Taller Practico Clase #5

1) Implemente la aplicación de muestra do100-probes en el clúster de Kubernetes y exponga la aplicación.
1.1) Cree una nueva implementación mediante kubectl.

```
kubectl create deployment do100-probes --image quay.io/redhattraining/do100-probes:latest
```

1.2) Exponga la implementación en el puerto 8080.

```
kubectl expose deployment/do100-probes --port 8080
```

1.3) Use un editor de texto para crear un archivo en su directorio actual llamado probes-ingress.yml.
Cree el archivo probes-ingress.yml con el siguiente contenido. Asegúrese de que la identacion sea correcta (usando espacios en lugar de tabulaciones) y luego guarde el archivo.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: do100-probes

  labels:

    app: do100-probes

spec:

  rules:

    - host: INGRESS-HOST
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: do100-probes
                port:
                  number: 8080
```

Reemplace INGRESS-HOST con el nombre de host asociado con su clúster de Kubernetes, como hello.example.com o do100-probes-USER-dev.apps.sandbox.x8i5.p1.openshiftapps.com.

El archivo en https://github.com/RedHatTraining/DO100-apps/blob/main/probes/probes-ingress.yml contiene el contenido correcto para el archivo probes-ingress.yml. Puede descargar el archivo y usarlo para comparar.

1.4) Utilice el comando kubectl create para crear el recurso Ingress

```
kubectl create -f probes-ingress.yml
```

2) Pruebe manualmente las aplicaciones `/ready` and `/healthz` endpoints.
2.1) Muestre la informacion del `do100-probes` ingress. Si el commando no muestra una dirección IP, espere un minuto e intente ejecutar el comando nuevamente.

```
kubectl get ingress/do100-probes
```

El valor de la columna HOST coincide con la línea de host especificada en su archivo probes-ingress.yml. Es probable que su dirección IP sea diferente de la que se muestra aquí.
2.2) Pruebe el `/ready` endpoint:

```
curl -i hello.example.com/ready
```

En Windows, elimine el flag `-i`:

```
curl hello.example.com/ready
```

El endpoint /ready simula un inicio lento de la aplicación y, por lo tanto, durante los primeros 30 segundos después de que se inicia la aplicación, devuelve un código de estado HTTP de 503 y la siguiente respuesta:

```
HTTP/1.1 503 Service Unavailable

...output omitted...

Error! Service not ready for requests...
```

Después de que la aplicación se ha estado ejecutando durante 30 segundos, devuelve:

```
HTTP/1.1 200 OK
```

```
...output omitted...

Ready for service requests...
```

2.3) Pruebe el endpoint de la aplicacion `/healthz` :

```
curl -i hello.example.com/healthz
```

En Windows, elimine el flag `-i`:

```
curl hello.example.com/healthz
```

2.4) Pruebe la respuesta de la aplicacion:

```
curl hello.example.com
```

3) Active el readiness y el liveness probes para la aplicacion.

3.1) Utilice el comando de edición de kubectl edit para editar la definición de implementación y agregar readiness y liveness probes.

- Para liveness probe, use el endpoint `/healthz` en el puerto `8080`.
- Para readiness probe, use el endpoint `/ready` en el puerto `8080`.
- Para ambos probes:

  o Configurar un retraso inicial de 2 segundos.

  o Configure el tiempo de espera en 2 segundos.

```
kubectl edit deployment/do100-probes...output omitted..
```

Este comando abre su editor de sistema predeterminado. Realice cambios en la definición para que se muestre de la siguiente manera.

```
...output omitted...

spec:

  ...output omitted...

  template:

    ...output omitted...

    spec:
```

```
      containers:

      - image: quay.io/redhattraining/do100-probes:latest

        ...output omitted...readinessProbe:
          httpGet:
            path: /ready
            port: 8080
          initialDelaySeconds: 2
          timeoutSeconds: 2
        livenessProbe:
          httpGet:
            path: /healthz
            port: 8080
          initialDelaySeconds: 2
          timeoutSeconds: 2
```

Guarde y salga del editor para aplicar sus cambios.

3.2) Verifique el valor en el `livenessProbe` y `readinessProbe` :

```
kubectl describe deployment do100-probes...output omitted...Liveness:     htt
p-get http://:8080/healthz delay=2s timeout=2s period=10s #success=1 #failure
=3
    Readiness:    http-get http://:8080/ready delay=2s timeout=2s period=10s
#success=1 #failure=3
...output omitted...
```

3.3) Espere a que el módulo de la aplicación se vuelva a implementar y cambie al estado LISTO:

```
[user@host ~]$ kubectl get pods

NAME                              READY   STATUS    RESTARTS      AGE

...output omitted...

do100-probes-7794c5cb4f-vwl4x   0/1     Running   0             6s
```

El estado LISTO muestra 0/1 si el valor de Age es inferior a aproximadamente 30 segundos. Después de eso, el estado LISTO es 1/1. Anote el nombre del pod para los siguientes pasos.

```
kubectl get pods

NAME                              READY   STATUS    RESTARTS      AGE

...output omitted...do100-probes-7794c5cb4f-vwl4x1/1     Running   0
62s
```

3.4) Utilice el comando kubectl logs para ver los resultados de las pruebas de actividad y preparación. Utilice el nombre del pod del paso anterior.

```
kubectl logs -f do100-probes-7794c5cb4f-vwl4x...output omitted...
```

```
nodejs server running on http://0.0.0.0:8080

ping /healthz => pong [healthy]

ping /ready => pong [notready]

ping /healthz => pong [healthy]

ping /ready => pong [notready]

ping /healthz => pong [healthy]

ping /ready => pong [ready]

...output omitted...
```

Observe que el readiness probe falla durante unos 30 segundos después de la reimplementación y luego se realiza correctamente. Recuerde que la aplicación simula una inicialización lenta de la aplicación estableciendo a la fuerza un retraso de 30 segundos antes de que responda con un estado de listo.

No finalice este comando. Continuará monitoreando la salida de este comando en el siguiente paso.

4) Simule una falla en la red.

En caso de una falla en la red, un servicio deja de responder. Esto significa que fallan los liveness y readiness probes

Kubernetes puede resolver el problema al recrear el contenedor en un nodo diferente.

4.1) En una ventana o pestaña de terminal diferente, ejecute los siguientes comandos para simular una falla en el liveness probe:

```
 curl http://hello.example.com/flip?op=kill-health

Switched app state to unhealthy...

curl http://hello.example.com/flip?op=kill-ready

Switched app state to not ready...
```

4.2) Regrese a la terminal donde está monitoreando la implementación de la aplicación:

```
 kubectl logs -f do100-probes-7794c5cb4f-vwl4x

...output omitted...

Received kill request for health probe.

Received kill request for readiness probe.

...output omitted...

ping /ready => pong [notready]

ping /healthz => pong [unhealthy]

...output omitted...

Received kill request for health probe.

...output omitted...

Received kill request for readiness probe.

...output omitted...
```

Kubernetes restarts the pod when the liveness probe fails repeatedly (three consecutive failures by default). This means Kubernetes restarts the application on an available node not affected by the network failure.

You see this log output only when you immediately check the application logs after you issue the kill request. If you check the logs after Kubernetes restarts the pod, then the logs are cleared and you only see the output shown in the next step.

Kubernetes reinicia el pod cuando liveness probe falla repetidamente (tres fallas consecutivas de forma predeterminada). Esto significa que Kubernetes reinicia la aplicación en un nodo disponible que no se ve afectado por la falla de la red.

Verá esta salida de logs solo cuando verifique inmediatamente los logs de la aplicación después de emitir la solicitud de finalización. Si revisa los logs después de que Kubernetes reinicia el pod, los logs se borran y solo vera el resultado que se muestra en el siguiente paso.

4.3). Verifique que Kubernetes reinicie el pod en mal estado. Siga revisando la salida del comando kubectl get pods. Observe la columna

RESTARTS y verifique que el conteo sea mayor que cero. Anote el nombre del nuevo pod.

```
kubectl get pods
NAME                            READY   STATUS    RESTARTS      AGE
do100-probes-95758759b-4cm2j   1/1     Running   1 (11s ago)   62s
```

4.4 Revise los logs de la aplicación. El liveness tiene éxito y la aplicación informa un estado correcto.

```
 kubectl logs -f do100-probes-95758759b-4cm2j
...output omitted...
ping /ready => pong [ready]
ping /healthz => pong [healthy]
...output omitted...
```

Delete the deployment, ingress, and service resources to clean your cluster. Kubernetes automatically deletes the associated pods.

Elimine los recursos deployment, ingress, y service para limpiar su clúster. Kubernetes elimina automáticamente los pods asociados.

```
[user@host ~]$ kubectl delete deployment do100-probes
deployment.apps "do100-versioned-hello" deleted
[user@host ~]$ kubectl delete service do100-probes
service "do100-probes" deleted
[user@host ~]$ kubectl delete ingress do100-probes
ingress.networking.k8s.io "do100-probes" deleted
```