Práctica 3 - Sistemas Críticos

Miguel Ángel Dorado Maldonado

1. Objetivo de la práctica

El objetivo de la práctica es diseñar e implementar un sistema utilizando Zookeeper como servicio, de forma que varios nodos generadores de mediciones se sincronicen y manden valores medios a una api externa.

2. Descripción general del sistema

El sistema está compuesto por nodos que generan mediciones cada cierto tiempo, un cluster de Zookeeper y una api externa. De forma iterativa los nodos generarán una medición aleatoria, el nodo líder hará una media y la enviará a la api externa.

Cada nodo ejecuta la aplicación desarrollada y puede asumir de forma dinámica el rol de líder o seguidor. El líder se encarga de coordinar las iteraciones, calcular la media y desbloquear a los demás nodos en el caso de la aplicación final, donde ya se usan barreras.

3. Arquitectura del sistema

- N nodos de aplicación: generan mediciones cada cierto tiempo.
- Zookeeper (3 nodos en el apartado final): proporciona coordinación distribuida, elecciones y sincronización.
- API externa: recibe las medias calculadas por el líder.

La relación entre componentes se realiza gracias a docker compose y una red compartida

4. Uso de Zookeeper y recetas empleadas

Se han utilizado las siguientes funcionalidades de Zookeeper mediante la librería Kazoo:

- Election: encargado de producir la selección de líder.
- Barrier: encargado de sincronizar a los nodos para evitar depender de timers.
- Counter: lleva el contador global de cada iteración.
- DataWatch y ChildrenWatch: se encargan de reaccionar a cambios ocurridos en configuraciones o rutas de control.

Las rutas principales empleadas son:

/election
/mediciones
/barrier
/counter
/config

5. Funcionamiento normal del sistema

En condiciones normales de funcionamiento:

1. Los nodos se conectan a Zookeeper.
2. Mediante un Election se realiza una elección de líder.
3. Cada nodo genera una medición y actualiza su valor en zookeeper.
4. El líder calcula la media de las mediciones disponibles.
5. El líder envía la media a la api.
6. El líder libera la barrera para permitir la siguiente iteración.

Este ciclo se repite indefinidamente mientras el sistema esté activo.

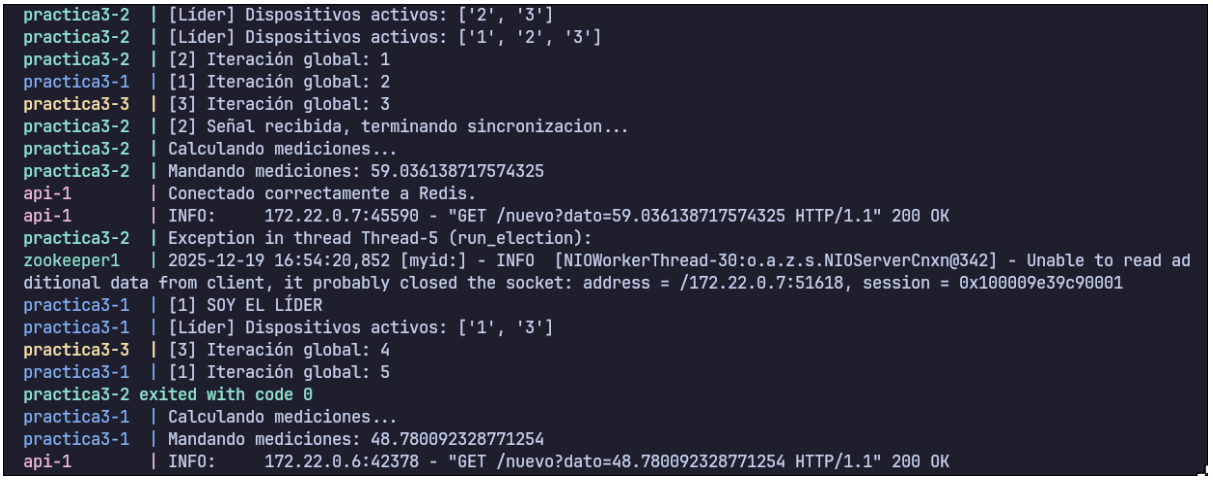
6. Gestión de fallos

El sistema gestiona correctamente:

- La caída de nodos de aplicación.
- La caída del nodo líder de la aplicación (se produce una nueva elección y el sistema continúa funcionando).

Sin embargo, no se ha conseguido gestionar completamente la caída de un nodo de Zookeeper.

Aquí podemos ver el caso en el que el líder de los nodos cierra. Otro nodo adopta el rol de líder y continúa la ejecución



7. Configuración por defecto y gestión global

La aplicación define varios valores por defecto que permiten controlar el funcionamiento de la misma

- Periodo de muestreo (sampling_period): por defecto se establece en 5 segundos.
- URL de la api (api_url): por defecto se establece en http://api:80, que corresponde al servicio api definido en el docker-compose.

Estos valores se cargan por defecto en el constructor y en caso de existir valores en zookeeper, se actualizan.

Se facilita la gestión de las variables con un script init_config.py que permite modificar los valores sin necesidad de reiniciar el sistema. Este permite:

- Crear los nodos de configuración bajo /config en ZooKeeper.
- Inicializar o actualizar los valores de sampling_period y api_url de forma centralizada.

De este modo, todos los dispositivos reciben automáticamente los cambios mediante DataWatch, sin necesidad de reiniciar los contenedores.

8. Problemas encontrados y limitaciones

El principal problema no resuelto aparece cuando uno de los nodos de Zookeeper se cae:

- ZooKeeper realiza correctamente una nueva elección interna de líder.
- Los clientes Kazoo se reconectan automáticamente al clúster.
- El listener de estado detecta la reconexión y se reconfiguran las recetas.

El problema que me ha surgido:

No he conseguido que los nodos se vuelvan a reconectar con la instancia de zookeeper. Al matar al proceso líder de zookeeper los 3 nodos lanzados de mi aplicación no resuelven el host de zookeeperx y cierran. No se si es problema de docker o de mi código. Las tres aplicaciones siguen ejecutando pero de forma individual sin ningún tipo de relación. El leader_loop sigue ejecutándose mandando reiteradas veces la última medición antes del fallo.

9. Estado final de la solución

La práctica cumple correctamente los requisitos funcionales principales:

- Elección de líder.
- Sincronización mediante barreras.
- Uso de contadores distribuidos.
- Reacción a cambios mediante watchers.

El único problema que queda por resolver es el manejo ante una caída de un nodo zookeeper en el cluster.

10. Despliegue

La imágen ha sido subida a docker hub:
https://hub.docker.com/repository/docker/migueel15/practica3-kazoo/general

En la entrega contará con dos archivos docker compose:

- **docker-compose.yml**: Ejecuta la aplicación sin uso de cluster zookeeper.
- **docker-compose-cluster.yml**: Ejecuta la aplicación con cluster zookeeper.

Para abrir los servicios:

- docker compose -f docker-compose.yml up –build

- docker compose -f docker-compose-cluster.yml up –build

Para cerrar los servicios:

- docker compose -f docker-compose.yml down
- docker compose -f docker-compose-cluster.yml down

Aplicación grafana abierta en http://localhost:3000 donde podrá ver las nuevas mediciones.
Api abierta en http://localhost:4000 accesible desde dentro de los contenedores en api:80.