Practical Assignment 5 - Distributed Systems

Kubernetes and Raft

Authors:

Fernando Pastor Peralta (897113)
Javier Murillo Jimenez (897154)

Submission date: 11/27/2025

Group: Pair: 1-6

Table of contents

Introduction:

The objective of this exercise is to deploy the distributed key/value storage service based on the Raft protocol designed in the previous exercise within a Kubernetes cluster, ensuring that each replica runs on a different worker node and that, in the event of a replica or an entire node in the cluster going down, Kubernetes automatically restarts it and Raft is able to restore its previous state without any data loss.

To do this, we had to adapt the code from the previous practice to allow connection via DNS in the Kubernetes service in a container environment.

Design of our solution:

Adapting the Raft server to Kubernetes

To adapt the code from the previous exercise, we made the necessary changes so that the nodes connect correctly to the DNS service we created (in svraft_go.yaml), explicitly setting the DNS address of each of the raft servers on the client, and on the raft servers, checking that the given parameter is an expected server name, constructing its address and that of the others, knowing in advance the DNS subdomain and the number of nodes in the system.

Next, we compiled the server and client .go code and obtained their executables in the raft/ folder.

Next, we created a Dockerfile to create the images for each one, and made them accessible in a local Docker container registry (localhost:5001) so that they could be obtained by Kubernetes (the registry and kind were created using the provided shell file "kind-with-registry.sh"). In these Dockerfiles, we mounted the Alpine Linux distribution that will run on Docker and decided on the files where the container will start and work.

```Shell
 #build creates the docker image and pushes it into the local
    registry #---Raft server image---
    docker build -t localhost:5001/srvraft -f Dockerfile.srvraft .
    docker push localhost:5001/srvraft
```

```
#---Raft client image---

docker build -t localhost:5001/cltraft -f Dockerfile.cltraft .

docker push localhost:5001/cltraft
```

The Dockerfiles and .yaml will be available in the appendix.

Running each node type in its specific Pod

This second part can be seen reflected in the .yaml files, where we will apply these configuration manifests for Kubernetes. For the client (cltraft_go.yaml), we have decided to use a basic Pod since it does not require fault tolerance. We also connect directly as clients, automatically taking the client image from the local registry. In contrast, in the Raft replicas (srvraft_go.yaml), we have used a StatefulSet since we need a stable and unique identity for each node, a fixed DNS (service on port 6000), and fault tolerance. The following applies directly launch the different pods:

```
Shell

  kubectl apply -f srvraft_go.yaml

  kubectl apply -f cltraft_go.yaml
```

Structure of the system deployed in Kubernetes

Our solution is organized into three components:

- 3 Raft replicas of the storage server, executed using a StatefulSet.
- A headless Service for DNS.
- A pod that runs the client.

The StatefulSet provides us with stable DNS names for each replica, which is necessary for each node to build the RPC addresses of the others. The headless Service allows Kubernetes to register each Pod in the DNS without the need to associate a virtual IP.

Docker containers

To isolate the executables and allow their deployment in Kubernetes, we have generated two independent Docker images:

- Image of the Raft server, built from Alpine Linux and containing only the "srvraft" executable.
- Image of the automatic client, also based on Alpine, which executes the "cltraft" binary.

Both images, found in the appendix, have been statically built from Golang code (compiled with "CGO_ENABLED = 0") and uploaded to the local registry "localhost:5001" that Kubernetes uses as a container repository. This ensures that the cluster nodes can download the images without the need for external repositories.

Kubernetes manifests

We have designed the following manifests:

1. Client pod

Runs the "cltraft" binary with a "Never" policy so that Kubernetes does not recreate it if it dies.

2. Headless service

Define the "raft-service" service, which is necessary for Kubernetes to register Raft replicas in its internal DNS. We set "clusterIp: None" so that Kubernetes does not assign a virtual IP and instead each one gets its own stable DNS name.

3. StatefulSet of replicas

The StatefulSet creates three replicas of the Raft server "srvraft," each running in a different container. Using the environment variable "MINOMBREPOD," each node obtains its name within the cluster and, from there, calculates the addresses of the other replicas.

The manifests are included in the appendix.

Validation tests

For the validation tests, we created a bash script that checks whether the client logs indicate that the operations have been successful. The script performs the following tests:

1. Deployment and normal operation: after cleaning the environment, we start up the servers and client. The client performs four reads and four writes (one of them on the same key to confirm that the value is changed correctly). If everything goes well, the client prints "SUCCESS" in one of its strings. When this happens, the bash script detects it and passes the first test.

2. We kill the Pod of the leader raft node: after analyzing the logs to identify the leader node, we forcibly remove it to verify that Kubernetes regenerates it and that Raft is able to choose a new leader and continue to operate correctly with the client.

3. Node fault tolerance test (Docker crash): In bash, we identify the node (Docker container) running the new leader and stop it (simulating a network outage) to verify that the system continues to function correctly with the remaining majority. We then restore the node.

Appendix

A. Dockerfiles

a. Dockerfile.cltraft

```
None

#We mount the Alpine Linux distribution in our image.

FROM alpine

#From now on, everything you do will go to /usr/local/bin.

WORKDIR /usr/local/bin

#The application inside the container listens on port 6000.

EXPOSE 6000

#Copy the Raft server executable. COPY

srvraft /usr/local/bin/srvraft #Start the

server.

ENTRYPOINT ["/usr/local/bin/srvraft"]
```

b. Dockerfile.srvraft

```
None

#Mount the Alpine Linux distribution in our image.

FROM alpine

#From now on, everything that is done will go to /usr/local/bin.

WORKDIR /usr/local/bin

#Copy the Raft server executable. COPY

cltraft /usr/local/bin/cltraft #Start the

server.

ENTRYPOINT ["/usr/local/bin/cltraft"]
```

B. Yaml

a. cltraft_go.yaml (Client pod)

```
None
#Automatic client, connected to the Raft

service apiVersion: v1

kind: Pod

metadata:

  name: clientraft

spec:

  restartPolicy: Never

  containers:

  - name: cltraft

    image: localhost:5001/cltraft:latest

    ports:

    - containerPort: 7000
```

b. srvraft_go.yaml (Headless service and StatefulSet)

```
None
#Headless service for Kubernetes to register the pods in the DNS. apiVersion: v1

kind: Service

metadata:

  name: raft-service

  labels:

    app: raft

spec:

  clusterIP: None

  selector: #must match the label defined in the Pods.
```

```
    app: raft
  ports:
  - port: 6000
    name: raft-port
    protocol: TCP
    targetPort: 6000
---
#StatefulSet of Raft replicas
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: raft
spec:
  serviceName: raft-service
  replicas: 3
  podManagementPolicy: Parallel
  selector:
    matchLabels:
      app: raft          # must correspond to
.spec.template.metadata.labels
  template:
    metadata:
      labels:
        app: raft
    spec:
      terminationGracePeriodSeconds: 10
      containers:
      - name: srvraft
```

```
        image: localhost:5001/srvraft:latest

        env:

        - name: MYPODNAME    #first replica raft-0, second raft-1, etc...

          valueFrom:

            fieldRef:

              fieldPath: metadata.name

        command:

        - /usr/local/bin/srvraft

        - $(MYPODNAME)

        ports:

        - containerPort: 6000
```

C. Execution and commands used

a. Image building and uploading

```
Shell

# Compile the client and server executables

CGO_ENABLED=0 go build -o cltraft pkg/cltraft/main.go

CGO_ENABLED=0 go build -o srvraft cmd/srvraft/main.go

# Upload the images to kind-registry

#---Raft server image---

docker build -t localhost:5001/srvraft -f Dockerfile.srvraft .

docker push localhost:5001/srvraft

#---Raft client image ---

docker build -t localhost:5001/cltraft -f Dockerfile.cltraft .

docker push localhost:5001/cltraft
```

b. System startup

```
Shell

# Apply the Kubernetes configuration manifests kubectl apply -f

srvraft_go.yaml

kubectl apply -f cltraft_go.yaml
```

c. Testing

```
Shell

bash test.bash

# Next, test.bash #!/bin/bash

# Configuration

CLIENT="cltraft_go.yaml"

SERVER="srvraft_go.yaml"

POD_CLI="clientraft"

# Function to run client and validate success. verify_client() {

    kubectl delete pod $POD_CLI --grace-period=0 --force 2>/dev/null

    kubectl apply -f $CLIENT

    echo "   ...Waiting for client execution..."

    #The script pauses here until the pod finishes its work (Succeeded
or Failed status).

    while true; do

        #Get the current status (Running, Pending, Succeeded,
Failed, etc.).

        PHASE=$(kubectl get pod $POD_CLI -o jsonpath='{.status.phase}'
2>/dev/null)

        # If it has already finished (successfully or unsuccessfully), we exit
        the loop.

        if [[ "$PHASE" == "Succeeded" || "$PHASE" == "Failed" ]]; then

            break
        fi

        #If it is still running, we wait a

        little. sleep 2

    done

    LOGS=$(kubectl logs $POD_CLI)

    if echo "$LOGS" | grep -q "SUCCESS";

        then echo "  -> OK: Test passed.\n"

        echo "--------------------------------------------------------"

        kubectl get pods -o wide

        echo "--------------------------------------------------------"

        echo "$LOGS"

        echo "--------------------------------------------------------"

    else

        echo "   -> ERROR: The client

        failed." echo "$LOGS"

        exit 1

    fi

}

echo "=== STARTING RAFT TEST IN KUBERNETES ==="

echo "[1] Deploying Servers..."

kubectl delete pod --all --grace-period=0 --force 2>/dev/null

kubectl apply -f $SERVER

echo "   Waiting for pods to be ready..." sleep 5

echo "[2] Basic Test (Write/Read)" verify_client

# Save logs to find the leader
```

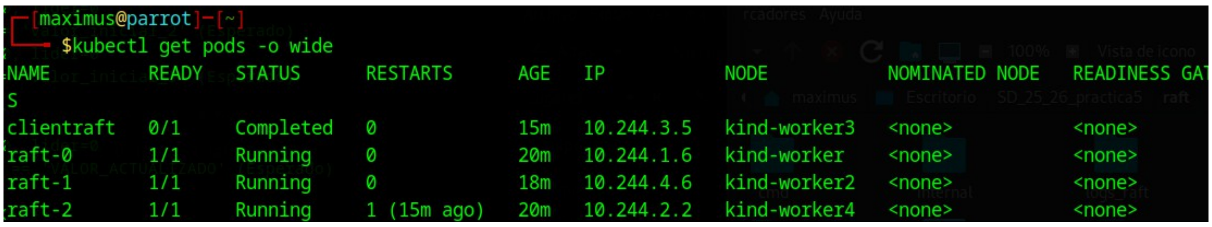
```
PREVIOUS_LOGS=$(kubectl logs $POD_CLI)

#We search for the ID of the leader to kill in the logs from the previous

execution. LEADER_ID=$(echo "$PREVIOUS_LOGS" | grep -o 'leader=[0-9]*' | tail -1

| cut -d=
-f2)

echo "[3] Deleting Leader Pod (raft-$LEADER_ID)..."

kubectl delete pod raft-$LEADER_ID --grace-period=0 --force

2>/dev/null echo "   Waiting for regeneration..."

sleep 10

verify_client # Check that it is still working #

Save logs to find the leader

PREVIOUS_LOGS=$(kubectl logs $POD_CLI)

#We search for the ID of the leader to kill in the logs from the previous
execution.

LEADER_ID=$(echo "$PREVIOUS_LOGS" | grep -o 'leader=[0-9]*' | tail -1 | cut -d=
-f2)

LEADER_NODE=$(kubectl get pods -o wide | grep -E "raft-$LEADER_ID.*" | awk
'{print $7}' )

echo "[4] Stopping Docker container on node ($LEADER_ID)..."

docker stop $LEADER_ID

sleep 8

echo "   Testing with node down..."

verify_client

echo "[5] Restoring node..."

docker start $LEADER_NODE

echo " "

kubectl get pods -o wide

echo "=== END ==="
```

d. Delete everything

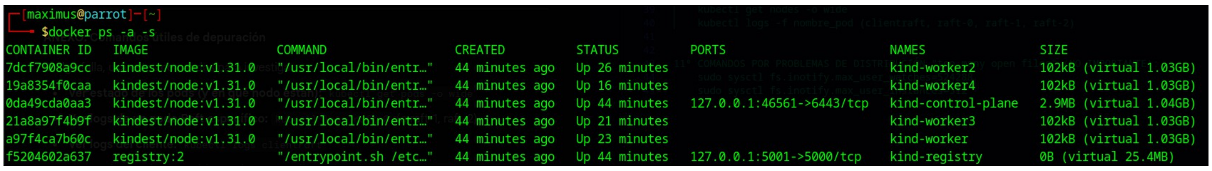
```
Shell

kind delete cluster

docker stop kind-registry

docker rm kind-registry
```

e. Other commands

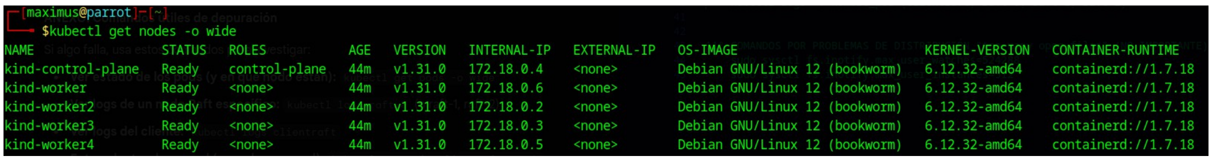
```
Shell

# Display running pods kubectl get

pods -o wide

# Display Docker containers on the host system docker

ps -a -s

# Display all nodes in the Kubernetes cluster kubectl

get nodes -o wide

# Display logs in real time

kubectl logs -f pod_name (clientraft, raft-0, raft-1, raft-2)
```



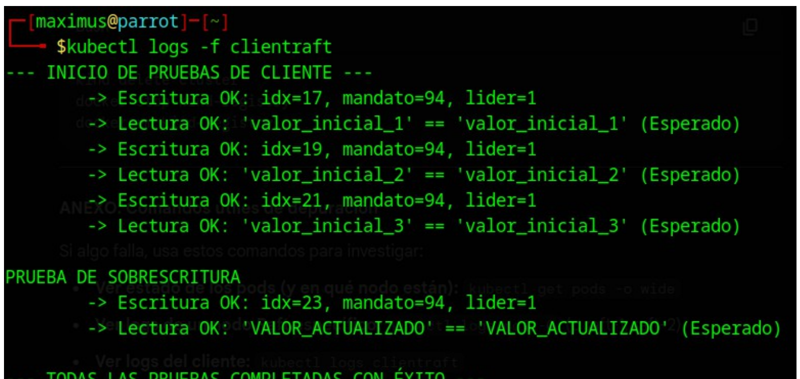
```
┌─[maximus@parrot]─[~]
└──╼ $kubectl get pods -o wide
NAME         READY   STATUS      RESTARTS        AGE   IP            NODE           NOMINATED NODE   READINESS GAT
S
clientraft   0/1     Completed   0               15m   10.244.3.5    kind-worker3   <none>           <none>
raft-0       1/1     Running     0               20m   10.244.1.6    kind-worker    <none>           <none>
raft-1       1/1     Running     0               18m   10.244.4.6    kind-worker2   <none>           <none>
raft-2       1/1     Running     1 (15m ago)     20m   10.244.2.2    kind-worker4   <none>           <none>
```



```
┌─[maximus@parrot]─[~]
└──╼ $docker ps -a -s
CONTAINER ID   IMAGE                  COMMAND                  CREATED          STATUS          PORTS                       NAMES                SIZE
7dcf7908a9cc   kindest/node:v1.31.0   "/usr/local/bin/entr…"   44 minutes ago   Up 26 minutes                               kind-worker2         102kB (virtual 1.03GB)
19a8354f0caa   kindest/node:v1.31.0   "/usr/local/bin/entr…"   44 minutes ago   Up 16 minutes                               kind-worker4         102kB (virtual 1.03GB)
0da49cda0aa3   kindest/node:v1.31.0   "/usr/local/bin/entr…"   44 minutes ago   Up 44 minutes   127.0.0.1:46561->6443/tcp   kind-control-plane   2.9MB (virtual 1.04GB)
21a8a97f4b9f   kindest/node:v1.31.0   "/usr/local/bin/entr…"   44 minutes ago   Up 21 minutes                               kind-worker3         102kB (virtual 1.03GB)
a97f4ca7b60c   kindest/node:v1.31.0   "/usr/local/bin/entr…"   44 minutes ago   Up 23 minutes                               kind-worker          102kB (virtual 1.03GB)
f5204602a637   registry:2             "/entrypoint.sh /etc…"   44 minutes ago   Up 44 minutes   127.0.0.1:5001->5000/tcp    kind-registry        0B (virtual 25.4MB)
```



```
┌─[maximus@parrot]─[~]
└──╼ $kubectl get nodes -o wide
NAME                 STATUS   ROLES           AGE   VERSION   INTERNAL-IP   EXTERNAL-IP   OS-IMAGE                         KERNEL-VERSION   CONTAINER-RUNTIME
kind-control-plane   Ready    control-plane   44m   v1.31.0   172.18.0.4    <none>        Debian GNU/Linux 12 (bookworm)   6.12.32-amd64    containerd://1.7.18
kind-worker          Ready    <none>          44m   v1.31.0   172.18.0.6    <none>        Debian GNU/Linux 12 (bookworm)   6.12.32-amd64    containerd://1.7.18
kind-worker2         Ready    <none>          44m   v1.31.0   172.18.0.2    <none>        Debian GNU/Linux 12 (bookworm)   6.12.32-amd64    containerd://1.7.18
kind-worker3         Ready    <none>          44m   v1.31.0   172.18.0.3    <none>        Debian GNU/Linux 12 (bookworm)   6.12.32-amd64    containerd://1.7.18
kind-worker4         Ready    <none>          44m   v1.31.0   172.18.0.5    <none>        Debian GNU/Linux 12 (bookworm)   6.12.32-amd64    containerd://1.7.18
```



```
┌─[maximus@parrot]─[~]
└──╼ $kubectl logs -f clientraft
--- INICIO DE PRUEBAS DE CLIENTE ---
    -> Escritura OK: idx=17, mandato=94, lider=1
    -> Lectura OK: 'valor_inicial_1' == 'valor_inicial_1' (Esperado)
    -> Escritura OK: idx=19, mandato=94, lider=1
    -> Lectura OK: 'valor_inicial_2' == 'valor_inicial_2' (Esperado)
    -> Escritura OK: idx=21, mandato=94, lider=1
    -> Lectura OK: 'valor_inicial_3' == 'valor_inicial_3' (Esperado)

PRUEBA DE SOBRESCRITURA
    -> Escritura OK: idx=23, mandato=94, lider=1
    -> Lectura OK: 'VALOR_ACTUALIZADO' == 'VALOR_ACTUALIZADO' (Esperado)

--- TODAS LAS PRUEBAS COMPLETADAS CON ÉXITO
```

D. The output of our script

```
Shell

 === START RAFT TEST IN KUBERNETES ===

[1] Deploying servers...

pod "clientraft" force deleted from default namespace

pod "raft-0" force deleted from default namespace

pod "raft-1" force deleted from default namespace

pod "raft-2" force deleted from default namespace

service/raft-service unchanged

statefulset.apps/raft unchanged

   Waiting for pods to be ready...

[2] Basic Test (Write/Read)

pod/clientraft created

   ...Waiting for client execution...

   -> OK: Test passed.

--------------------------------------

NAME         READY   STATUS      RESTARTS    AGE     IP             NODE
NOMINATED NODE   READINESS GATES

clientraft   0/1     Completed   0           2m18s   10.244.4.5
kind-worker2   <none>           <none>

raft-0       1/1     Running     0           2m24s   10.244.1.6
kind-worker    <none>           <none>

raft-1       1/1     Running     0           2m24s   10.244.3.3
kind-worker3   <none>           <none>

raft-2       1/1     Running     0           2m24s   10.244.2.14
kind-worker4   <none>           <none>

--------------------------------------

--- START OF CLIENT TESTS ---

       -> Write OK: idx=1, command=38, leader=1
       -> Read OK: 'initial_value_1' == 'initial_value_1' (Expected)

       -> Write OK: idx=3, command=38, leader=1

       -> Read OK: 'initial_value_2' == 'initial_value_2' (Expected)

       -> Write OK: idx=5, command=38, leader=1

       -> Read OK: 'initial_value_3' == 'initial_value_3' (Expected)


OVERWRITE TEST

       -> Write OK: idx=7, command=38, leader=1

       -> Read OK: 'UPDATED_VALUE' == 'UPDATED_VALUE' (Expected)


--- ALL TESTS COMPLETED SUCCESSFULLY ---

-----------------------------------------

[3] Deleting Leader Pod (raft-1)...

pod "raft-1" force deleted from default namespace

   Waiting for regeneration...

pod "clientraft" force deleted from default namespace

pod/clientraft created

   ...Waiting for client execution...

   -> OK: Test passed.\n

-----------------------------------------

NAME          READY   STATUS      RESTARTS    AGE     IP             NODE
NOMINATED NODE   READINESS GATES

clientraft    0/1     Completed   0           2m17s   10.244.3.4
kind-worker3    <none>           <none>

raft-0        1/1     Running     0           4m51s   10.244.1.6
kind-worker     <none>           <none>

raft-1        1/1     Running     0           2m27s   10.244.4.6
kind-worker2    <none>           <none>
```

```
raft-2        1/1     Running     0          4m51s   10.244.2.14
kind-worker4    <none>           <none>

-------------------------------------

--- START OF CLIENT TESTING ---

       -> Write OK: idx=9, command=93, leader=2

       -> Read OK: 'initial_value_1' == 'initial_value_1' (Expected)

       -> Write OK: idx=11, command=93, leader=2

       -> Read OK: 'initial_value_2' == 'initial_value_2' (Expected)

       -> Write OK: idx=13, command=93, leader=2

       -> Read OK: 'initial_value_3' == 'initial_value_3' (Expected)


OVERWRITE TEST

       -> Write OK: idx=15, command=93, leader=2

       -> Read OK: 'UPDATED_VALUE' == 'UPDATED_VALUE' (Expected)


--- ALL TESTS COMPLETED SUCCESSFULLY ---

-------------------------------------

[4] Stopping node Docker container (kind-worker4)...

kind-worker4

   Testing with node down...

pod "clientraft" force deleted from default namespace

pod/clientraft created

   ...Waiting for client execution...

   -> OK: Test passed.\n

-------------------------------------

NAME          READY   STATUS      RESTARTS   AGE     IP              NODE
NOMINATED NODE   READINESS GATES

clientraft    0/1     Completed   0          5s      10.244.3.5
kind-worker3    <none>           <none>
```

```
raft-0        1/1     Running     0          5m4s    10.244.1.6
kind-worker     <none>           <none>

raft-1        1/1     Running     0          2m40s   10.244.4.6
kind-worker2    <none>           <none>

raft-2        1/1     Running     0          5m4s    10.244.2.14
kind-worker4    <none>           <none>

-------------------------------------

--- START OF CLIENT TESTS ---

       -> Write OK: idx=17, command=94, leader=1

       -> Read OK: 'initial_value_1' == 'initial_value_1' (Expected)

       -> Write OK: idx=19, command=94, leader=1

       -> Read OK: 'initial_value_2' == 'initial_value_2' (Expected)

       -> Write OK: idx=21, command=94, leader=1

       -> Read OK: 'initial_value_3' == 'initial_value_3' (Expected)


OVERWRITE TEST

       -> Write OK: idx=23, command=94, leader=1

       -> Read OK: 'UPDATED_VALUE' == 'UPDATED_VALUE' (Expected)


--- ALL TESTS COMPLETED SUCCESSFULLY        ---

-------------------------------------

[5] Restoring node...

kind-worker4


NAME          READY   STATUS      RESTARTS   AGE     IP             NODE
NOMINATED NODE   READINESS GATES

clientraft    0/1     Completed   0          5s      10.244.3.5
kind-worker3    <none>           <none>

raft-0        1/1     Running     0          5m4s    10.244.1.6
kind-worker     <none>           <none>
```

```
raft-1        1/1     Running     0           2m40s   10.244.4.6
kind-worker2   <none>           <none>

raft-2        1/1     Running     0           5m4s    10.244.2.14
kind-worker4   <none>           <none>

=== END ===
```