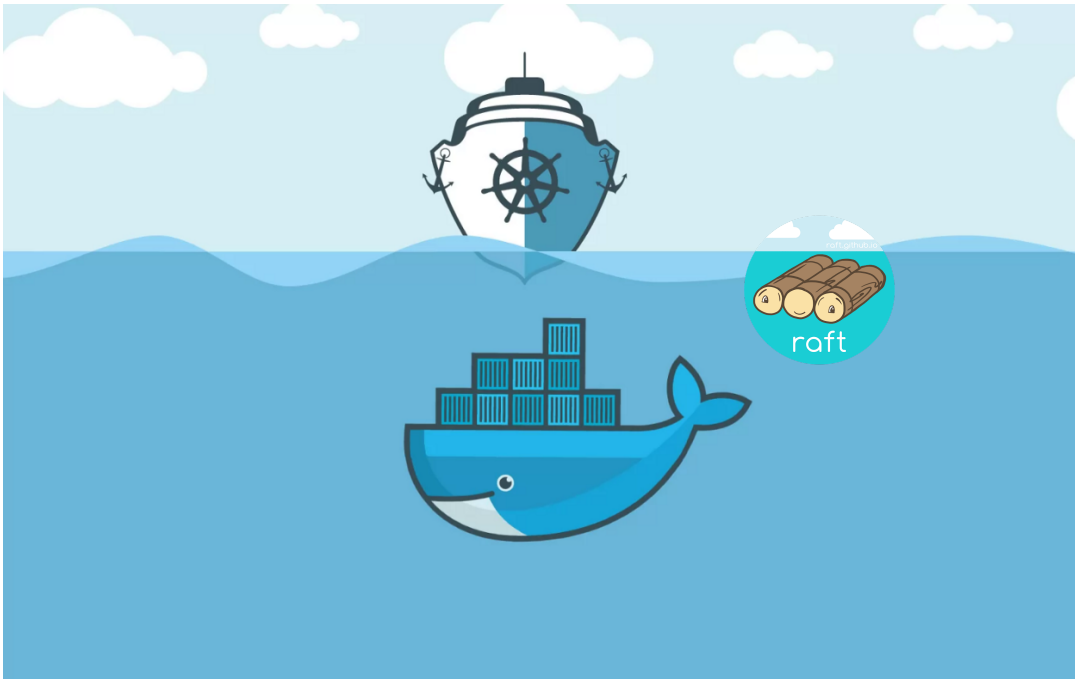
Práctica 5:
Kubernetes y Raft

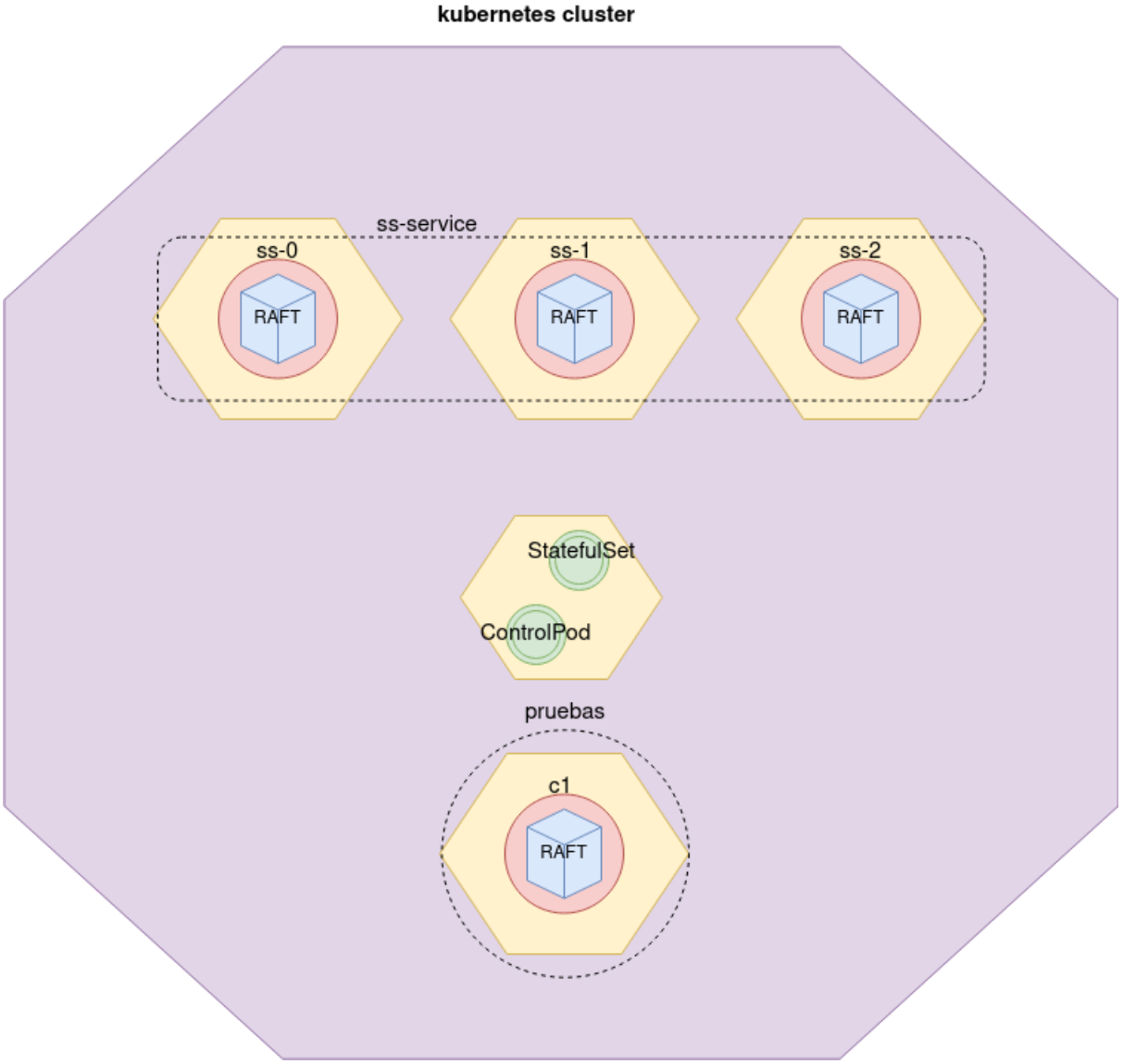


Sistemas Distribuidos
Universidad de Zaragoza, curso 2022/2023

Ayelen Nuño Gracia 799301
Loreto Matinero Angusto 796598

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

Descripción del problema:
En esta práctica, se plantea utilizar Kubernetes para poner en funcionamiento un servicio replicado basado en Raft. El cual se ha almacenado en contenedores docker, la estructura que seguirá el sistema lanzado se define en el siguiente esquema:



En los pods se ejecutará raft, el cual sigue el esquema presentado en las prácticas anteriores.

Implementación con kubernetes.

En primer lugar se ha realizado la puesta en marcha de kubernetes tal y como se indica en el enunciado de la práctica.
Para la creación del cluster de kubernetes se ha reutilizado el script *kind-with-registry.sh* sin llevar a cabo ninguna modificación.

Para la ejecución de la aplicación golang en kubernetes se ha implementado un cliente y un servidor para la puesta en marcha de raft. En primer lugar se consideró modificar el fichero de *testintegracionraft1_test.go* para poder realizar una correcta depuración del algoritmo y comprobar que cumple con las especificaciones planteadas, pero finalmente se ha visto más sencillo realizar esta comprobación a través de un cliente manual. Este cliente consiste en un búcle en el cual se mandan 5 entradas al nodo líder para someterlas.

Una vez se tienen los ficheros correctos correspondientes al cliente y al servidor se ha creado un compilado de estos y situado en la carpeta donde se encuentran los ficheros Dockerfiles (de cliente y servidor) para proceder a la creación de los contenedores docker.
La creación de contenedores y subida de la imagen para cliente y servidor se ha realizado de la siguiente forma:

Desde Dockerfiles/cliente, se ha ejecutado:
docker build . -t localhost:5001/cliente:latest
docker push localhost:5001/cliente:latest

Desde Dockerfiles/servidor, ejecutar ./Docker-Servidor.sh
docker build . -t localhost:5001/servidor:latest
docker push localhost:5001/servidor:latest

En nuestro caso se ha optado por utilizar como imagen de los contenedores la imagen mínima de Linux “alpine”.

El siguiente paso ha sido decidir cuál de los 3 controladores usar para desplegar el sistema Raft. El primero de ellos ha sido el *controlador de pods individuales***,** este consiste en desplegar por separado las tres réplicas pero sin la propiedad de tolerancia a fallos, por ello se ha descartado, ya que no cumple uno de los requisitos críticos, dado que Raft es una solución tolerante a fallos de servidores con estado.
El segundo que se ha tenido en cuenta ha sido el *controlador Deployment* , este se encarga de mantener vivas todas las réplicas de forma automática, pero a pesar de ello no es el controlador adecuado para servidores con estado como Raft, ya que este controlador solo dispone de un único volumen común para todas las réplicas.
Por último, y el que se ha elegido ha sido el *controlador SatatefulSet* este además de asegurar el mantenimiento de las réplicas, proporciona un volumen de almacenamiento por cada pod, por esto es el más adecuado para Raft.

Una vez se ha elegido el tipo de recurso, se ha procedido a su implementación, similar a la proporcionada en el fichero *statefulset_go.yaml*.

Además de esto, para la creación del pod del cliente se ha decidido basarse en el controlador de pods individuales, basándose en el fichero *pods_go.yaml* pero adaptado para un solo pod.

Cambios respecto a la práctica anterior.

En el caso del algoritmo de raft tan solo se ha modificado el valor de la variable kLogToStdout a true para que de esta forma el contenido que antes iba dirigido a los logs

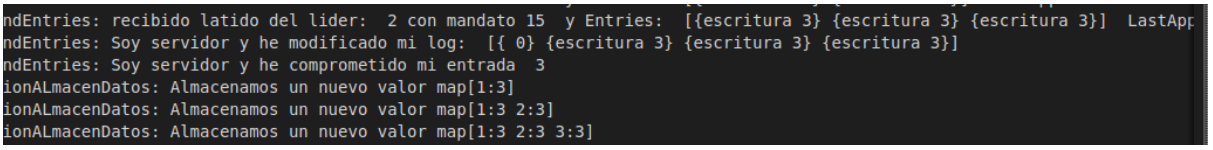
salga en terminal para poder acceder a ella cuando se ejecute en kubernetes y dirigirla a un fichero, facilitando así el proceso de debugeo del programa.
Además se han ajustado los tiempos de las llamadas de CallTimeout, al igual que el tiempo límite de los timeout, en este caso del latido, y de la elección. En todos estos casos se ha aumentado el tiempo para solventar problemas y permitir así que los mensajes lleguen a su correspondiente destino.

Pruebas realizadas.

1. Se consigue acuerdos de varias entradas de registro a pesar de que un réplica (de un grupo Raft de 3) se desconecta del grupo

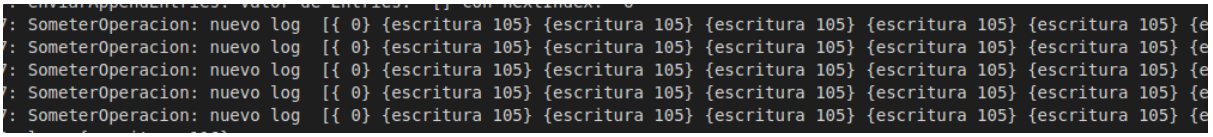
La prueba consiste en tirar uno de los nodos de raft, de esta forma sigue existiendo mayoría, por tanto al solicitar el cliente una entrada esta es comprometida por los otros dos nodos.
En la siguiente captura observamos como en el nodo caído al levantarse se han registrado las 3 entradas que el resto de nodos había comprometido previamente. Y como esta lo almacena en su almacén de datos.
Para la realización de esta prueba se ha tirado un pod el cual no fuese el líder (lo cual se ha comprobado a través de los logs generados por los nodos raft), justo antes de lanzar el cliente, dado que así nos aseguramos de que el nodo estaba caído antes de someter las entradas.

Para poder repetir esta prueba se ha decidido implementar un script que automatiza la caída del nodo, justo antes del lanzamiento del cliente, este fichero es prueba1.sh.



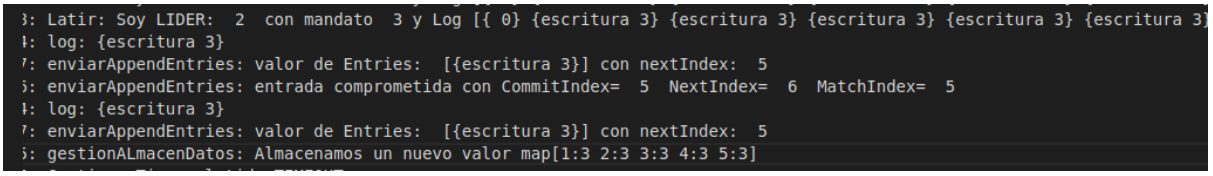
2. NO se consigue acuerdo de varias entradas al desconectarse 2 nodos Raft de 3.

Dado que el número de nodos es 3 para poder comprometer una entrada se necesita que al menos dos de los nodos respondan y guarden esa entrada. Al desconectarse dos de las tres entradas es imposible que la solicitud sea comprometida, por lo que quedará sometida en el Log del líder, pero sin llegar a comprometerse en ningún instante. Es por eso que nunca llega a comprometerse la entrada, aunque esta se someta.

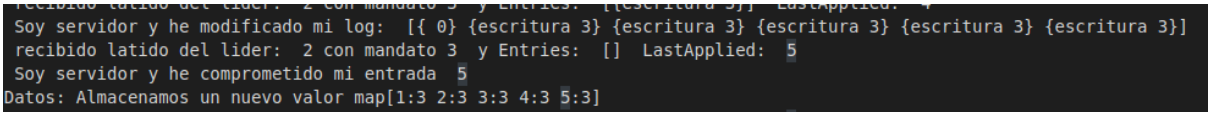


3. Someter 5 operaciones cliente de forma concurrente y comprobar avance de índice del registro.

Esta última prueba es sencilla consiste en solicitar cinco operaciones a los nodos, funcionando correctamente todos ellos, de esta forma las cinco entradas son comprometidas en la máquina de estados de cada uno de los nodos.
A continuación se muestra el logg del líder, en la sección que se muestra como se ha comprometido la última entrada.



Y el de uno de los dos seguidores, en el instante en que recibe el mensaje de comprometer la última entrada que se ha recibido.



Esta prueba se ha limitado a ejecutar el fichero prueba3.sh en el que no se elimina ningún pod, y se lanzaba el cliente, previamente configurado para realizar tres peticiones.