

# Trabajo Administración de Sistema - Kubernetes

## Parte 2

Hayk Kocharyan  
757715@unizar.es

3 de junio de 2020

## Índice

<b>1. Resumen</b>	<b>1</b>
<b>2. Arquitectura Y Despliegue</b>	<b>1</b>
2.1. Puesta en marcha de Ceph . . . . .	1
2.2. Despliegue Aplicación . . . . .	1
2.2.1. MySQL y Wordpress . . . . .	2
2.2.2. Depliegue de una Aplicación mas compleja . . . . .	2
2.3. Registro de repositorio privado de contenedores y almacenamiento de sistema de ficheros distribuido	3
<b>3. Problemas</b>	<b>3</b>
<b>4. Anexo - Comandos Usado</b>	<b>3</b>
<b>5. Anexo - Imágenes</b>	<b>4</b>
<b>Referencias</b>	<b>5</b>

## 1. Resumen

En esta segunda parte del proyecto, se ha desplegado una aplicación web y un repositorio privado en un sistema de fichero con despliegue de Ceph. Para ello, se usó el orquestrador Rook. Se ha tratado almacenamiento de bloques y de sistema de ficheros, haciendo uso del primero a través de la aplicación web, y del segundo a través del repositorio privado.

## 2. Arquitectura Y Despliegue

### 2.1. Puesta en marcha de Ceph

Para comenzar con la puesta en marcha de Ceph [1], en primer lugar se realizará con el despliegue de **Rook**. **Rook** Es un orquestrador nativo de almacenamiento para K8s, nos provee una plataforma, un *framework* y soporte para el manejo de almacenamiento nativo cloud.

Para comenzar se despliegan los recursos que necesita el operador Rook para Ceph a través del fichero **common.yaml**. Este fichero define el namespace y los recursos llamados **Custom Resource Definitions**, también conocidos como **CRDs**. Estos recursos normalmente no existen en K8s y son usados por el Operador<sup>1</sup>.

A continuación, se despliega el operador *Ceph*, que será el encargado de poner en marcha y gestionar la orquestación de *Ceph* en el cluster. Esto se lleva a cabo con el fichero **operator.yaml**.

Una vez todo en marcha, con el siguiente comando podemos obtener el estado del *cluster* y su salida en la **figura 1**.

```
kubectl get pod -n rook-ceph -o wide
```

En este momento se ha desplegado un agente **rook-discovery** en cada uno de los nodos existentes del *cluster*. También podemos observar el despliegue de un *daemonset*. El objetivo principal de este controlador es controlar que todos los nodos dispongan de un pod concreto, en este caso es el pod que contiene el proceso es **rook-discovery**. Con este servicio corriendo, todo nodo que entre en el *cluster*, automáticamente dispondrá de un pod con el proceso **rook-discovery** corriendo en él. Además si un nodo se borra, estos Pods son recolectados.

Nuestro siguiente paso es desplegar el *cluster* a través de **cluster.yaml**. Básicamente definimos la configuración del *cluster*. Un aspecto a destacar de este fichero de configuración es que se exige que todos los nodos de K8s se usen para desplegar procesos de Ceph y que todos los dispositivos vinculados a los nodos se usen para tener una buena persistencia.

En la **figura 2** podemos ver como tras crear el *cluster*, han aumentado notablemente los Pods en ejecución, esto es debido a que el operador de Ceph ha notificado de la creación del *cluster Ceph* y avisa a través de la API Server para crear todos los Pods relacionados con Ceph. Podemos ver Pods **mon** corriendo en los 3 nodos, estos son Pods con un proceso *Monitor* responsable del mantenimiento del mapeado del estado del cluster. También vemos un Pod con proceso **mgr**, el *Manager*, demonio responsables de mantener un registro de las métricas y del estado del cluster Ceph. Por último, vemos los **osd**, el demonio responsable de almacenar objetos, manejar la replicación, recuperación y el balanceo.

Por último, vamos a desplegar un kit de herramientas para *Ceph* a través de **toolbox.yaml**, este contiene las herramientas más comunes para depurar y testear el *cluster Ceph*.

Para acceder al Pod que contiene las herramientas podemos hacerlo a través del siguiente comando:

Una vez dentro del Pod, podemos ejecutar comando como *Ceph status* y obtener la salida que se puede observar en esta **figura 3**. Podemos observar detalles como en cuantas máquinas está corriendo el demonio de monitorización o el manager.

### 2.2. Despliegue Aplicación

Para que Rook nos pueda proveer de almacenamiento, es necesario crear un *StorageClass* y un *CephBlockPool*, esto permite a K8s interoperar con Rook para proveer los volúmenes persistentes necesarios. Pondremos todo lo

---

<sup>1</sup>proceso que corre en un pod y contiene toda la lógica de manejo de la aplicación

anterior en marcha con el manifiesto **storageClassRbdBlock.yaml**. A continuación, si ejecutamos *ceph status* en el *toolbox*, podemos ver como ahora disponemos de un pool activo y limpio.

Para hacer uso del almacenamiento de bloques creado anteriormente, vamos a desplegar dos aplicaciones diferentes

### 2.2.1. MySQL y Wordpress

Lanzaremos una bbdd MySQL y un app simple en Wordpress para comprobar el uso del almacenamiento a través de los manifiestos **mysql.yaml** y **wordpress.yaml**. Previamente, será necesario modificar los ficheros para completar la información que falta. Los cambios son los siguientes:

- En la declaración del PVC lo vincularemos al StorageClass, creado previamente, llamado *rook-ceph-block*.
- El *volumeMounts* de los contenedores se asignará al **volumen** declarado en el mismo manifiesto, asegurando que los nombres coinciden.
- El *volumen* que se declara hay que vincularlo con el PVC declarado al principio de cada manifiesto.

Si el proceso no ha fallado, tendremos acceso a wordpress y podremos crear contenido. También tendremos acceso a la base de datos de MySQL y podremos manipular la base .

Como nota adicional cabe destacar que si borramos un PVC ahora, el volumen rbd también se eliminará, esto es debido a que al crear el StorageClass *reclaimPolicy* se ha establecido en *Delete*, si optásemos por *Retain*, este se mantendría y tendríamos que borrarlo manualmente.

Adicionalmente podemos desplegar un pod *direct-mount* desde el manifiesto *direct-mount.yaml* y manipular los bloques montándolos en el pod y comprobando la disponibilidad desmontando y apagando máquinas para comprobar que los datos persisten. También se pueda hacer la prueba con dos pods *direct-mount* y montando el mismo dispositivo, de esta manera comprobamos que los bloques son compartidos, además si desmontamos uno de los direct-mounts podemos comprobar que el otro sigue en funcionamiento y disponible.

### 2.2.2. Depliegue de una Aplicación mas compleja

Para ampliar esta sección, se ha decidido desplegar una aplicación mas compleja. La aplicación se desplegó previamente en *docker*, por lo que se disponía del fichero *docker-compose*, y los *Dockerfile* de los diferentes contenedores. La estructura es la siguiente:

- Base de datos en *PostgreSQL* que usa la última imagen disponible en *Docker Hub*. 1 volumen asignado para almacenar los datos.
- Api REST en Django, contenedor desplegado con un Dockerfile propio. Tiene asignados 3 volúmenes para sus diferentes datos.
- Aplicación Web en Node.js y Vue, desplegado con un Dockerfile propio. Con un volumen asignado para sus datos.

Para completar el despliegue ha sido necesario el uso de la herramienta **Komposer**, quien ha traducido el fichero *docker-compose* a manifiestos kubernetes. Sin embargo, esto no ha sido suficiente, ya que muchas cosas no se traducían correctamente o estaban incompletos, por lo que ha sido necesario un estudio de los ficheros y una modificación manual. Adicionalmente, se ha asignado a cada uno de los PVC creados un StorageClass como en el apartado anterior con MySQL. Finalmente, se dispone de 3 manifiestos principales:

- *postgres-deployment.yaml* para el despliegue de la base
- *django-deployment.yaml* para el despliegue de la api
- *web-deployment.yaml* para el despliegue de la web

Para terminar, también ha sido necesario la creación de dos ficheros de **NetworkPolicy** para permitir tráfico entre los 3 elementos desplegados.

## 2.3. Registro de repositorio privado de contenedores y almacenamiento de sistema de ficheros distribuido

En esta parte vamos a crear un sistema de ficheros compartido [4] y distribuido para los pods de nuestro sistema. En primer lugar, definiremos el sistema de ficheros con el manifiesto *filesystem.yaml*, el operador de Rook se encargará de poner en marcha todos los pool y servicios necesarios. Para confirmar el correcto despliegue, tendremos que asegurarnos que se han creado los pods mds. Otra comprobación que podemos realizar es un *ceph status* en el pod *toolbox* y asegurarnos que nos aparece la línea *mds: myfs:1 0=myfs-b=up:active 1 up:standby-replay*. Para poder usar el sistema de ficheros con PV es necesario crear un StorageClass para ello aplicamos el manifiesto *storageClassCephFs.yaml*.

Una vez disponemos del sistema de ficheros, podemos crear el repositorio privado[3], para ello aplicamos el manifiesto *kube-registry.yaml* y lo exponemos a través de *kubeRegistryService.yaml*. Por último, para que las nodos del cluster tengan acceso al servicio, usaremos el manifiesto *kubeRegistryProxy.yaml*. Si queremos usar el repositorio desde host, buscaremos el pod que se encarga del registro y habilitaremos una redirección de puertos hacia este. En nuestro caso no realizaremos el push y pull desde el repositorio local, ya que Podman únicamente funciona con un kernel de Linux. Las pruebas las realizamos desde un nodo del cluster, para ello pulleamos una imagen busybox a través de podman, y pusheamos este al repositorio local. Para comprobar que el repositorio funciona correctamente, dentro del pod *direct-mount*, montamos el sistema de ficheros y buscamos la imagen que hemos pusheado.

## 3. Problemas

Los principales problemas se han encontrado a la hora de desplegar rook y ceph, ya que si no se esperaba el tiempo exacto, había problemas de despliegue. También se ha encontrado problemas con el despliegue de la aplicación extra, ya que *Komposer* dejaba ciertas cosas incompletas y se tenía que comprender todo el manifiesto para ver que fallos había y poder solucionarlos. Destacar que por problemas de tiempo, no se ha habilitado el sistema de ficheros para esta aplicación, no obstante, esta tarea queda pendiente para la siguiente entrega. Otro de los fallos ha estado en el apartado del repositorio. En primer lugar, podman no era compatible sin un kernel de linux, por lo que en MacOS no era posible su uso. Para solucionarlo, se ha trabajado desde uno de los workers, se ha conseguido pushear una imagen, pero tras muchos intentos y con muchos errores 502, y eso si, sin llegar a completarse un push de principio a final. Como tarea adicional se ha incorporado TLS para el repositorio[2], no obstante, no se ha podido probar debido al problema explicado con podman. De todas formas, se han creado los certificados autofirmados, se han creado los secrets de kubernetes, y se ha desplegado un nuevo kube registry con el manifiesto *rcRegistry.yaml* y se ha expuesto el servicio con *regServ.yaml*. Tras seguir los pasos, no se ha conseguido poner en marcha TLS.

## 4. Anexo - Comandos Usado

```
kubectl -n rook-ceph exec -it nombrePodToolbox bash
```

```
#para obtener el nombre del pod
kubectl -n rook-ceph get pod -l "app=rook-ceph-tools"
-o jsonpath='{.items[0].metadata.name}'
```

```
POD=$(kubectl get pods --namespace kube-system -l k8s-app=kube-registry \
-o template --template '{{range _ .items}}{{.metadata.name}}_ \
{{.status.phase}}\n'}} \
| grep Running | head -1 | cut -f1 -d'_')
```

```
kubectl port-forward --namespace kube-system $POD 5000:5000 &
```

```
# Entrar en el pod de MySQL
```

```
kubectl exec -it wordpress-mysql-7f9db94b75-m2fxx -- mysql --password
```

```
#confirmar que el sistema de ficheros se ha configurado correctamente
kubectl -n rook-ceph get pod -l app=rook-ceph-mds
```

```
podman pull quay.io/prometheus/busybox
```

```
podman push --tls-verify=false 0a74f8eccb8c localhost:5000/hayk/busybox
```

```
creamos el directorio del registro
mkdir /tmp/registry
```

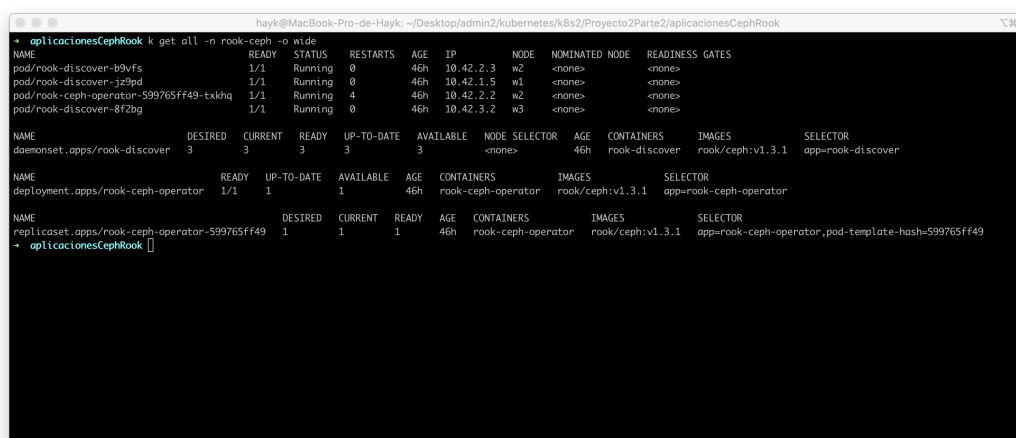
```
# buscamos los mon endpoints y el user secret para la conexión
mon_endpoints=$(grep mon_host /etc/ceph/ceph.conf | awk '{print_$3}')
my_secret=$(grep key /etc/ceph/keyring | awk '{print_$3}')
```

```
# montamos el sistema de ficheros
mount -t ceph -o mds_namespace=myfs,name=admin,secret=$my_secret
$mon_endpoints:/ /tmp/registry
```

```
# comprobamos que está
df -h
```

```
#buscamos
find . -iname nombreImagen
```

## 5. Anexo - Imágenes



```
hayk@MacBook-Pro-de-Hayk: ~/Desktop/admin2/kubernetes/k8s2/Proyecto2Parte2/aplicacionesCephRook
+ aplicacionesCephRook k get all -n rook-ceph -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE   NOMINATED NODE   READINESS GATES
pod/rook-discover-b9vfs             1/1     Running   0           46h   10.42.2.3       w2     <none>            <none>
pod/rook-discover-1z9pd             1/1     Running   0           46h   10.42.1.5       w1     <none>            <none>
pod/rook-ceph-operator-599765ff49-txkhq 1/1     Running   4           46h   10.42.2.2       w2     <none>            <none>
pod/rook-discover-8f2bg             1/1     Running   0           46h   10.42.3.2       w3     <none>            <none>

NAME                                DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE   CONTAINERS   IMAGES           SELECTOR
daemonset.apps/rook-discover        3         3         3       3             3           <none>          46h   rook-discover  rook/ceph:v1.3.1 app=rook-discover

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE   CONTAINERS   IMAGES           SELECTOR
deployment.apps/rook-ceph-operator  1/1     1             1           46h   rook-ceph-operator  rook/ceph:v1.3.1 app=rook-ceph-operator

NAME                                DESIRED   CURRENT   READY   AGE   CONTAINERS   IMAGES           SELECTOR
replicaset.apps/rook-ceph-operator-599765ff49 1       1         1       46h   rook-ceph-operator  rook/ceph:v1.3.1 app=rook-ceph-operator,pod-template-hash=599765ff49
+ aplicacionesCephRook []
```

Figura 1: Salida obtenida del estado del despliegue de Rook

```
hayk@MacBook-Pro-de-Hayk: ~/Desktop/admin2/kubernetes/k8s2/Proyecto2Parte2/aplicacionesCephRook
+ aplicacionesCephRook k get pods -n rook-ceph -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
rook-discover-b9vfs                 1/1     Running   0           47h   10.42.2.3       w2       <none>            <none>
rook-discover-jz9pd                 1/1     Running   0           47h   10.42.1.5       w1       <none>            <none>
rook-ceph-operator-599765ff49-txkhq 1/1     Running   4           47h   10.42.2.2       w2       <none>            <none>
rook-discover-8f2bg                 1/1     Running   0           47h   10.42.3.2       w3       <none>            <none>
csi-rbdplugin-jkss5                 3/3     Running   0           15m   192.168.0.95    w3       <none>            <none>
csi-cephfsplugin-d2gd7              3/3     Running   0           15m   192.168.0.95    w3       <none>            <none>
csi-rbdplugin-laj5p                 3/3     Running   0           15m   192.168.0.93    w1       <none>            <none>
csi-cephfsplugin-w6rpx              3/3     Running   0           15m   192.168.0.93    w1       <none>            <none>
csi-rbdplugin-59jhd                 3/3     Running   0           15m   192.168.0.94    w2       <none>            <none>
csi-cephfsplugin-4frs6              3/3     Running   0           15m   192.168.0.94    w2       <none>            <none>
csi-cephfsplugin-provisioner-674847b584-4r2zh 5/5     Running   0           15m   10.42.1.6       w1       <none>            <none>
csi-cephfsplugin-provisioner-674847b584-hrpgv 5/5     Running   0           15m   10.42.2.5       w2       <none>            <none>
csi-rbdplugin-provisioner-5777f9cf96-d27nz 6/6     Running   0           15m   10.42.1.7       w1       <none>            <none>
csi-rbdplugin-provisioner-5777f9cf96-sxsn5 6/6     Running   0           15m   10.42.2.4       w2       <none>            <none>
rook-ceph-mon-a-594445468b-9cdb4     1/1     Running   0           14m   10.42.3.7       w3       <none>            <none>
rook-ceph-mon-b-6577fcf877-pptkj     1/1     Running   0           14m   10.42.2.7       w2       <none>            <none>
rook-ceph-mon-c-7d7c4d764-zgtq5     1/1     Running   0           13m   10.42.1.9       w1       <none>            <none>
rook-ceph-mgr-a-586d4b4c4b-78zpt     1/1     Running   0           13m   10.42.3.8       w3       <none>            <none>
rook-ceph-crashcollector-w1-5d74c5755f-6rg4l 1/1     Running   0           13m   10.42.1.11      w1       <none>            <none>
rook-ceph-crashcollector-w2-75dc597ad-hzfrk 1/1     Running   0           14m   10.42.2.9       w2       <none>            <none>
rook-ceph-osd-prepare-w2-5vb48       0/1     Completed 0           12m   10.42.2.8       w2       <none>            <none>
rook-ceph-osd-prepare-w1-zc729       0/1     Completed 0           12m   10.42.1.10      w1       <none>            <none>
rook-ceph-osd-prepare-w3-ff7r8       0/1     Completed 0           12m   10.42.3.10      w3       <none>            <none>
rook-ceph-crashcollector-w3-fc5cb4c89-s5ljg 1/1     Running   0           12m   10.42.3.12      w3       <none>            <none>
rook-ceph-osd-0-548556c9d7-qjz94    1/1     Running   0           12m   10.42.2.10      w2       <none>            <none>
rook-ceph-osd-1-54f65b48bb-8pp4l    1/1     Running   0           12m   10.42.1.12      w1       <none>            <none>
rook-ceph-osd-2-5f788f7bc8-wsb7x    1/1     Running   0           12m   10.42.3.11      w3       <none>            <none>
+ aplicacionesCephRook
```

Figura 2: Pods en ejecución tras desplegar el *cluster Ceph*

```
@rook-ceph-tools-877c4d966-cln9b/
[root@rook-ceph-tools-877c4d966-cln9b /]# ceph status
cluster:
  id:      e0feaa9e-0d2c-49e7-bf3e-f014a4da2dff
  health: HEALTH_OK

services:
  mon: 3 daemons, quorum a,b,c (age 44m)
  mgr: a(active, since 44m)
  osd: 3 osds: 3 up (since 43m), 3 in (since 43m)

data:
  pools:   0 pools, 0 pgs
  objects: 0 objects, 0 B
  usage:   3.0 GiB used, 87 GiB / 90 GiB avail
  pgs:

[root@rook-ceph-tools-877c4d966-cln9b /]#
```

Figura 3: Estado de Ceph

## Referencias

- [1] L. Juggery. Deploy a ceph cluster on k8s with rook. <https://itnext.io/deploy-a-ceph-cluster-on-kubernetes-with-rook-d75a20c3f5b1>.
- [2] Kubernetes. Registro con tls. <https://github.com/kubernetes/kubernetes/blob/7ef585be224dae4ec5deae0f135653116e21a6e0/cluster/addons/registry/tls/README.md>.
- [3] Kubernetes. Repositorio privado. <https://github.com/kubernetes/kubernetes/blob/7ef585be224dae4ec5deae0f135653116e21a6e0/cluster/addons/registry/README.md#expose-the-registry-on-each-node>.
- [4] Rook. Filesystem. <https://rook.io/docs/rook/v0.7/filesystem.html>.