Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

adsis2-tf

Administración de Sistemas II

Autor 1:	Toral Pallás, Héctor - 798095
Grado:	Ingeniería Informática
Curso:	2022-2023

9 de junio de 2023

Índice

1. Resumen

En el marco de este proyecto, se ha llevado a cabo un despliegue exitoso de un clúster Rook-ceph, el cual ha sido aprovechado para implementar varias aplicaciones de manera simultánea, entre ellas una aplicación web que trabaja con WordPress y MySQL, y un repositorio de contenedores privado. Para lograr este despliegue de manera eficiente y reproducible, se ha utilizado una combinación de herramientas como Vagrant y scrips shells, se intentarón realizar pruebas con puppet y terraform pero debido a diversos problemas técnicos y de tiempo tuvieron que dejarse de lado a medio hacer.

El proceso de creación y configuración del clúster Rook-ceph ha sido clave para garantizar una infraestructura de almacenamiento distribuido confiable y escalable. A través del despliegue de los OSDs y la configuración adecuada de los monitores, se ha logrado establecer un entorno de alta disponibilidad, donde los datos se distribuyen y replican de manera efectiva para evitar la pérdida de información en caso de fallos.

Posteriormente, se ha aprovechado este clúster Rook-ceph como base para desplegar una serie de aplicaciones populares. WordPress, como plataforma de gestión de contenido, ha sido implementado en el clúster, aprovechando las ventajas de rendimiento y escalabilidad que ofrece la infraestructura de almacenamiento distribuido. Por otro lado, MySQL ha sido configurado como el sistema de gestión de bases de datos para respaldar las necesidades de almacenamiento de la aplicación. Además, se ha establecido un repositorio de contenedores privado, lo que permite almacenar y distribuir imágenes de contenedores de manera eficiente y segura dentro del clúster Rook-ceph.

La elección de herramientas como Vagrant ha facilitado en gran medida el despliegue y la configuración del clúster y las aplicaciones en un entorno automatizado. Mediante la creación de scripts y la definición de infraestructura como código, se ha logrado agilizar el proceso de implementación y permitir su replicabilidad en diferentes entornos. Esta aproximación basada en DevOps ha demostrado ser eficaz en la gestión y operación de la infraestructura, alineando el desarrollo y las operaciones de manera eficiente.

2. Introducción y objetivos

En esta práctica, se busca aplicar los conocimientos relacionados con almacenes de datos, alta disponibilidad, alta escalabilidad, monitorización distribuida, DevOps y aspectos de automatización de infraestructuras; Para ello se plantea desplegar un clúster Rook-ceph y utilizarlo como base para implementar una aplicación web. A través de la cual, se podrán adquirir conocimientos y habilidades prácticas en el uso de los RBD (bloques de datos distribuidos) y un repositorio privado para trabajar con CEPHFS (sistema de archivos distribuido de Ceph).

El despliegue del clúster Rook-ceph facilitará la comprensión de la organización y gestión de un sistema de almacenamiento distribuido, así como entender las ventajas de la alta disponibilidad y la escalabilidad inherentes a esta tecnología. A través del uso de RBD, se explorarán las capacidades de almacenamiento de bloques distribuidos y se aprenderá cómo gestionar y acceder a estos bloques de manera eficiente.

Además, la práctica incluye la implementación de un repositorio privado utilizando CEPHFS, lo que permitirá a los participantes experimentar con el sistema de archivos distribuido de Ceph, lo que implica aprender a trabajar con directorios y archivos en CEPHFS, así como comprender las ventajas de utilizar un sistema de archivos distribuido en un entorno de almacenamiento de alto rendimiento y escalabilidad.

A lo largo de la práctica, los participantes también tendrán la oportunidad de aplicar conceptos y técnicas relacionadas con la monitorización distribuida para garantizar el correcto funcionamiento del clúster y la aplicación web desplegada. Asimismo, se explorarán aspectos de automatización de infraestructuras y DevOps, permitiendo a los participantes adquirir habilidades en la gestión eficiente de la infraestructura de TI y la implementación ágil de servicios y aplicaciones.

3. Arquitectura del sistema

La arquitectura de la infraestructura basada en Rook-Ceph sobre Kubernetes en la que se desarrolla el proyecto combina dos potentes proyectos de código abierto para ofrecer un almacenamiento distribuido y altamente escalable a entornos de contenedores.

Para simplificar la implementación y administración de servicios de almacenamiento distribuido, tenemos a Rook, un orquestador de almacenamiento para Kubernetes que complementa a Ceph. Rook se encarga de automatizar la implementación, el aprovisionamiento y la gestión del clúster de Ceph, brindando una capa de abstracción que facilita la configuración y el mantenimiento de la infraestructura de almacenamiento. Por otro lado, tenemos Ceph, un sistema de almacenamiento distribuido que ofrece la capacidad de almacenar datos de manera eficiente y confiable, con características avanzadas como replicación, tolerancia a fallos y escalabilidad horizontal.

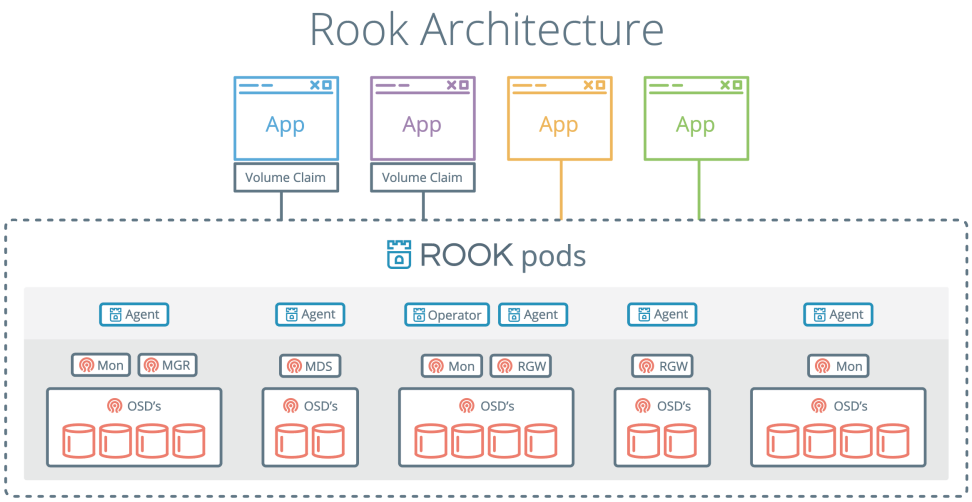

Figura 1: Arquitectura de almacenamiento con Rook.

3.1. Componentes

Figura 2: División del trabajo.

- **Agent**: El Agente (Agent) de Rook es responsable de ejecutar tareas específicas en los nodos del clúster de Kubernetes. Estas tareas incluyen la configuración del almacenamiento en los nodos, la gestión de servicios de red y otros aspectos relacionados con el despliegue y operación del almacenamiento distribuido.
- **Operator**: El Operador (Operator) de Rook es un controlador que se encarga de administrar la implementación, configuración y operaciones del clúster de almacenamiento distribuido. El Operador es responsable de crear y gestionar los recursos de Kubernetes necesarios para el funcionamiento de Rook, como los pods, servicios, volúmenes y configuraciones.

  El operador Rook define el estado deseado para el almacenamiento del clúster y ejecuta bucles de reconciliación para detectar cualquier cambio y volver al estado deseado.

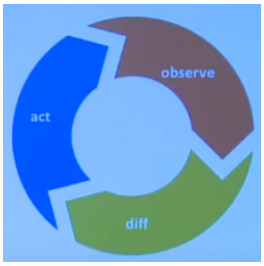

Figura 3: Ciclo de vida del operator.

- **MON (Monitor)**: Los Monitores (MON) son componentes esenciales del backend de Ceph. Los MONs se encargan de almacenar y distribuir los metadatos del clúster, así como de gestionar la configuración y el estado global del clúster. Los MONs se despliegan en múltiples nodos y cooperan para mantener la consistencia y disponibilidad del clúster.
- **MGR (Manager)**: El Gestor (MGR) es otro componente del backend de Ceph. El MGR proporciona servicios adicionales de administración y monitoreo para el clúster, como la generación de estadísticas, el seguimiento del estado de los OSDs y la interfaz de administración a través de una API.
- **OSD** (Object Storage Daemon): Los Daemons de Almacenamiento de Objetos (OSD) son los componentes finales del backend de Ceph. Los OSDs se encargan del almacenamiento y la recuperación de los datos en el clúster. Cada OSD administra su propio disco o dispositivo de almacenamiento y replica los datos en otros OSDs para garantizar la redundancia y la tolerancia a fallos.

3.2. CEPH

Ceph es un sistema de almacenamiento distribuido diseñado para ofrecer alta escalabilidad y confiabilidad en entornos de infraestructura a gran escala. Su arquitectura se compone de varios componentes clave, cada uno diseñado para manejar diferentes tipos de almacenamiento y casos de uso específicos.

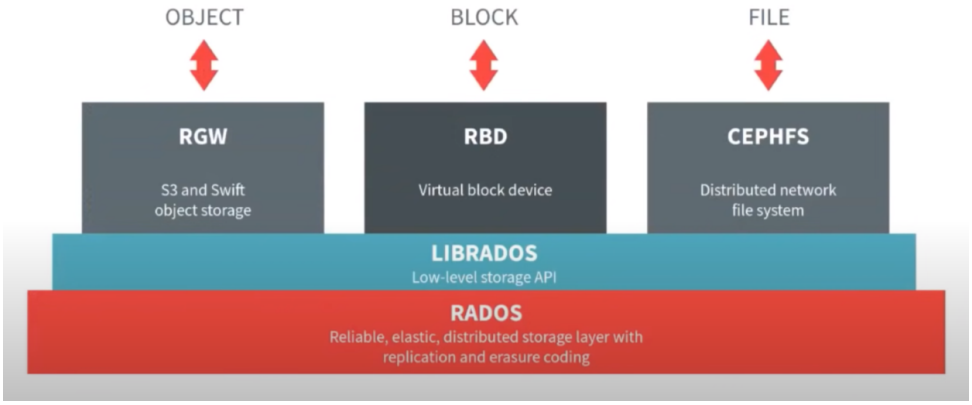

Figura 4: Arquitectura del sistema de almacenamiento distribuido CEPH.

Uno de los componentes principales de Ceph es **RADOS** (Reliable Autonomic Distributed Object Storage), que es el corazón del sistema. RADOS es responsable de almacenar y distribuir los datos de manera eficiente en el clúster de Ceph. Utiliza una arquitectura descentralizada en la que los datos se dividen en objetos y se distribuyen en múltiples OSDs (Object Storage Devices) en el clúster. Esto garantiza una alta disponibilidad y tolerancia a fallos, ya que los datos se replican y distribuyen de manera redundante en varios OSDs.

Librados (Ceph librados) es una interfaz de biblioteca que permite a las aplicaciones interactuar directamente con RADOS. Proporciona una API para acceder, leer y escribir objetos en el clúster de Ceph. Librados ofrece una interfaz de programación flexible y potente para que las aplicaciones puedan aprovechar las capacidades de almacenamiento distribuido de Ceph.

Ceph **RGW** (RADOS Gateway) es un componente que proporciona una interfaz compatible con S3 y Swift para el almacenamiento de objetos. Permite a las aplicaciones y usuarios acceder y almacenar objetos directamente a

través de HTTP/HTTPS utilizando protocolos como S3 (Amazon Simple Storage Service) y Swift (OpenStack Object Storage). RGW es una opción popular para casos de uso en la nube y aplicaciones web que requieren un almacenamiento de objetos altamente escalable y accesible.

RBD (RADOS Block Device) es un componente de Ceph que ofrece almacenamiento de bloques similar a una unidad de disco. Permite la creación de imágenes de disco virtual que se pueden usar como dispositivos de almacenamiento en servidores y máquinas virtuales. RBD es compatible con la mayoría de las tecnologías de virtualización, lo que lo convierte en una solución versátil para el almacenamiento de bloques en entornos de virtualización. Ver demo en la sección 7.1

Por último, **CephFS** es el sistema de archivos distribuido de Ceph. Proporciona un espacio de nombres unificado y compartido que se puede montar en múltiples nodos en un clúster de Ceph. CephFS ofrece un acceso de lectura/escritura de alto rendimiento y permite a los usuarios y aplicaciones trabajar con datos de manera similar a un sistema de archivos tradicional. Es una opción popular para casos de uso que requieren un almacenamiento de archivos distribuido y escalable. Ver demo en la sección 7.2

3.3. Gestión de objetos en CEPH

En Ceph, un archivo (como una imagen) se divide en **objetos**. Cada objeto es una unidad lógica de datos que se almacena en el clúster de Ceph. Estos objetos se distribuyen y replican en los OSDs (Object Storage Devices) dentro del clúster.

Un **pool** en Ceph es un contenedor lógico que agrupa un conjunto de objetos relacionados. Se puede pensar en un pool como un espacio de almacenamiento lógico que alberga objetos de un tipo específico. Por ejemplo, puedes tener un pool para imágenes, otro para archivos de configuración, etc. Cada pool tiene su propia configuración, su cantidad de réplicas y su propia políticas de almacenamiento.

Los grupos de ubicación (**placement groups**) son un mecanismo de particionado utilizado en Ceph para distribuir y gestionar los objetos de manera eficiente. Cada "placement groups" contiene un subconjunto de objetos dentro de un pool. La cantidad de "placement groups" en un pool depende de la configuración y el tamaño del clúster. Los "placement groups" permiten una distribución equitativa de los objetos en los OSDs y mejoran el rendimiento y la escalabilidad del sistema.

Los **OSDs** (Object Storage Devices) son los componentes físicos en el clúster de Ceph que almacenan y administran los objetos. Cada OSD es responsable de almacenar una parte de los datos y mantener su integridad y disponibilidad. Los OSDs manejan operaciones de lectura y escritura de objetos, así como la replicación y distribución de los datos en el clúster.

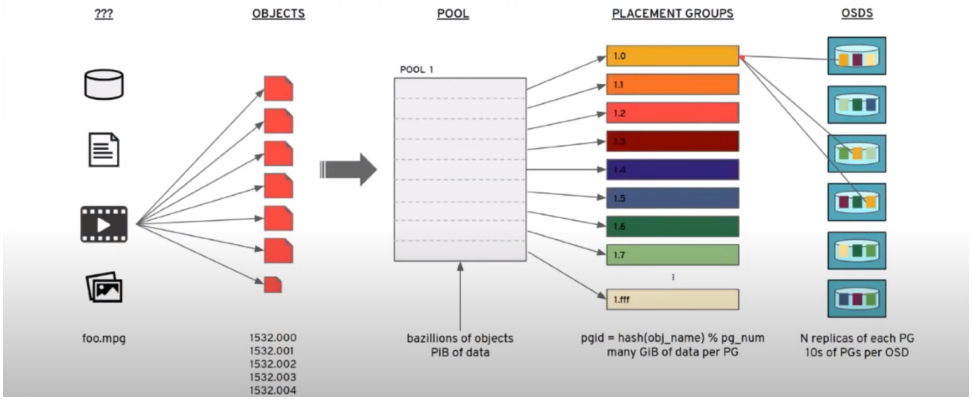

Figura 5: Transformación de un archivo mediante el sistema de Ceph.

En resumen, cuando un archivo (como un vídeo) se almacena en Ceph, se divide en objetos que se distribuyen en los OSDs dentro del clúster. Los objetos se agrupan en pools, que proporcionan un espacio de almacenamiento lógico para objetos del mismo tipo. Los "placement groups" se utilizan para particionar y gestionar los objetos dentro de los pools. Y los OSDs son los componentes físicos que almacenan y administran los objetos en el clúster de Ceph. Esta relación jerárquica entre archivos, objetos, pools, "placement groups" y OSDs permite un almacenamiento distribuido y altamente escalable.

A modo de curiosidad, el algoritmo empleado para distribuir los "PGs" en los OSDs se llama "CRUSH" y consiste en un algoritmo de colocación pseudo-aleatorio similar a "Consistent hashing"

3.4. RBD Block vs CEPHFS

La diferencia principal entre RBD Block y Ceph Filesystem radica en el nivel de abstracción y las características que ofrecen.

RBD (Rados Block Device) es un sistema de almacenamiento de bloques distribuido en Ceph que proporciona almacenamiento persistente basado en bloques a nivel de dispositivo. Los volúmenes RBD Block se implementan como dispositivos de bloques virtuales que se pueden montar en los pods de Kubernetes. Estos son adecuados para casos de uso que requieren acceso a bajo nivel, como bases de datos o aplicaciones que trabajan con almacenamiento a nivel de bloques. Además permite la replicación de datos y la tolerancia a fallos a nivel de bloque. Se puede ver una demo en la sección 7.1

Ceph Filesystem (CephFS) es un sistema de ficheros distribuido en Ceph que proporciona almacenamiento persistente a nivel de sistema de ficheros que se puede compartir entre varios pods de Kubernetes. Los volúmenes Ceph Filesystem se montan como puntos de montaje de sistema de archivos accesibles en los pods de Kubernetes. Estos a su vez son adecuados para casos de uso que requieren un acceso compartido a nivel de sistema de ficheros, como almacenamiento compartido entre varios pods o aplicaciones que necesitan escribir y leer archivos de manera concurrente. Por último, proporcionan replicación de datos y tolerancia a fallos a nivel de sistema de archivos. Se puede ver una demo en la sección 7.2

3.5. Bluestore

mBlueStore es el nuevo backend de almacenamiento para Ceph. Ofrece un mejor rendimiento (aproximadamente el doble para escrituras), suma de comprobación de datos completa y compresión integrada. Es el nuevo backend de almacenamiento predeterminado para los OSD de Ceph en Luminous v12.2.z

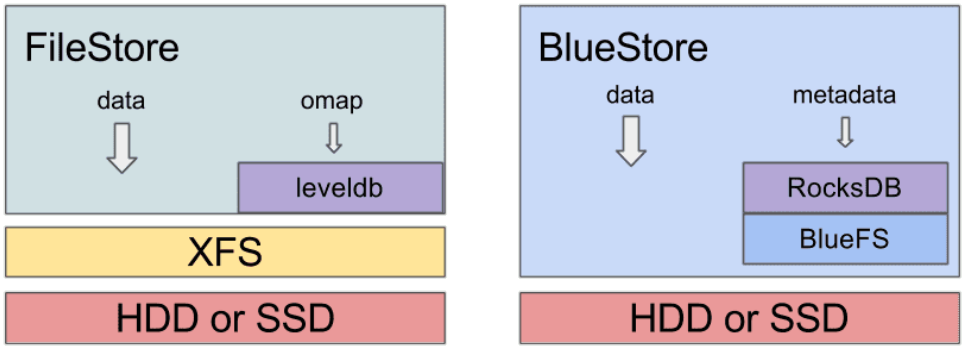

Figura 6: Arquitectura de despliegue de la app y fs con Rook.

La mayor diferencia que se nota con los antiguos OSD basados en FileStore y los de BlueStore es el aspecto de las particiones y el punto de montaje. Se observa que el directorio de datos es ahora una partición diminuta con sólo un puñado de archivos, y que el resto del dispositivo parece una gran partición sin usar, con un enlace simbólico de bloque en el directorio de datos que apunta a ella. Aquí es donde BlueStore está poniendo todos sus datos, y está realizando IO directamente al dispositivo sin procesar (usando la infraestructura asíncrona libaio de Linux) desde el proceso ceph-osd. Para mas información, consultar aquí.

4. Despliegue de la aplicación web y fs

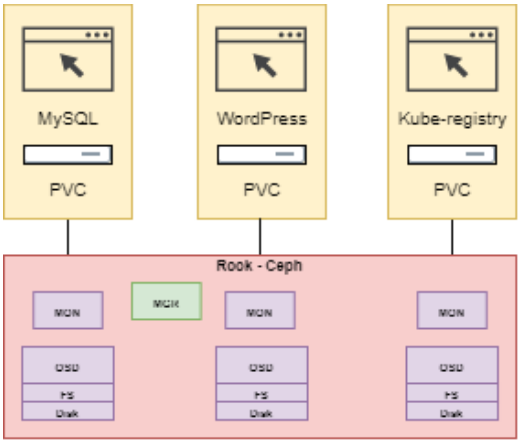

Figura 7: Arquitectura de despliegue de la app y fs con Rook.

En la figura proporcionada se representa la arquitectura del clúster Rook-ceph sobre el que se ha trabajado en este proyecto, en este se puede observar la organización y la interacción de los componentes clave. En esta configuración, el disco se utiliza como medio de almacenamiento y se monta el sistema de archivos sobre él. Sobre el sistema de archivos, se despliegan los Object Storage Daemons (OSDs), que son responsables de almacenar y administrar los datos distribuidos en el clúster.

Además de los OSDs, se pueden identificar otros elementos en el clúster. En particular, se muestra la presencia de monitores (monitors) que supervisan y controlan el estado del clúster. En el caso de la figura, se ilustran tres monitores, que trabajan en conjunto para garantizar la disponibilidad y confiabilidad del clúster. Además de los monitores, se muestra un pod de gestión (mgr pod), que proporciona funcionalidades de administración y gestión del clúster.

Sobre esta infraestructura de almacenamiento distribuido, se despliegan diversos servicios y aplicaciones según las necesidades del entorno. En la figura se representan ejemplos como una base de datos MySQL, un sitio web basado en WordPress y un repositorio privado.

5. Problemas

"HEALT_WARN" al realizar ceph status.

Se solucionó gracias a lo mencionado en la sección 7.1 donde se menciona la sincronización de los workers mediante ntp y dejando mas tiempo tras haber activado la sincronización ntp.

Perdida del acceso a la herramienta kubectl inesperadamente.

Se soluciono liberando espacio del home del usuario del lab. Para ello se utilizo el comando para detectar aquello directorios con mayor tamaño

```
du -sk * | sort -n
```

Además se configuró la variable de entorno"VAGRANT_HOME para que este dentro de /misc y asi las boxes de vagrant se generen dentro de una carpeta especifica. Por último, se siguió el guión del proyecto para terminar de configurar correctamente el path de virtual box.

Fallo al hacer "pull" del repositorio privado

Se solucionó dejando de lado la opción del manifiesto y se opto por el siguiente comando:

```
kubectl run -i -- tty p -- image = localhost :5000/ my - busybox -- sh
```

6. Bibliografía

Documentación técnica

- rook
- ceph
- redhat - ceph
- k3s
- vagrant
- virtualbox
- puppet-ceph

Charlas de CEPH

- Bassam Tabbara, Fundador y Consejero Delegado de Upbound. (24 abril 2018). Making Ceph Awesome on Kubernetes with Rook - Bassam Tabbara
- Sage Weil fundador y arquitecto jefe de Ceph (8 julio 2019). 2019-JUN-27 :: Ceph Tech Talk - Intro to Ceph
- Lars Marowsky-Brée Ingeniero distinguido de SUSE (24 mayo 2019). Failing Better - When Not To Ceph and Lessons Learned - Lars Marowsky-Brée, SUSE

7. Anexos

7.1. Despliegue sobre RBD de una aplicación web

En esta sección se detallan los pasos necesarios para llevar a cabo el despliegue en los laboratorios de la universidad, abarcando desde la etapa 1 hasta la etapa 2. La etapa 3 y última, se encuentra en la sección 7.2.

Etapa inicial : puesta en marcha básica de Ceph en Kubernetes

En esta primera etapa de puesta en marcha del sistema Rook-Ceph, los comandos ejecutados son los siguientes:

```
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f common.yaml
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f operator.yaml

lab102-192:/ kubectl -n rook-ceph get pods
NAME                                  READY   STATUS    RESTARTS   AGE
rook-ceph-operator-6d5456dddd-f87t8   1/1     Running   0          5m39s
rook-discover-5vmmn                   1/1     Running   0          3m12s
rook-discover-jf88z                   1/1     Running   0          3m12s
rook-discover-wvz22                   1/1     Running   0          3m12s
```

Tras la creación de los manifiestos “common.yaml” y “operator.yaml” se ha desplegado la primera parte de la infraestructura básica sobre la que desplegar nuestro sistema de almacenamiento distribuido Ceph.

Al desplegar el operator, se inicia el componente encargado de administrar y supervisar el despliegue de los clústeres de Ceph (rook-ceph-operator-...). Además es el encargado de poner en marcha 3 pods responsables de descubrir y detectar los dispositivos de almacenamiento disponibles en los nodos de la infraestructura.

A continuación se crea el manifiesto “cluster.yaml” que mone en funcionamiento el resto de componentes como rook-ceph-mon, rook-ceph-mgr y rook-ceph-osd, los cuales corresponden a los nombres mencionados en la sección de arquitectura (3) y que completan nuestra infraestructura para desplegar RBD en la sección 7.1 y CEPHFS en la sección 7.2.

```
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f cluster.yaml

lab102-192:/ kubectl -n rook-ceph get pods
NAME                                            READY   STATUS      RESTARTS   AGE
rook-ceph-operator-6d5456dddd-f87t8             1/1     Running     0          20m
rook-discover-5vmmn                             1/1     Running     0          17m
rook-discover-jf88z                             1/1     Running     0          17m
rook-discover-wvz22                             1/1     Running     0          17m
csi-rbdplugin-cxgh8                             3/3     Running     0          13m
csi-cephfsplugin-j5zch                          3/3     Running     0          13m
csi-cephfsplugin-g5vqn                          3/3     Running     0          13m
csi-rbdplugin-p2dwm                             3/3     Running     0          13m
csi-cephfsplugin-provisioner-64d4468bbf-ns5gz   5/5     Running     0          13m
csi-cephfsplugin-provisioner-64d4468bbf-hx7l8   5/5     Running     0          13m
csi-rbdplugin-provisioner-76789bdcc7-j84kx      6/6     Running     0          13m
csi-rbdplugin-provisioner-76789bdcc7-m5gqm      6/6     Running     0          13m
csi-rbdplugin-g8dwz                             3/3     Running     0          13m
csi-cephfsplugin-kdnmm                          3/3     Running     0          13m
rook-ceph-mon-a-84c84946ff-f2j8c                1/1     Running     0          9m46s
rook-ceph-mon-b-d8d98dd6b-5vnnv                 1/1     Running     0          9m41s
rook-ceph-mon-d-85bc97c78d-czxlp                1/1     Running     0          6m5s
rook-ceph-mgr-a-5b5b7fd568-hzm7h                1/1     Running     0          3m11s
rook-ceph-osd-prepare-w2-z99rk                  0/1     Completed   0          2m52s
rook-ceph-osd-prepare-w1-nwj77                  0/1     Completed   0          2m52s
rook-ceph-crashcollector-w2-564f7c67b5-b74zg    1/1     Running     0          2m17s
rook-ceph-osd-prepare-w3-fn4sj                  0/1     Completed   0          2m52s
rook-ceph-osd-0-5cfcfcdd7b-7pwwp                1/1     Running     0          2m17s
rook-ceph-osd-1-54ddf94b7c-z7wtn                1/1     Running     0          2m13s
rook-ceph-osd-2-74b648ff6c-khx6w                1/1     Running     0          2m11s
rook-ceph-crashcollector-w3-8659c6997b-cr4xh    1/1     Running     0          6m5s
rook-ceph-crashcollector-w1-6965bf654-sqdjx     1/1     Running     0          9m41s
```

Para finalizar esta etapa inicial, se implementa un pod de herramientas Ceph utilizando el manifiesto “toolbox”. Una vez creado, se puede observar que se ha generado el pod “rook-ceph-tools-58df894b89-v8dpn”. Dentro de este pod podremos verificar el correcto funcionamiento del sistema Ceph mediante herramientas como “ceph status”, “ceph df” y “rados df” entre otras.

```
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f toolbox.yaml‘

lab102-192:/  kubectl -n rook-ceph get pod -l "app=rook-ceph-tools"
NAME                               READY   STATUS    RESTARTS   AGE
rook-ceph-tools-58df894b89-v8dpn   1/1     Running   0          14s

lab102-192:/ kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-
    tools" -o jsonpath='{.items[0].metadata.name}') -- bash
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_COLLATE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_MESSAGES: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_NUMERIC: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_TIME: cannot change locale (en_US.UTF-8): No such file or
    directory
[root@rook-ceph-tools-58df894b89-v8dpn /]# ceph status
  cluster:
    id:     04bfe2d0-f221-4c28-a88f-89d22c6763c3
    health: HEALTH_WARN
            clock skew detected on mon.b, mon.d

  services:
    mon: 3 daemons, quorum a,b,d (age 11m)
    mgr: a(active, since 10m)
    osd: 3 osds: 3 up (since 9m), 3 in (since 9m)

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

Tras realizar un primer análisis mediante la utilidad "ceph status" se ha podido observar el estado del cluster de Ceph. A continuación, se muestra una descripción de los diferentes componentes y métricas que se presentan en la salida:

- **Cluster**
  - ID: Identificador único del clúster de Ceph.
  - Health: Estado de salud del clúster. En la salida se puede observar como hay una discrepancia de tiempo entre los monitores b y d.
- **Services**
  - Mon: Muestra la cantidad de demonios monitor en el clúster. En este caso, hay 3 demonios en total, y los monitores a, b y d forman el quórum.
  - Mgr: Indica el estado del gestor del clúster.
  - Osd: Muestra la cantidad de demonios de almacenamiento (OSD) en el clúster. En este caso, hay 3 OSD en total.
- **Data**
  - Pools: Indica la cantidad de pools en el clúster y el número de "Placement Groups" (PGs) asociados. En este caso, no hay pools ni PGs creados en el clúster ya que este está vacío.
  - Objects: Muestra la cantidad de objetos almacenados en el clúster y el tamaño total en bytes.
  - Usage: Proporciona información sobre el uso de almacenamiento en el clúster. Se muestra la cantidad de almacenamiento utilizado, la capacidad disponible y la capacidad total.
  - PGs: No se proporciona información en este caso dado que el cluster esta vacío.

A continuación, se reinicia el demonio systemd-timesyncd en las máquinas virtuales para facilitar la sincronización de los relojes. Esto se realiza para que el estado del demonio pase de "loaded" a "active" activando asi la sincronización con ntp. Dicha acción se ha llevado a cabo mediante el "one-liner" mostrado en el siguiente bloque de código.

```
lab102-192:/vagrantk3s/ vagrant ssh m -c "timedatectl | grep -q 'System clock synchronized:
    no' && sudo systemctl restart systemd-timesyncd"
lab102-192:/vagrantk3s/ vagrant ssh w1 -c "timedatectl | grep -q 'System clock synchronized:
     no' && sudo systemctl restart systemd-timesyncd"
lab102-192:/vagrantk3s/ vagrant ssh w2 -c "timedatectl | grep -q 'System clock synchronized:
     no' && sudo systemctl restart systemd-timesyncd"
lab102-192:/vagrantk3s/ vagrant ssh w3 -c "timedatectl | grep -q 'System clock synchronized:
     no' && sudo systemctl restart systemd-timesyncd"
```

Tras ejecutar los comandos en la parte superior y esperar un pequeño periodo de tiempo, se ha vuelto a ejecutar el comando "ceph status" dentro del pod de herramientas "toolbox" y como se puede observar el estado del cluster a pasado de "HEALTH_WARN" a "HEALTH_OK".

En cuando a los otros comandos ejecutados como pueden ser "ceph df" o "rados df" ambos proporcionan información sobre el uso de almacenamiento en un clúster Ceph.

- **ceph df**: Proporciona información detallada y específica sobre el uso del almacenamiento en el clúster Ceph. Este comando muestra detalles sobre los pools de almacenamiento, el número de objetos, el tamaño utilizado y disponible, entre otros datos.
- **rados df**: Ofrece una visión general del estado del almacenamiento en el clúster Ceph al mostrar información resumida. Incluye el espacio total utilizado y disponible en el clúster, el número total de objetos y el número total de PGs (grupos de colocación).

En definitiva, en el estado actual del clúster, el uso del comando "ceph status" es más que suficiente para obtener una idea general de su estado.

```
[root@rook-ceph-tools-58df894b89-v8dpn /]# ceph status
  cluster:
    id:     04bfe2d0-f221-4c28-a88f-89d22c6763c3
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum a,b,d (age 19s)
    mgr: a(active, since 25m)
    osd: 3 osds: 3 up (since 24m), 3 in (since 24m)

  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-58df894b89-v8dpn /]# ceph df
RAW STORAGE:
    CLASS     SIZE        AVAIL       USED         RAW USED      %RAW USED
    hdd       90 GiB      87 GiB      4.7 MiB       3.0 GiB           3.34
    TOTAL     90 GiB      87 GiB      4.7 MiB       3.0 GiB           3.34

POOLS:
    POOL     ID     STORED     OBJECTS     USED     %USED     MAX AVAIL

[root@rook-ceph-tools-58df894b89-v8dpn /]# rados df
POOL_NAME USED OBJECTS CLONES COPIES MISSING_ON_PRIMARY UNFOUND DEGRADED RD_OPS RD WR_OPS WR
     USED COMPR UNDER COMPR

total_objects    0
total_used       3.0 GiB
total_avail      87 GiB
total_space      90 GiB
```

Etapa 2 : Aplicación web y almacenamiento distribuidos en dispositivos de bloques

En esta segunda etapa, utilizaremos el almacenamiento de bloques para montar un almacenamiento de datos en un único Pod. En este contexto, se ha utilizado para crear una aplicación web con WordPress y MySQL que aprovecha los "Persistent Volumes" a partir del despliegue inicial de Rook-Ceph.

Para ello, en primer lugar se definen dos recursos: un CephBlockPool y una StorageClass. Estos recursos se utilizan para configurar y gestionar el almacenamiento distribuido de bloques utilizando la solución Rook-Ceph.

El CephBlockPool especifica un pool de bloques bajo el nombre de "replicapool" en el espacio de nombres "rook-ceph". Este se encuentra configurado con un failureDomain en "host" que se utiliza para determinar cómo se distribuyen y replican los datos en un clúster Ceph indicando que los datos almacenados en un host estarán disponibles en los otros debido a la replicación seleccionada, y se replica en 3 ubicaciones para mayor redundancia y durabilidad.

Por otro lado, La StorageClass define los parámetros de configuración para la creación de volúmenes RBD (block devices) en el clúster Ceph. Los parámetros especificados incluyen el ID del clúster "rook-ceph", el pool en el que se crearán las imágenes RBD ("replicapool"), el formato de imagen RBD ("2"), las características de la imagen (en este caso, "layering"), y las credenciales de administrador de Ceph necesarias para la provisión de volúmenes. Además, se establece la política de reclamación de recursos en "Delete", lo que significa que cuando un PersistentVolumeClaim (PVC) se elimine, el volumen RBD asociado también se eliminará.

La característica "Layering" permite la creación de "snapshots" y clonar imágenes RBD de manera eficiente, para ver más información de ella se puede consultar la documentación oficial aquí.

```
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f storageclassRbdBlock.yaml
```

Una vez que el entorno está preparado, se procede a crear los manifiestos “mysql.yaml” y “wordpress.yaml” para desplegar nuestra aplicación.

```
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f mysql.yaml
service/wordpress-mysql created
persistentvolumeclaim/mysql-pv-claim created
deployment.apps/wordpress-mysql created
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f wordpress.yaml
service/wordpress created
persistentvolumeclaim/wp-pv-claim created
deployment.apps/wordpress created

lab102-192:/ kubectl get pods
NAME                               READY   STATUS    RESTARTS   AGE
svclb-wordpress-bg542              1/1     Running   0          7m48s
svclb-wordpress-mthcr              1/1     Running   0          7m48s
svclb-wordpress-k8ssr              1/1     Running   0          7m48s
wordpress-mysql-6965fc8cc8-xq6xv   1/1     Running   0          8m16s
wordpress-7b989dbf57-fvjsc         1/1     Running   0          7m48s
```

```
lab102-192:/ kubectl get svc -o wide
NAME              TYPE           CLUSTER-IP     EXTERNAL-IP                                  PORT
    (S)         AGE   SELECTOR
kubernetes        ClusterIP      10.43.0.1      <none>                                       443/
    TCP         68m   <none>
wordpress-mysql   ClusterIP      None           <none>
    3306/TCP     11m   app=wordpress,tier=mysql
wordpress         LoadBalancer   10.43.28.62    192.168.1.71,192.168.1.72,192.168.1.73
    80:30592/TCP   10m   app=wordpress,tier=frontend
```

Una vez se ha verificado que todo está en estado “Running”, se procede a comprobar la disponibilidad del sitio web y la persistencia de los datos en nuestro sistema de almacenamiento por bloques.

Como se puede ver a continuación, al realizar una peticion GET mediante la herramienta CURL a la dirección donde se encuentra alojada una de las replicas de la aplicación web se obtienen las cabeceras de la petición donde podemos encontrar que la aplicación esta siendo ejecutada sobre un servidor Apache/2.4.10 y la url a la que se debe redirigir el cliente es “http://192.168.1.71/wp-admin/install.php”.

```
lab102-192:/ curl -i 192.168.1.71
HTTP/1.1 302 Found
Date: Thu, 08 Jun 2023 08:57:25 GMT
Server: Apache/2.4.10 (Debian)
X-Powered-By: PHP/5.6.28
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Location: http://192.168.1.71/wp-admin/install.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

Posteriormente, se procede a la comprobación de la utilización del nuevo pool de dispositivos de bloques de Ceph (RBD), mediante un pod de creación y uso manual de dispositivos de bloques a partir de ese pool.

```
lab102-192:/aplicacionesCephRook/ceph/ kubectl create -f direct-mount.yaml
lab102-192:/ kubectl -n rook-ceph get pod -l app=rook-direct-mount
NAME                                  READY   STATUS    RESTARTS   AGE
rook-direct-mount-68ccfbd4f5-vx8rv    1/1     Running   0          19s
lab102-192:/ kubectl -n rook-ceph exec -it $( kubectl -n rook-ceph get pod -l app=rook-
    direct-mount --no-headers -o custom-columns=":metadata.name" ) -- bash
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_COLLATE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_MESSAGES: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_NUMERIC: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_TIME: cannot change locale (en_US.UTF-8): No such file or
    directory
[root@w1 /]#
```

Listing 1: Comprobación del estado inicial de los dispositivos de bloque en el sistema.

```
[root@w1 /]# lsblk | grep rbd
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
rbd0   252:0    0   2G  0 disk
rbd1   252:16   0   1G  0 disk
```

Listing 2: Comprobación del estado inicial de los pools de Ceph existentes en el clúster.

```
[root@w1 /]# rados lspools
replicapool
```

Listing 3: Comprobación de la existencia de un pool “replicapool” y la existencia de un dispositivo RBD dentro de este.

```
[root@w1 /]# rbd ls -p replicapool
csi-vol-e0a5e57d-05d8-11ee-94c6-ba275707341d
csi-vol-f49b13a8-05d8-11ee-94c6-ba275707341d
```

Crea un bloque de almacenamiento virtual que se puede utilizar como un disco duro.

Listing 4: Creación de un dispositivo RBD (“test”) en el pool “replicapool” con un tamaño de 10MiB.

```
[root@w1 /]# rbd create replicapool/test --size 10
```

Listing 5: Muestra la existencia del dispositivo RBD “test” en el pool “replicapool”.

```
[root@w1 /]# rbd ls -p replicapool
csi-vol-e0a5e57d-05d8-11ee-94c6-ba275707341d
csi-vol-f49b13a8-05d8-11ee-94c6-ba275707341d
test
```

Listing 6: Información detallada sobre el dispositivo RBD “test”.

```
[root@w1 /]# rbd info replicapool/test
rbd image 'test':
        size 10 MiB in 3 objects
        order 22 (4 MiB objects)
        snapshot_count: 0
        id: 675ecb47576
        block_name_prefix: rbd_data.675ecb47576
        format: 2
        features: layering
        op_features:
        flags:
        create_timestamp: Thu Jun  8 09:13:30 2023
        access_timestamp: Thu Jun  8 09:13:30 2023
        modify_timestamp: Thu Jun  8 09:13:30 2023
```

Intenta deshabilitar varias características específicas del dispositivo RBD “test” en el pool “replicapool”. Las características a deshabilitar incluyen “fast-diff” (diferencial rápido), “deep-flatten” (aplanamiento profundo) y “object-map” (mapa de objetos).

```
[root@w1 /]# rbd feature disable replicapool/test fast-diff deep-flatten object-map
rbd: failed to update image features: (22) Invalid argument
2023-06-08 09:25:11.554 7f4ee797eb00 -1 librbd::Operations: one or more requested features
    are already disabled
```

Listing 7: Muestra los dispositivos RBD (Block Device) mapeados en el sistema.

```
[root@w1 /]# rbd showmapped
id pool        namespace image                                          snap device
0  replicapool           csi-vol-e0a5e57d-05d8-11ee-94c6-ba275707341d -    /dev/rbd0
1  replicapool           csi-vol-f49b13a8-05d8-11ee-94c6-ba275707341d -    /dev/rbd1
```

Un dispositivo RBD mapeado es aquel que ha sido asignado a un nodo y puede ser utilizado como un dispositivo de bloques regular en ese nodo.

Listing 8: Mapeo del dispositivo RBD “test” en el pool “replicapool” a un dispositivo de bloque en el sistema operativo para poder utilizar el dispositivo RBD como un disco duro normal.

```
[root@w1 /]# rbd map replicapool/test
/dev/rbd2
```

Listing 9: Muestra el dispositivos RBD “test” mapeado en el sistema.

```
[root@w1 /]# rbd showmapped
id pool        namespace image                                          snap device
0  replicapool           csi-vol-e0a5e57d-05d8-11ee-94c6-ba275707341d -    /dev/rbd0
1  replicapool           csi-vol-f49b13a8-05d8-11ee-94c6-ba275707341d -    /dev/rbd1
2  replicapool           test
```

Listing 10: Muestra una lista de todos los dispositivos de bloque en el sistema

```
[root@w1 /]# lsblk | grep rbd
rbd0   252:0    0   2G  0 disk
rbd1   252:16   0   1G  0 disk
rbd2   252:32   0  10M  0 disk
```

Listing 11: Formatea el dispositivo de bloque RBD /dev/rbd2

```
[root@w1 /]# mkfs.ext4 -m0 /dev/rbd2
mke2fs 1.42.9 (28-Dec-2013)
Discarding device blocks: done
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
Stride=4096 blocks, Stripe width=4096 blocks
2560 inodes, 10240 blocks
0 blocks (0.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=10485760
2 block groups
8192 blocks per group, 8192 fragments per group
1280 inodes per group
Superblock backups stored on blocks:
        8193

Allocating group tables: done
Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
```

Una vez ya está preparado el dispositivo RBD (/dev/rbd2) se procede a probar su correcto funcionamiento mediante el montaje de un nuevo sistema de ficheros llamado “mountpoint”.

```
[root@w1 /]# mkdir /tmp/mountpoint
[root@w1 /]# mount /dev/rbd2 /tmp/mountpoint
[root@w1 mountpoint]# df -h | grep mountpoint
/dev/rbd2       8.6M   14K  8.4M   1% /tmp/mountpoint
[root@w1 mountpoint]# echo 'Hello, World!' >> /tmp/mountpoint/testfile
[root@w1 mountpoint]# cat /tmp/mountpoint/testfile
Hello, World!
```

Una vez creado, y llenado con el fichero testfile, se procede a parar el nodo que no tenga un servicio critico como el DNS, las metricas, o el ceph-mgr.

Listing 12: Se fuerza la caida del w3.

```
lab102-192:/ kubectl get pods --all-namespaces -o wide | sort -k8,8 | grep -E "coredns|
    metrics-server|rook-ceph-mgr"
kube-system   coredns-854c77959c-hlj4x                         1/1     Running     0
     161m   10.42.1.4       w1      <none>           <none>
kube-system   metrics-server-86cbb8457f-dkqsm                  1/1     Running     0
     161m   10.42.1.3       w1      <none>           <none>
rook-ceph     rook-ceph-mgr-a-5b5b7fd568-hzm7h                 1/1     Running     0
     131m   10.42.2.8       w2      <none>           <none>

lab102-192:/vagrantk3s/ vagrant halt w3
==> w3: Attempting graceful shutdown of VM...
==> w3: Forcing shutdown of VM...
lab102-192:/vagrantk3s/ vagrant status
Current machine states:

m                         running (virtualbox)
w1                        running (virtualbox)
w2                        running (virtualbox)
w3                        poweroff (virtualbox)

This environment represents multiple VMs. The VMs are all listed
above with their current state. For more information about a specific
VM, run `vagrant status NAME`.
```

Una vez parado el nodo, se procede a comprobar el funcionamiento de la aplicación web y el punto de montaje creado anteriormente y como se puede verificar abajo todo funciona correctamente.

Listing 13: La aplicación web sigue funcionando con el W3 caído.

```
lab102-192:/vagrantk3s/ curl -i 192.168.1.71
HTTP/1.1 302 Found
Date: Thu, 08 Jun 2023 10:40:33 GMT
Server: Apache/2.4.10 (Debian)
X-Powered-By: PHP/5.6.28
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Location: http://192.168.1.71/wp-admin/install.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

Listing 14: El fichero “testfile” sigue siendo accesible con el W3 caído.

```
lab102-192:/vagrantk3s/ kubectl -n rook-ceph exec -it $( kubectl -n rook-ceph get pod -l app
    =rook-direct-mount --no-headers -o custom-columns=":metadata.name" ) -- bash
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_COLLATE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_MESSAGES: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_NUMERIC: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_TIME: cannot change locale (en_US.UTF-8): No such file or
    directory
[root@w1 /]# cat /tmp/mountpoint/testfile
Hello, World!
```

Una vez acabadas las pruebas se ha procedido a desmontar y desmapear el dispositivo RDB del sistema.

```
[root@w1 /]# umount /tmp/mountpoint
[root@w1 /]# rbd unmap /dev/rbd2
[root@w1 /]# rbd showmapped
id pool         namespace image                                         snap device
0  replicapool           csi-vol-e0a5e57d-05d8-11ee-94c6-ba275707341d -    /dev/rbd0
1  replicapool           csi-vol-f49b13a8-05d8-11ee-94c6-ba275707341d -    /dev/rbd1
3  replicapool           test                                          -    /dev/rbd3
[root@w1 /]# df -h | grep mountpoint
```

7.2. Despliegue sobre CEPHFS

Etapa 3 : Registro de repositorio privado de contenedores y almacenamiento de sistema de ficheros distribuido

Dado que se puede configurar un sistema de ficheros distribuido y compartido con permisos de lectura/escritura para ser utilizado por múltiples Pods se va a emplear para habilitar el registro de un repositorio privado de contenedores sin utilizar seguridad TLS.

Para ello, se comenzará dejando el cluster en un estado con 1 master y 3 workers activos y con los relojes otra vez sincronizados.

```
lab102-192:/vagrantk3s/ vagrant up w3
lab102-192:/vagrantk3s/ vagrant ssh w3 -c "timedatectl | grep -q "System clock synchronized:
     no" && sudo systemctl restart systemd-timesyncd"
```

Acto seguido, se creará el sistema de ficheros Ceph, definiendo la configuración para el pool de metadatos, los pools de datos y el servidor de metadatos en Ceph mediante el manifiesto "filesystem.yaml".

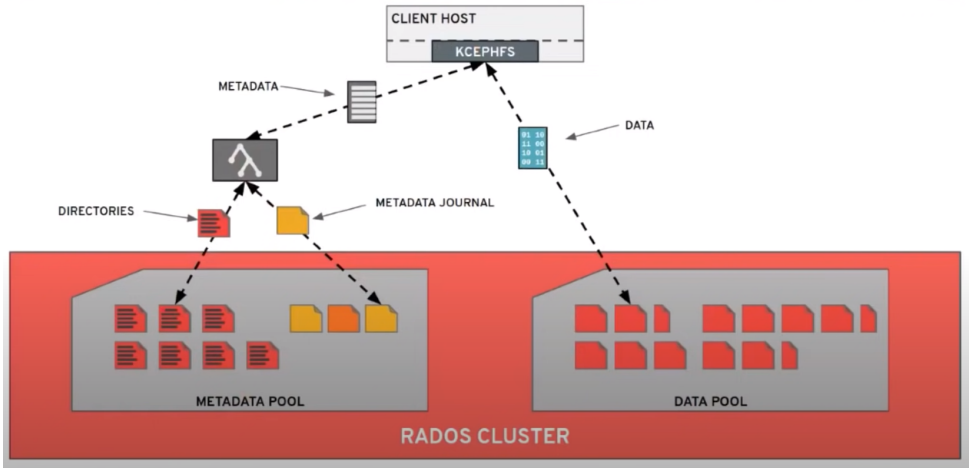

Figura 8: Distribución de metadatos en el cluster RADOS.

```
lab102-192:/ kubectl create -f filesystem.yaml

lab102-192:/ kubectl -n  rook-ceph get pod -l app=rook-ceph-mds
NAME                                    READY   STATUS    RESTARTS   AGE
rook-ceph-mds-myfs-a-64dddcbbcd-hxnj2   1/1     Running   0          27s
rook-ceph-mds-myfs-b-86f88f5db8-c8mtc   1/1     Running   0          27s
```

```
lab102-192:/ kubectl -n rook-ceph exec -it $(kubectl -n rook-ceph get pod -l "app=rook-ceph-
    tools" -o jsonpath='{.items[0].metadata.name}') -- bash
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_COLLATE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_MESSAGES: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_NUMERIC: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_TIME: cannot change locale (en_US.UTF-8): No such file or
    directory
[root@rook-ceph-tools-58df894b89-v8dpn /]# ceph status
  cluster:
    id:     04bfe2d0-f221-4c28-a88f-89d22c6763c3
    health: HEALTH_OK

  services:
    mon: 3 daemons, quorum a,b,d (age 10m)
    mgr: a(active, since 104m)
    mds: myfs:1 {0=myfs-a=up:active} 1 up:standby-replay
    osd: 3 osds: 3 up (since 10m), 3 in (since 10m)

  data:
    pools:   3 pools, 96 pgs
    objects: 102 objects, 214 MiB
    usage:   3.5 GiB used, 86 GiB / 90 GiB avail
    pgs:     96 active+clean

  io:
    client:   1.2 KiB/s rd, 2 op/s rd, 0 op/s wr
```

Como se puede apreciar, los pods estan corriendo y el sistema de ficheros esta funcionando correctamente como se puede observar en la línea que aparece seguido.

```
mds: myfs:1 {0=myfs-a=up:active} 1 up:standby-replay
```

Ahora, para crear los volumenes persistentes mediante el sistema de ficheros Ceph, se necesita crear el StorageClass asociado mediante el manifiesto "storageclassCephFS.yaml" proporcionando unas opciones muy similares a las definidas en el mencionado en las sección 7.1

```
kubectl create -f storageclassCephFS.yaml
```

Una vez definida la storage class se procede con la creación de los componentes del registro.

```
lab102-192:/aplicacionesCephRook/ kubectl create -f kube-registry.yaml
persistentvolumeclaim/cephfs-pvc created
deployment.apps/kube-registry created
lab102-192:/aplicacionesCephRook/ kubectl create -f kubeRegistryService.yaml
service/kube-registry created
lab102-192:/aplicacionesCephRook/ kubectl create -f kubeRegistryProxy.yaml
daemonset.apps/kube-registry-proxy created
```

Una vez puesto en funcionamiento nuestro registro privado, se procede a realizar un redireccionamiento de puertos para poder acceder a este desde la maquina host.

```
lab102-192:/ POD=$(kubectl get pods --namespace kube-system -l k8s-app=kube-registry -o
    template --template '{{range .items}}{{.metadata.name}} {{.status.phase}}{{"\n"}}{{end
    }}' | grep Running | head -1 | cut -f1 -d' ')
lab102-192:/ kubectl port-forward --namespace kube-system $POD 5000:5000 &
[1] 26879
lab102-192:/ Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000

lab102-192:/ curl http://localhost:5000
Handling connection for 5000
```

Después de realizar el mapeo, crearemos una imagen que poder subir a nuestro repositorio. A modo de ejemplo se ha creado una imagen con busybox y se ha subido al repositorio en "localhost:5000/".

```
lab102-192:/ echo "FROM busybox" > Dockerfile

lab102-192:/ docker build -t localhost:5000/my-busybox .
Sending build context to Docker daemon  92.16kB
Step 1/1 : FROM busybox
latest: Pulling from library/busybox
325d69979d33: Pull complete
Digest: sha256:560af6915bfc8d7630e50e212e08242d37b63bd5c1ccf9bd4acccf116e262d5b
Status: Downloaded newer image for busybox:latest
 ---> 8135583d97fe
Successfully built 8135583d97fe
Successfully tagged localhost:5000/my-busybox:latest

lab102-192:/ docker push localhost:5000/my-busybox:latest
The push refers to repository [localhost:5000/my-busybox]
Handling connection for 5000
Handling connection for 5000
9547b4c33213: Preparing
Handling connection for 5000
Handling connection for 5000
9547b4c33213: Pushing [==================================================>]  5.092MB
Handling connection for 5000
9547b4c33213: Pushed
Handling connection for 5000
Handling connection for 5000
Handling connection for 5000
Handling connection for 5000
Handling connection for 5000
Handling connection for 5000
latest: digest: sha256:5cd3db04b8be5773388576a83177aff4f40a03457a63855f4b9cbe30542b9a43 size
    : 528
```

A continuación comprobamos que las imágenes están creadas en nuestro repositorio local y en el respositorio privado mediante los 2 siguiente bloques de terminal.

Listing 15: Imágenes en el repositorio local o del host.

```
lab102-192:/ docker images
REPOSITORY                  TAG       IMAGE ID       CREATED         SIZE
busybox                     latest    8135583d97fe   2 weeks ago     4.86MB
localhost:5000/my-busybox   latest    8135583d97fe   2 weeks ago     4.86MB
```

Listing 16: Imágenes en el repositorio privado o localhost:5000.

```
lab102-192:/ curl http://localhost:5000/v2/_catalog
Handling connection for 5000
{"repositories":["my-busybox"]}
```

Por último se procede con la creación del pod con la imagen subida al repositorio privado mediante uno de los siguientes 2 métodos.

Listing 17: Metodo 1: manifiesto

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
spec:
  containers:
    - name: my-container
      image: localhost:5000/my-busybox
      command: ["sleep", "3600"]
```

Listing 18: Metodo 2: one-liner

```
lab102-192:/ kubectl run -i --tty p --image=localhost:5000/my-busybox -- sh
```

Aquí podemos ver como el pod se encuentra en ejecución y devuelve su hostname (p).

```
lab102-192:/ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
p                                   1/1     Running   0          11m
lab102-192:/ kubectl exec -it p -- sh
/ # hostname
p
```

Para acabar, se procede a realizar las mismas pruebas que en la sección 7.1

Listing 19: Muestra que la imagen esta presente en el montaje realizado sobre /tmp/registry

```
lab102-192:/ kubectl -n rook-ceph exec -it $( kubectl -n rook-ceph get pod -l app=rook-
    direct-mount --no-headers -o custom-columns=":metadata.name" ) -- bash
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_COLLATE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_MESSAGES: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_NUMERIC: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_TIME: cannot change locale (en_US.UTF-8): No such file or
    directory
[root@w1 /]# mon_endpoints=$(grep mon_host /etc/ceph/ceph.conf | cut -f3 -d' ')
[root@w1 /]# my_secret=$(grep key /etc/ceph/keyring | cut -f3 -d' ')
[root@w1 /]# mkdir /tmp/registry
[root@w1 /]# mount -t ceph -o mds_namespace=myfs,name=admin,secret=$my_secret $mon_endpoints
    :/ /tmp/registry
[root@w1 /]# find /tmp/registry -name my-busybox
/tmp/registry/volumes/csi/csi-vol-23e896e8-05ed-11ee-8221-5ede4f76ef32/267a5223-222c-467e-
    bfbc-e6905ba13b93/docker/registry/v2/repositories/my-busybox
```

Listing 20: El contenido sigue presente tras desmontar y montar nuevamente el dispositivo.

```
[root@w1 registry]# echo 'Hello Registry!' > file.txt
[root@w1 registry]# ls
file.txt  volumes
[root@w1 registry]# cat file.txt
Hello Registry!
[root@w1 tmp]# umount /tmp/registry
[root@w1 tmp]# ls
mountpoint  registry
[root@w1 tmp]# ls registry
[root@w1 tmp]# mon_endpoints=$(grep mon_host /etc/ceph/ceph.conf | cut -f3 -d' ')
[root@w1 tmp]# my_secret=$(grep key /etc/ceph/keyring | cut -f3 -d' ')
[root@w1 tmp]# mount -t ceph -o mds_namespace=myfs,name=admin,secret=$my_secret
    $mon_endpoints:/ /tmp/registry
[root@w1 tmp]# ls registry
file.txt  volumes
```

Listing 21: Se fuerza la caida del w3.

```
lab102-192:/ kubectl get pods --all-namespaces -o wide | sort -k8,8 | grep -E "coredns|
    metrics-server|rook-ceph-mgr"
kube-system   coredns-854c77959c-hlj4x                        1/1     Running     0
     6h47m   10.42.1.4       w1     <none>           <none>
kube-system   metrics-server-86cbb8457f-dkqsm                 1/1     Running     0
     6h47m   10.42.1.3       w1     <none>           <none>
rook-ceph     rook-ceph-mgr-a-5b5b7fd568-hzm7h                1/1     Running     0
     6h17m   10.42.2.8       w2     <none>           <none>
lab102-192:/vagrantk3s/ vagrant halt w3
==> w3: Attempting graceful shutdown of VM...
E0608 16:37:16.033833   26879 portforward.go:234] lost connection to pod
==> w3: Forcing shutdown of VM...
[1]+  Done                    kubectl port-forward --namespace kube-system $POD 5000:5000  (
    wd: /misc/alumnos/as2/as22022/a798095/proyecto/aplicacionesCephRook)
(wd now: /misc/alumnos/as2/as22022/a798095/proyecto/vagrantk3s)
```

Listing 22: La aplicación web sigue funcionando con el W3 caído

```
lab102-192:/ curl -i 192.168.1.71
HTTP/1.1 302 Found
Date: Thu, 08 Jun 2023 14:43:54 GMT
Server: Apache/2.4.10 (Debian)
X-Powered-By: PHP/5.6.28
Expires: Wed, 11 Jan 1984 05:00:00 GMT
Cache-Control: no-cache, must-revalidate, max-age=0
Location: http://192.168.1.71/wp-admin/install.php
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

Listing 23: El fichero file.txt sigue siendo accesible con el W3 caído

```
lab102-192:/ kubectl -n rook-ceph exec -it $( kubectl -n rook-ceph get pod -l app=rook-
    direct-mount --no-headers -o custom-columns=":metadata.name" ) -- bash
bash: warning: setlocale: LC_CTYPE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_COLLATE: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_MESSAGES: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_NUMERIC: cannot change locale (en_US.UTF-8): No such file or
    directory
bash: warning: setlocale: LC_TIME: cannot change locale (en_US.UTF-8): No such file or
    directory
[root@w1 registry]# ls /tmp/registry
file.txt  volumes
[root@w1 registry]# cat /tmp/registry/file.txt
Hello Registry!
```

7.3. Despliegue sobre con puppet

Listing 24: provision.pp

```
class provision {
  $hostname = $facts['hostname']
  $nodeip   = $facts['node_ip']
  $masterip = $facts['master_ip']
  $nodetype = $facts['type']

  # Configura la zona horaria
  class { 'timezone':
    timezone => 'Europe/Madrid',
  }

  # Configura el archivo de hosts
  file { '/etc/hosts':
    ensure  => file,
    content => "# This file is managed by Puppet\n
                192.168.1.70 m\n
                192.168.1.71 w1\n
                192.168.1.72 w2\n
                192.168.1.73 w3\n",
  }

  # Copia el archivo "k3s" a /usr/local/bin/
  file { '/usr/local/bin/k3s':
    source => '/vagrant/k3s',
    mode   => '0755',
  }

  # Comprueba el tipo de nodo y realiza la instalaci n correspondiente
  if $nodetype == 'master' {
    # Instalaci n del nodo maestro
    exec { 'install_k3s_master':
      command => "/bin/bash /vagrant/install.sh server \
                  --token 'wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy5OZ/5'
     \
                  --flannel-iface enp0s8 \
                  --bind-address $nodeip \
                  --node-ip $nodeip --node-name $hostname \
                  --disable traefik \
                  --node-taint k3s-controlplane=true:NoExecute",
      require => File['/usr/local/bin/k3s'],
    }

    # Copia el archivo de configuracion k3s.yaml al directorio /vagrant
    file { '/vagrant/k3s.yaml':
      ensure  => present,
      source  => '/etc/rancher/k3s/k3s.yaml',
      require => Exec['install_k3s_master'],
    }
  } else {
    # Instalacion del nodo agente
    exec { 'install_k3s_agent':
      command => "/bin/bash /vagrant/install.sh agent \
                  --server https://$masterip:6443 \
                  --token 'wCdC16AlP8qpqqI53DM6ujtrfZ7qsEM7PHLxD+Sw+RNK2d1oDJQQOsBkIwy5OZ/5'
     \
                  --node-ip $nodeip --node-name $hostname --flannel-iface enp0s8",
      require => File['/usr/local/bin/k3s'],
    }
  }
}

include provision
```

Listing 25: Vagrantfile

```
#boxes
U = 'ubuntu/bionic64'

MASTER = '192.168.1.70'
NODES = [
  { hostname: 'm', type: "master", ip: MASTER, mem: 1000, m: MASTER },
  { hostname: 'w1', type: "worker", ip: '192.168.1.71', mem: 1800, m: MASTER },
  { hostname: 'w2', type: "worker", ip: '192.168.1.72', mem: 1800, m: MASTER },
  { hostname: 'w3', type: "worker", ip: '192.168.1.73', mem: 1800, m: MASTER },
]

Vagrant.configure("2") do |config|
  NODES.each do |node|
    config.vm.define node[:hostname] do |nodeconfig|
      nodeconfig.vm.box = U
      nodeconfig.vm.hostname = node[:hostname]

      # Additional Network
      nodeconfig.vm.network   :public_network,
        # substituir por vuestra interfaz de red
        bridge: "br1",
        ip: node[:ip],
        nic_type: "virtio"

      # Virtual hardware configuration
      nodeconfig.vm.provider "virtualbox" do |v|
        v.memory = node[:mem]
        v.cpus = 1
        v.default_nic_type = "virtio"

        v.customize ["modifyvm", :id, "--name", node[:hostname]]

        # CREATE DISK
        if node[:type] == "worker"
          vdi_file = "disk-#{node[:hostname]}.vdi"
          unless File.exist?(vdi_file)
            v.customize [ "createmedium", "--filename", vdi_file, "--size", 30*1024 ]
            v.customize [ "storageattach", :id, "--storagectl", "SCSI", "--port", 2, "--
    device", 0,"--type", "hdd", "--medium", vdi_file ]
          end
        end

        nodeconfig.vm.boot_timeout = 600
        nodeconfig.vm.provision "puppet" do |puppet|
          puppet.manifests_path = "."
          puppet.manifest_file  = "provision.pp"
          puppet.options        = "--verbose --environment=<nombre_del_entorno> --no-
    daemonize --debug"
          puppet.facter         = { "masterip" => node[:m], "nodeip" => node[:ip], "hostname
    " => node[:hostname], "nodetype" => node[:type] }
        end

        if node[:type] == "master"
          nodeconfig.trigger.after :up do |trigger|
            trigger.run = {inline: "sh -c 'cp k3s.yaml /home/a798095/.kube/config'"}
          end
        end
      end
    end
  end
end
```