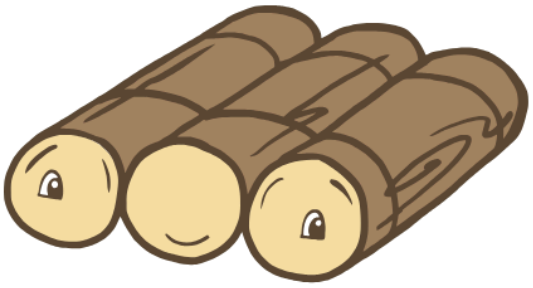
Práctica 5: Kubernetes

Jorge Gallardo, 868801
Enrique Baldovin, 869402

Índice



Introducción

Para esta práctica se ha solicitado adaptar el servicio de almacenamiento clave/valor basado en Raft de la práctica 4 para poder desplegarlo mediante el uso de Kubernetes, de modo que cada servidor se ejecuta en un nodo distinto, y cada servidor dentro de un pod distinto.

Además de adaptar la práctica anterior se nos pide crear un cliente para que sea el cliente quien vaya mandando peticiones al servidor, y será a través de este cliente la manera de comprobar el debido funcionamiento del despliegue de raft.

Para llevar a cabo lo anterior hemos creado un contenedor para cada servidor y uno más para el cliente, y se ha debido de adaptar el código del servidor para su funcionamiento por DNS.

Toda la práctica ha sido realizada en una máquina virtual linux usando Ubuntu 24.04.1 con 6 cores, 8GB de Ram y 50GB de almacenamiento.

Desarrollo de la práctica

Lo primero que tuvimos que hacer fue la puesta en marcha de la máquina virtual mencionada anteriormente y la instalación tanto de Docker como de Kind y ejecutar y estudiar los anexos A y B que nos proporcionais en el código de prueba, y tras mirar bien los archivos de creación (.yaml), vimos que statefulset era el mejor, por lo que lo usamos como base.

En cmd/srvraft, el código del servidor cambiamos como se gestionan los parámetros, puesto que desde la creación del statefulset ahora solo enviamos como parámetro la dirección propia del servidor, para ser usada por el DNS, ya que ahora al usar kubernetes las direcciones ip no son fijas, además para cada nodo hemos construyendo un vector de todos los nombres DNS de todos los nodos incluido el mismo

Por otro lado en pkg/cltraft, el código del cliente, se ha desarrollado en esta práctica, se trata de un servidor interactivo, el cual despliega un menú para mostrar por pantalla las operaciones disponibles, en este código definimos para que el cliente conozca, los nombres de los servidores con los que se quiere comunicar, de forma que le permita realizar llamadas RPC a estos servidores .

Para que todo funcione correctamente en kubernetes, hemos creado una carpeta llamada Dockerfiles, dentro de la cual hay otras 2 carpetas, una para el cliente y otra para el servidor, en las cuales guardaremos los dockerfiles y los binarios respectivos de cada uno.

Para los dockerfiles hemos usado alpine por su bajo costo de almacenamiento y eficiencia, además como se indica hemos usado los puertos 6000 y 7000.

A la hora de crear los pods, para el cliente hemos optado por uno básico, sin tolerancia a fallos, puesto que no hemos visto importancia en que relance al cliente.

Mientras que para los servidores, están configurados para que si hay algún problema estos se vuelvan a lanzar, usando un controlador statefulset, lo cual por definición es buena opción para situaciones en las cuales queremos conservar identidades estables y almacenamiento persistente para cada nodo, además statefulset asegura que los Pods se crean, escalan y terminan de manera ordenada, finalmente en la configuración del servidor hemos definido que cada nodo tenga como nombre único en el cluster, nodo-”x” siendo x un numero del 0 al 2.

A todos los nodos de raft se les pone en marcha con el servicio headless (clusterIP=none), lo que permite que estos nodos sean accesibles a través de nombres DNS en vez de tener que asignarles una IP, cosa que es dificil en kubernetes porque las IPs son difíciles de controlar, ya que las asigna automáticamente kubernetes, y asi creamos el dominio del DNS a usar en este cluster, raft.default.svc.cluster.local.

El uso del DNS es crucial porque nos permite que los nodos sean capaces de identificarse y comunicarse entre sí directamente sin necesitar nada más que el nombre del nodo, puerto y DNS.

Puesta en marcha

En esta práctica se utilizan 3 servidores y un único cliente, cada proceso se ejecuta en un pod distinto.

Por un lado los servidores, kubernetes, los mantendrá siempre vivos, gracias a la gestión de estos mediante statefulset, lo cual nos garantiza que ante la parada, error o reinicio de un nodo de raft siempre se va a volver a ejecutar, haciendo que los servidores sean tolerantes a fallos.

Hemos de tener en cuenta que el cliente no se reinicia, ya que lo que queremos asegurar es el funcionamiento del sistema de raft, por lo que al no tener controlador, si falla,quien esté utilizando el cliente será el responsable de reiniciarlo.

Se han creado varios scripts para facilitar el lanzamiento del conjunto de pods:

-borrar.sh: un script que se usa para limpiar cualquier cluster existente antes de crear uno nuevo, detiene y elimina los contenedores, y elimina imágenes, volúmenes, redes no utilizadas y finalmente el cluster en sí.

-lanzar.sh: un script que se asegura de que el path esté exportado para que se pueda acceder a este a la hora de compilar, luego a continuación mediante el uso de borrar.sh, borra cualquier traza de otros posibles clusters en ejecución en el instante de ejecución, crea un cluster nuevo y borra los ejecutables antiguos, compila los códigos del cliente y servidor y hace el push de las imágenes a sus respectivas carpetas, finalmente ejecuta el stateful.yaml para la creación de los pods, y hace una espera a cuando el pod del cliente este en estado “running” para entonces ejecutar en nuestra terminal el cliente y asi abrir el menú de interacción con los nodos

He de recordar que para ejecutar los scripts hay que darles permisos de ejecución mediante chmod u+x, y todo debe de ser ejecutado desde la carpeta en la que están los scripts, de lo contrario no funcionarán bien las rutas y fallaran

Prueba de funcionamiento

Esta vez no hemos podido hacer uso de nuestro módulo de test usado en la práctica 4, por lo que hemos tenido que hacer uso del cliente e ir poniendo operaciones y verificar se ejecuta correctamente.

Este cliente dispone de varias operaciones, como son: Obtener el estado general del nodo, obtener el estado de los logs del nodo, parar el nodo y someter operaciones de lectura o escritura al líder de los nodos.

Al ser una máquina virtual bastante lenta hemos tenido que aumentar los timeouts de las llamadas rpc, dado que si no darían problemas

Con las operaciones explicadas anteriormente, hemos verificado que, todos los nodos tienen el mismo líder, cuando se para al líder se provocan elecciones, al someter una operación de escritura, si secuencialmente sometes una de lectura con la clave de la de escritura nos devuelve el valor esperado y finalmente comprobar que cuando se someten operaciones, todos los nodos tienen el mismo estado

Todos los nodos tienen el mismo líder:

```
>obtenerEstado 2
=== Estado del Nodo ===
IdNodo:  2
Mandato: 39
EsLider: false
IdLider: 0
======================
```

```
>obtenerEstado 1
=== Estado del Nodo ===
IdNodo:  1
Mandato: 39
EsLider: false
IdLider: 0
======================
```

```
>obtenerEstado 0
=== Estado del Nodo ===
IdNodo:  0
Mandato: 39
EsLider: true
IdLider: 0
======================
```

Se provocan elecciones al parar al líder:

```
>obtenerEstado 1
=== Estado del Nodo ===
IdNodo:  1
Mandato: 54
EsLider: false
IdLider: 2
======================
```

```
>parar 2
Nodo 2 detenido con éxito.
```

```
>obtenerEstado 1
=== Estado del Nodo ===
IdNodo:  1
Mandato: 55
EsLider: false
IdLider: 0
======================
```

Comprobar que una escritura se somete y se puede leer

```
>escribir 0 escribir dato
=== Resultado de la Operación ===
IndiceRegistro: 0
Mandato: 48
EsLider: true
IdLider: 0
ValorADevolver: escrito
================================
Comandos disponibles:
  obtenerEstado <idNodo>
  obtenerEstadoSometido <idNodo>
  parar <idNodo>
  leer <idNodoLider> <clave>
  escribir <idNodoLider> <clave> <valor>
  salir

>leer 0 escribir
=== Resultado de la Operación ===
IndiceRegistro: 1
Mandato: 48
EsLider: true
IdLider: 0
ValorADevolver: dato
================================
```

Todos los nodos tienen el mismo estado del log

```
>obtenerEstadoSometido 0
=== Estado Sometido ===
Índice:  1
Mandato: 48
Longuitud del log: 2
========================
```

```
>obtenerEstadoSometido 1
=== Estado Sometido ===
Índice:  1
Mandato: 48
Longuitud del log: 2
========================
```

```
>obtenerEstadoSometido 2
=== Estado Sometido ===
Índice:  1
Mandato: 48
Longuitud del log: 2
========================
```