varios pods según el caso, como los vemos?

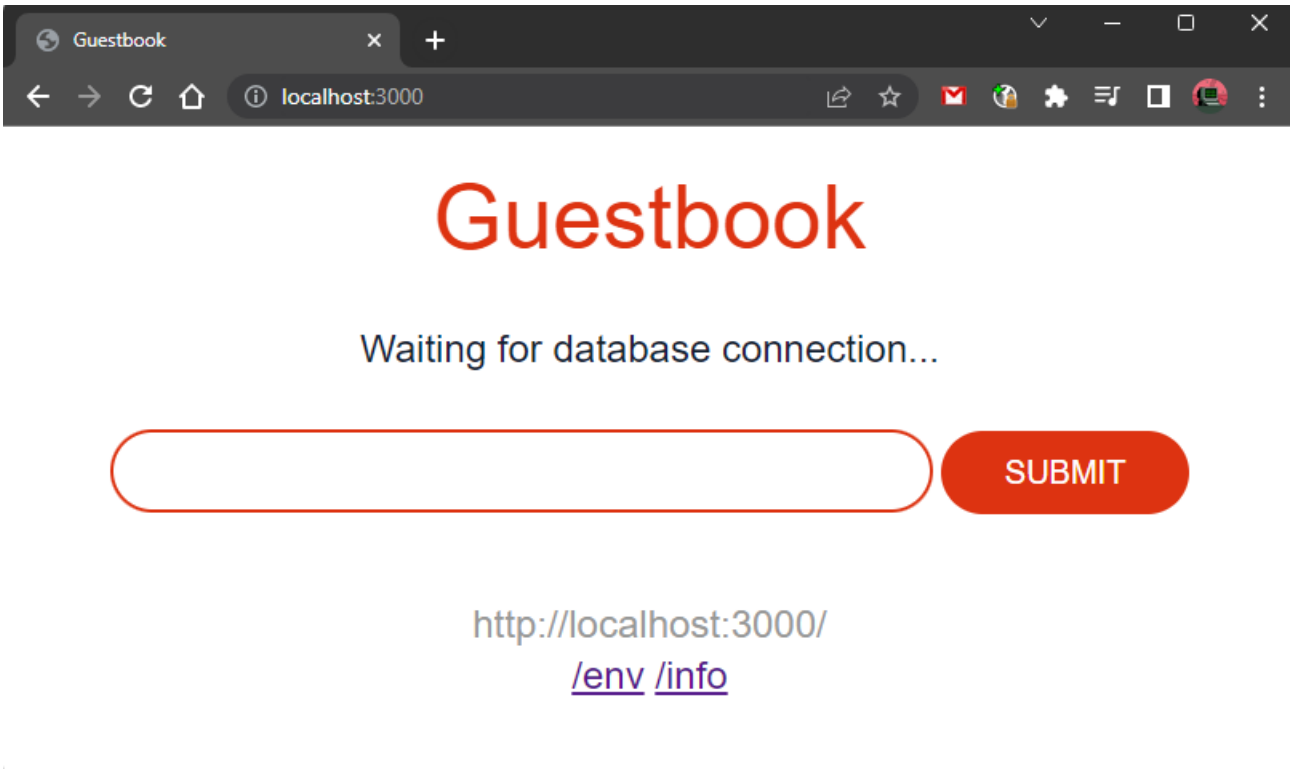
#kubectl get pods -n app

```
→  clase10 k get pods -n app
NAME                           READY   STATUS    RESTARTS   AGE
guestbook-65f547f597-5ft8c     1/1     Running   0          34m
guestbook-65f547f597-chndv     1/1     Running   0          12m
guestbook-65f547f597-mq5w4     1/1     Running   0          34m
redis-master-7764fcd499-8pg6j  1/1     Running   0          34m
redis-slave-74d9f75698-gzsv2   1/1     Running   0          34m
redis-slave-74d9f75698-tpspq   1/1     Running   0          34m
→  clase10
```

8. Ahora corremos nuestra app para ver cómo funciona:

kubectl port-forward `--address 0.0.0.0` deployment/guestbook -n app 3000:3000

9. Accedemos de nuestro navegador al localhost:3000



Como vemos, nuestro frontend funciona, pero no se está conectando a la DB 🤔

No se conecta porque no la está encontrando, y esto es porque nos falta crear más recursos, debemos crear los servicios asociados a los deployments 😅

10. Creamos el servicio para el frontend dentro del namespace creado

    # kubectl apply -f frontend-svc.yaml -n app

11. Creamos el servicio para la db redis master dentro del namespace creado

    # kubectl apply -f redis-master-svc.yaml -n app

12. Creamos el servicio para la db redis slave dentro del namespace creado

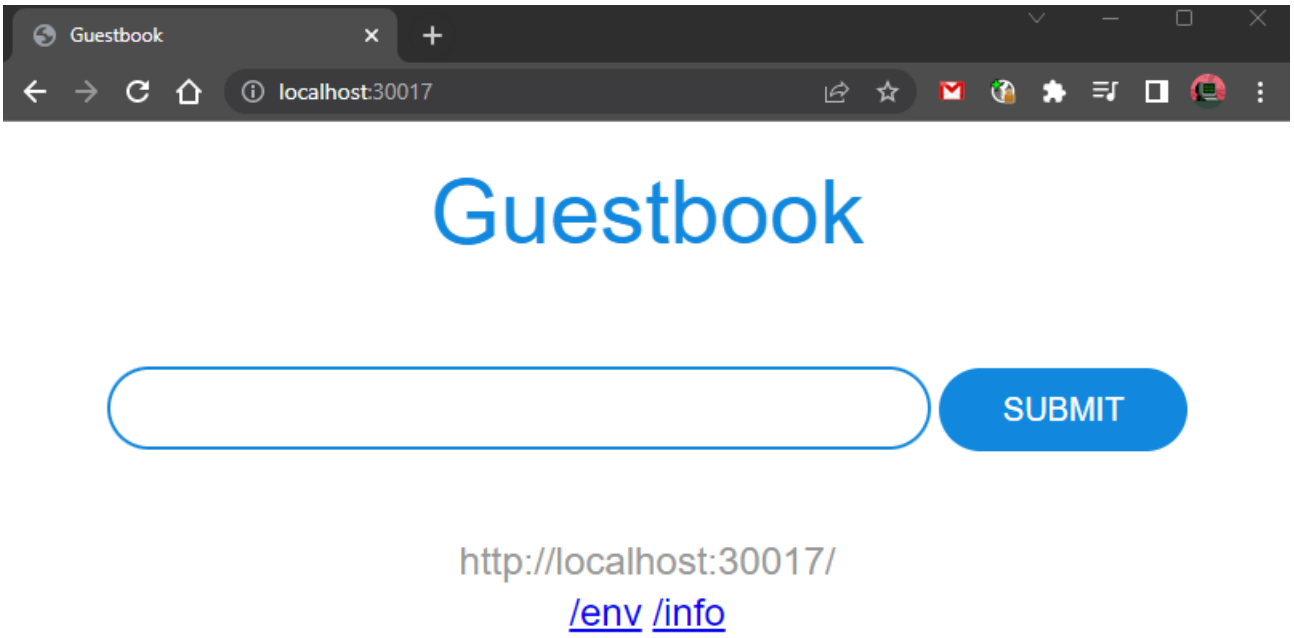
    # kubectl apply -f redis-slaver-svc.yaml -n app

13. Comprobamos que los servicios estén activos y vemos el puerto que nos asignó para el frontend

    # kubectl get services -n app

```
→  clase10 kubectl get services -n app
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
frontend       NodePort    10.102.210.220   <none>        80:30017/TCP   103s
redis-master   ClusterIP   10.100.18.160    <none>        6379/TCP       89s
redis-slave    ClusterIP   10.106.40.42     <none>        6379/TCP       85s
→  clase10
```

    Vemos que, en mi caso, me asignó el puerto 30017, con esto puedo acceder desde el navegador y ver si ahora está funcionando correctamente.



    Efectivamente, ya podemos ver que ingresamos y no tenemos error de conexión con la DB. 🚀

Los que estén usando una EC2 deben ejecutar el siguiente comando para acceder correctamente, cambiando el puerto port el que corresponda
kubectl port-forward --address 0.0.0.0 service/frontend -n app 30017:80

Y el acceso en el navegador será IP_PUBLICA_EC2:PUERTO

14. Nuestro frontend está compuesto por 2 réplicas, vamos a aumentarlas a 3, para esto debemos modificar el manifiesto de deployment del mismo.

    # nano frontend-deployment.yaml

    y cambiamos el valor de réplicas por 3, guardamos y salimos

15. Aplicamos el cambio que acabamos de realizar

    # kubectl apply -f frontend-deployment.yaml -n app

16. comprobemos nuestras replicas ahora

    # kubectl get replicaset -n app

```
→  clase10 kubectl get replicaset -n app
NAME                     DESIRED   CURRENT   READY   AGE
guestbook-65f547f597     3         3         3       21m
redis-master-7764fcd499  1         1         1       21m
redis-slave-74d9f75698   2         2         2       21m
→  clase10
```

    Ahora tenemos 3.

17. Veamos ahora todo lo que creamos

    # kubectl get all -n app

```
→  clase10 kubectl get all -n app
NAME                                READY   STATUS    RESTARTS   AGE
pod/guestbook-65f547f597-5ft8c      1/1     Running   0          41m
pod/guestbook-65f547f597-chndv      1/1     Running   0          19m
pod/guestbook-65f547f597-mq5w4      1/1     Running   0          41m
pod/redis-master-7764fcd499-8pg6j   1/1     Running   0          41m
pod/redis-slave-74d9f75698-gzsv2    1/1     Running   0          41m
pod/redis-slave-74d9f75698-tpspq    1/1     Running   0          41m

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
service/frontend       NodePort    10.102.210.220   <none>        80:30017/TCP   29m
service/redis-master   ClusterIP   10.100.18.160    <none>        6379/TCP       29m
service/redis-slave    ClusterIP   10.106.40.42     <none>        6379/TCP       29m

NAME                           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/guestbook      3/3     3            3           41m
deployment.apps/redis-master   1/1     1            1           41m
deployment.apps/redis-slave    2/2     2            2           41m

NAME                                      DESIRED   CURRENT   READY   AGE
replicaset.apps/guestbook-65f547f597      3         3         3       41m
replicaset.apps/redis-master-7764fcd499   1         1         1       41m
replicaset.apps/redis-slave-74d9f75698    2         2         2       41m
→  clase10
```

18. **Identifiquen qué objetos se crearon y cuantos**:
    a. Deployments
    b. ReplicaSets
    c. Statefulsets
    d. Daemonsets
    e. Services
    f. Pods
    g. ConfigMaps
    h. Secrets
    i. namespace
    j. Persistent Volumes

19. Ahora ya podemos eliminar lo creado, para lo cual tenemos opciones.

    Podemos eliminar los recursos 1 a 1: # kubectl delete -f nombre-archivo -n app

    Podemos eliminar todos de una vez: # kubectl delete -f . -n app

Al eliminar el namespace eliminará todo su contenido: # kubectl delete namespace app

Pro Tip: Asi como vimos que se podía eliminar todo de una vez, también se puede crear: # kubectl apply -f . -n app 🙌