DISTRIBUTED SYSTEMS

Kubernetes and Raft

14/12/2023

GUILLERMO BAJO LABORDA, 842748@unizar.es

ÁLVARO DE FRANCISCO NIEVAS, 838819@unizar.es

Universidad Zaragoza
1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Index

1. Introduction

In this practice, we have used Kubernetes to test the "raft" system developed in practices 3 and 4.
We adapted the code from the previous practices to support its use with Kubernetes, which is different from using SSH for launching. By using Kubernetes, we launched a cluster that allows three instances of Raft nodes to communicate with each other, with which a client interacts to submit operations.

2. Changes Made to the Code

We made small changes to our Raft code to adapt it for deployment with Kubernetes. Mainly, we modified the files cltraft.go and main.go, which initialize the client and Raft nodes, respectively.

cltraft.go:

We retrieve the URL where the server should listen as an argument. We construct the string from the name we assign to the client, the DNS, and the port where it will listen.
After that, we call SometerOperacion while we don't find a leader node, and we verify the write we've made.

main.go:

We only changed the way each Raft node receives its name in the network and how it establishes the listening address.

3. Deployment with Kubernetes

Once the code was modified, we built the two main components we will use to launch our cluster.

Launch script

We start by deleting the cluster if it already existed previously to avoid errors. Then, we create a new cluster using kind.
Next, we compile and build the images for the client and server, and we push them to the local registry to make them accessible.
Finally, the script manages Kubernetes resources by removing previous instances of the application and deploying new resources using a specific configuration file.

Kubernetes Architecture

We use the .yaml files provided in the practice material as the base. We deploy three replicas of "raft" using a StatefulSet pattern, ensuring the unique identity of each instance (replica) and guaranteeing persistent state. For this, we use the "server" image that was built and exposed in the previous script.
We also provide a service that acts as an abstraction to access the "raft" service replicas within the Kubernetes cluster.
Additionally, we run a "client" pod to interact with the "raft" service. Like with "raft", we use the image built in the previous script.

In our system, the client communicates with the "service-raft" service through Kubernetes' internal network using the internal DNS service name. The "service-raft" service routes requests to one of the "raft" StatefulSet replicas on port 6000

4. Testing

To validate the correct functioning of the system, we checked the logs of the various services using the following commands: *kubectl logs client and kubectl logs raft-x* to view the output of the different processes.