ALMACENAMIENTO DISTRIBUIDO PARA KUBERNETES

Javier Vega Correa

2ºASIR

ÍNDICE

1. Motivación del proyecto

Soy Javier Vega Correa alumno de el Ciclo Formativo de Grado Superior Administración de Sistemas Informáticos en Red en el instituto IES Gonzalo Nazareno de Dos Hermanas, Sevilla.

Antes de desvelar cual es la motivación de mi proyecto he de decir que mi asignatura favorita este curso ha sido HLC (Hora de Libre Configuración), en la cuál hemos estudiado los principales conceptos del almacenamiento distribuido y kubernetes.

Con esta premisa llegamos a la motivación principal de hacer este proyecto. La cuál es conocer más a fondo los cluster de kubernetes y entender mejor el almacenamiento distribuido, por suerte encontré la herramienta Longhorn para ahondar mejor en ambos conceptos.

2. Introducción del proyecto

2.1. Creación y presentación del escenario

El escenario se compone de 4 maquinas linux creadas sobre Proxmox. Concretamente las máquinas tienen instalado el sistema operativo Debian GNU/Linux 11 (bullseye) , conectadas por una red nat con conexión al exterior.

Las características hardware de los cuatro nodos son las siguientes:

- 4 cores.
- 4 gigabyte de memoria RAM.
- Un disco de 10 gigabyte.

En ese escenario hemos instalado un cluster de kubernetes k3s, con un nodo master y tres nodos worker.

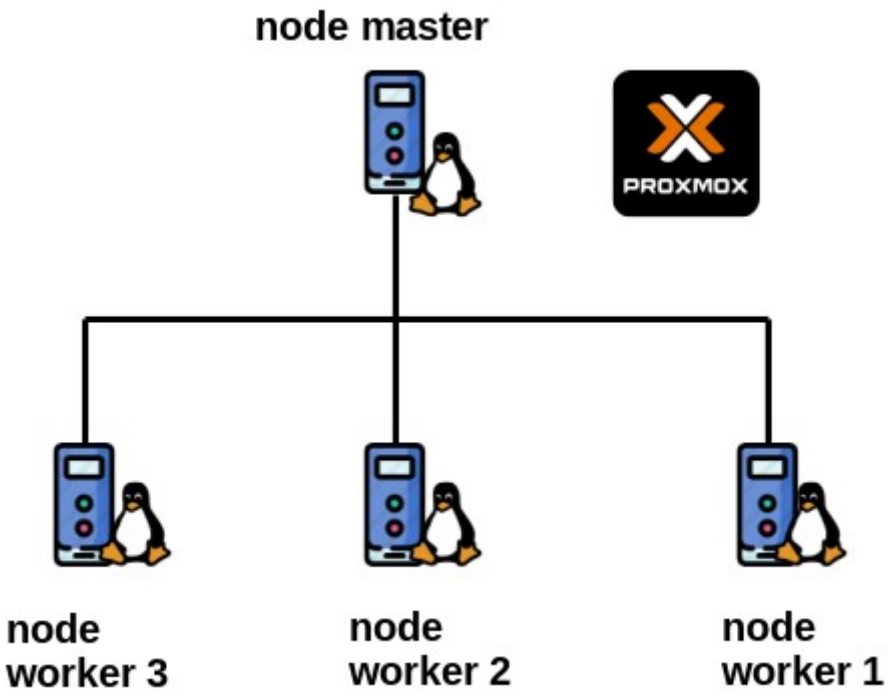


Figura 1: escenario

2.1.1. Creación de máquinas en Promox

Las cuatro máquinas han sido creadas a partir de una imagen de Debian 11, desde la interfaz web de Promox nos dirigimos a las imágenes que tenemos disponibles y seleccionamos la de debian y clicamos en clone.

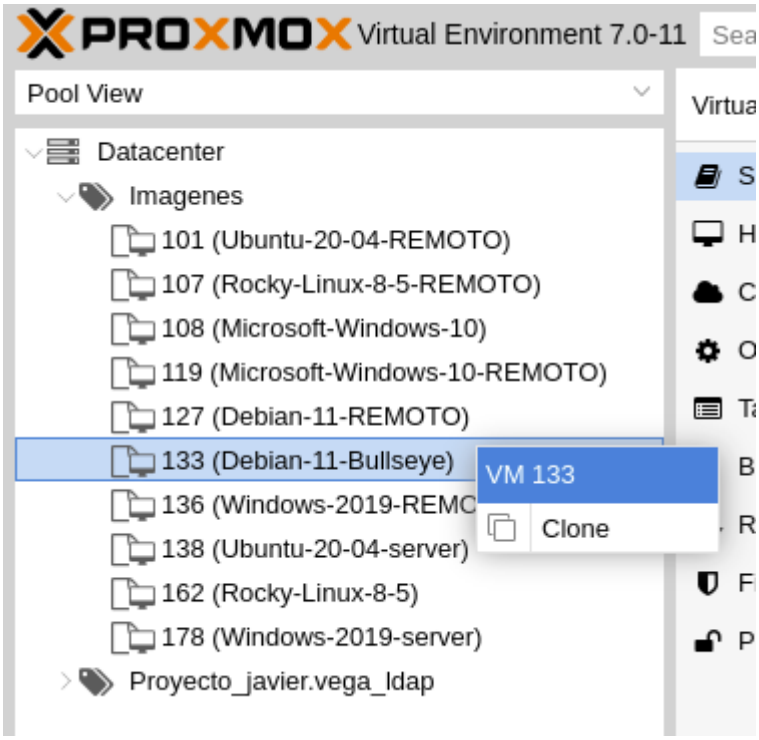


Figura 2: Clone image promox

Se nos abrirá una ventana en la que seleccionamos nombre y el recurso en el cual se va a crear la máquina.

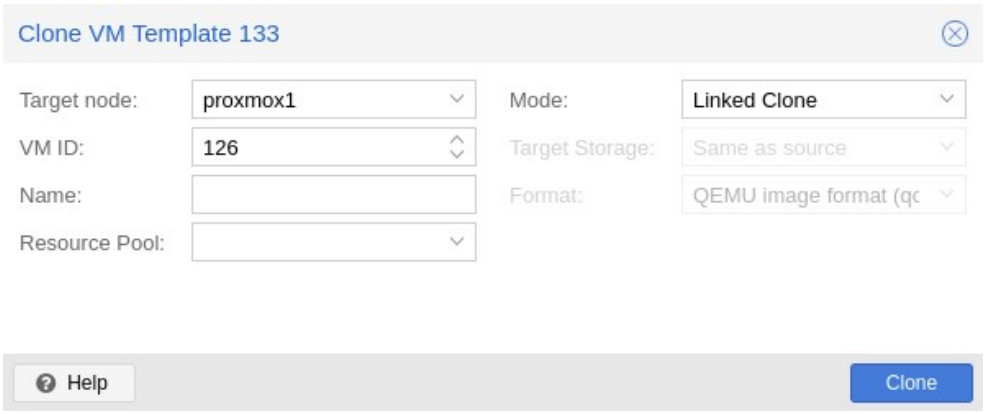


Figura 3: Seleccionar Resorce Pool

Una vez la tenemos creada y asignada en nuestro recurso, vamos a dirigirnos a la configuración de la máquina y modificaremos la cpu y la memoria ram para establecer la que necesitamos, también podemos establecer un nombre de usuario y una contraseña para poder conectarnos, cabe recalcar que es imporante que este paso se haga con la máquina apagada.

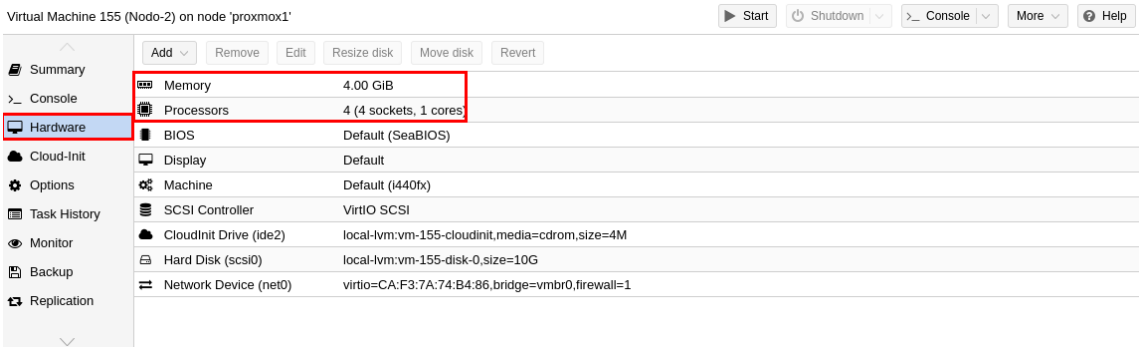


Figura 4: Seleccionar CPU y memoria RAM

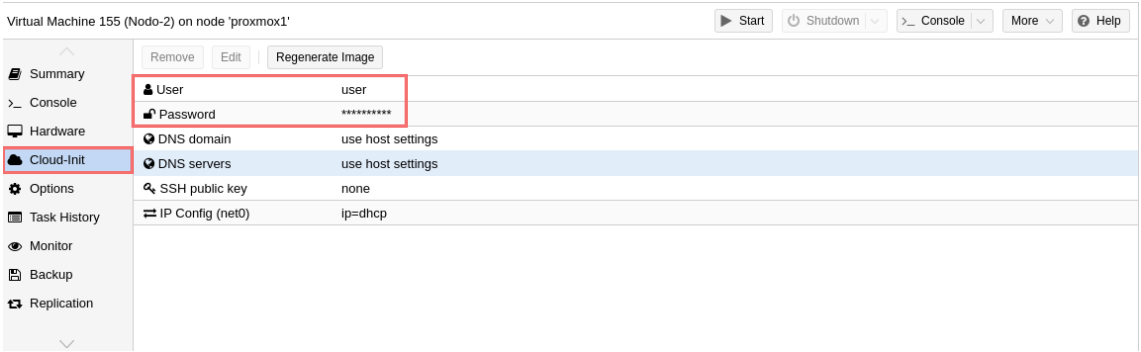


Figura 5: Seleccionar CPU y memoria RAM

Una vez modificados los valores que necesitábamos, iniciamos la máquina y se le asignará la ip automáticamente, cuando esto pase ya podemos acceder por ssh desde nuestra máquina personal.

2.2. ¿ Que es Kubernetes ?

Kubernetes es un software de orquestación de contenedores desarrollado inicialmente por Google, pero que hoy en día es un proyecto libre independiente utilizado en gran cantidad de entornos diferentes y que se ha convertido en muchos casos en la solución preferida para orquestar aplicaciones basadas en contenedores en entornos en producción.

Kubernetes es extensible, cuenta con gran cantidad de objetos, módulos, plugins , y un largo etc. Fue desarrollado en el lenguaje Go como diversas aplicaciones de este sector.

Este software es declarativo, es decir, declaramos los elementos que queremos desplegar en ficheros yaml. Un ejemplo claro sería el siguiente:

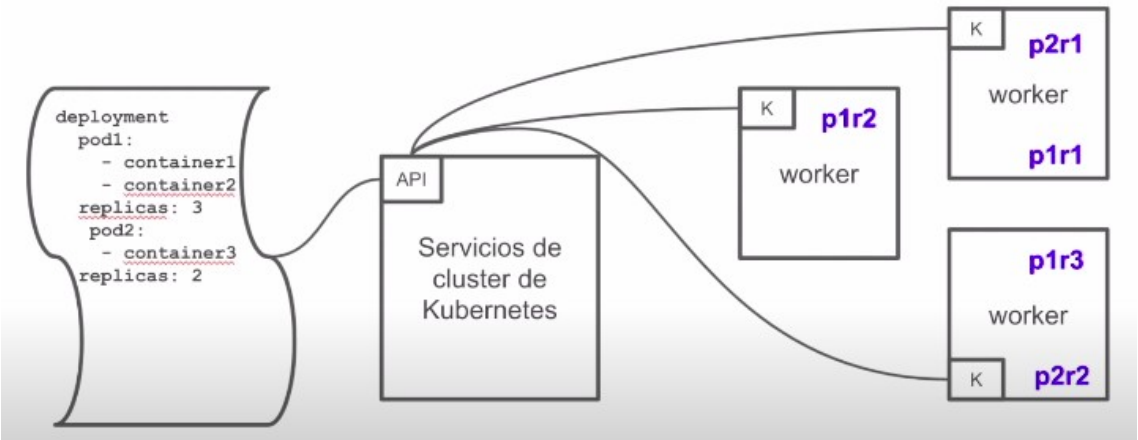
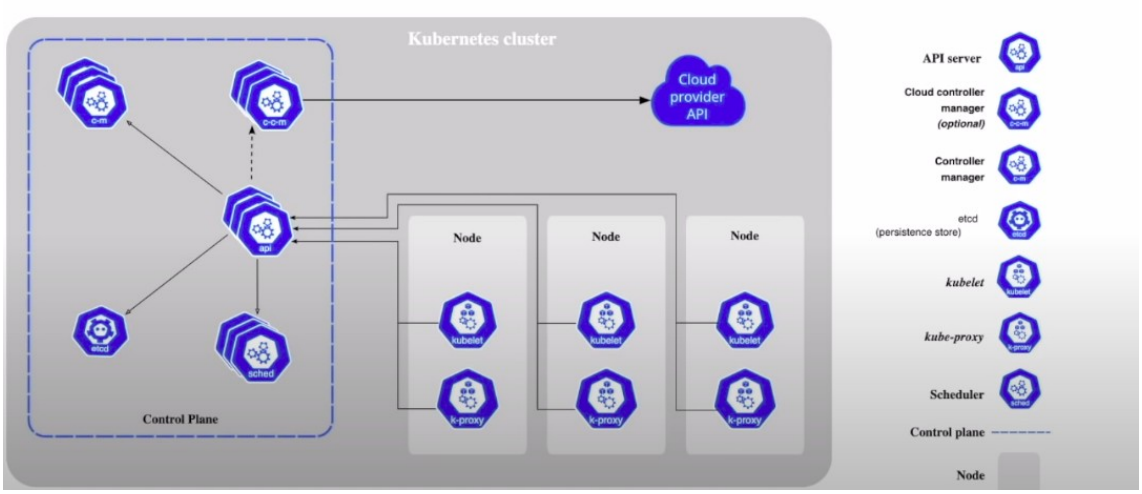


Figura 6: Ejemplo funcionamiento de kubernetes

Definimos un deployment en el que especificamos el número de pods y de réplicas que queremos desplegar en nuestro cluster. Ese fichero pasa por la API y es esta la que los distribuye los pods entre los nodos worker. Un factor importante es que el cluster siempre va a hacer todo lo posible por desplegar exactamente lo que pedimos.

Componentes de Kubernetes



Control-plane

Los componentes que forman el plano de control toman decisiones globales sobre el clúster, detectan y responden a eventos del clúster, como la creación de un nuevo pod cuando la propiedad replicas de un controlador de replicación no se cumple.

- **kube-apiserver**: expone la API de Kubernetes. Recibe las peticiones y actualiza acordemente el estado en etcd.

- **etcd**: Almacén de datos persistente, consistente y distribuido de clave-valor utilizado para almacenar toda a la información del clúster de Kubernetes.

- **kube-scheduler**: Está pendiente de los Pods que no tienen ningún nodo asignado y seleciona uno donde ejecutarlo. Decide en qué nodo se ejecutará el pod, para ellos tiene en cuenta diversos factores: requisitos de recursos, restricciones de hardware/software, localización de datos dependientes, entre otros.

- **kube-controller-manager**: Componente del plano de control que ejecuta los controladores de Kubernetes. Incluyen:

    - Controlador de nodos: es el responsable de detectar y responder cuándo un nodo deja de funcionar.

    - Controlador de replicación: es el responsable de mantener el número correcto de pods para cada controlador de replicación del sistema.

    - Controlador de endpoints: construye el objeto Endpoints, es decir, hace una unión entre los Services y los Pods.

    - Controladores de tokens y cuentas de servicio: crean cuentas y tokens de acceso a la API por defecto para los nuevos Namespaces.

- **cloud-controller-manager**: ejecuta controladores que interactúan con proveedores de la nube. Incluye:

    - Controlador de nodos: es el responsable de detectar y actuar cuándo un nodo deja de responder.

- Controlador de rutas: para configurar rutas en la infraestructura de nube subyacente.

- Controlador de servicios: para crear, actualizar y eliminar balanceadores de carga en la nube.

- Controlador de volúmenes: para crear, conectar y montar volúmenes e interactuar con el proveedor de la nube para orquestarlos

Componentes de nodo

Los componentes de nodo corren en cada nodo, manteniendo a los pods en funcionamiento y proporcionando el entorno de ejecución de Kubernetes.

- **kubelet**: Agente que se ejecuta en cada nodo de un clúster. Se asegura de que los contenedores estén corriendo en un pod.

- **kube-proxy**: permite abstraer un servicio en Kubernetes manteniendo las reglas de red en el anfitrión y haciendo reenvío de conexiones.

- **Runtime de contenedores**: es el software responsable de ejecutar los contenedores.

Addons

Los addons son pods y servicios que implementan funcionalidades del clúster.

Interfaz Web (Dashboard)

El Dashboard es una interfaz Web de propósito general para clusters de Kubernetes. Le permite a los usuarios administrar y resolver problemas que puedan presentar tanto las aplicaciones como el clúster.

Monitor de recursos de contenedores

El Monitor de recursos de contenedores almacena de forma centralizada series de tiempo con métricas sobre los contenedores, y provee una interfaz para navegar estos datos.

Registros del clúster

El mecanismo de registros del clúster está a cargo de almacenar los registros de los contenedores de forma centralizada, proporcionando una interfaz de búsqueda y navegación.

Características principales de Kubernetes

- Service Discovery y load balancing.
- Orquestación del almacenamiento.
- Despliegues y rollbacks automáticos.
- Ejecución Batch.
- Planificación.
- Autorreparación.
- Gestión de la configuración y secrets.

- Escalado y auto-escalado.

2.3. ¿ Que es Helm ?

Como hemos visto Kubernetes tiene mucha complejidad, ofrece muchos objetos y opciones muy diferentes para operar nuestras aplicaciones, para esto precisamente surge Helm.

Helm simplifica el uso de kubernetes ya que es un gestor de paquetes que permite instalar, compartir, actualizar y personalizar cualquier aplicación. Helm tiene el poder de instalar aplicaciones con una sola linea de comandos como veremos posteriormente con Longhorn.

Ventajas de Helm:

- Tiene cientos de paquetes disponibles.
- Permite personalizar aplicaciones
- Contamos con aplicaciones actualizadas.
- Facilita la gestión de la aplicación. Por ejemplo el control de versiones.

2.4. ¿ Que es Longhorn ?

Longhorn es un sistema de almacenamiento de bloques distribuido ligero, fiable y fácil de usar para Kubernetes. Compatible con la arquitectura AMD64 y de manera experimental con ARM64.

Es un software libre y de código abierto. Originalmente desarrollado por Rancher Labs, en la actualidad se desarrolla como un proyecto de incubación de la Cloud Native Computing Foundation.

2.4.1. Características principales de Longhorn

Almacenamiento persistente de alta disponibilidad para Kubernetes

En el pasado, a ITOps y DevOps les resultó difícil agregar almacenamiento replicado a los clusters de Kubernetes. Como resultado, muchos clusters de Kubernetes no alojados en la nube no admiten el almacenamiento persistente. Los arreglos de almacenamiento externo no son portátiles y pueden ser extremadamente costosos.

Longhorn ofrece almacenamiento en bloque persistente simplificado, fácil de implementar y actualizar, 100% de código abierto y nativo de la nube sin los costos generales de las alternativas de núcleo abierto o propietarias.

Copias de seguridad e instantáneas incrementales sencillas

Las funciones de copia de seguridad e instantáneas incrementales integradas de Longhorn mantienen los datos del volumen seguros dentro o fuera del clúster de Kubernetes.

Las copias de seguridad programadas de volúmenes de almacenamiento persistentes en clusters de Kubernetes se simplifican con la interfaz de usuario de administración gratuita e intuitiva de Longhorn.

Recuperación ante desastres entre clusters

Las soluciones de replicación externa se recuperarán de una falla del disco volviendo a replicar todo el almacén de datos. Esto puede llevar días, tiempo durante el cual el clúster funciona mal y tiene un mayor riesgo de fallar.

Con Longhorn, puede controlar la granularidad al máximo, crear fácilmente un volumen de recuperación ante desastres en otro clúster de Kubernetes y realizar una conmutación por error en caso de emergencia.

2.4.2. Posibilidades de Longhorn

Simplificar el almacenamiento en bloque distribuido con microservicios

Longhorn puede simplificar el sistema de almacenamiento partiendo un gran controlador de almacenamiento en bloque en varios controladores de almacenamiento más pequeños, siempre que esos volúmenes puedan seguir construyéndose a partir de un conjunto común de discos. Al utilizar un controlador de almacenamiento por volumen, Longhorn convierte cada volumen en un microservicio. El controlador se llama Longhorn Engine.

El componente Longhorn Manager orquesta los Longhorn Engine, para que trabajen juntos de forma coherente.

Utilizar el almacenamiento persistente en Kubernetes sin depender de un proveedor de la nube

Los pods pueden hacer referencia al almacenamiento directamente, pero esto no se recomienda porque no permite que el pod o el contenedor sean portátiles. En su lugar, los requisitos de almacenamiento de las cargas de trabajo deben definirse en los volúmenes persistentes (PV) de Kubernetes y en las reclamaciones de volúmenes persistentes (PVC). Con Longhorn, puede especificar el tamaño del volumen, los requisitos de IOPS y el número de réplicas sincrónicas que desea en los hosts que suministran el recurso de almacenamiento para el volumen.

Programar múltiples réplicas en múltiples hosts de almacenamiento

Para aumentar la disponibilidad, Longhorn crea réplicas de cada volumen. Las réplicas contienen una cadena de instantáneas del volumen, en la que cada instantánea almacena el cambio de una instantánea anterior. Cada réplica de un volumen también se ejecuta en un contenedor, por lo que un volumen con tres réplicas da lugar a cuatro contenedores.

El número de réplicas de cada volumen es configurable en Longhorn, así como los nodos donde se programarán las réplicas. Longhorn monitoriza la salud de cada réplica y realiza reparaciones, reconstruyendo la réplica cuando es necesario.

Especificar los horarios para las operaciones recurrentes de instantáneas y copias de seguridad

Especificar la frecuencia de estas operaciones (cada hora, cada día, cada semana, cada mes y cada año), la hora exacta a la que se realizan estas operaciones (por ejemplo, a las 3:00 de la mañana todos los domingos) y cuántas instantáneas recurrentes y conjuntos de copias de seguridad se mantienen.

2.4.3. Como funciona Longhorn

Para entender su funcionamiento tenemos que empezar por conocer que el diseño de Longhorn tiene dos capas: el plano de datos y el plano de control. Longhorn Engine es un controlador de almacenamiento que corresponde al plano de datos, y Longhorn Manager corresponde al plano de control.

Longhorn-manager es el responsable de crear y gestionar los volúmenes en el clúster de Kubernetes, y maneja las llamadas a la API desde la UI o los plugins de volumen. Los pods de Longhorn Manager se ejecutan en cada nodo del cluster Longhorn.

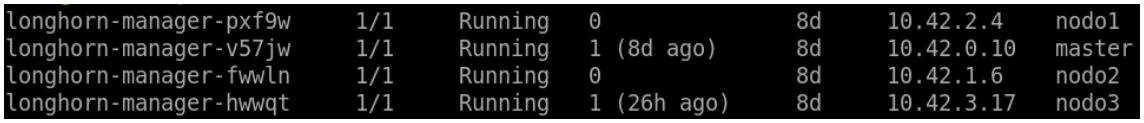


Figura 7: pods longhorn-manager

Cuando solicitamos al Longhorn-manager que cree un volumen, este crea una instancia de Longhorn Engine en el nodo al que se enlaza el volumen y crea una réplica en cada nodo. Las réplicas deben ser colocadas nodos diferentes para asegurar la alta disponibilidad.

Incluso si se produce un problema con una determinada réplica o con el motor, el problema no afectará a todas las réplicas o al acceso que tiene el Pod con el volumen.

Ejemplo de funcionamiento de Longhorn-engine

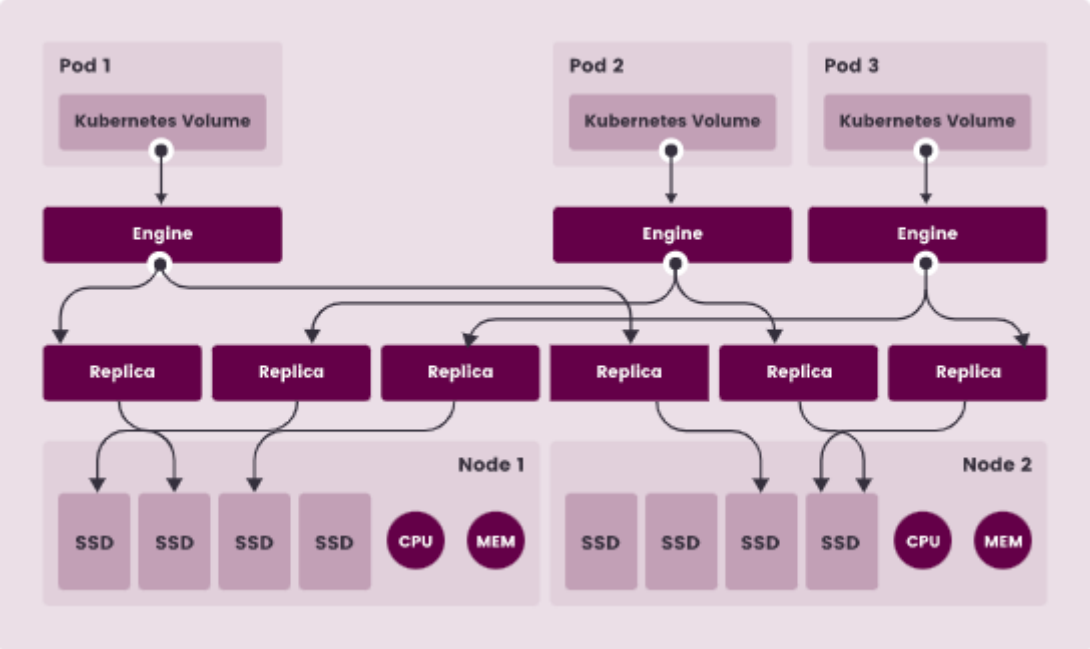


Figura 8: Funcionamiento de Longhorn

Como observamos en la imagen hay 3 instancias con sus volúmenes , cada una tiene un controlador Longhorn-engine y también tienen 2 réplicas de sus volúmenes una por cada nodo.

Al crear un controlador Longhorn-engine para cada volumen, si uno falla, la función de los otros dos volúmenes no se ve afectada.

Podemos ver los elementos del ejemplo comentado sobre nuestro escenario.

Vemos que existe un volumen de 1G.

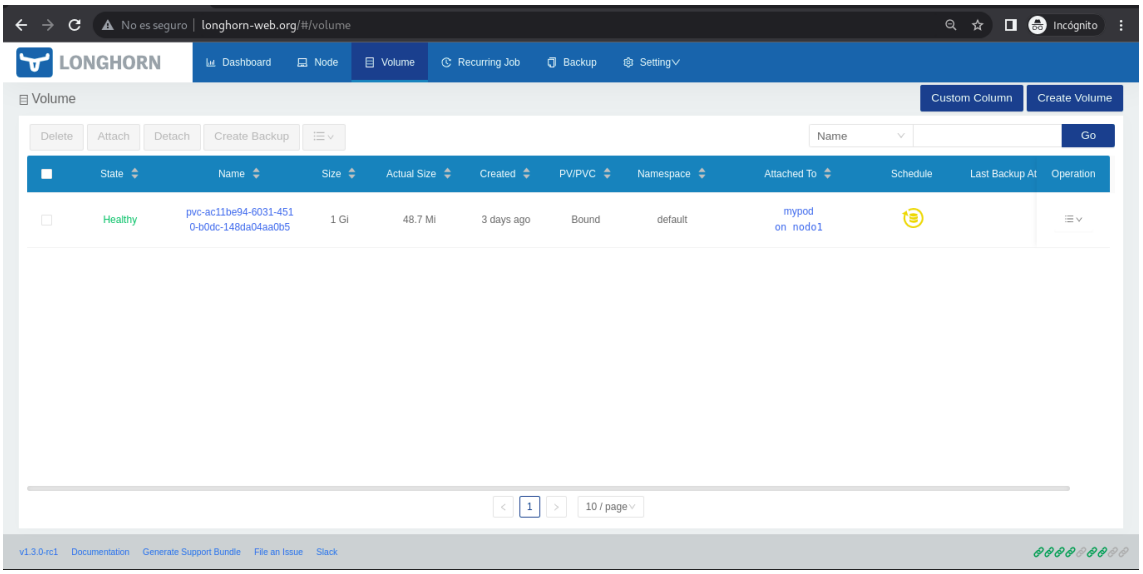


Figura 9: Dashboard > Volumes Longhorn

Si observamos los detalles vemos que esta creado en el nodo1 y tiene dos replicas una en nodo2 y otra en el nodo master.

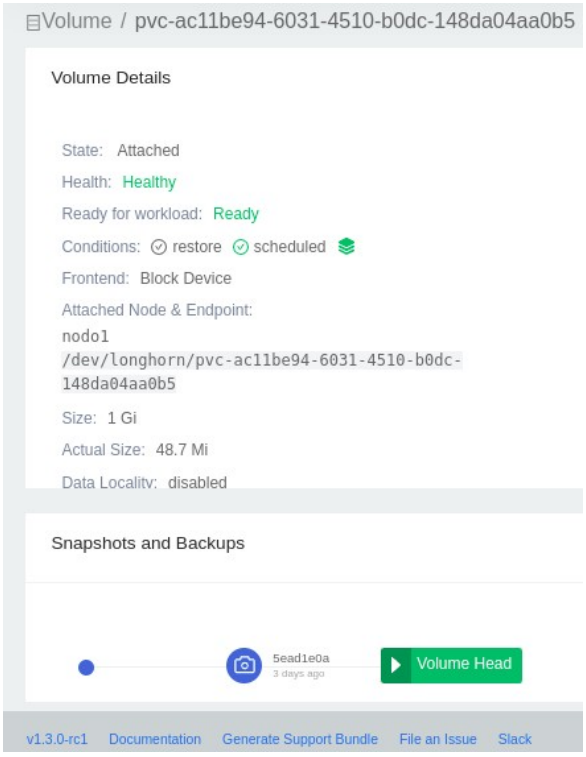


Figura 10: Detalles Volumen Ejemplo

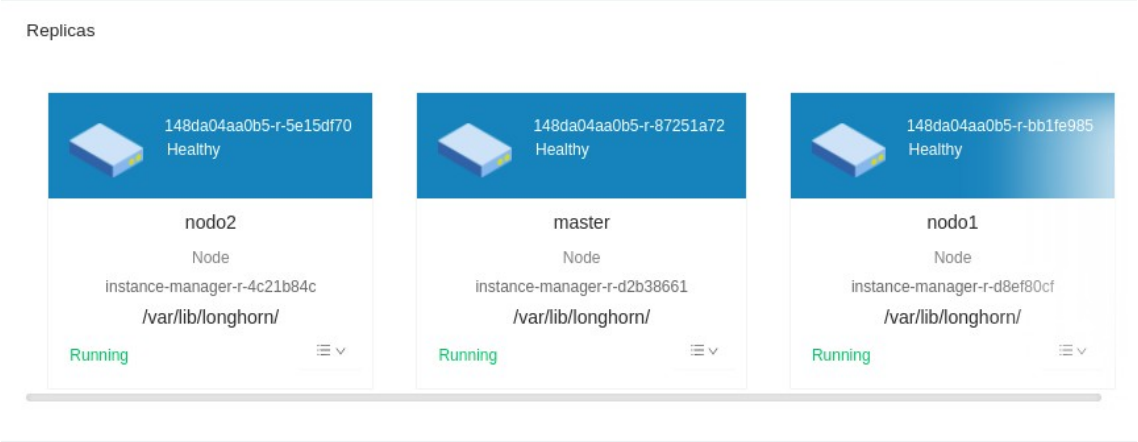


Figura 11: Replicas volumen-ejemplo

Y tiene un controlador longhorn-engine en el nodo1

```
user@master:~$ kubectl -n longhorn-system get engine
NAME           STATE     NODE    INSTANCEMANAGER
pvc-ac11be94   running   nodo1   instance-manager
```

3. Instalación y uso básico de Longhorn

3.1. Maneras de instalar Longhorn

Longhorn viene con una interfaz de usuario independiente y puede instalarse mediante Helm, kubectl o el catálogo de aplicaciones de Rancher.

3.2. Pre-requisitos

- Mínimo 3 nodos de 4 cpus y 4 gb de ram por nodo.
- Un cluster de kubernetes operativo.
- El paquete open-iscsi instalado en todos los nodos del cluster.
- El paquete nfs en todos los nodos del cluster.
- Tener instalado helm

3.2.1. Instalación de cluster k3s

K3S es una distribución de Kubernetes que está pensada para poner en producción, no es un proyecto oficial de Kubernetes, sino que lo comenzó a desarrollar la empresa Rancher y hoy en día lo mantiene la Cloud Native Computing Foundation.

Paquetes previos

El único requisito, al margen de tener recursos hardware suficientes, para instalar el cluster de k3s es instalar la herramienta kubectl . En Debian podemos instalarla desde apt de la siguiente manera:

```
apt install kubernetes-client -y
```

Instalación del nodo Master

En la maquina que vayamos a utilizar como nodo master ejecutamos:

```
curl -sfL https://get.k3s.io | sh -
```

Nos aseguramos que se ha instalado correctamente observando el estado del servicio k3s.service

```
root@master:~$ systemctl status k3s.service
● k3s.service - Lightweight Kubernetes
   Loaded:loaded (/etc/systemd/system/k3s.service;enabled;vendor
preset:enabled)
   Active:active(running)since Sun2022-05-2922:00:58CEST;27min
ago
     Docs:https://k3s.io
      Process:9620ExecStartPre=/bin/sh-xc!/usr/bin/systemctl is-
enabled--quiet
nm-cloud-setup.service(code=exited,status=0/SUCCESS)
   Process:9622ExecStartPre=/sbin/modprobe
br_netfilter(code=exited,status=0/SUCCESS)
   Process:9623ExecStartPre=/sbin/modprobe
overlay(code=exited,status=0/SUCCESS)
   Main PID:9624(k3s-server)
        Tasks:127
   Memory:1.4G
      CPU:5min 15.723s
```

Para acceder al cluster desde fuera con kubectl tendremos que copiar el fichero /etc/rancher/k3s/k3s.yaml en la ruta ~/.kube/config. Si no existise la carpeta .kube el fichero config la creamos.

```
cp /etc/rancher/k3s/k3s.yaml ~/.kube/config
```

En este momento ya podemos observar el estado de los nodos , aunque ahora mismo solo tenemos instalado y configurado el nodo master.

```
user@master:~$ kubectl get nodes
NAME   STATUS ROLES                  AGE VERSION
```

```
master Ready  control-plane,master 1m  v1.23.6+k3s1
```

Instalación de nodos Worker

En todos los nodos worker ejecutaremos el siguiente comando:

```
curl -sfL https://get.k3s.io |
K3S_URL=https://myserver:6443 K3S_TOKEN=mynodetoken sh -
```

Sustituyendo 'myserver' por la ip del nodo master y 'mynodetoken' por el contenido del fichero node-token , el cuál esta ubicado el la ruta /var/lib/rancher/k3s/server/ del nodo master. Para observar el contenido utilizamos el comando cat.

```
cat /var/lib/rancher/k3s/server/node-token
```

Nos aseguramos que se ha instalado correctamente en todos los nodos observando el estado del servicio en cada uno de los nodos.

```
root@nodo2:~$ systemctl status k3s-agent.service
● k3s-agent.service-Lightweight Kubernetes
  Loaded: loaded(/etc/systemd/system/k3s-agent.service;
enabled; vendor preset: enabled)
  Active: active(running)since Sun2022-05-2922:07:04 CEST;
36min ago
   Docs: https://k3s.io
  Process:923ExecStartPre=/bin/sh-xc/usr/bin/systemctl is-
enabled -quiet nm-cloud-setup.service(code=exited,status=>
  Process:925ExecStartPre=/sbin/modprobe
br_netfilter(code=exited,status=0/SUCCESS)
```

```
  Process:926ExecStartPre=/sbin/modprobe
overlay(code=exited,status=0/SUCCESS)
   Main PID:927(k3s-agent)
    Tasks:42
   Memory: 328.8M
    CPU: 1min 35.536s
```

Ya podemos comprobar el estado de todos los nodos del cluster, por tanto nos dirigimos al nodo master y volvemos a ejecutar:

```
user@master:~$ kubectl get nodes
NAME   STATUS ROLES                  AGE  VERSION
master Ready  control-plane,master 37m  v1.23.6+k3s1
nodo2  Ready  <none>                 31m  v1.23.6+k3s1
nodo1  Ready  <none>                 31m  v1.23.6+k3s1
nodo3  Ready  <none>                 31m  v1.23.6+k3s1
```

En el caso de que el estado de los nodos se iniciara con el estado NotReady podemos ejecutar el comando :

```
user@master:~$ kubectl describe node 'nombre_del_nodo'
```

Con la información que obtenemos podemos resolver todos los errores que no nos permitan tener el cluster operativo.

3.2.2. Helm

Requisitos para instalar Helm:

- Tener Kubernetes instalado.
- Tener configurado localmente kubectl

Existen múltiples maneras de instalar Helm en esta documentación se va a mostrar la instalación con apt en Debian/Ubuntu. Siguiendo la documentación de la página oficial ejecutamos los siguientes pasos.

1. Agregamos los datos necesarios para instalación.

```
curl https://baltocdn.com/helm/signing.asc | gpg --dearmor | sudo tee /usr/share/keyrings/helm.gpg > /dev/null
```

2. Instalar el paquete apt-transport-https, este paquete habilita el soporte https para comunicarte con los repositorios.

```
apt-get install apt-transport-https --yes
```

3. Añadimos el repositorio donde se encuentran los binarios de helm para poder instalarlos.

```
echo "deb [arch=$(dpkg –print-architecture) signed-by=/usr/share/keyrings/helm.gpg] https://baltocdn.com/helm/stable/debian/ all main" | sudo tee /etc/apt/sources.list.d/helm-stable-debian.list
```

4. Actualizamos la lista de repositorios que definimos en el archivo /etc/apt/sources.list ya que hemos añadido el de helm e instalamos el binario.

```
apt-get update && apt-get install helm -y
```

3.2.3 ISCSI

open-iscsi es necesario, ya que Longhorn depende de iscsiadm en el host para proporcionar volúmenes persistentes a Kubernetes.

La instalación de open-iscsi depende de la distribución que utilicemos, en mi caso es Debian Bullseye , por tanto ejecutamos en todos los nodos:

```
apt-get update -y && apt-get install open-iscsi -y
```

Para asegurarnos que esta instalado comprobamos el servicio

```
root@master:~$ systemctl status iscsi
```

```
● open-iscsi.service-Login to default iSCSI targets
  Loaded: loaded (/lib/systemd/system/open-iscsi.service;
enabled; vendor preset: enabled)
  Active: active (exited) since Mon 2022-05-30 18:05:59
EST; 3min 8s ago
    Docs: man:iscsiadm(8)
          man:iscsid(8)
  Process: 34529 ExecStartPre=/bin/systemctl –quiet is-
active iscsid.service (code=exited, status>
  Process: 4530 ExecStart=/sbin/iscsiadm -m node -
loginall=automatic (code=exited,status=21)
  Process: 34531 ExecStart=/lib/open-iscsi/activate-
storage.sh (code=exited, status=0/SUCCESS)
  Main PID: 34531 (code=exited, status=0/SUCCESS)
```

```
    CPU: 16ms
```

3.2.4. NFS

Para instalar un cliente NFSv4, ejecutamos en todos los nodos

```
apt-get update && apt-get install nfs-common
```

3.3. Instalación de Longhorn con Helm

Comenzamos por añadir el repositorio de longhorn a helm:

```
root@master:~#helm repo add longhorn https://charts.longhorn.io
   "longhorn" has been added to your repositories
```

Actualizamos los repositorios para poder usarlo.

```
root@master:~# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "longhorn" chart repository
Update Complete. ⎈Happy Helming!⎈
```

Podemos ver la configuración por defecto del chart de longhorn ejecutando el siguiente comando:

```
helm show values longhorn/longhorn
```

En este caso vamos a guardar la configuración en un fichero temporal, de esta manera se puede hacer una instalación más personalizada, vamos a instarlo con la configuración por defecto pero es conveniente tener este archivo porque si en el futuro queremos hacer cambios , esta es la única manera de saber con que configuración esta instalado. Para guardar la información redirigimos el resultado del comando anterior a un fichero.

```
helm show values longhorn/longhorn > /tmp/longhorn-values.yaml
```

Ya podemos instalar longhorn para ello ejecutamos:

```
user@master:~$  helm install longhorn longhorn/longhorn --values /tmp/longhorn-values.yaml --namespace longhorn-system –create-namespace
```

```
NAME: longhorn
LAST DEPLOYED: Mon May 30 19:48:58 2022
NAMESPACE: longhorn-system
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Longhorn is now installed on the cluster!
```

```
Please wait a few minutes for other Longhorn components
such as CSI deployments, Engine Images, and Instance
Managers to be initialized.
```

con la opción --values indicamos el archivo con todos los valores de la configuración

Como podemos obervar con este comando también creamos el namespace llamado longhorn-system, el cual se va a crear antes de implementar longhorn en nuestro cluster.

Podemos observar todos los recursos que se están instalados, ejecutando:

```
kubectl -n longhorn-system get all
```

Y para comprobar que los pods ya se van repartiendo entre los nodos ejecutamos:

```
user@master:~$ kubectl -n longhorn-system get pod -o wide |
grep longhorn-manager

longhorn-manager-hwwqt 1/1 Running 0  46h  10.42.3.5   nodo3
longhorn-manager-pxf9w 1/1 Running 0  46h  10.42.2.4   nodo1
longhorn-manager-v57jw 1/1 Running 1  46h  10.42.0.10  master
longhorn-manager-fwwln 1/1 Running 0  46h  10.42.1.6   nodo2
```

3.3.1. Acceso al Dashboard de Longhorn

Al instalar Longhorn en un clúster de Kubernetes con Helm tenemos que crear un Ingress Controller para acceder a la interfaz gráfica que tiene Longhorn. Un objeto Ingress nos permite utilizar un proxy inverso que por medio de reglas de encaminamiento, que obtiene de la API de Kubernetes, accedemos a las aplicaciones del cluster por medio de nombres, en este caso a la UI de Longhorn, la herramienta web para controlar el estado de los nodos, espacio de disco ... , entre otros datos.

Lo primero que tenemos que hacer es crear el fichero de autenticación básica, es imporante que el fichero se llame auth de lo contrario el Ingress devuelve un error 503.

Para crear el fichero ejecutamos:

```
USER=<USERNAME_HERE>; PASSWORD=<PASSWORD_HERE>; echo "$
{USER}:$(openssl passwd -stdin -apr1 <<< ${PASSWORD})" >> auth
```

Cambiando USERNAME_HERE y PASSWORD_HERE, por un usuario y una contraseña, el fichero se genera con el siguiente formato:

```
user@master:~$ cat auth
javier:$apr1$xTMl7UGi$6Vooyi954NLG0ZkVjGy7v0
```

Una vez tenemos el fichero creamos un objeto tipo Secret. Los objetos de tipo Secret en Kubernetes te permiten almacenar y administrar información confidencial, como contraseñas, tokens OAuth y llaves ssh. Poniendo esta información en un Secret es más seguro y más flexible que ponerlo en la definición de un Pod o en un container image.

Para crear el secret ejecutamos:

```
kubectl -n longhorn-system create secret generic basic-auth --from-file=auth
```

A continuación definimos el ingress en un fichero yaml y lo creamos, el fichero yaml es el siguiente:

```
apiVersion:networking.k8s.io/v1
kind:Ingress
metadata:
  name:longhorn-ingress
  namespace: longhorn-syetem
  annotations:
   # type of authentication
   nginx.ingress.kubernetes.io/auth-type: basic
   # prevent the controller from redirecting (308) to HTTPS
   nginx.ingress.kubernetes.io/ssl-redirect: 'false'
  # name of the secret that contains the user/password definitions
   nginx.ingress.kubernetes.io/auth-secret: basic-auth
   # message to display with an appropriate context why the authentication is required
   nginx.ingress.kubernetes.io/auth-realm:'Authentication Required'
   # custom max body size for file uploading like backing image uploading
   nginx.ingress.kubernetes.io/proxy-body-size:10000m
spec:
  rules:
  -host: www.longhorn-web.org
  -http:
     paths:
     -pathType: Prefix
      path:"/"
      backend:
        service:
          name:longhorn-frontend
          port:
           number:80
```

Una vez definido, ejecutamos:

```
kubectl -n longhorn-system apply -f longhorn-ingress.yml
```

Si observamos datos del ingress nos percataremos de que se puede acceder con cualquier ip del nodo:

```
user@master:~$ kubectl -n longhorn-system get ingress
NAME                CLASS   HOSTS   ADDRESS         PORTS   AGE
longhorn-ingress    <none>  *       172.22.7.216    80      102s
                                    172.22.7.86
                                    172.22.8.170
                                    172.22.8.255
```

Añadimos la ip de la maquina donde se encuentre el cluster, y el nombre que hemos indicado en el ingress, en el fichero /etc/hosts de la maquina donde se encuentre el navegador con el que vamos a acceder.

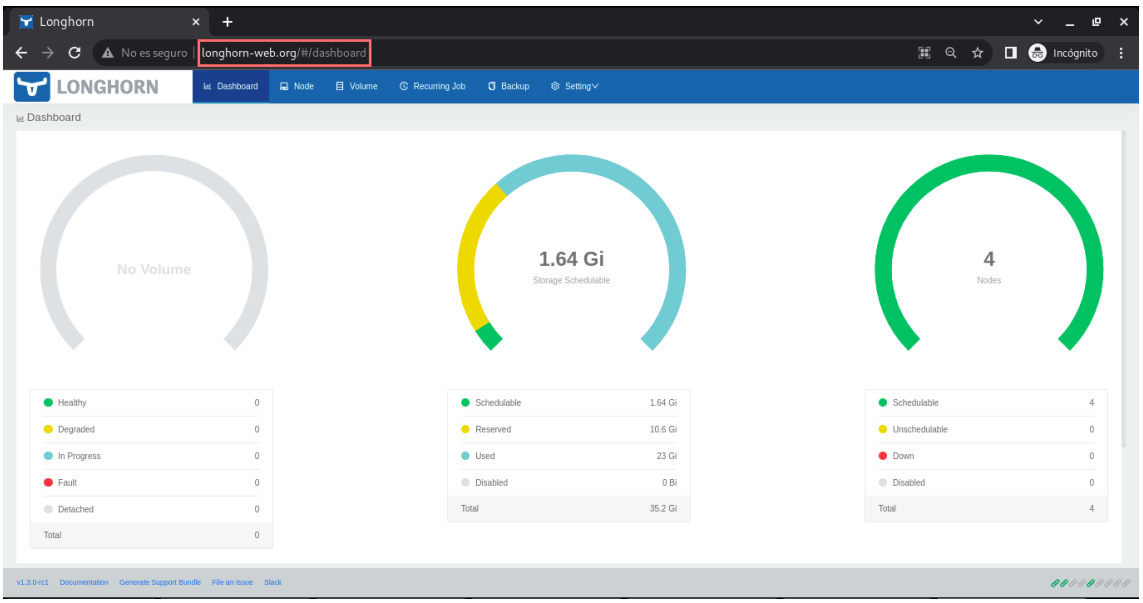


Figura 12: Dashboard Longhorn

4. Pruebas de Longhorn

En este apartado vamos a realizar pruebas con Longhorn para mostrar un poco su funcionamiento.

4.1. Creación de volumen desde interfaz Longhorn

Antes de proceder a crearlo podemos comprobar que Volúmenes tenemos creados y en funcionamiento en nuestro cluster. En mi caso tengo 2 volúmenes, uno para wordpress y otro para mysql.

```
user@master:~$ kubectl get pvc
NAME  STATUS VOLUME CAPACITY ACCESSMODES STORAGECLASS AGE
db-pvc Bound pvc-b5 2Gi      RWO          longhorn     89m
wp-pvc Bound pvc-e57 1Gi     RWO          longhorn     89m
```

Ahora nos dirigimos a la pantalla principal de la interfaz de Longhorn y accedemos al apartado Volume.

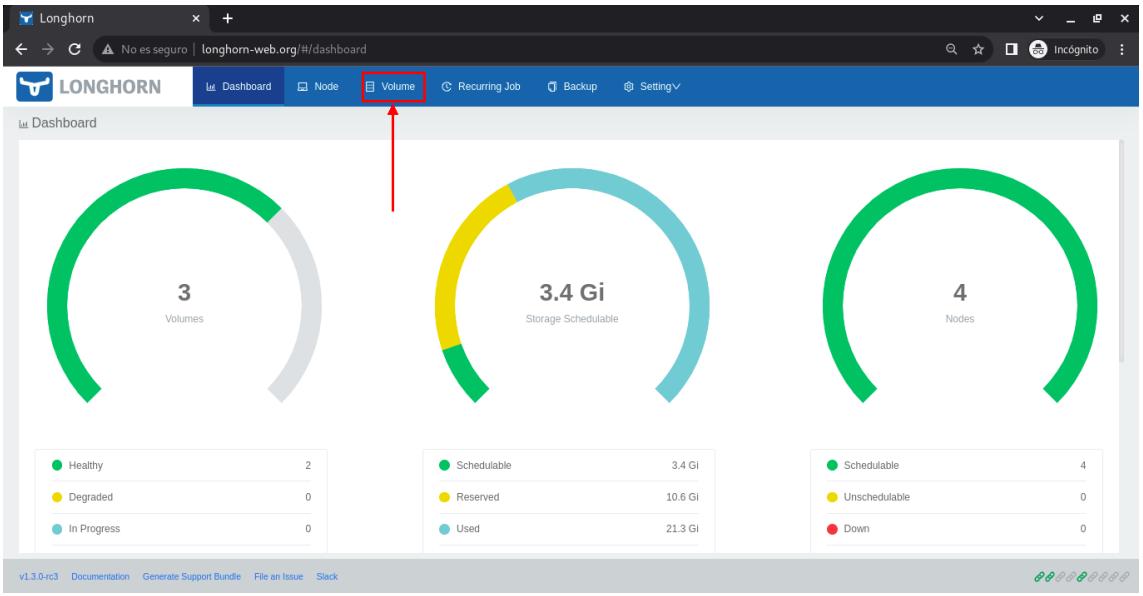


Figura 13: Creación volumen desde Longhorn-1

Ahí tenemos un apartado para crear volúmenes en el que especificamos las características del mismo.

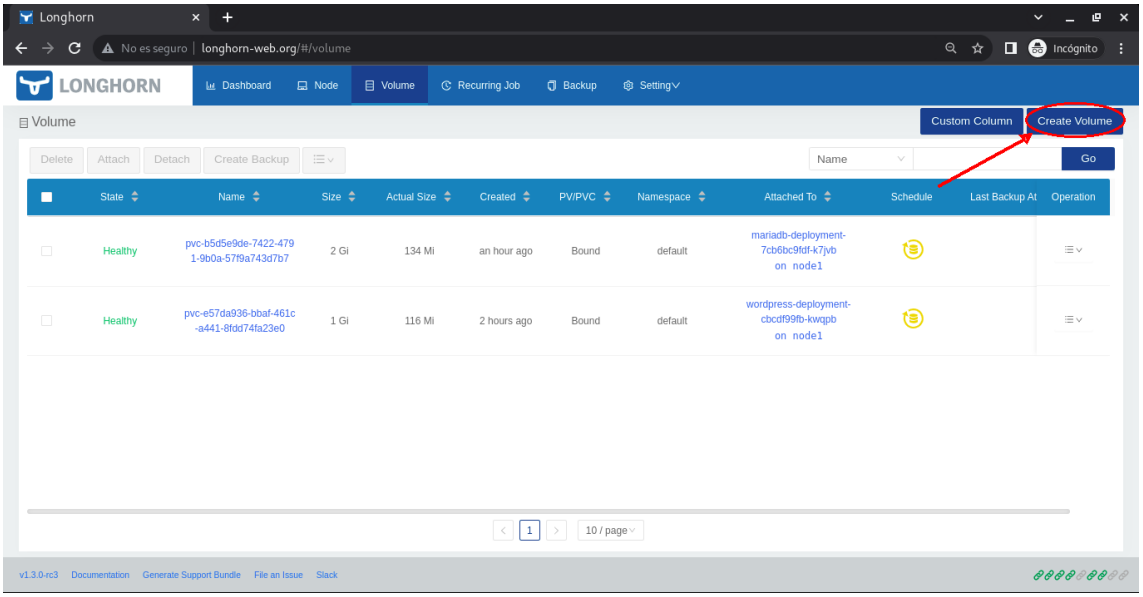


Figura 14: Creación de volumen desde Longhorn-2

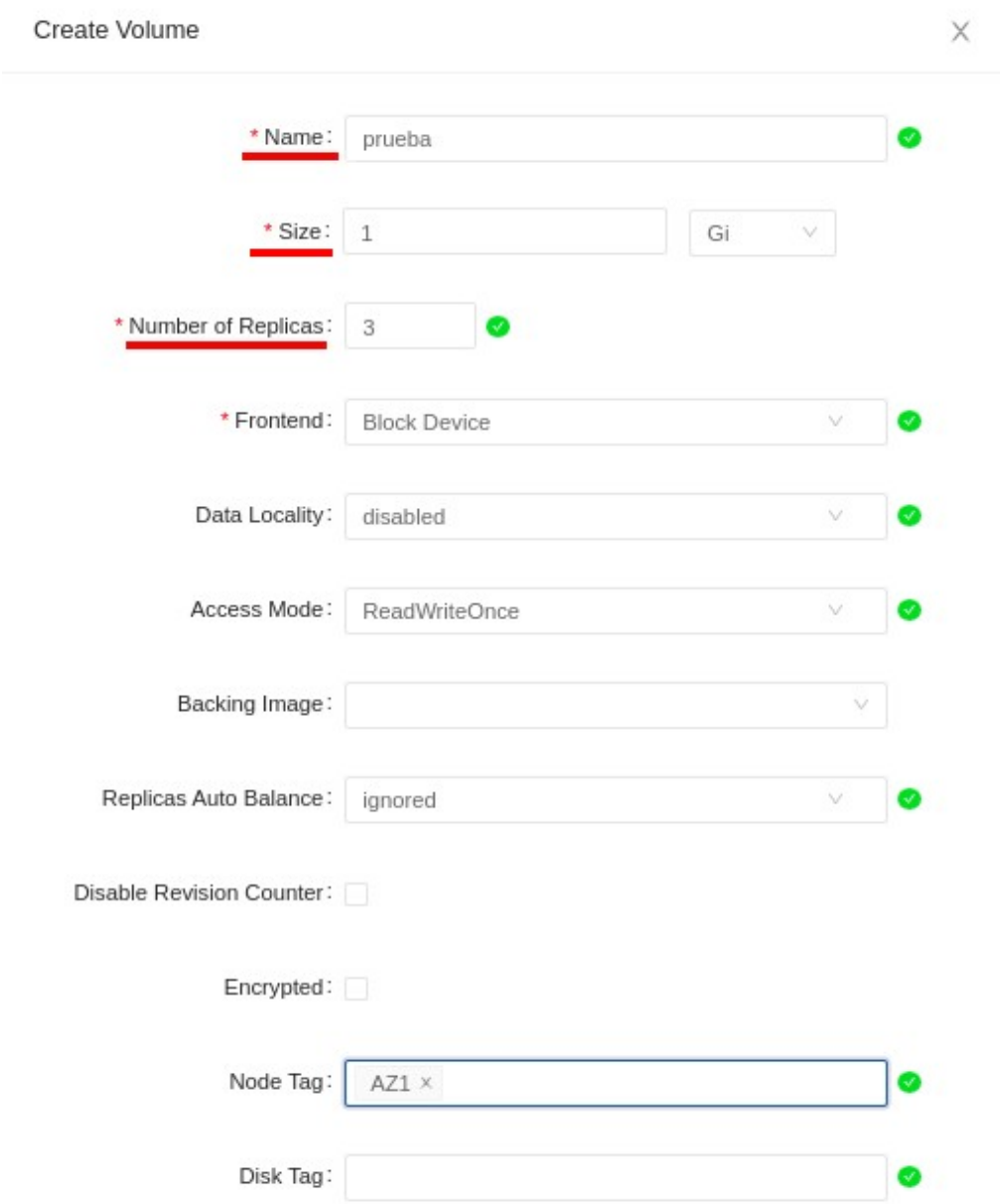


Figura 15: Creación de volumen desde interfaz de Longhorn-3

Automáticamente se crea el volumen ,

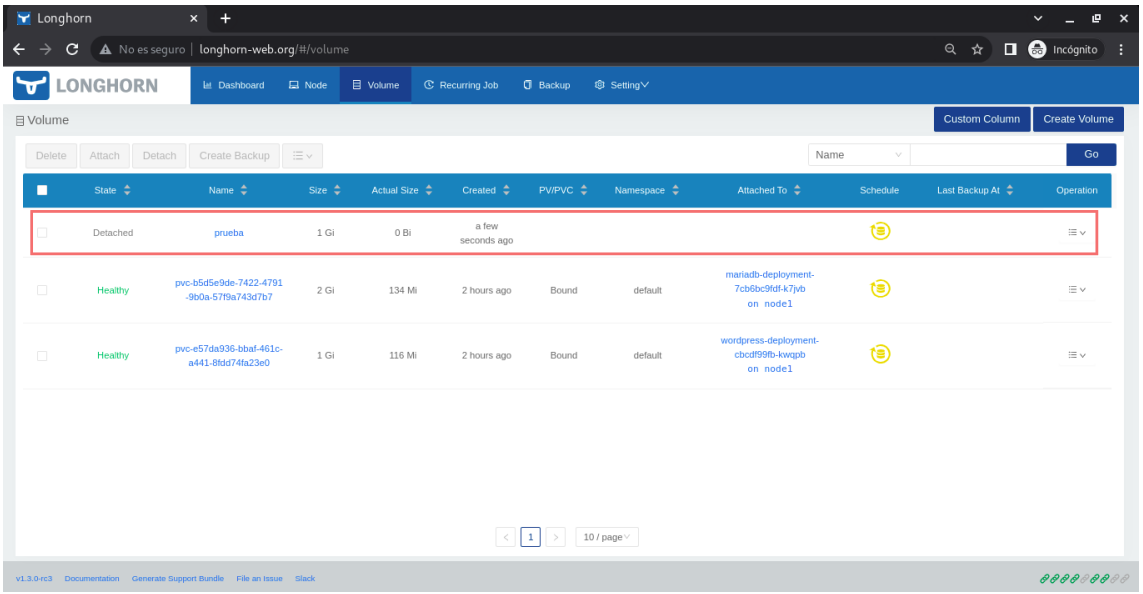


Figura 16: Creación de volumen desde interfaz Longhorn-4

pero tenemos que crear un recurso pvc para el. Para ello seleccionamos el volumen pulsamos en el icono que se ve en la imagen

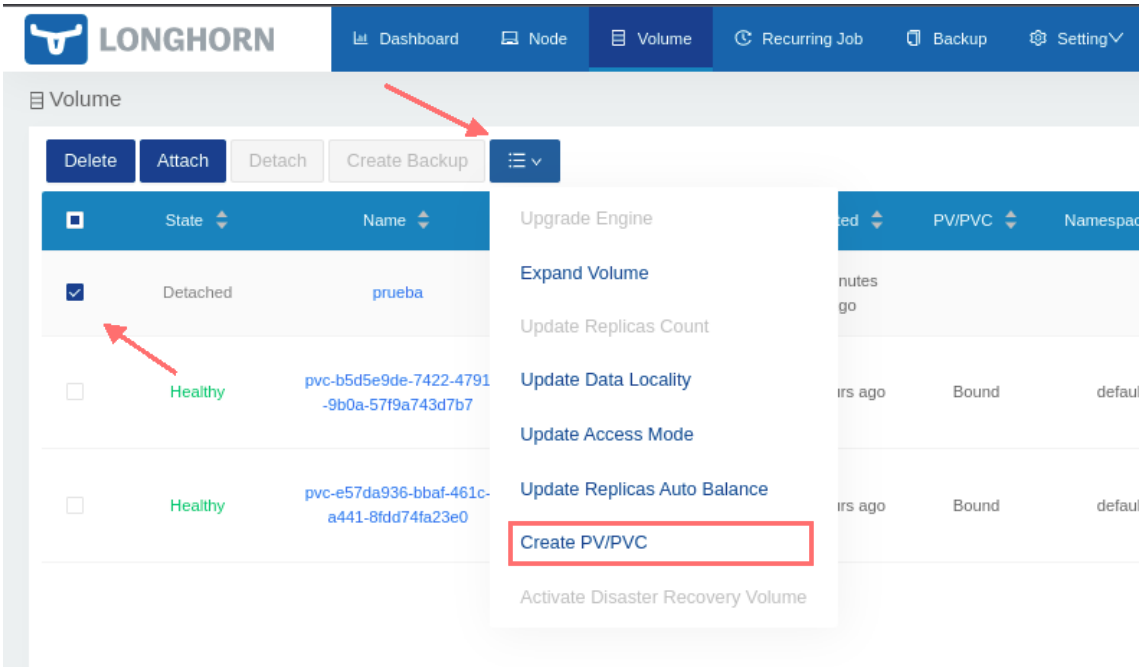


Figura 17: Creación de volumen desde interfaz Longhorn-5

Y tan solo tendríamos que seleccionar el namespace al que lo queremos asociar, en mi caso default.

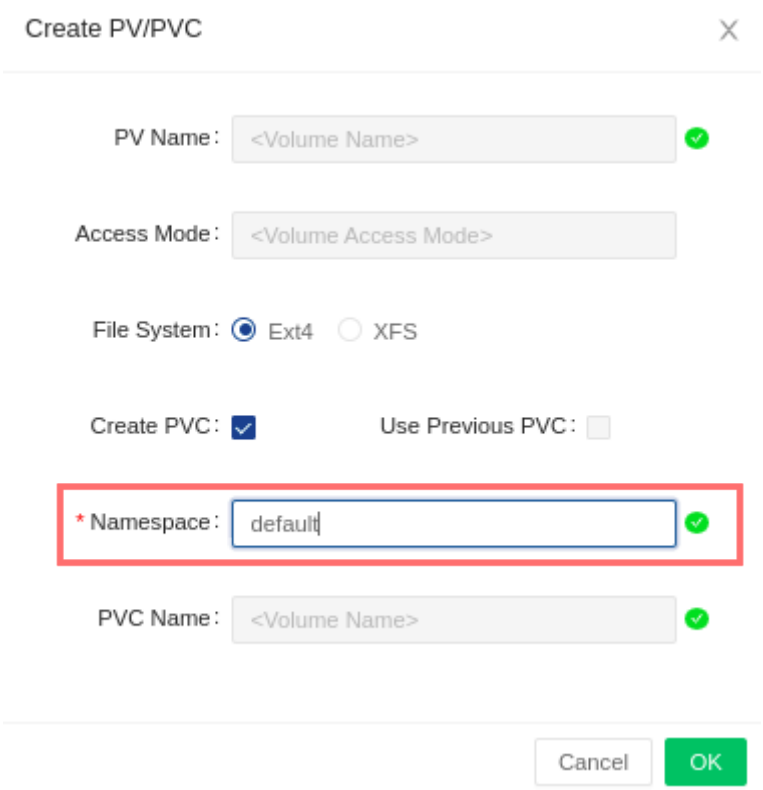


Figura 18: Creación de volumen desde la interfaz de Longhorn-6

Si volvemos a observar los volúmenes con kubectl observaremos que aparece nuestro nuevo volumen:

```
user@master:~$ kubectl get pvc
NAME STATUS VOLUME  CAPACITY ACCESSMODES STORAGECLASS AGE
db-pvc Bound pvc-b5 2Gi      RWO          longhorn     91m
wp-pvc Bound pvc-e5 1Gi      RWO          longhorn     91m
prueba Bound prueba 1Gi      RWO          longhorn     51s
```

Aunque esta no es la manera en la que generalmente creamos volúmenes en kubernetes ya que se suelen desplegar junto con los pods.

4.2. Despliegue de un pod junto a un volumen con kubectl

Como hemos visto en el apartado anterior se puede gestionar la creación de volúmenes desde la interfaz pero la forma común de desplegar pods y volúmenes es desde la linea de comandos.

En este apartado vamos a desplegar un pod y un volumen con kubectl y vamos a comprobar como se comporta longhorn frente a esto.

Para ello vamos a definir un pod y un volumen en ficheros yaml.

pod

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx
spec:
  containers:
  - name: nginx
    image: nginx:1.14.2
    ports:
    - containerPort: 80
  volumes:
```

```
- name: mypd
  persistentVolumeClaim:
    claimName: mypvc
```

pvc

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mypvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
```

Una vez estén definidos ejecutamos:

```
user@master:~/ejemplo-podypvc-basico$ kubectl apply -f .
  pod/nginx created
  persistentvolumeclaim/mypvc created
```

Y al momento tenemos toda la información en longhorn.

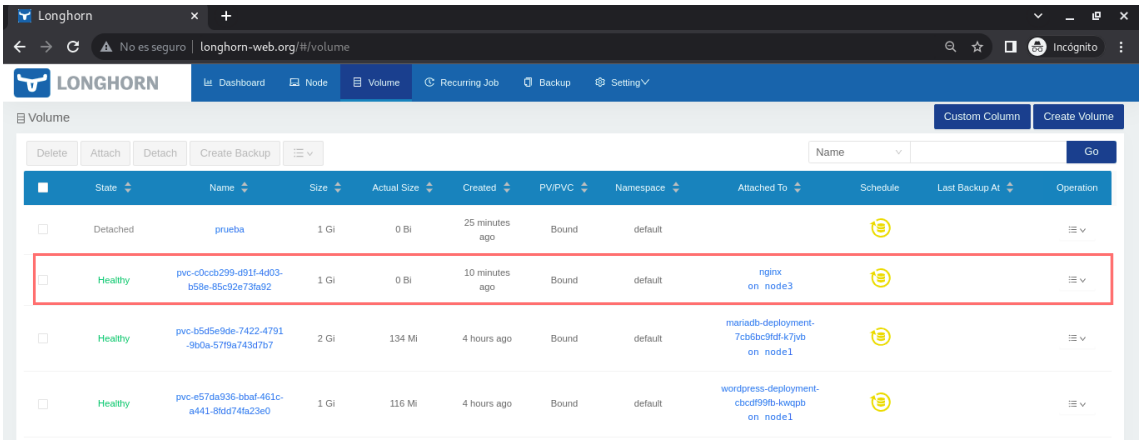


Figura 19: Creación de un volumen con kubectl

Si nos fijamos esta enlazado con el pod nginx en el nodo3

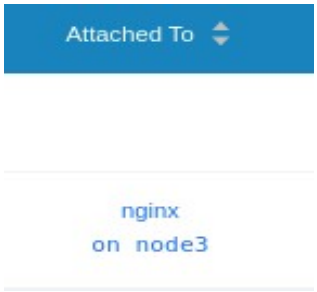


Figura 20: Attach to Node3

4.3. Prueba de backups con Longhorn

En este apartado vamos a comprobar la facilidad que nos aporta Longhorn a la hora de restaurar una aplicación , o la información de la misma, que tengamos en nuestro cluster.

Para ello vamos a desplegar un wordpress con una base de datos mysql, en el cuál vamos a crear 2 posts. Cuando creemos el primero vamos a hacer una snapshot y posteriormente crearemos el segundo, actos seguido eliminaremos y levantaremos el deployment y podremos comprobar que la información estaba guardada hasta la snapshot que hemos realizado.

Por tanto vamos a comenzar por desplegar la aplicación, es por eso que vamos a clonar un repositorio github dónde se encuentran todos los ficheros yaml que necesito.

```
git clone https://github.com/javivegaa01/wordpress-kubernetes.git
```

Me coloco en la carpeta que se me ha clonado y ejecuto el comando

```
user@master:~/ejemplo-wordpress_mysql$ kubectl apply -f

configmap/bd-datos created
secret/bd-passwords created
deployment.apps/mariadb-deployment created
persistentvolumeclaim/mariadb-pvc created
service/mariadb-service created
deployment.apps/wordpress-deployment created
ingress.networking.k8s.io/wordpress-ingress created
persistentvolumeclaim/wordpress-pvc created
service/wordpress-service created
```

para levantar la app.

Comprobamos que se esté desplegando bien ejecutando

```
user@master:~/ejemplo-wordpress_mysql$ kubectl get deployments.apps
NAME                  READY UP-TO-DATE AVAILABLE AGE
mariadb-deployment    1/1    1          1         75s
wordpress-deployment  1/1    1          1         75s

user@master:~/ejemplo-wordpress_mysql$ kubectl get svc | grep service
mariadb-service  ClusterIP  10.43.230.157  <none>  3306/TCP 98s
wordpress-service     NodePort     10.43.17.251     <none> 80:30993/TCP,443:30011/TCP 98

user@master:~/ejemplo-wordpress_mysql$ kubectl get pvc
NAME STATUS VOLUME CAPACITY ACCESS MODES STORAGECLASS AGE
mariadb-pvc Bound pvc-20a7204c-8058-4985-bc6a-1a7a6d164214 2Gi RWO longhorn 2m35s
wordpress-pvc Bound pvc-8f5f6278-a03a-4c56-b5cf-f15c8b74bd2c 1Gi RWO longhorn 2m35
```

Automáticamente en Longhorn se han generado los volúmenes con sus respectivas réplicas:

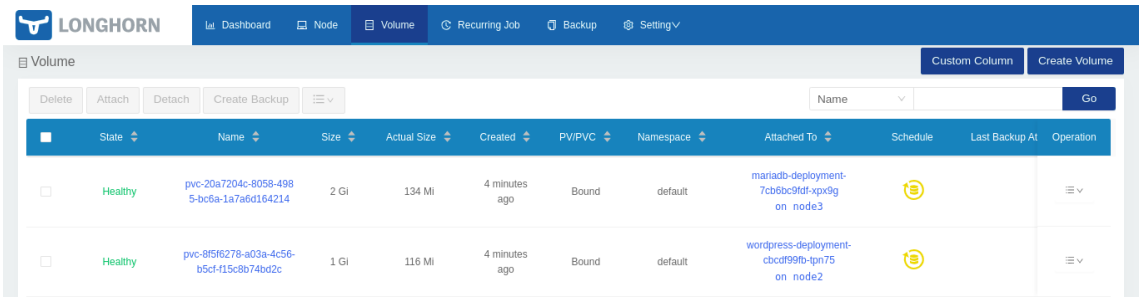


Figura 21: Backups-longhorn-1

Volumen para mariabd:

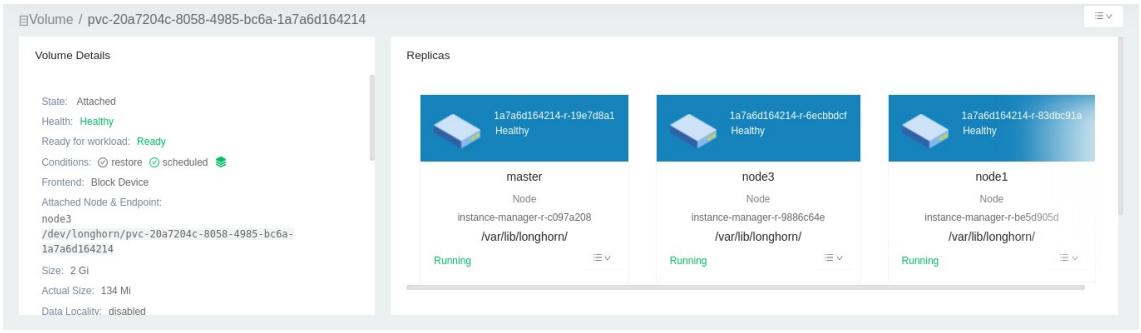


Figura 22: Backups-longhorn-2

Volumen para wordpress:

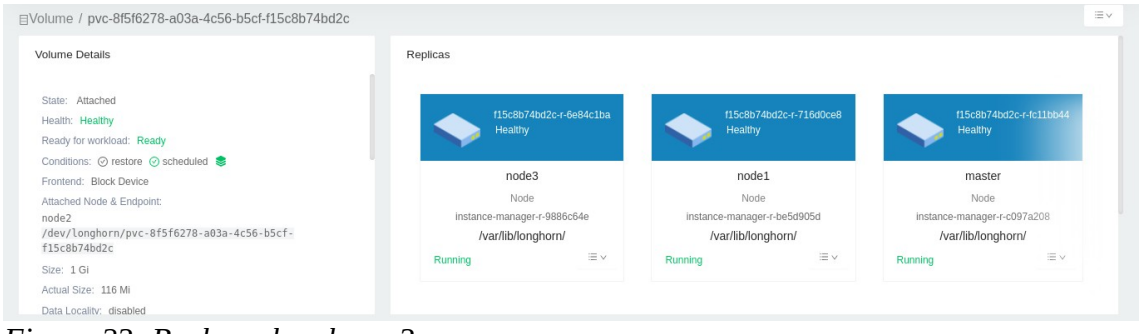


Figura 23: Backups-longhorn-3

Ahora vamos acceder a wordpress mediante el ingress y lo instalamos.

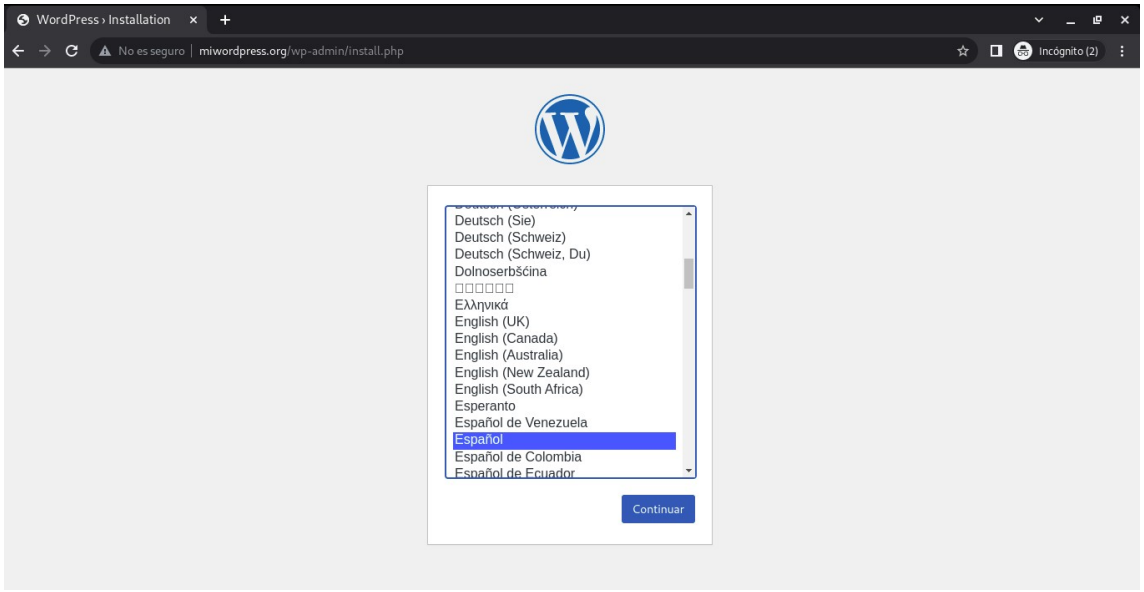


Figura 24: Backups-longhorn-4

Una vez instalado creamos el primer posts.

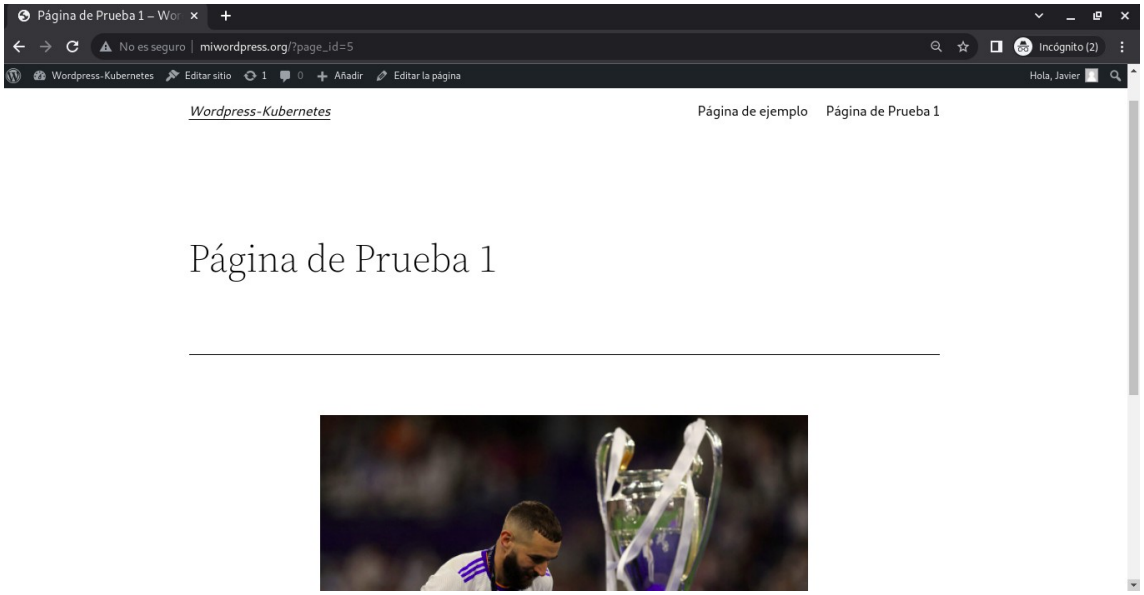


Figura 25: Backups-longhorn-5

Y nos dirigimos a la interfaz de Longhorn para hacer una snapshot. En el apartado Volumes seleccionamos el volumen de mariadb y pulsamos en 'Take a snapshots'.

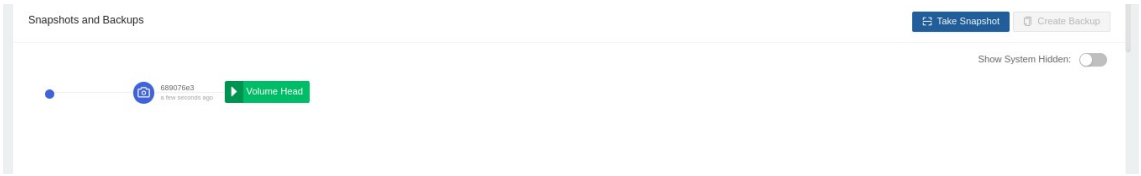


Figura 26: Backups-longhorn-7

A continuación vamos a crear otro posts para hacer la prueba.

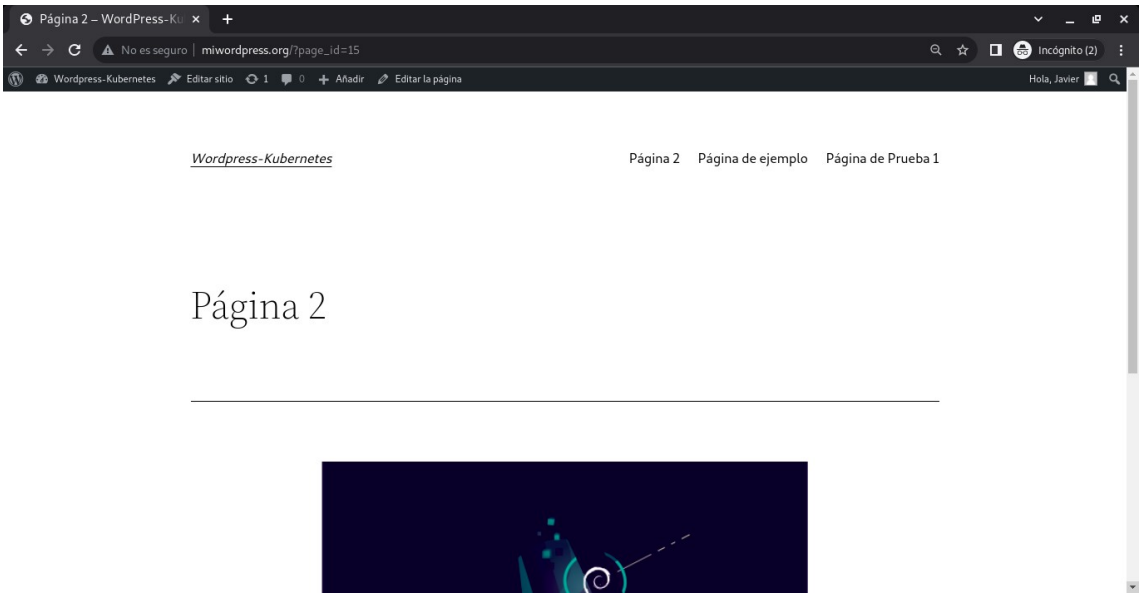


Figura 27: Backups-longhorn-7

Ahora vamos a borrar los deployments:

```
user@master:~/ejemplo-wordpress_mysql$ kubectl delete deployment.apps/mariadb-deployment
deployment.apps "mariadb-deployment" deleted
```

Lógicamente wordpress nos mostrará que se le ha caído la base de datos.

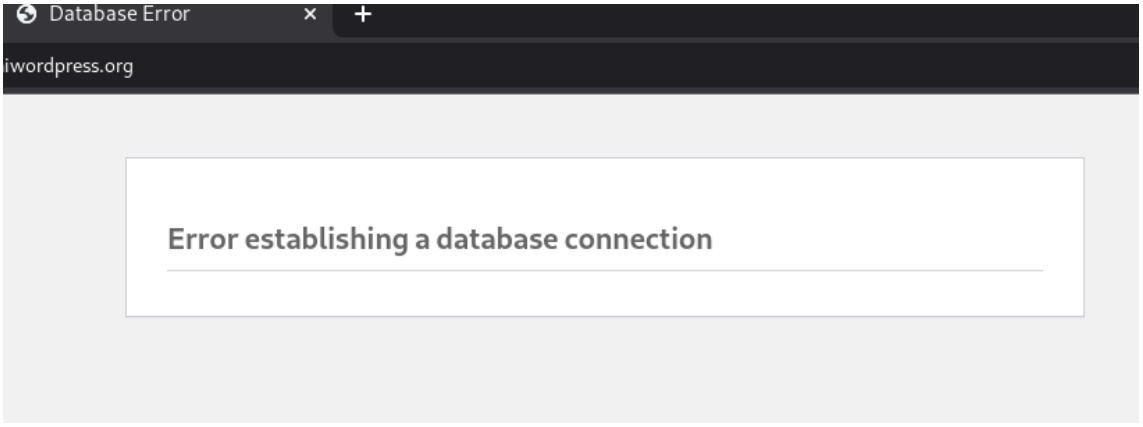


Figura 28: Backups-longhorn-8

Para recuperarlo asociamos el volumen a cualquier nodo del cluster en modo mantenimiento.

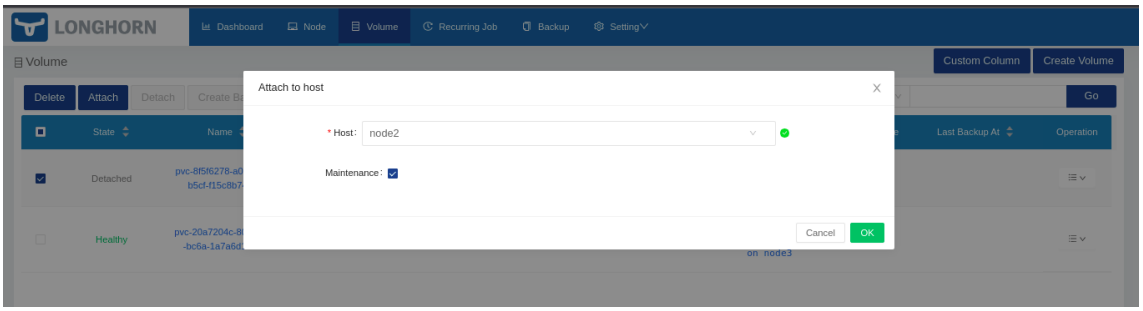
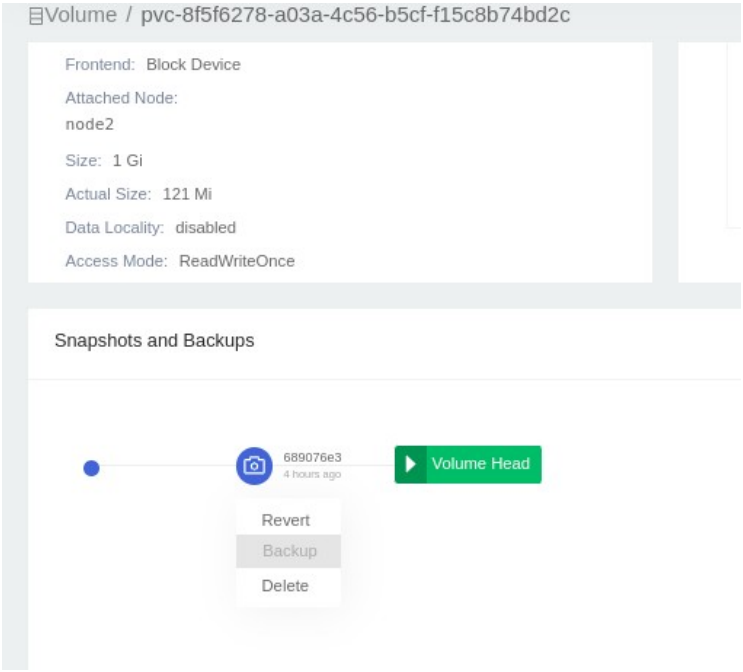


Figura 29: Backups-longhorn-9

Y nos dirigimos al snpashot y seleccionamos recuperar.

Figura 30: Backups-longhorn-10



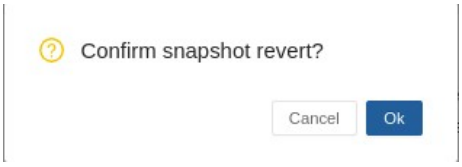


Figura 31: Bakcups-longhorn-11

Una vez este recuperada desasociamos el volumen y leventamos nuevamente el deployment de mariadb.

```
user@master:~/ejemplo-wordpress_mysql$ kubectl apply -f mariadb-deployment.yaml
deployment.apps/mariadb-deployment created
```

Comprobamos como rápidamente se vuelve a asociar con el pod

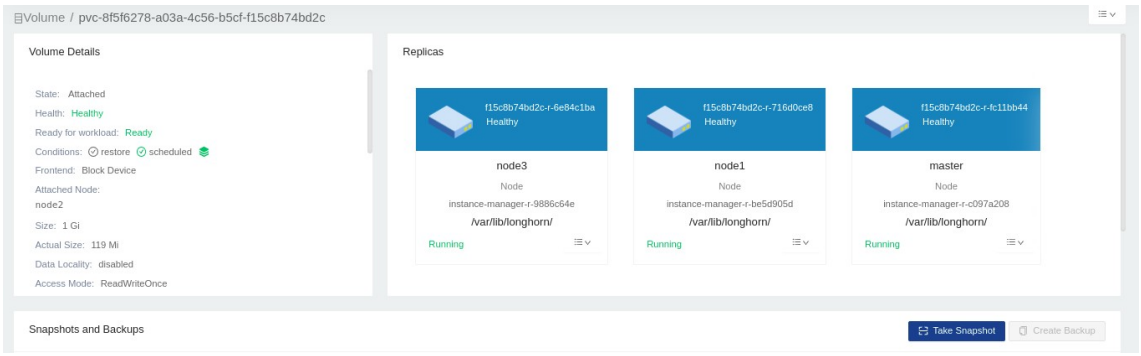


Figura 32: Backups-longhorn-13

Y si nos dirigimos a la página comprobamos como solo tenemos la primera página, la cuál es la única que existía cuando se hizo la snapshot.

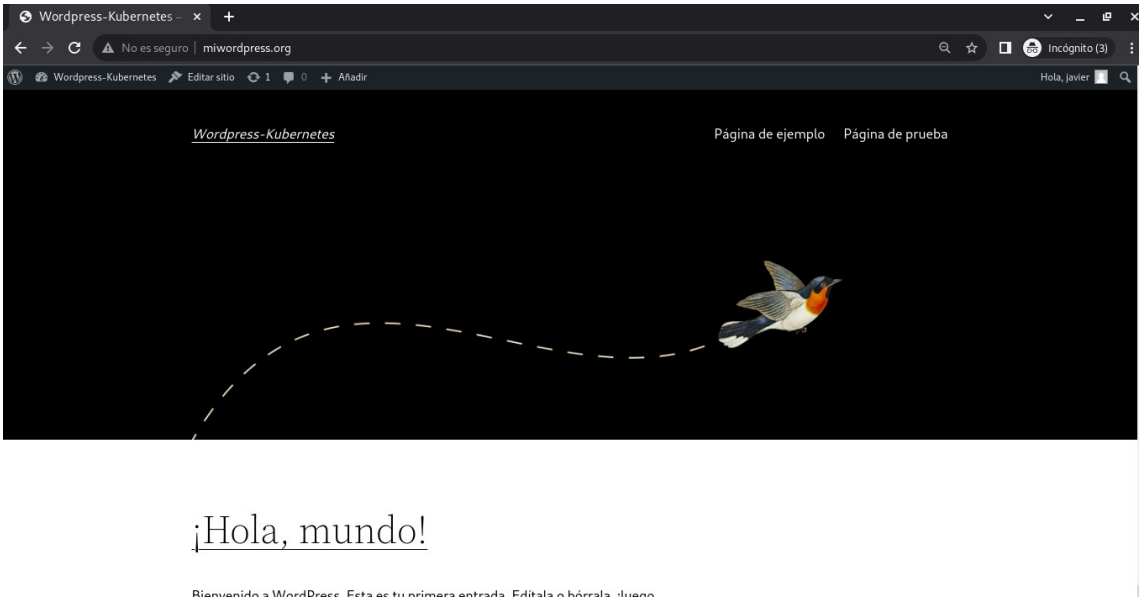


Figura 33: Backups-longhorn-14

4.4. Añadir discos de almacenamiento al cluster

Longhorn nos permite añadir nuevos discos a almacenamiento, la principal ventaja de esto es que no tendríamos que preocuparnos por la falta de espacio ya que podemos añadir discos en cualquier momento sin necesidad que longhorn los registre durante su instalación, para ello vamos a agregar un disco de 5 Gbi a alguno de los nodos.

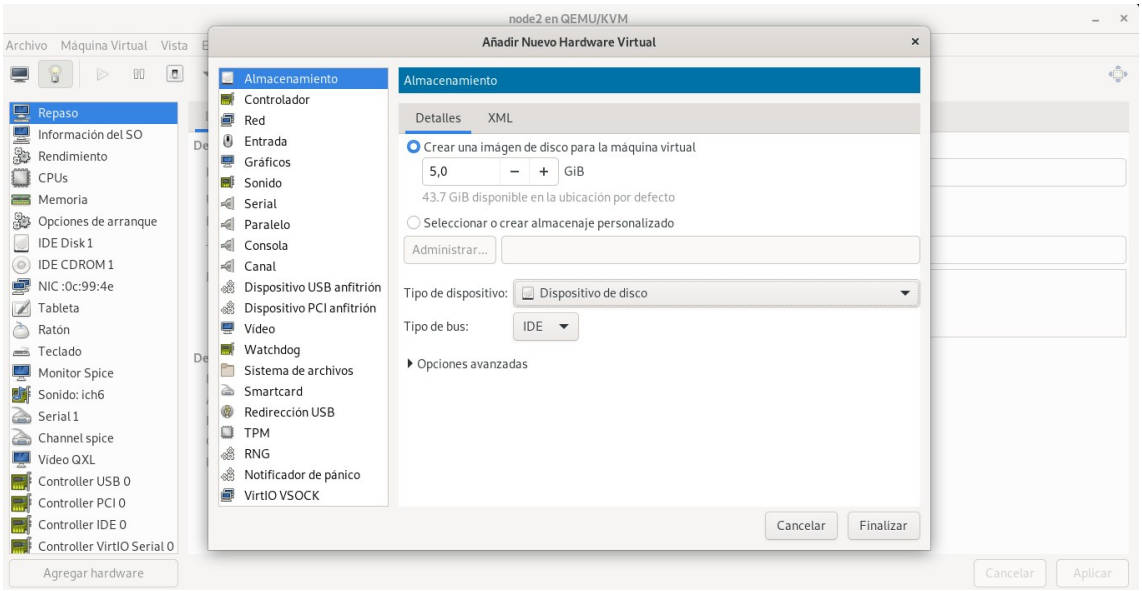


Figura 34: Añadir-discos-1

Tras añadirlo primero comprobamos que aparezca en nuestra máquina ejecutando el comando lsblk, en mi caso es /dev/sdb.

```
debian@node2:~$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda 8:0 0 10G 0 disk
├─sda1 8:1 0 9G 0 part /
├─sda2 8:2 0 1K 0 part
└─sda5 8:5 0 975M 0 part [SWAP]
```

```
sdb 8:16 0 5G 0 disk
sr0 11:0 1 1024M 0 rom
```

Tenemos que proporcionarle un sistema de ficheros y montarlo en algún directorio, en mi caso:

```
root@node2:~# mkfs.ext4 /dev/sdb
Allocating group tables: done
Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

```
root@node2:~# mkdir /mnt/disco-2
```

```
root@node2:~# mount /dev/sdb /mnt/disco-2/
```

Una vez montado nos dirigimos al interfaz de Longhorn, concretamente a Node y seleccionamos 'Operation' > 'Edit Node and Disks' en el nodo2.

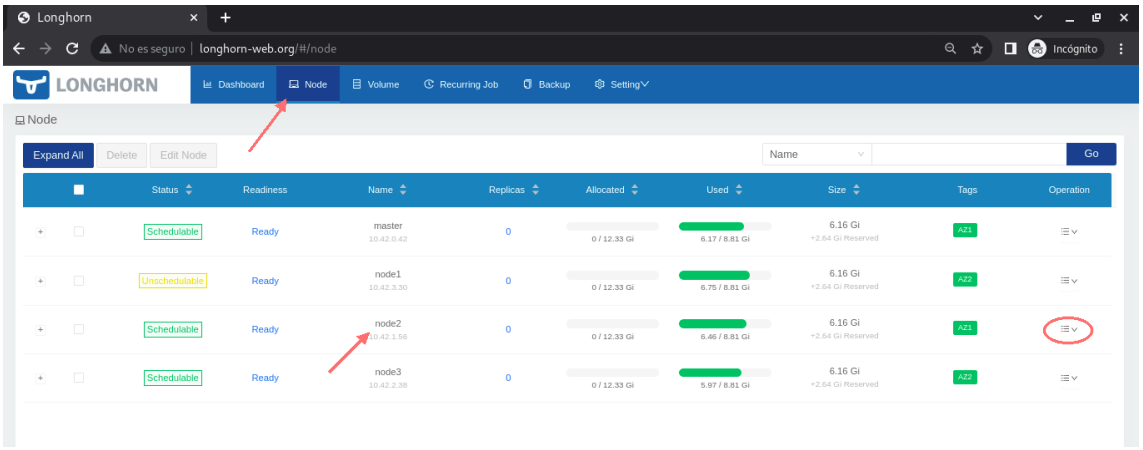


Figura 35: Añadir-discos-2

Ahí si pueden cambiar algunos parámetros pero nos vamos a centrar el botón 'Add Disk' ubicado al final del menú desplegado.

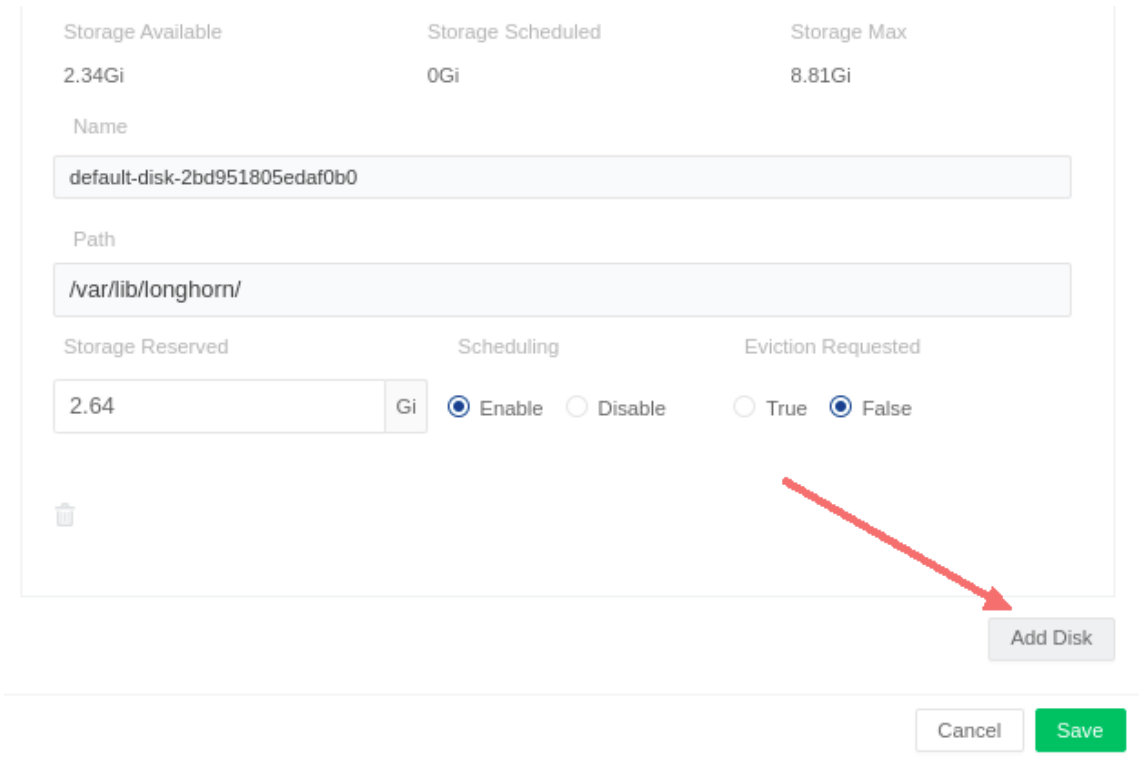


Figura 36: Añadir-discos-3

Rellenamos los campos necesarios, en Path escribimos la ruta en la cúal hemos montado el disco, también vamos añadirle una etiqueta para poder elegir si montarlo en ese disco o en otro.

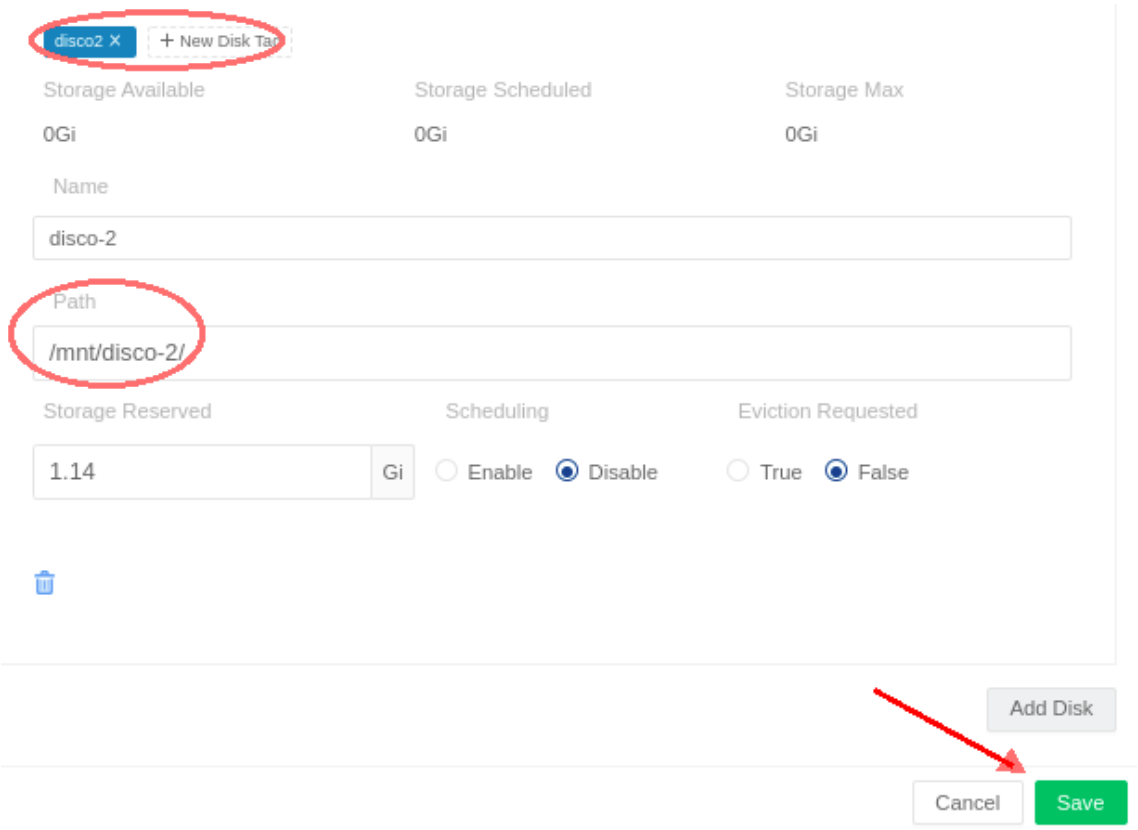


Figura 37: Añadir-discos-4

Cuando tengamos los campos completos pulsaremos en Save y comprobamos que el espacio disponible haya aumentado.

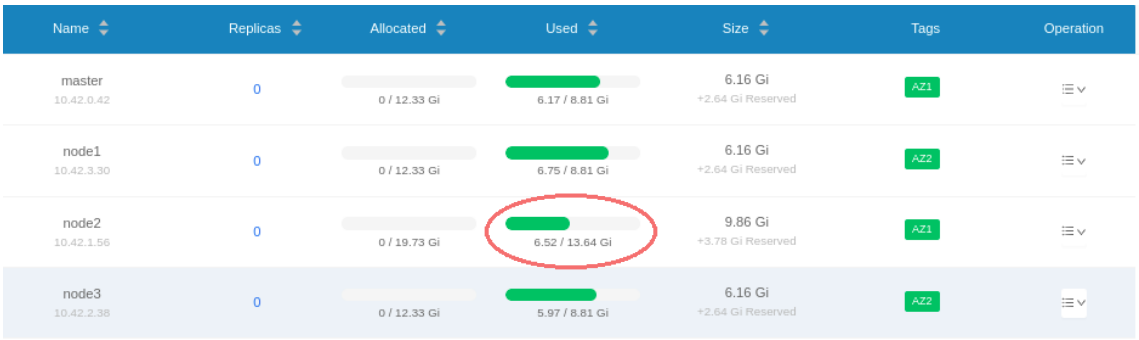


Figura 38: Añadir-discos-5

Ahora cuando creemos un pod podemos elegir que se encuentre en el espacio específicamente, para ello creamos un volumen y especificamos la etiqueta de disco que hemos indicado.

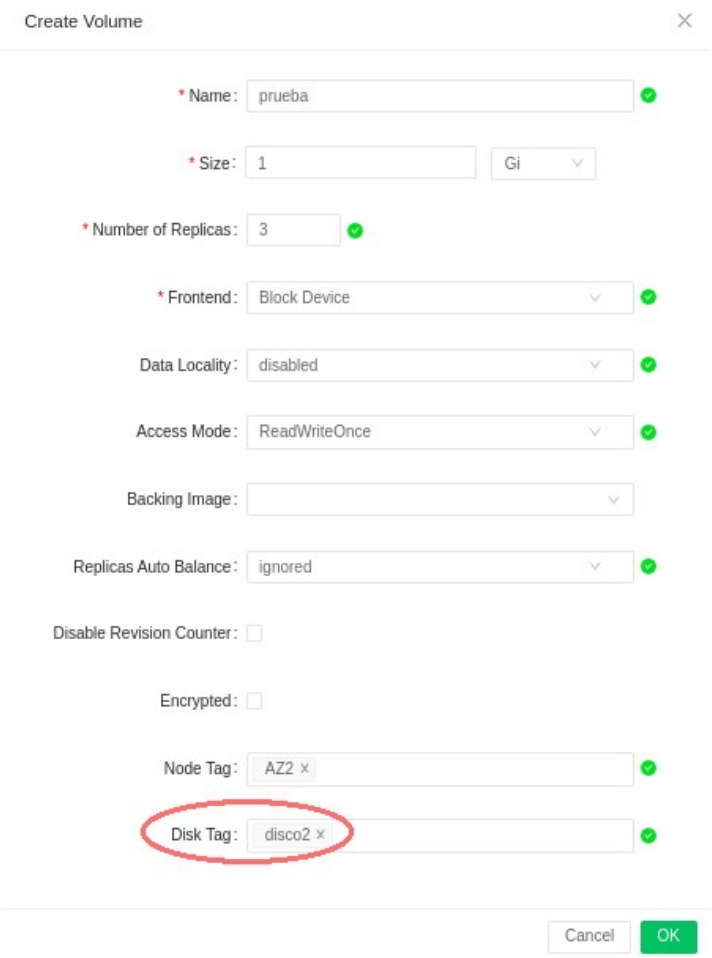


Figura 39: Añadir-discos-6

5. Bibliografía

Install k3s: https://rancher.com/docs/k3s/latest/en/installation/

Install Helm: https://helm.sh/docs/intro/install/

Longhorn docs: https://longhorn.io/docs/1.2.4/

Kubernetes docs: https://kubernetes.io/es/docs/home/