Sistemas Distribuidos - Práctica 5

Kubernetes y Raft

Selivanov Dobrisan, Cristian Andrei - 816456@unizar.es

Wozniak, Dorian Boleslaw - 817570@unizar.es

Índice

1 Introducción

En las prácticas anteriores se ha desarrollado la implementación de un sistema de almacenamiento clave-valor distribuido. Para ello, se ha implementado también el algoritmo de consenso Raft, el cual es capaz de recuperarse de caídas (mediante un mecanismo de elección de líder) así como garantizar el mantenimiento de una máquina de estados distribuida consistente en todas las máquinas que forman la red (mediante un mecanismo de replicación de entradas que garantiza la persistencia de todas las operaciones comprometidas por una mayoría de nodos). Además, el sistema acepta solicitudes de clientes para leer y escribir datos, obtener su estado, o parar alguno de los nodos. Por último, se ha diseñado una serie de pruebas de integración que garantizan el correcto funcionamiento del sistema en un escenario de uso real.

En las anteriores prácticas, se ha utilizado para el despliegue el protocolo ssh para lanzar remotamente nodos en una serie de máquinas conectadas a un sistema de ficheros en red. En esta práctica, en cambio, se realizará el despliegue mediante Kubernetes, un orquestador para la puesta en marcha de servicios distribuidos. Para el despliegue además se utilizará la herramienta Kind, que permite realizar despliegues mediante Kubernetes en una máquina local mediante contenedores anidados

2 Despliegue

El funcionamiento de Kind y Kubernetes es el siguiente: al crear un cluster, Kind crea y arranca una serie de contenedores Docker que están ejecutando Ubuntu, los cuales actuarán como nodos del cluster en vez de máquinas remotas. Kind ordena a Kubernetes asociarse al nodo, y a partir de entonces se podrá interactuar con Kubernetes para crear pods, los cuales se encargará de asociar a uno de los contenedores creados por Kind, que a su vez arracarán un contenedor añidado que ejecutará la imagen obtenida de un repositorio asociado.

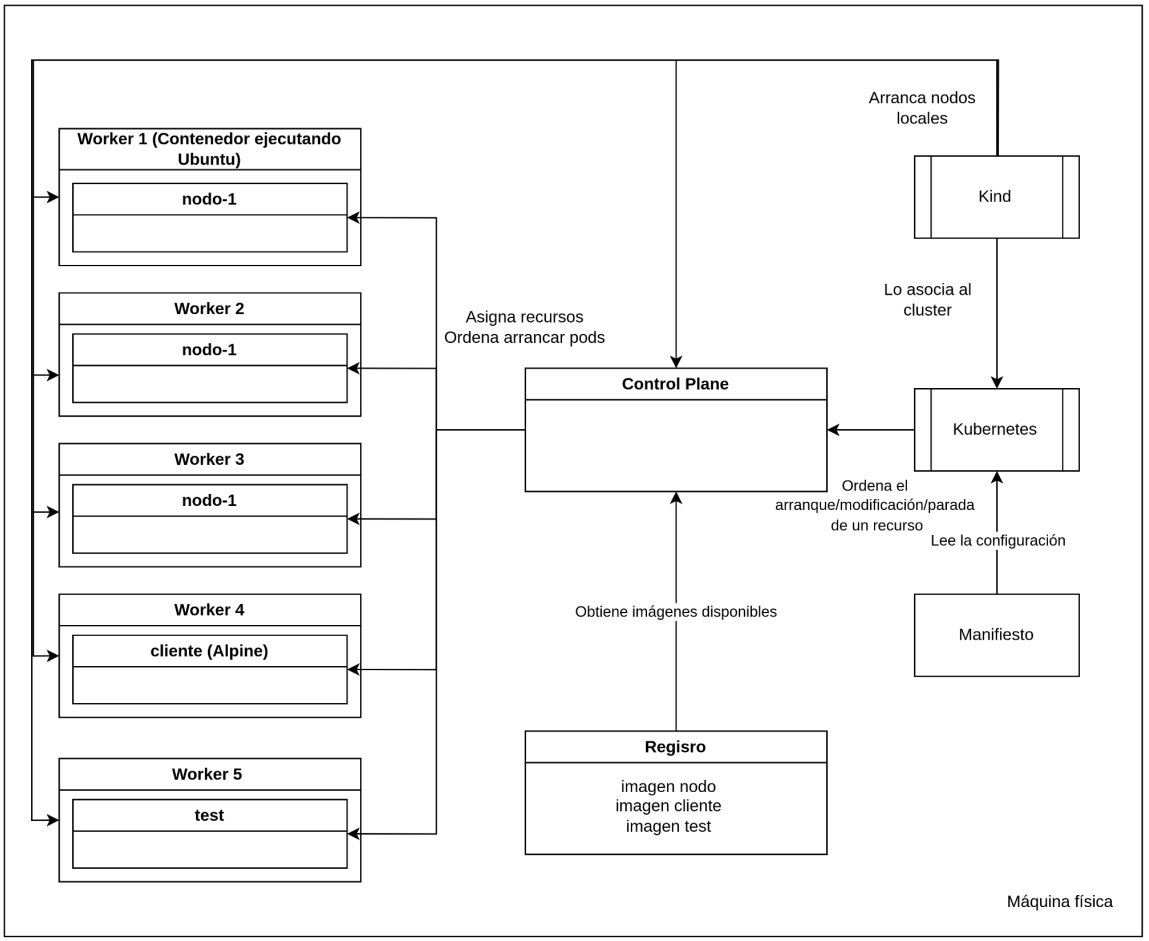

Fig. 1: Diagrama de un despliegue del sistema utilizando Kubernetes y Kind

2.1: Evaluación de controladores

Para la puesta en marcha de los pods que contendrán los nodos Raft, se ofrecen tres soluciones:

- **Pods individuales**: Cada pod del conjunto de nodos a arrancar se especifica individualmente. Los pods son configurables tal que no se reinicien nunca, se reinicien al fallar o se reinicien incluso en caso de terminación normal. Además, se pueden asociar a un servicio que ofrece funcionalidad de DNS para los nodos tal que se puedan comunicar a través de su nombre en la red.
- **Deployments**: En vez de especificar cada pod de forma individual, permite lanzar múltiples pods a partir de una plantilla mediante un conjunto de réplicas. Los deployments no ofrecen un servicio DNS; en vez de eso, los nodos deben contactarse mediante su IP. Los deployments reinician por defecto los nodos al terminar o fallar sus pods asociados.
- **Stateful sets**: Similar al Deployment, pero este ofrece un servicio DNS que permite asignar un nombre a cada pod generado de forma consistente. También reinicia los nodos terminados o fallidos.

Para el despliegue, se ha decidido utilizar un *Stateful Set* para arrancar los nodos Raft. Comparando con el uso de *Deployments*, el segundo daría problemas a la hora de arrancar los nodos, puesto que no se conocen las IPs de cada máquina hasta ser arrancada, y cada nodo requiere, en el arranque, conocer las direcciones de todos los nodos. Además, en caso de haber más máquinas, si se cayese un nodo del *cluster*, y hubiese otros nodos disponibles, Kubernetes podría decidir arrancar el pod en otra máquina. Si la IP del contenedor no es estática, el resto de nodos no serán capaces de contactar con el nodo reiniciado.

Además, se prefiere utilizar un *Stateful set* en vez de pods individuales por sencillez al crear el manifiesto de arranque. Además, aunque los pods también pueden ser configurados para recuperarse de caídas, los *Stateful Sets* son tolerantes a fallos por defecto.

Para los pods para el cliente y el test de integración, se pueden lanzar como un pod básico sin tolerancia a fallos.

2.2: Pasos para arrancar el cluster

Para arrancar el cluster mediante Kind, se dispone de un *script* parecido al dado como ejemplo en la documentación oficial. Este *script* (*"kind-create.sh"*) crea seis nodos: 5 nodos *worker* y un *control-plane* que hará de *master* dentro del *cluster*. Además, si no existe, se arrancará un registro local que contendrá un repositorio de imágenes Docker disponibles para el *cluster*, accesible mediante el puerto 29310. Finalmente, el *script* asocia el servicio de Kubernetes al cluster creado.

Por otro lado, para crear las imágenes Docker a utilizar, se ha creado un *script* (*"docker-create.sh"*) que compila los binarios para un nodo Raft, un cliente y el test de integración tal que no utilice librerías dinámicas, y crea una imágen a partir de estos binarios. La imágen de un nodo contiene únicamente su binario y expone para su uso el puerto 29311. La imágen del test funciona de forma similar, pero está asociado al puerto 29314. La imágen del cliente, a diferencia de las anteriores, en vez de ser creada de cero importa una imagen de Alpine Linux (una distribución diseñada para dispositivos de bajo rendimiento y máquinas virtuales), al que se añade a */usr/local/bin* el cliente para que sea accesible desde la *PATH*. La imágen del cliente expone el puerto 29319. Estas imágenes son creadas mediante sus respectivos Dockerfiles y cargadas al repositorio del registro anteriormente creado.

2.3: Despliegue mediante un *Stateful Set*

Inicialmente se realizó un despliegue mediante pods básicos de tres nodos y un cliente interactivo utilizando pods básicos sin tolerancia a fallos. Se dispone de un *script* para arrancar los pods (*"k8s-pods.sh"*) y un manifiesto para la configuración del despliegue de los pods (*"k8s/raft-cliente-fallos.sh"*). Si se ha desplegado correctamente todos los pods, deberá ejecutarse el programa del cliente para interactuar con los nodos de forma interactiva.

Para el despliegue del *Stateful Set*, es similar al de los pods básicos (ejecutar *"k8s-stateful.sh"*, que a su vez lee el manifiesto *"raft-stateful.yaml"*). El manifiesto se ha creado siguiendo el ejemplo dado y adaptándolo para que se lancen tres réplicas que contacten con las otras dos que no sean la misma. Al no permitir obtener el índice asignado a la réplica directamente, en vez de pasar como argumento el índice se pasa el nombre del pod, y el programa, al arrancar, obtiene el índice. El nombre del pod debe seguir el formato *"algo-índice"*, siendo *índice* un entero.

Adicionalmente, se ha añadido al despliegue una sentencia *topologySpreadConstraint* que asegura que se desplegarán los nodos en nodos distintos si hay *workers* libres disponibles.

3 Validación

Se han realizado una serie de pruebas con un programa diseñado como cliente interactivo, donde se verifican las ventajas ofrecidas por el despliegue del sistema raft en Kubernetes. El principal objetivo es comprobar que al eliminar un Pod asociado a un nodo raft, este volverá automáticamente a ejecutarse, tratando de ponerse en marcha y comunicarse con los demás nodos desplegados, replicando / recuperando las entradas sometidas / comprometidas, aplicándolo en su máquina de estado de vuelta. El nodo a eliminar podrá ser cualquiera, desde un líder, hasta un seguidor.

Ambas pruebas se realizan en una misma ejecución de un cliente tras un despliegue de los distintos nodos Raft.

Comprobación de que se crean los distintos nodos para mantener los Pods.

```
NAME                 STATUS  ROLES          AGE   VERSION   INTERNAL-IP   EXTERNAL-IP
OS-IMAGE             KERNEL-VERSION        CONTAINER-RUNTIME
kind-control-plane   Ready   control-plane   66s   v1.25.3   172.18.0.4  <none>
Ubuntu 22.04.1 LTS   5.15.0-56-generic   containerd://1.6.9

kind-worker          Ready   <none>         42s   v1.25.3   172.18.0.3   <none>
Ubuntu 22.04.1 LTS   5.15.0-56-generic   containerd://1.6.9

kind-worker2         NotReady   <none>           29s   v1.25.3   172.18.0.7   <none>
       Ubuntu 22.04.1 LTS   5.15.0-56-generic   containerd://1.6.9

kind-worker3         Ready   <none>         42s   v1.25.3   172.18.0.2   <none>
Ubuntu 22.04.1 LTS   5.15.0-56-generic   containerd://1.6.9

kind-worker4         Ready   <none>         42s   v1.25.3   172.18.0.5   <none>
Ubuntu 22.04.1 LTS   5.15.0-56-generic   containerd://1.6.9

kind-worker5         Ready   <none>         42s   v1.25.3   172.18.0.6   <none>
Ubuntu 22.04.1 LTS   5.15.0-56-generic   containerd://1.6.9
```

Comprobación de que cada Pod de almacenamiento está situado en nodos distintos.

```
NAME           READY   STATUS RESTARTS   AGE   IP           NODE           NOMINATED NODE
READINESS GATES
cliente        1/1     Running   0          69s   10.244.4.3   kind-worker4   <none>
       <none>

raft-ss-0   1/1       Running   0          69s   10.244.5.3   kind-worker2   <none>
       <none>

raft-ss-1   1/1       Running   0          69s   10.244.1.3   kind-worker    <none>
       <none>

raft-ss-2   1/1       Running   0          69s   10.244.2.3   kind-worker5   <none>
       <none>
```

Puesta en marcha de nodos y cliente comprueba a comprometer entradas iniciales, para verificar que posteriormente no se pierden.

```
--------- Esperar un poco para dar tiempo a arrancar réplicas
Indique una acción {leer | escribir | estado | parar}:
estado 0
0 106 true 0 [{0 0 {  }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 106 false 0 [{0 0 {  }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 2
2 106 false 0 [{0 0 {  }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
escribir k1 v1
1 106 true 0 v1 <nil>

Indique una acción {leer | escribir | estado | parar}:
escribir k2 v2
2 106 true 0 v2 <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 0
0 106 true 0 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 106 false 0 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 2
2 106 false 0 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}}] <nil>
```

En primer lugar trataremos de borrar un nodo seguidor para verificar que al volver a ponerse en marcha recuperara las entradas para aplicarlas a su máquina de estados.

```
~$ kubectl delete pod raft-ss-2
pod "raft-ss-2" deleted
```

En el instante que se elimina el nodo raft 2 (Seguidor), también se verifica si una operación se compromete y aplica correctamente fallando menos de la mitad de los nodos que componen el sistema. Acto seguido se espera unos segundos para la vuelta del nodo 2 al cluster, comprobando si mantiene las nuevas entradas comprometidas y aplicadas por el cluster superviviente.

```
[El nodo-2 ha sido eliminado, simulación de fallo producido en dicho nodo]

Indique una acción {leer | escribir | estado | parar}:
leer k1
3 106 true 0 v1 <nil>

[Se espera a que nodo-2 vuelva a comunicarse con cluster]

Indique una acción {leer | escribir | estado | parar}:
estado 0
0 133 true 0 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 133 false 0 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 2
2 133 false 0 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>
```

En segundo lugar, se procede a eliminar al líder, con el propósito de comprobar que uno de los seguidores retoma su rol, además de verificar si a la hora de volver el nodo correspondiente recupera las entradas perdidas. (El líder durante el mandato mostrado en las salidas anteriores es el nodo 0)

```
~$ kubectl delete pod raft-ss-0
pod "raft-ss-0" deleted
```

Se puede observar como tras producir un fallo en el nodo líder, los seguidores llegan como deberían a un acuerdo para elegir un nuevo líder, acto seguido se reconectara el nodo 0 eliminado anteriormente, recuperando el registro de entradas comprometido y aplicado hasta el momento.

```
[El nodo-0 ha sido eliminado, simulación de fallo producido en dicho nodo]

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 134 false 2 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 2
2 134 true 2 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>

[Se espera a que nodo-0 vuelva a comunicarse con cluster]

Indique una acción {leer | escribir | estado | parar}:
estado 0
0 223 false 2 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 223 false 2 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 2
2 223 true 2 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }}] <nil>
```

Se ha realizado una última prueba con el propósito de verificar la recuperación de las entradas por parte de un nodo líder de raft caído, pero estas entradas tendrán nuevas operaciones comprometidas y aplicadas por los otros dos nodos que no han sufrido fallos.

```
~$ kubectl delete pod raft-ss-2
pod "raft-ss-2" deleted
```

A continuación se puede apreciar como los nodos restantes llegan a consenso para escoger un nuevo líder, comprometer y aplicar la nueva operación durante la ausencia del nodo 2 en el que se ha simulado el fallo. Además del nodo 2 recuperar las nuevas entradas y aplicarlas tras su vuelta al cluster.

```
[El nodo-2 ha sido eliminado, simulación de fallo producido en dicho nodo]

[Se procede a someter operación entre nodos restantes que llegan a consenso]

Indique una acción {leer | escribir | estado | parar}:
leer k2
4 224 true 1 v2 <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 0
0 224 false 1 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }} {4 224 {leer k2 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 224 true 1 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }} {4 224 {leer k2 }}] <nil>

[Se verifica que efectivamente el nodo-2 sigue caído, y por tanto recuperándose]

Indique una acción {leer | escribir | estado | parar}:
estado 2
0 0 false 0 [] dial tcp: lookup raft-ss-2.raft-ss-service.default.svc.cluster.local on
10.96.0.10:53: no such host

[Se espera a que nodo-2 vuelva a comunicarse con cluster, recuperando nuevas entradas]

Indique una acción {leer | escribir | estado | parar}:
estado 1
1 318 true 1 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }} {4 224 {leer k2 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 2
2 318 false 1 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }} {4 224 {leer k2 }}] <nil>

Indique una acción {leer | escribir | estado | parar}:
estado 0
0 318 false 1 [{0 0 {  }} {1 106 {escribir k1 v1}} {2 106 {escribir k2 v2}} {3 106 {leer
k1 }} {4 224 {leer k2 }}] <nil>
```