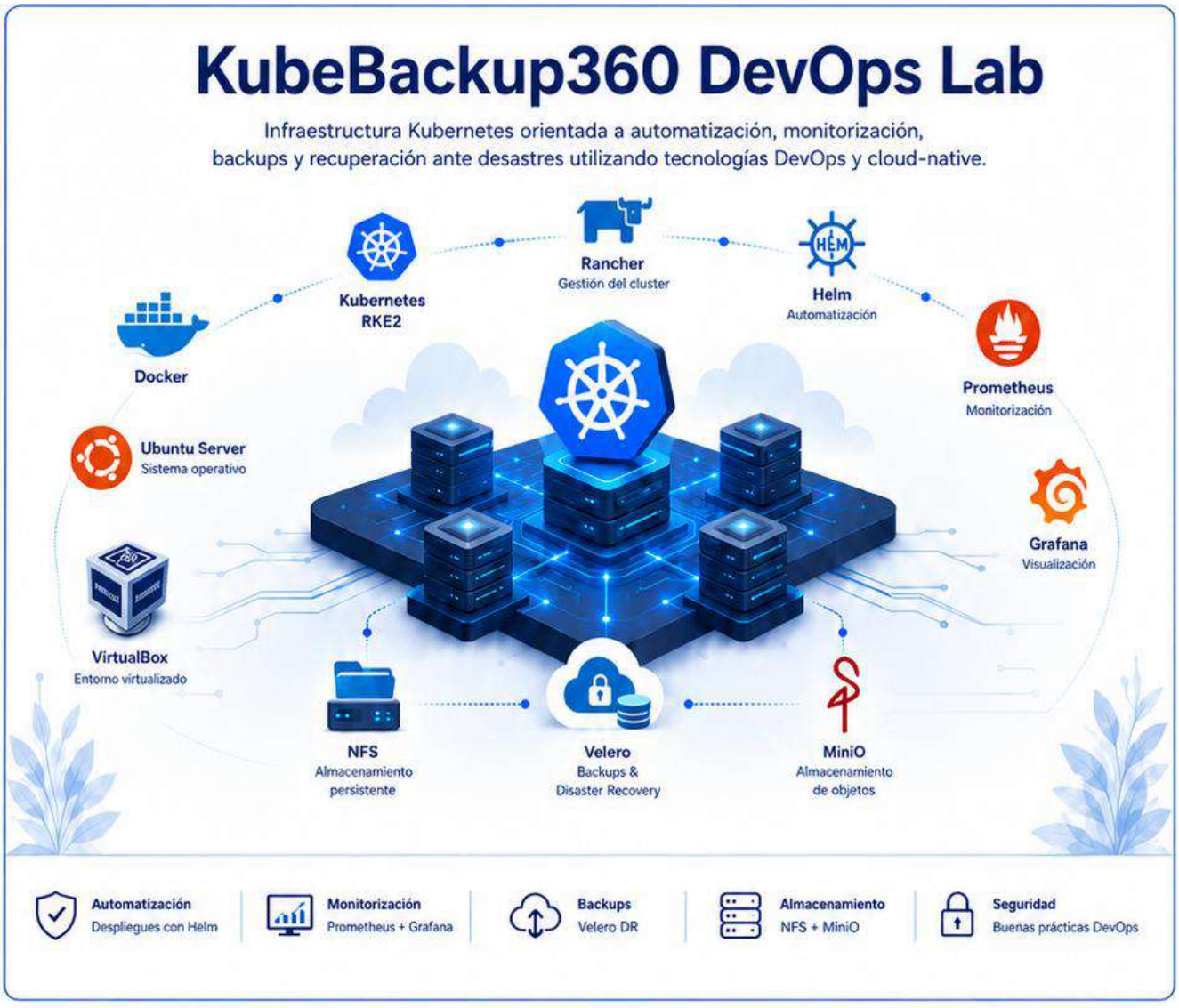
INSTITUT
PUIG CASTELLAR

# KUBEBACKUP360

## Sistema de copias de seguridad y recuperación para Kubernetes




Proyecto de Desarrollo e Investigación

ASIR Administració de Sistemes Informàtics i Xarxes

Centro : INSTITUT EL PUIG CASTELLAR


**Autores**

-Luis Alfredo Florez
-Clara Ines Nchama


**Grup : ASIX 2**

# INDICE

# 1 RESUMEN TÉCNICO

El proyecto KubeBackup360 consiste en el diseño, implementación y validación de una infraestructura Kubernetes resiliente orientada a la gestión de aplicaciones containerizadas, almacenamiento persistente, copias de seguridad y recuperación ante desastres en un entorno virtualizado. La solución desarrollada se ha desplegado íntegramente sobre máquinas virtuales Ubuntu Server utilizando VirtualBox como plataforma de virtualización, permitiendo simular un entorno distribuido similar al utilizado en infraestructuras empresariales reales.

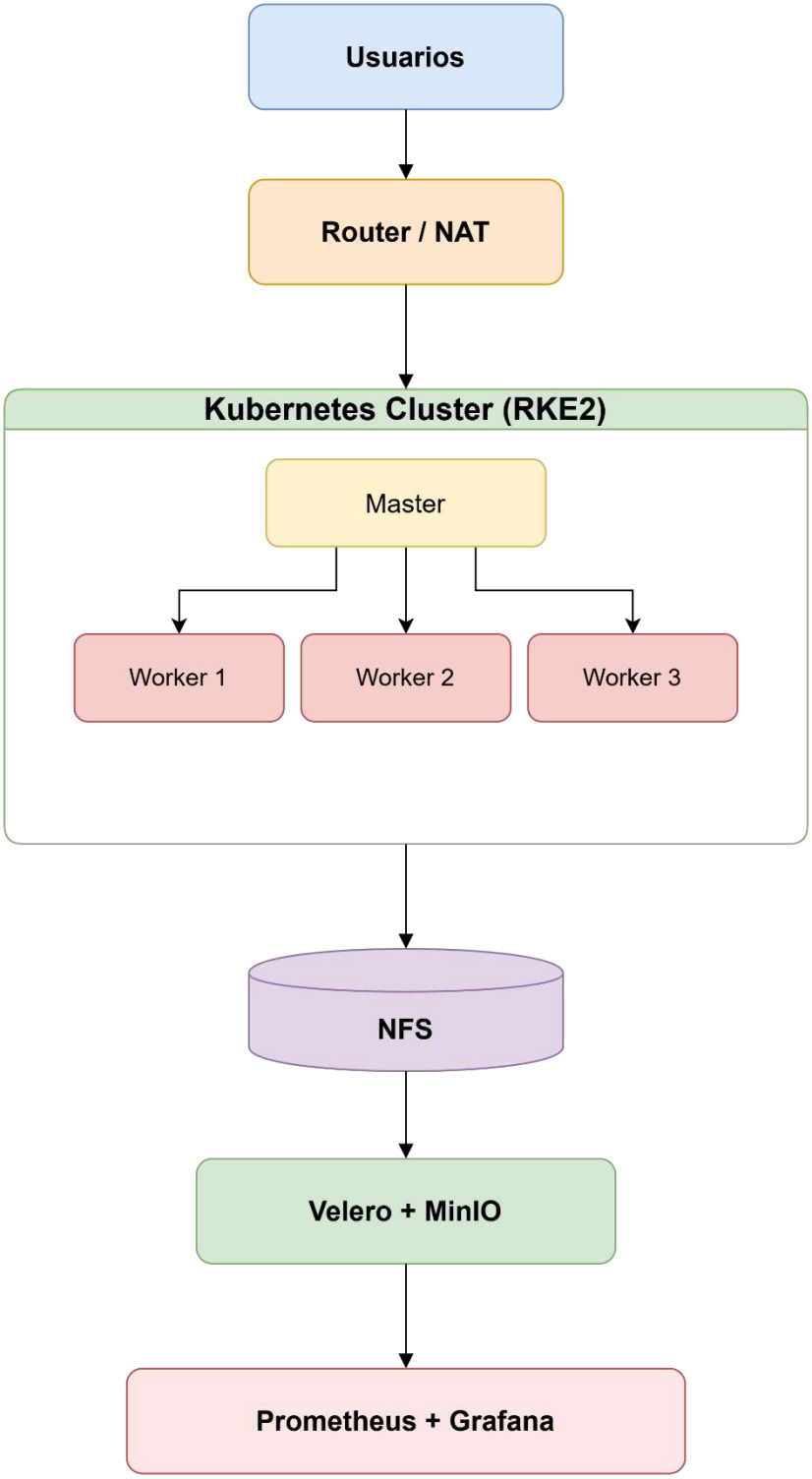
La arquitectura implementada está basada en un clúster Kubernetes desplegado mediante RKE2 (Rancher Kubernetes Engine 2), compuesto por un nodo master y varios nodos worker conectados a través de una red segmentada y gestionada mediante servicios de routing, DHCP y DNS internos. Sobre esta infraestructura se han integrado distintos componentes orientados a garantizar disponibilidad, persistencia de datos, observabilidad y recuperación ante fallos.

Entre las principales tecnologías utilizadas destacan Kubernetes, Docker, RKE2, NFS, Velero, MinIO, Prometheus y Grafana. Docker se empleó para la creación y distribución de imágenes containerizadas, mientras que Kubernetes permitió la orquestación automática de aplicaciones, la gestión de Pods y Deployments, la escalabilidad horizontal y la autorecuperación de servicios. Para garantizar persistencia de datos se implementó un sistema de almacenamiento compartido mediante NFS integrado con Persistent Volumes y Persistent Volume Claims.

Asimismo, se desplegó una plataforma de Disaster Recovery basada en Velero y MinIO, permitiendo realizar copias de seguridad completas del clúster Kubernetes, automatizar backups periódicos y validar procesos reales de restauración ante desastres simulados. Adicionalmente, se incorporó una solución de monitorización y observabilidad utilizando Prometheus y Grafana, facilitando la supervisión de recursos, nodos, Pods y métricas del clúster en tiempo real.

Como resultado final, se obtuvo una infraestructura Kubernetes completamente funcional capaz de desplegar aplicaciones reales a partir de imágenes Docker propias, mantener persistencia de datos, automatizar copias de seguridad, recuperar servicios tras fallos controlados y monitorizar el estado operativo del sistema. El proyecto demuestra la integración práctica de tecnologías DevOps y cloud-native en un entorno académico, proporcionando además una base sólida para futuras ampliaciones orientadas a entornos enterprise, alta disponibilidad avanzada, automatización CI/CD y arquitecturas Kubernetes de producción.

# Diagrama general de arquitectura



## 2. INTRODUCCIÓN TÉCNICA

KubeBackup360 es un proyecto orientado al diseño e implementación de una infraestructura Kubernetes resiliente dentro de un entorno virtualizado. La plataforma desarrollada integra tecnologías de contenerización, almacenamiento persistente, recuperación ante desastres y monitorización, utilizando herramientas ampliamente utilizadas en entornos cloud-native y DevOps.

La infraestructura se ha desplegado sobre máquinas virtuales Ubuntu Server utilizando VirtualBox, implementando un clúster Kubernetes mediante RKE2 compuesto por un nodo master y varios nodos worker. Sobre esta arquitectura se integraron distintos servicios complementarios orientados a garantizar disponibilidad, persistencia de datos y supervisión del entorno.

El proyecto incorpora almacenamiento persistente mediante NFS, un sistema de backups y recuperación basada en Velero y MinIO, así como una plataforma de observabilidad utilizando Prometheus y Grafana. Asimismo, se desarrolló y desplegó una aplicación web propia containerizada mediante Docker e integrada posteriormente dentro del clúster Kubernetes.

El objetivo principal del proyecto consiste en validar el funcionamiento conjunto de distintas tecnologías cloud-native dentro de una infraestructura distribuida, aplicando conceptos relacionados con Kubernetes, automatización, escalabilidad, tolerancia a fallos y recuperación de servicios.

## 2.1 Arquitectura general del proyecto

La infraestructura desarrollada para el proyecto KubeBackup360 se basa en una arquitectura distribuida orientada al despliegue y administración de aplicaciones containerizadas mediante Kubernetes. El entorno fue implementado sobre máquinas virtuales Ubuntu Server utilizando VirtualBox como plataforma de virtualización.

La arquitectura está compuesta por un clúster Kubernetes RKE2 formado por un nodo master y varios nodos worker encargados de ejecutar las aplicaciones desplegadas dentro del entorno. Asimismo, se integraron distintos servicios auxiliares destinados a proporcionar persistencia, backups, recuperación ante desastres y monitorización de la infraestructura.

Para el almacenamiento persistente se implementó un servidor NFS integrado con Kubernetes mediante volúmenes persistentes. El sistema de backups y recuperación se desarrolló utilizando Velero y MinIO, mientras que la monitorización del entorno se realizó mediante Prometheus y Grafana.

La contenerización y distribución de aplicaciones se realizó utilizando Docker y Docker Hub, permitiendo desplegar aplicaciones propias dentro del clúster Kubernetes mediante Deployments y Services.

**Arquitectura general de la infraestructura KubeBackup360**

| Componente | Función principal |
|---|---|
| Router/NAT | Routing y acceso a Internet |
| RKE2 Master | Administración del clúster Kubernetes |
| Workers | Ejecución de Pods y aplicaciones |
| NFS | Persistencia de datos compartida |
| Velero | Backups y restauración Kubernetes |
| MinIO | Almacenamiento compatible con S3 |
| Docker | Contenerización de aplicaciones |
| Docker Hub | Repositorio de imágenes Docker |
| Prometheus | Recolección de métricas |
| Grafana | Visualización y monitorizaci |

## 2.2 Tecnologías utilizadas

La infraestructura KubeBackup360 integra distintas tecnologías orientadas a virtualización, orquestación de contenedores, almacenamiento persistente, recuperación ante desastres y monitorización del entorno Kubernetes.

| Tecnología | Función principal |
|---|---|
| VirtualBox | Virtualización del laboratorio |
| Ubuntu Server | Sistema operativo base |
| RKE2 | Despliegue del clúster Kubernetes |
| Kubernetes | Orquestación de contenedores |
| Docker | Contenerización de aplicaciones |
| Docker Hub | Repositorio de imágenes Docker |
| NFS | Almacenamiento persistente |
| Velero | Backups y restauración Kubernetes |
| MinIO | Almacenamiento S3 para backups |
| Prometheus | Recolección de métricas |
| Grafana | Visualización y monitorización |

# 3. DISEÑO Y PREPARACIÓN DE LA INFRAESTRUCTURA

## 3.1 Preparación del entorno de virtualización

El entorno de laboratorio utilizado para el proyecto KubeBackup360 se desplegó sobre VirtualBox como plataforma principal de virtualización, permitiendo implementar una infraestructura aislada y segmentada compuesta por múltiples máquinas virtuales Ubuntu Server.

Como punto de partida se utilizó una imagen base Ubuntu Server 24.04 en formato OVA, importada mediante la herramienta “Importar servicio virtualizado” de VirtualBox. Esta máquina se utilizó como plantilla inicial para la creación del resto de nodos del laboratorio.

Una vez importada la máquina base, se realizó una actualización inicial del sistema operativo:

**sudo apt update**
**sudo apt upgrade -y**

*Posteriormente, la máquina fue apagada y utilizada como base para la clonación del resto de sistemas virtuales.*

### 3.2 Clonación y organización de máquinas virtuales

A partir de la máquina base se generaron múltiples máquinas virtuales mediante clonación completa (Full Clone), permitiendo disponer de nodos totalmente independientes dentro del entorno de laboratorio. Durante el proceso de clonación se reinicializaron las direcciones MAC de las interfaces de red para evitar conflictos de direccionamiento IP.

Las máquinas virtuales creadas fueron las siguientes:

| Nodo | Función |
|---|---|
| kb360-router | Router, DHCP y DNS |
| kb360-master | Nodo master Kubernetes |
| kb360-w1 | Nodo worker 1 |
| kb360-w2 | Nodo worker 2 |
| kb360-w3 | Nodo worker 3 |
| kb360-nfs | Servidor NFS |
| kb360-client | Cliente de pruebas |


La nomenclatura utilizada permitió mantener una estructura homogénea y fácilmente identificable dentro del laboratorio.

### 3.3 Configuración hardware

Una vez creadas las máquinas virtuales, se asignaron recursos hardware en función del rol de cada nodo dentro de la infraestructura. Esta distribución permitió optimizar el consumo de recursos del sistema anfitrión manteniendo estabilidad en el entorno Kubernetes.

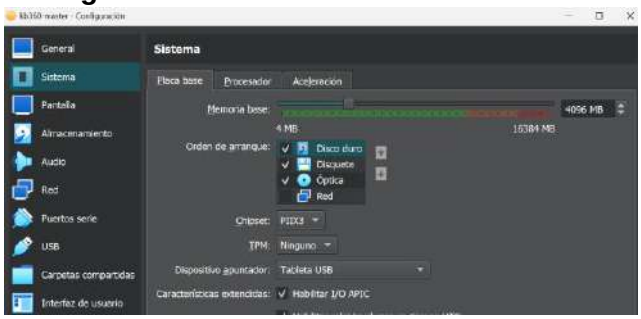
| Máquina | RAM | CPU | Disco |
|---|---|---|---|
| kb360-router | 1 GB | 1 | 20 GB |
| kb360-master | 4 GB | 2 | 20 GB |
| kb360-w1 / w2 / w3 | 2 GB | 1 | 20 GB |
| kb360-nfs | 2 GB | 1 | 40 GB |

Con el objetivo de disponer de mayor capacidad para backups y almacenamiento persistente, el disco del servidor NFS fue ampliado de 20 GB a 40 GB.

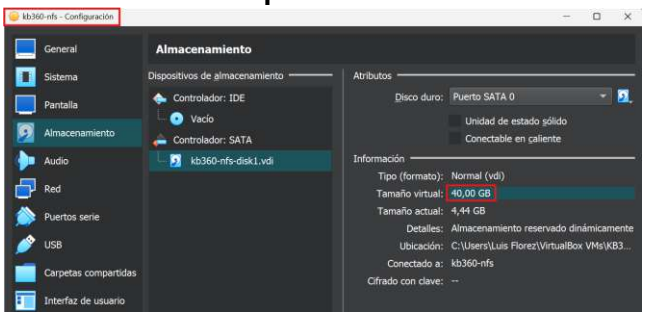
Comandos utilizados para ampliar la partición:

**lsblk**
**sudo growpart /dev/sda 1**
**sudo resize2fs /dev/sda1**
**df -h**

**Configuración hardware del nodo master**




**Verificación de ampliación de almacenamiento en el servidor NFS**




### 3.4 Diseño de red y segmentación

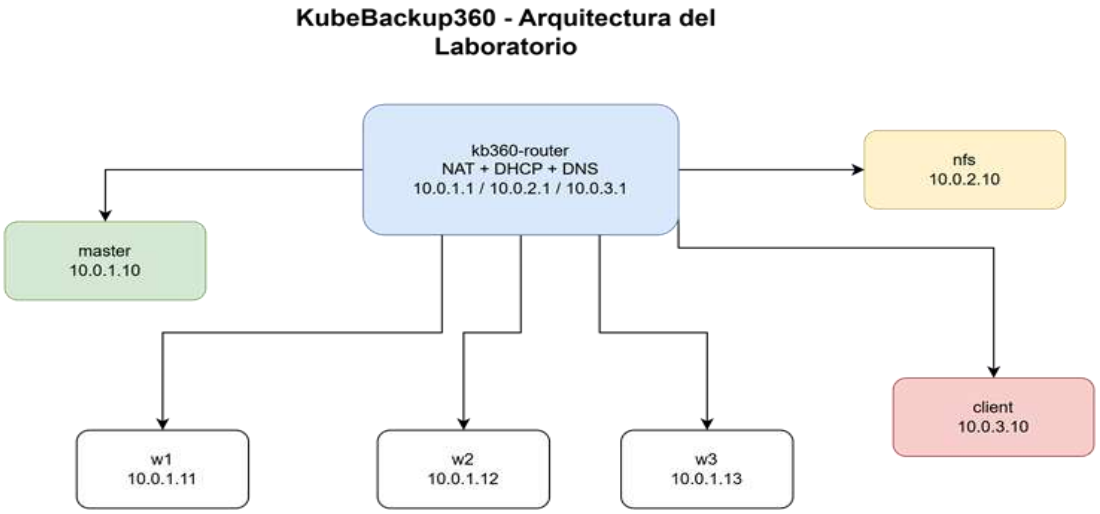
La infraestructura de red del laboratorio se diseñó utilizando redes internas segmentadas en VirtualBox, permitiendo aislar el tráfico y simular una arquitectura empresarial distribuida.

Las redes implementadas fueron las siguientes:

| Red | Subred | Función |
|---|---|---|
| intnet-cluster | 10.0.1.0/24 | Clúster Kubernetes |
| intnet-nfs | 10.0.2.0/24 | Almacenamiento NFS |
| intnet-client | 10.0.3.0/24 | Cliente de acceso |

El nodo kb360-router se configuró como elemento central de red, proporcionando acceso a Internet mediante NAT, routing entre subredes y servicios DHCP/DNS.

**Diagrama de segmentación de red del laboratorio KubeBackup360**




### 3.5 Configuración hostname y Netplan

Se configuró un hostname específico para cada nodo utilizando hostnamectl, facilitando la identificación y administración de la infraestructura.

**sudo hostnamectl set-hostname kb360-master**

**hostname**

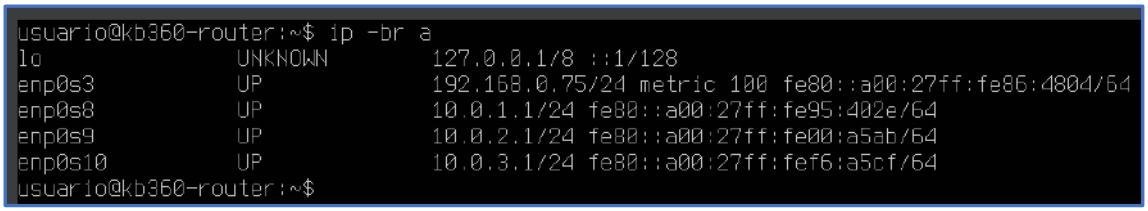
Asimismo, se actualizó el archivo /etc/hosts en todas las máquinas del laboratorio para permitir resolución local de nombres.

Posteriormente, se configuraron las interfaces de red mediante Netplan utilizando direcciones IP estáticas durante la fase inicial de despliegue.

Archivo de configuración:

**sudo nano /etc/netplan/50-installer-config.yaml**

**Configuración Netplan aplicada correctamente**




### 3.6 Routing y acceso a Internet

El nodo kb360-router fue configurado para proporcionar routing entre subredes y acceso a Internet mediante NAT utilizando iptables.

Se habilitó el reenvío de paquetes IP:

**echo 1 > /proc/sys/net/ipv4/ip_forward**

**sudo sysctl -p**

Posteriormente, se configuró NAT:

**sudo iptables -t nat -A POSTROUTING -o enp0s3 -j MASQUERADE**

*Finalmente, se definieron reglas de forwarding para permitir la comunicación entre redes internas y acceso a Internet.*

### 3.7 Configuración DHCP y DNS

Se implementó un servidor DHCP mediante KEA DHCP en el nodo kb360-router con el objetivo de automatizar la asignación de direcciones IP dentro del laboratorio.

El servicio fue configurado utilizando reservas DHCP basadas en direcciones MAC para mantener IPs consistentes en todos los nodos del entorno.

Tabla de reservas:

| Nodo | Dirección IP |
|---|---|
| kb360-master | 10.0.1.10 |
| kb360-w1 | 10.0.1.11 |
| kb360-w2 | 10.0.1.12 |
| kb360-w3 | 10.0.1.13 |
| kb360-nfs | 10.0.2.10 |
| kb360-client | 10.0.3.10 |

Asimismo, se implementó un servidor DNS mediante Bind9 integrado con KEA DHCP, permitiendo la resolución de nombres dentro del dominio interno kb360.lan.

#### 3.7.1 Implementación del servicio DHCP con KEA

El servicio DHCP fue implementado mediante KEA DHCP sobre el nodo kb360-router, permitiendo automatizar la asignación de direcciones IP dentro de las distintas subredes del laboratorio.

El fichero principal de configuración utilizado fue:

**sudo nano /etc/kea/kea-dhcp4.conf**


La configuración distribuye automáticamente:

- gateway,
- DNS,
- dominio interno,
- y reservas IP asociadas a cada nodo.

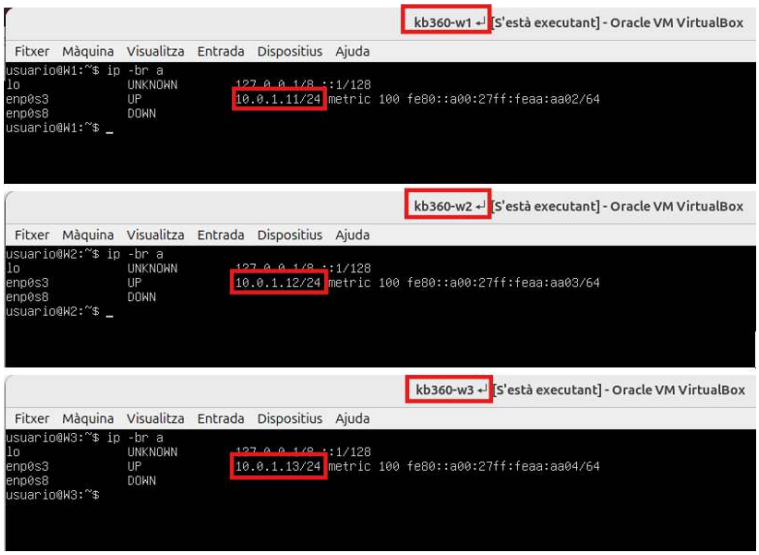
Una vez completada la configuración, se validó la sintaxis del fichero:

**sudo kea-dhcp4 -t /etc/kea/kea-dhcp4.conf**

Posteriormente, el servicio fue reiniciado y habilitado:

**sudo systemctl restart kea-dhcp4-server**
**sudo systemctl enable kea-dhcp4-server**

Las pruebas realizadas confirmaron la correcta asignación automática de direcciones IP en todas las subredes del laboratorio.

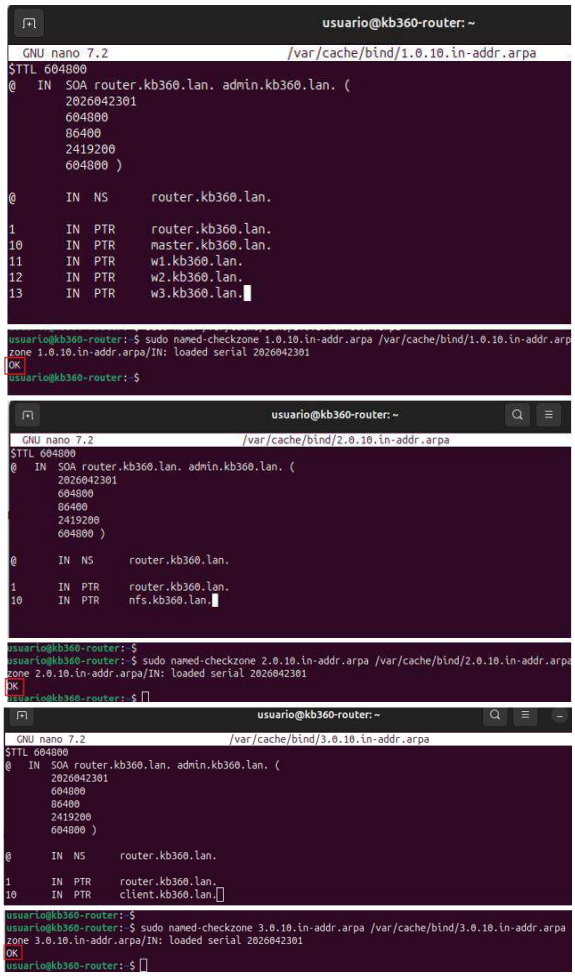


### 3.7.2 Implementación del servicio DNS con Bind9

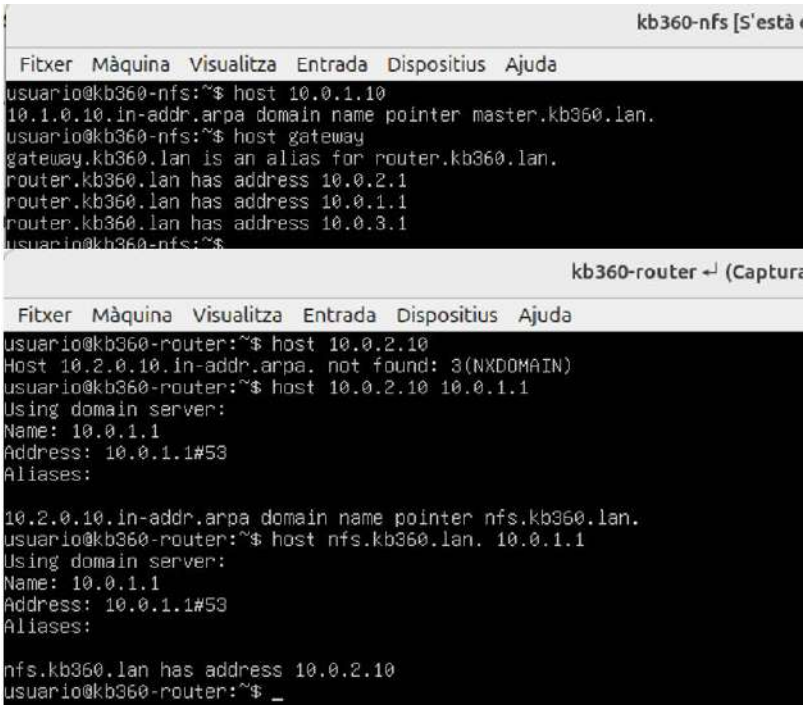
El servicio DNS se implementó mediante Bind9 integrado en el nodo kb360-router, proporcionando resolución de nombres dentro del dominio interno kb360.lan.

Se configuraron:

- zona directa,
- zonas inversas,
- resolución interna,
- forwarders externos.




## Resolución DNS mediante Bind9



### 3.8 Resultado de la fase de infraestructura

La fase de preparación de infraestructura permitió disponer de un entorno virtualizado completamente funcional, segmentado y preparado para el despliegue del clúster Kubernetes. La arquitectura implementada proporciona conectividad entre subredes, acceso a Internet, automatización de direccionamiento IP y resolución DNS interna, estableciendo una base sólida para las siguientes fases del proyecto.

## 4. IMPLEMENTACIÓN DEL CLÚSTER KUBERNETES

### 4.1 Preparación de nodos

Una vez finalizada la fase de infraestructura y validada la conectividad entre todos los sistemas del laboratorio, se procedió a la preparación de los nodos destinados al despliegue del clúster Kubernetes. **Esta fase tuvo como objetivo garantizar que todos los sistemas cumplieran los requisitos necesarios para la instalación y funcionamiento de RKE2.**

**Antes de iniciar la instalación del clúster, se realizaron comprobaciones básicas de red y resolución DNS en todos los nodos del entorno, verificando:**

- **Conectividad entre máquinas,**
- **Acceso a Internet,**
- **Resolución de nombres**
- **Comunicación con el nodo router.**

Comandos de validación utilizados:
**hostname**
**ip a**
**ping -c 3 10.0.1.1**
**ping -c 3 kb360-master**
**ping -c 3 8.8.8.8**
**host kb360-master**

Las pruebas confirmaron el correcto funcionamiento de la infraestructura de red desplegada durante la fase anterior.

Asimismo, antes de instalar Kubernetes, se configuró el servicio de resolución local mediante systemd-resolved, estableciendo el dominio interno del laboratorio y el servidor DNS correspondiente:

**sudo nano /etc/systemd/resolved.conf**

Configuración aplicada:

**DNS=127.0.0.1**

**Domains=kb360.lan**

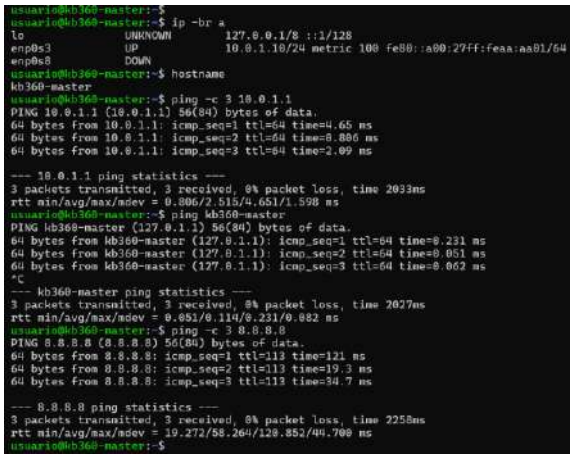
Posteriormente, se reinició el servicio:

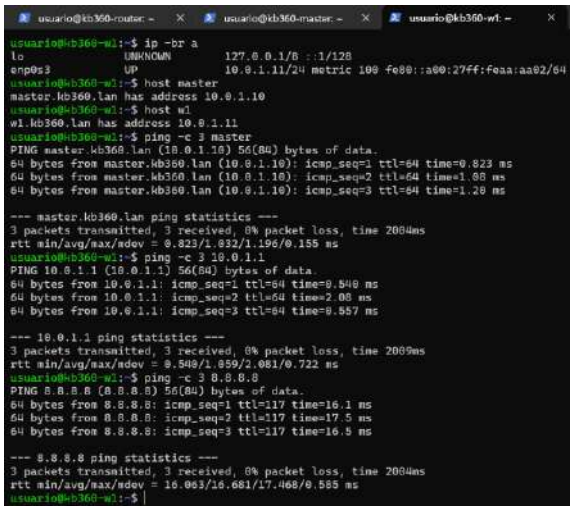
**sudo systemctl restart systemd-resolved**

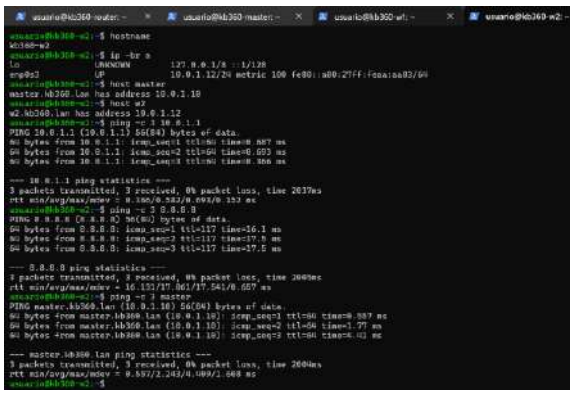
**resolvectl status**

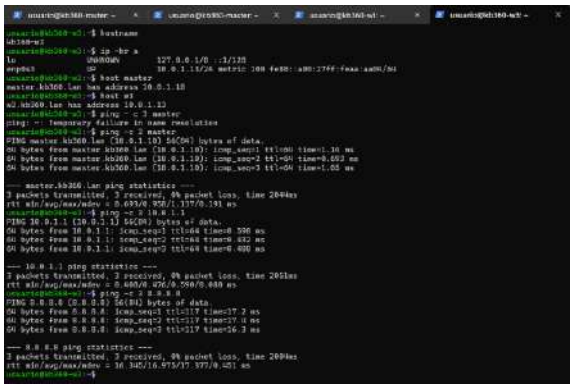
## Validación de conectividad y resolución DNS entre nodos

## Comprobación de los Workers en la red1 10.0.1.0

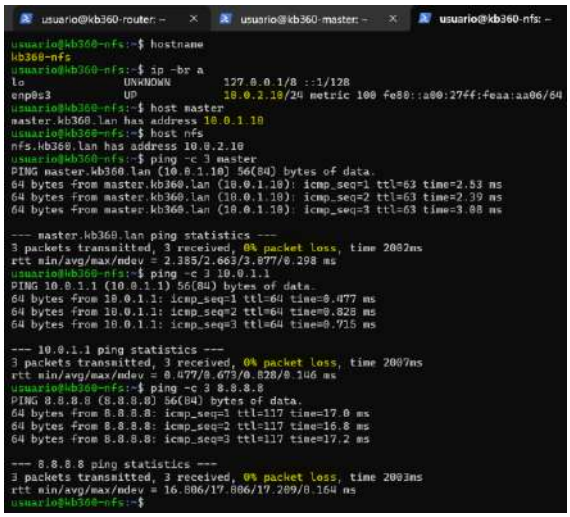


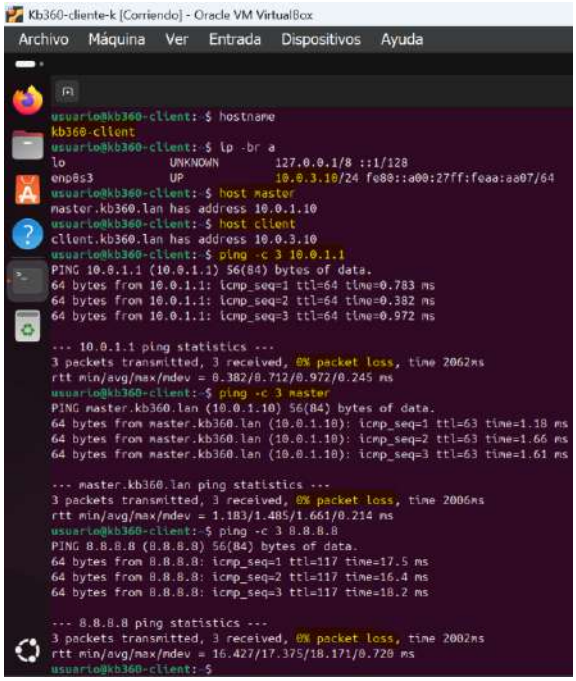





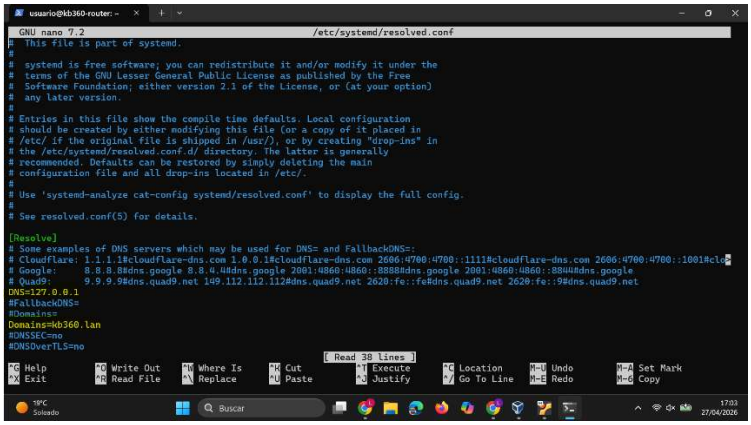
## Comprobación del nfs en la red2 10.0.2.0




## Comprobación al client en la red3 10.0.3.0



**Configuración de systemd-resolved y dominio kb360.lan**




**Desactivación de SWAP**

Como requisito obligatorio para el funcionamiento de Kubernetes, se procedió a desactivar la memoria swap en todos los nodos del clúster. Kubernetes requiere control directo de la memoria del sistema, por lo que el uso de swap puede provocar comportamientos inestables en la gestión de recursos y contenedores.

La desactivación se realizó inicialmente de forma temporal:

**sudo swapoff -a**

Posteriormente, se modificó el archivo /etc/fstab para desactivar swap de forma permanente:

**sudo nano /etc/fstab**

Línea comentada:

**# /swap.img none swap sw 0 0**

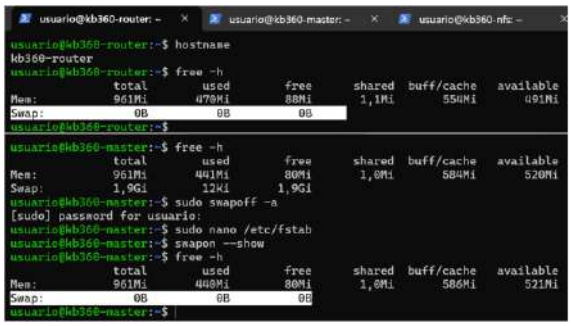
Finalmente, se verificó que el sistema ya no utilizaba memoria swap:

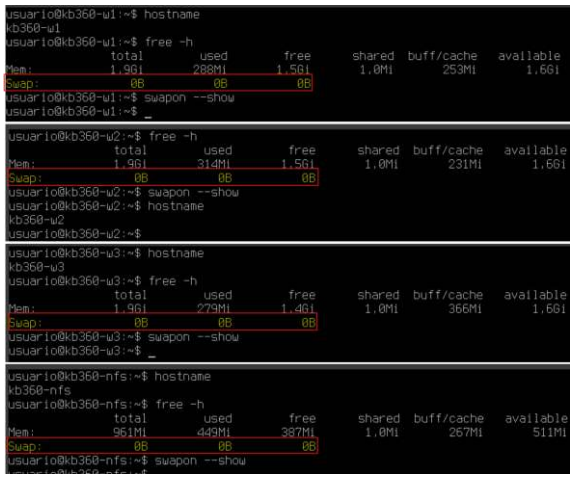
**free -h**

**swapon --show**

Las validaciones realizadas confirmaron que todos los nodos del entorno se encontraban correctamente preparados para la instalación del clúster Kubernetes.

**Desactivación de memoria SWAP en los nodos del clúster**






**Importancia de la infraestructura de red y conectividad del entorno**

*La correcta configuración del entorno de trabajo constituye un aspecto fundamental dentro del proyecto KubeBackup360. La conectividad entre nodos, la resolución DNS interna y el acceso controlado a Internet son elementos esenciales para garantizar el correcto funcionamiento del clúster Kubernetes y de los servicios desplegados.*

*La arquitectura implementada, basada en redes segmentadas y un nodo router encargado de proporcionar routing, NAT, DHCP y DNS, permite disponer de un entorno estable y controlado sobre el que desplegar la infraestructura Kubernetes del laboratorio.*

***Esta base de red robusta resulta imprescindible para asegurar la comunicación entre nodos, la integración de servicios y el correcto funcionamiento general del entorno cloud-native implementado.***

### 4.2 Instalación del nodo master

Una vez preparados los nodos del entorno, se procedió a la instalación del plano de control Kubernetes sobre el nodo kb360-master utilizando RKE2 (Rancher Kubernetes Engine 2). Esta instalación incluye los componentes principales del clúster, como API Server, Scheduler, Controller Manager, etcd y kubelet.

La instalación del servicio RKE2 Server se realizó mediante el siguiente comando:

**curl -sfL https://get.rke2.io | sudo sh -**

Una vez finalizada la instalación, el servicio fue habilitado y arrancado:

**sudo systemctl enable rke2-server.service**

**sudo systemctl start rke2-server.service**

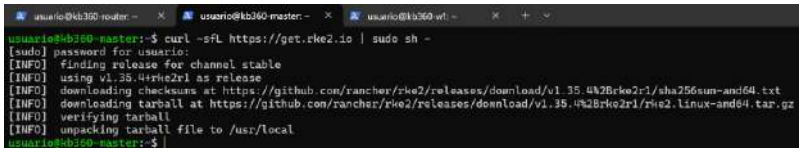
Posteriormente, se verificó el estado del servicio:

**sudo systemctl status rke2-server.service**

*El resultado confirmó que el servicio se encontraba en estado active (**running**).*

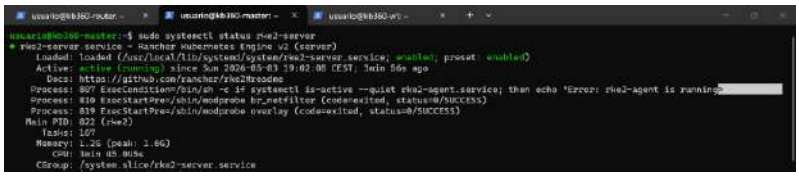
**Servicio rke2-server en ejecución**

Instalación del servicio RKE2 Server




Activar servicio


Ver estado




**Configuración de kubectl**

Tras la instalación del nodo master, se configuró el entorno de administración del clúster mediante la herramienta kubectl. Para ello, se copió el archivo kubeconfig generado automáticamente por RKE2 al directorio del usuario:

**mkdir -p ~/.kube**

**sudo cp /etc/rancher/rke2/rke2.yaml ~/.kube/config**

**sudo chown $USER:$USER ~/.kube/config**

Asimismo, se añadió el binario de RKE2 al PATH del sistema:

**export PATH=$PATH:/var/lib/rancher/rke2/bin**

**echo 'export PATH=$PATH:/var/lib/rancher/rke2/bin' >> ~/.bashrc**

**source ~/.bashrc**

Finalmente, se validó el funcionamiento inicial del clúster:

**kubectl get nodes**

Resultado esperado:

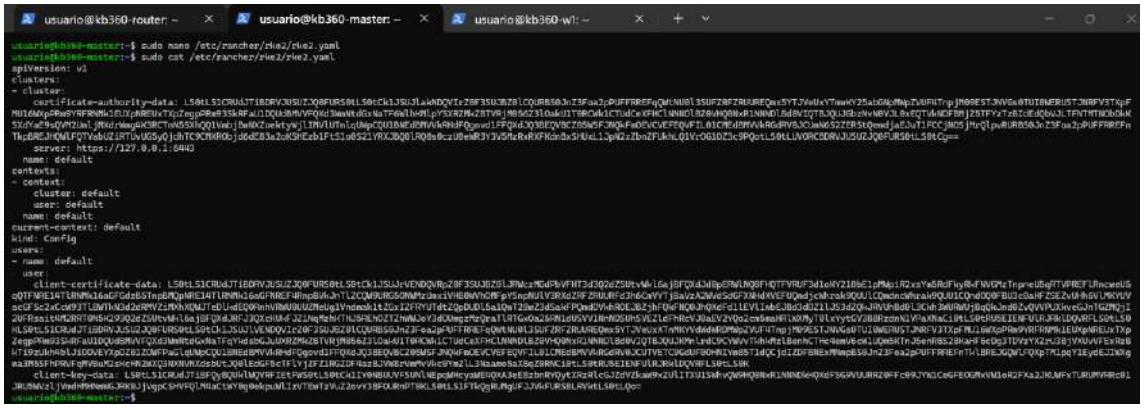
**kb360-master Ready**

También se verificó el despliegue correcto de los componentes internos del sistema:

**kubectl get pods -A**

Las comprobaciones confirmaron que el plano de control Kubernetes se encontraba completamente operativo y preparado para la incorporación de nodos worker.

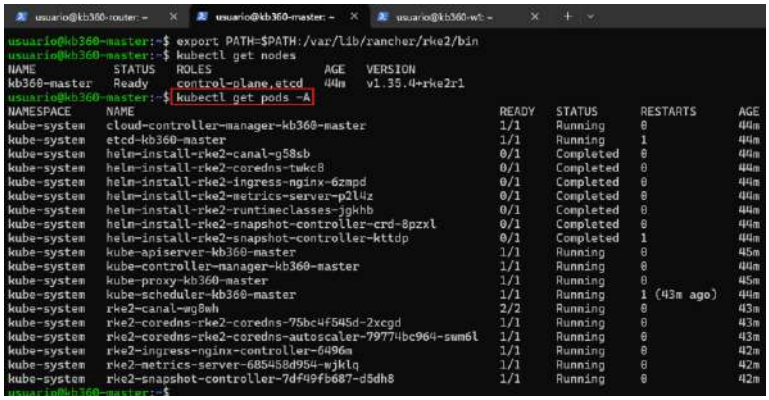
**Archivo kubeconfig generado automáticamente por RKE2**




**Configurar Kubectl (Usuario)**


**Validación final kubectl get pods -A**

Pods del sistema Kubernetes desplegados correctamente




- Kubernetes desplegado
- Plano de control activo
- Red del clúster operativa
- Servicios internos funcionando

### 4.3 Incorporación de nodos worker

Una vez desplegado y validado el nodo master del clúster Kubernetes, se procedió a la incorporación de los nodos worker mediante la instalación del componente rke2-agent en cada uno de ellos. Los nodos añadidos al clúster fueron:

- kb360-w1
- kb360-w2
- kb360-w3

La integración de los workers se realizó utilizando el token generado automáticamente por el nodo master, permitiendo la autenticación segura y el registro de los nodos dentro del clúster Kubernetes.

El token de unión fue obtenido desde el nodo master mediante el siguiente comando:

**sudo cat /var/lib/rancher/rke2/server/node-token**

Posteriormente, en cada nodo worker se instaló el agente RKE2:

**curl -sfL https://get.rke2.io | INSTALL_RKE2_TYPE="agent" sudo sh -**

Una vez instalado el agente, se creó el directorio de configuración:

**sudo mkdir -p /etc/rancher/rke2**

A continuación, se generó el fichero de configuración utilizado para conectar el worker con el nodo master:

**sudo nano /etc/rancher/rke2/config.yaml**

Configuración utilizada:

**server: https://10.0.1.10:9345**

token: TOKEN_DEL_CLUSTER

Finalmente, el servicio fue habilitado y arrancado:

**sudo systemctl enable rke2-agent**

**sudo systemctl start rke2-agent**

Tras unos minutos de inicialización y sincronización interna, los nodos worker fueron registrados correctamente dentro del clúster Kubernetes.

La validación se realizó desde el nodo master utilizando:

**kubectl get nodes**

Resultado esperado:

kb360-master Ready

kb360-w1 Ready

kb360-w2 Ready

kb360-w3 Ready

Durante la fase inicial de arranque, algunos nodos aparecieron temporalmente en estado NotReady, comportamiento habitual mientras Kubernetes completa la inicialización de componentes internos y networking del clúster.

Asimismo, se verificó la correcta integración de las interfaces de red Kubernetes mediante el comando:

**ip -br a**

Las comprobaciones confirmaron el funcionamiento del plugin de red del clúster y la creación de interfaces virtuales asociadas a Kubernetes y contenedores desplegados.

### Obtención del token de unión del clúster

```
usuario@kb360-master:~$ sudo cat /var/lib/rancher/rke2/server/node-token
K10e94677aab7bfab45c5921f34efdeb79a0d6461da5952287b404cd12c70ddd9fa::server:4c5c7b456590a32c5abd7ad703722e94
usuario@kb360-master:~$
```

### Instalar Rkee Agent en todos los Workers

```
usuario@kb360-w1:~$ hostname
kb360-w1
usuario@kb360-w1:~$ curl -sfL https://get.rke2.io | INSTALL_RKE2_TYPE="agent" sudo sh -
[sudo] password for usuario:
[INFO]  finding release for channel stable
[INFO]  using v1.35.4+rke2r1 as release
[INFO]  downloading checksums at https://github.com/rancher/rke2/releases/download/v1.35.4%2Brke2r1/sha256sum-amd64.txt
[INFO]  downloading tarball at https://github.com/rancher/rke2/releases/download/v1.35.4%2Brke2r1/rke2.linux-amd64.tar.gz
[INFO]  verifying tarball
[INFO]  unpacking tarball file to /usr/local
usuario@kb360-w1:~$
```

```
usuario@kb360-router: ~ | usuario@kb360-master: ~ | usuario@kb360-w2: ~ | usuario@kb360-w3: ~
usuario@kb360-w2:~$ hostname
kb360-w2
usuario@kb360-w2:~$ curl -sfL https://get.rke2.io | INSTALL_RKE2_TYPE="agent" sudo sh -
[sudo] password for usuario:
[INFO]  finding release for channel stable
[INFO]  using v1.35.4+rke2r1 as release
[INFO]  downloading checksums at https://github.com/rancher/rke2/releases/download/v1.35.4%2Brke2r1/sha256sum-amd64.txt
[INFO]  downloading tarball at https://github.com/rancher/rke2/releases/download/v1.35.4%2Brke2r1/rke2.linux-amd64.tar.gz
[INFO]  verifying tarball
[INFO]  unpacking tarball file to /usr/local
usuario@kb360-w2:~$
```

```
usuario@kb360-w3:~$ hostname
kb360-w3
usuario@kb360-w3:~$ curl -sfL https://get.rke2.io | INSTALL_RKE2_TYPE="agent" sudo sh -
[sudo] password for usuario:
[INFO]  finding release for channel stable
[INFO]  using v1.35.4+rke2r1 as release
[INFO]  downloading checksums at https://github.com/rancher/rke2/releases/download/v1.35.4%2Brke2r1/sha256sum-amd64.txt
[INFO]  downloading tarball at https://github.com/rancher/rke2/releases/download/v1.35.4%2Brke2r1/rke2.linux-amd64.tar.gz
[INFO]  verifying tarball
[INFO]  unpacking tarball file to /usr/local
```

### Configuración del fichero config.yaml en nodo worker

```
usuario@kb360-w1:~$ hostname
kb360-w1
usuario@kb360-w1:~$ sudo cat /etc/rancher/rke2/config.yaml
server: https://10.0.1.10:9345
token: K10781ba2d5da51f1945a9a1573d9c60a36879755c6b7e555ef442e59f56fca3983::server:45619a99a17a6fa0bae2ff0209033ac1
```

server: https://10.0.1.10:9345
token: TOKEN DEL CLUSTER

### Arrancar el servicio en cada uno de los agentes w1, w2, w3.

```
usuario@kb360-w1:~$ sudo systemctl enable rke2-agent
Created symlink /etc/systemd/system/multi-user.target.wants/rke2-agent.service → /usr/local/lib/systemd/system/rke2-agent.service.
usuario@kb360-w1:~$ sudo systemctl start rke2-agent
```

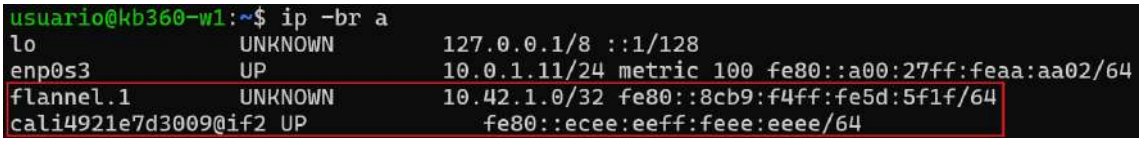
```
usuario@kb360-w2:~$ sudo systemctl enable rke2-agent
Created symlink /etc/systemd/system/multi-user.target.wants/rke2-agent.service → /usr/local/lib/systemd/system/rke2-agent.service.
usuario@kb360-w2:~$ sudo systemctl start rke2-agent
usuario@kb360-w2:~$
```

```
usuario@kb360-w3:~$ sudo systemctl enable rke2-agent
Created symlink /etc/systemd/system/multi-user.target.wants/rke2-agent.service → /usr/local/lib/systemd/system/rke2-agent.service.
usuario@kb360-w3:~$ sudo systemctl start rke2-agent
usuario@kb360-w3:~$
```

### Validación de nodos worker en estado Ready

```
usuario@kb360-master:~$ hostname
kb360-master
usuario@kb360-master:~$ kubectl get nodes
NAME           STATUS   ROLES                  AGE    VERSION
kb360-master   Ready    control-plane,etcd     138m   v1.35.4+rke2r1
kb360-w1       Ready    <none>                 61m    v1.35.4+rke2r1
kb360-w2       Ready    <none>                 34m    v1.35.4+rke2r1
kb360-w3       Ready    <none>                 25m    v1.35.4+rke2r1
```

Interfaces de red Kubernetes en nodo worker




**Resultado de la incorporación de nodos worker**

La incorporación de los nodos worker permitió completar la infraestructura distribuida del clúster Kubernetes, proporcionando capacidad de ejecución de cargas de trabajo y escalabilidad dentro del entorno.

Las validaciones realizadas confirmaron la correcta integración de los nodos, el funcionamiento del networking Kubernetes y la disponibilidad operativa del clúster para el despliegue de aplicaciones containerizadas.

### 4.4 Validación del clúster Kubernetes

Una vez incorporados los nodos worker, se realizaron distintas comprobaciones para validar el correcto funcionamiento del clúster Kubernetes y verificar la operatividad de los componentes desplegados por RKE2.

Las validaciones permitieron comprobar el estado general del clúster, la comunicación entre nodos y el despliegue correcto de los servicios internos del sistema.

La primera validación se realizó mediante:

**kubectl get nodes**

El resultado confirmó que todos los nodos del clúster se encontraban en estado Ready:

NAME STATUS ROLES AGE

kb360-master Ready control-plane,etcd,master

kb360-w1 Ready <none>

kb360-w2 Ready <none>

kb360-w3 Ready <none>

Posteriormente, se verificó el despliegue de los Pods internos del sistema Kubernetes:

**kubectl get pods -A**

Entre los componentes desplegados se validaron:

- CoreDNS
- Metrics Server
- Canal/Flannel
- Ingress Controller
- Helm Controller
- kube-proxy

La correcta ejecución de estos servicios confirmó el funcionamiento interno del clúster y de la infraestructura de networking Kubernetes.

Asimismo, se verificó el estado detallado de los nodos utilizando:

kubectl describe nodes

Las comprobaciones permitieron validar:

- Capacidad de CPU y memoria,
- Estado de kubelet,
- Networking,
- Direcciones IP,
- Recursos disponibles en cada nodo.

Con el objetivo de comprobar el funcionamiento del scheduler Kubernetes, se validó la distribución automática de Pods entre los distintos workers del clúster.

Finalmente, se realizaron comprobaciones de conectividad interna y funcionamiento de la red overlay desplegada automáticamente por RKE2.

**Estado general de los nodos del clúster Kubernetes**

```
kb360-master   Ready    control-plane,etcd   9d    v1.35.4+rke2r1
kb360-w1       Ready    <none>               9d    v1.35.4+rke2r1
kb360-w2       Ready    <none>               9d    v1.35.4+rke2r1
kb360-w3       Ready    <none>               9d    v1.35.4+rke2r1
```

**Información detallada de los nodos Kubernetes**

```
usuario@kb360-master:~$ kubectl describe nodes
Name:               kb360-master
Roles:              control-plane,etcd
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/instance-type=rke2
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=kb360-master
                    kubernetes.io/os=linux
                    node-role.kubernetes.io/control-plane=true
                    node-role.kubernetes.io/etcd=true
                    node.kubernetes.io/instance-type=rke2
Annotations:        alpha.kubernetes.io/provided-node-ip: 10.0.1.10
                    etcd.rke2.cattle.io/local-snapshots-timestamp: 2026-05-13T15:15:13+02:00
                    etcd.rke2.cattle.io/node-address: 10.0.1.10
                    etcd.rke2.cattle.io/node-name: kb360-master-b37d1cfe
                    flannel.alpha.coreos.com/backend-data: {"VNI":1,"VtepMAC":"86:2a:88:0e:18:f3"}
                    flannel.alpha.coreos.com/backend-type: vxlan
                    flannel.alpha.coreos.com/kube-subnet-manager: true
                    flannel.alpha.coreos.com/public-ip: 10.0.1.10
                    node.alpha.kubernetes.io/ttl: 0
                    rke2.io/encryption-config-hash: start-facf3720e3603955857344c63bc9e79f37cd29891487bf43c79485300ee2a8a1
                    rke2.io/hostname: kb360-master
                    rke2.io/internal-ip: 10.0.1.10
                    rke2.io/node-args: ["server"]
                    rke2.io/node-config-hash: MLFMUCBMRVINLJJKSG32TOUFWB4CN55GMSNY25AZPESQXZCYRN2A====
                    rke2.io/node-env: {}
                    volumes.kubernetes.io/controller-managed-attach-detach: true
CreationTimestamp:  Mon, 04 May 2026 11:33:17 +0200
Taints:             <none>
Unschedulable:      false
Lease:
  HolderIdentity:  kb360-master
```

```
usuario@kb360-master:~$ kubectl get pods -A | grep -E "coredns|metrics-server"
kube-system   helm-install-rke2-coredns-kksrj                    0/1     Completed   0               9d
kube-system   helm-install-rke2-metrics-server-bnsbn             0/1     Completed   0               9d
kube-system   rke2-coredns-rke2-coredns-75bc4f545d-794d9         1/1     Running     1 (91m ago)     9d
kube-system   rke2-coredns-rke2-coredns-75bc4f545d-dqkxd         1/1     Running     0               87m
kube-system   rke2-coredns-rke2-coredns-autoscaler-79774bc964-8fhdp   1/1     Running     1 (91m ago)     9d
kube-system   rke2-metrics-server-685458d954-l7jcz               1/1     Running     1 (91m ago)     9d
usuario@kb360-master:~$
```

**Resultado de la validación del clúster**

Las validaciones realizadas confirmaron el correcto funcionamiento del clúster Kubernetes desplegado mediante RKE2, verificando la comunicación entre nodos, la operatividad del scheduler y el despliegue correcto de los servicios internos del sistema.

El entorno quedó preparado para el despliegue de aplicaciones containerizadas y la ejecución de pruebas funcionales sobre la infraestructura Kubernetes del laboratorio.

**4.5 Primera prueba funcional Kubernetes**

Una vez validado el funcionamiento general del clúster Kubernetes, se realizó una primera prueba funcional mediante el despliegue de un contenedor NGINX. Esta prueba permitió verificar el correcto funcionamiento del scheduler Kubernetes, la ejecución de Pods sobre los nodos worker y la conectividad interna del clúster.

El despliegue inicial se realizó utilizando el siguiente comando:

**kubectl run nginx-test --image=nginx**

Tras la creación del Pod, se validó su estado y el nodo asignado automáticamente por Kubernetes:

**kubectl get pods -o wide**

La salida permitió comprobar:

- Estado Running,
- Dirección IP interna del Pod,
- Nodo worker donde fue desplegado.

Posteriormente, el Pod fue expuesto mediante un Service de tipo NodePort, permitiendo el acceso externo a la aplicación desde la red del laboratorio:

**kubectl expose pod nginx-test **
**--type=NodePort **
**--port=80**

Una vez creado el servicio, se verificó la asignación automática del puerto NodePort:

**kubectl get svc**

La validación funcional se completó accediendo al servicio NGINX desde un navegador web y mediante herramientas de consola como curl, confirmando el correcto funcionamiento del networking Kubernetes y la exposición del servicio.

Comando utilizado:

**curl http://10.0.1.X:PUERTO_NODEPORT**

Asimismo, se realizaron comprobaciones detalladas del Pod desplegado:

kubectl describe pod nginx-test

Las pruebas realizadas confirmaron:

- funcionamiento del scheduler,
- ejecución correcta del Pod,

- conectividad entre nodos,
- networking Kubernetes operativo,
- y acceso externo mediante Services NodePort.

kubectl get pods -o wide

**Despliegue del Pod NGINX en el clúster Kubernetes**

```
usuario@kb360-master:~$ kubectl run nginx-test --image=nginx
pod/nginx-test created
usuario@kb360-master:~$
```

```
usuario@kb360-master:~$ kubectl get pods -o wide
NAME         READY   STATUS    RESTARTS   AGE     IP          NODE      NOMINATED NODE   READINESS GATES
nginx-test   1/1     Running   0          7m51s   10.42.3.5   kb360-w3  <none>           <none>
usuario@kb360-master:~$
```

**Exposición del servicio mediante NodePort**

```
usuario@kb360-master:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.43.0.1       <none>        443/TCP        74m
nginx-test   NodePort    10.43.205.91    <none>        80:31741/TCP   2m11s
```

```
usuario@kb360-master:~$ kubectl expose pod nginx-test --port=80 --type=NodePort
service/nginx-test exposed
usuario@kb360-master:~$
```

```
usuario@kb360-router:~$ hostname
kb360-router
usuario@kb360-router:~$ curl http://10.0.1.13:31741
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```

**Información detallada del Pod nginx-test**

```
usuario@kb360-master:~$ kubectl describe pod nginx-test
Name:             nginx-test-6ff8854996-kzvpb
Namespace:        default
Priority:         0
Service Account:  default
Node:             kb360-w1/10.0.1.11
```

**Resultado de la prueba funcional**

La prueba funcional confirmó el correcto funcionamiento del clúster Kubernetes desplegado mediante RKE2, validando el despliegue de Pods, la asignación automática de nodos mediante el scheduler y el acceso externo a aplicaciones mediante servicios NodePort.

El entorno quedó preparado para continuar con pruebas de alta disponibilidad, autorecuperación y despliegue de aplicaciones más complejas dentro de la infraestructura Kubernetes del proyecto.

### 4.6 Simulación básica de fallo y autorecuperación

Con el objetivo de validar las capacidades de tolerancia a fallos y autorecuperación de Kubernetes, se realizaron distintas pruebas de disponibilidad sobre el clúster desplegado. Estas validaciones permitieron comprobar el comportamiento del scheduler y de los controladores Kubernetes ante la caída de nodos y servicios.

Inicialmente, se desplegó un Pod simple utilizando NGINX:

**kubectl run nginx-test --image=nginx**

Posteriormente, se verificó el nodo worker donde el Pod había sido desplegado:

**kubectl get pods -o wide**

Una vez identificado el nodo asignado, se simuló un fallo apagando el worker correspondiente desde VirtualBox.

Tras la caída del nodo, Kubernetes detectó automáticamente el estado NotReady del worker afectado:

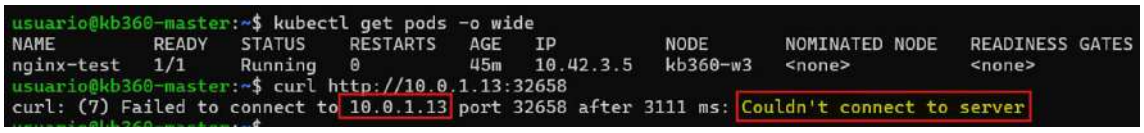
**kubectl get nodes**

*Sin embargo, al tratarse de un Pod simple creado manualmente,* ***Kubernetes no recreó automáticamente la carga de trabajo en otro nodo del clúster****. Esta prueba permitió comprobar que los Pods independientes no disponen de mecanismos de autorecuperación si no están gestionados mediante controladores Kubernetes*.

**Pod desplegado inicialmente en nodo worker**


**Apagar el nodo (kb360-w3)**

NodePort dejó de estar disponible, produciéndose un error de conexión.
*Los Pods independientes no disponen de mecanismos de autorecuperación*




***Kubernetes no recrea automáticamente la carga de trabajo en otro nodo del clúster.***

### 4.7 Implementación de autorecuperación mediante Deployment

Con el objetivo de validar el comportamiento real de autorecuperación de Kubernetes, se desplegó posteriormente un Deployment administrado por el controlador ReplicaSet.

El despliegue se realizó mediante:

**kubectl create deployment nginx-deploy --image=nginx --replicas=2**

Una vez desplegados los Pods, se validó su distribución entre los distintos workers del clúster:

**kubectl get pods -o wide**

Posteriormente, se simuló nuevamente la caída de uno de los nodos worker. En esta ocasión, Kubernetes detectó automáticamente la pérdida de los Pods afectados y recreó nuevas instancias en otros nodos disponibles del clúster.

La validación permitió comprobar:

- funcionamiento del scheduler,
- desired state,
- autorecuperación,
- y redistribución automática de cargas de trabajo.

Comprobación realizada:

**kubectl get deployments**

**kubectl get replicasets**

**kubectl get pods -o wide**

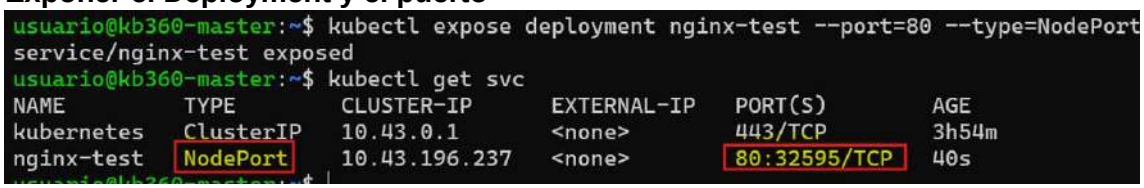
Las pruebas confirmaron que Kubernetes mantiene automáticamente el número de réplicas definidas dentro del Deployment incluso ante fallos de infraestructura.

**Creación del Deployment NGINX con múltiples réplicas**


**Distribución de Pods entre nodos worker**


**nginx-test-6ff8854996-59q4s** esto significa:
- Kubernetes lo controla Si falla = lo recrea.
- Ahora no es un Pod simple, es gestionado.

**Exponer el Deployment y el puerto**



**Simulación de fallo sobre nodo worker**
**Apagar kb360-w3**

```
usuario@kb360-router:~$ ping -c 3 10.0.1.13
PING 10.0.1.13 (10.0.1.13) 56(84) bytes of data.
From 10.0.1.1 icmp_seq=1 Destination Host Unreachable
From 10.0.1.1 icmp_seq=2 Destination Host Unreachable
From 10.0.1.1 icmp_seq=3 Destination Host Unreachable

--- 10.0.1.13 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2018ms
pipe 2
```

**Mostrando worker en estado NotReady.**

```
usuario@kb360-master:~$ kubectl get nodes
NAME           STATUS     ROLES                  AGE     VERSION
kb360-master   Ready      control-plane,etcd     4h12m   v1.35.4+rke2r1
kb360-w1       Ready      <none>                 3h53m   v1.35.4+rke2r1
kb360-w2       Ready      <none>                 3h46m   v1.35.4+rke2r1
kb360-w3       NotReady   <none>                 3h43m   v1.35.4+rke2r1
usuario@kb360-master:~$
```

**Redistribución automática de Pods tras el fallo**

```
usuario@kb360-master:~$ kubectl get nodes
NAME           STATUS     ROLES                  AGE     VERSION
kb360-master   Ready      control-plane,etcd     4h12m   v1.35.4+rke2r1
kb360-w1       Ready      <none>                 3h53m   v1.35.4+rke2r1
kb360-w2       Ready      <none>                 3h46m   v1.35.4+rke2r1
kb360-w3       NotReady   <none>                 3h43m   v1.35.4+rke2r1
usuario@kb360-master:~$
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                     READY   STATUS        RESTARTS   AGE   IP          NODE       NOMINATED NODE   READINESS GATES
nginx-test-6ff8854996-59q4s   1/1     Terminating   0          30m   10.42.3.6   kb360-w3   <none>           <none>
nginx-test-6ff8854996-v7pm6   1/1     Running       0          17s   10.42.2.5   kb360-w2   <none>           <none>
usuario@kb360-master:~$
```

**Validación del estado del Deployment Kubernetes**
**Validación desde nodo kb360-w1**

```
usuario@kb360-w1:~$ hostname
kb360-w1
usuario@kb360-w1:~$ curl http://10.0.1.12:32595
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

**Interpretación:**

Kubernetes ha hecho automáticamente: SIN intervención manual
- Detectar nodo caído
- Marcarlo como NotReady
- Eliminar el pod afectado
- Crear un nuevo pod
- Asignarlo a otro nodo disponible (kb360-w2)

**Prueba de alta disponibilidad mediante Deployment**

Tras la implementación del Deployment, se realizó una prueba de fallo controlado apagando el nodo worker kb360-w3, donde se encontraba inicialmente desplegado el pod.

El sistema detectó automáticamente la caída del nodo, marcándolo como *NotReady*. Como consecuencia, el pod asociado entró en estado *Terminating* y fue eliminado.

De forma automática, Kubernetes generó una nueva instancia del pod y la desplegó en otro nodo disponible, en este caso kb360-w2, sin intervención manual.

Este comportamiento demuestra la capacidad del sistema para garantizar la alta disponibilidad de las aplicaciones, asegurando la continuidad del servicio incluso ante fallos de infraestructura.

**CONCLUSIÓN TÉCNICA**

- El Deployment garantiza alta disponibilidad
- El servicio sigue funcionando
- El clúster se autorecupera
- El scheduler redistribuye la carga

**Prueba de acceso desde cliente externo**


**Acceso desde cliente externo**

Con el objetivo de validar el acceso real al servicio desplegado en el clúster, se realizó una prueba desde una máquina cliente independiente (kb360-client), situada en la misma red.

Desde el navegador web del cliente, se accedió a la dirección:

**http://10.0.1.12:32595**

Como resultado, se obtuvo correctamente la página por defecto de NGINX, lo que confirma que:

- El servicio NodePort está funcionando correctamente
- La comunicación de red entre cliente y clúster es operativa
- El servicio es accesible desde fuera de los nodos del clúster
- La reubicación automática del pod (tras el fallo previo) no ha afectado a la disponibilidad

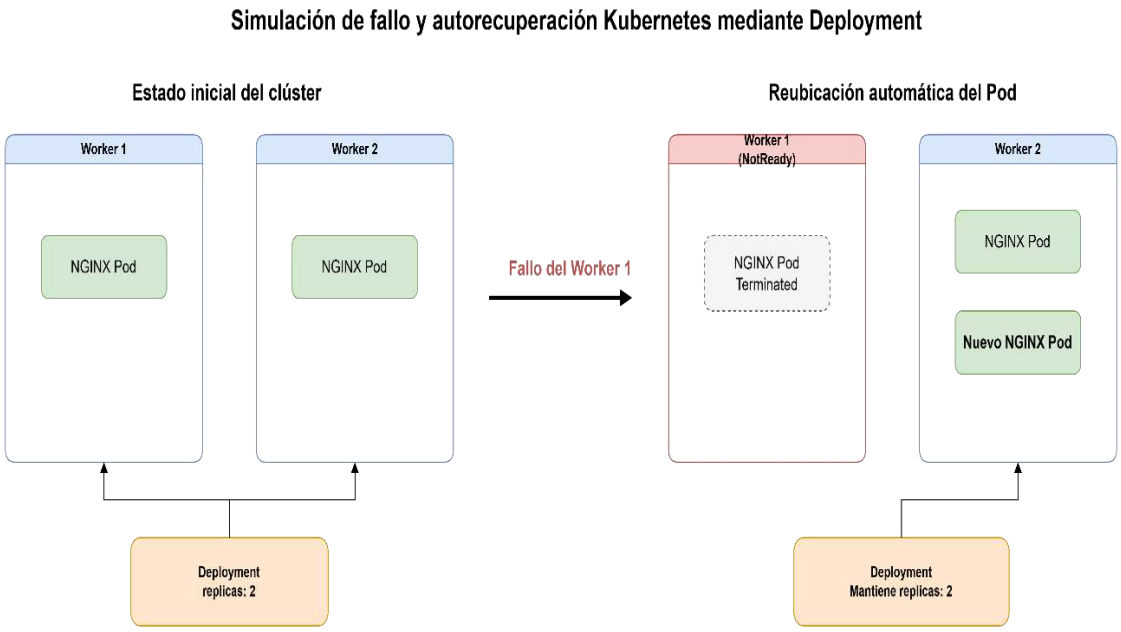
*Esta prueba demuestra que el sistema no solo funciona internamente, sino que también ofrece acceso externo real, cumpliendo con los requisitos de disponibilidad y conectividad del entorno Kubernetes desplegado*.

**Resultado de la simulación de fallo**

*Las pruebas realizadas permitieron validar el comportamiento de alta disponibilidad y autorecuperación proporcionado por Kubernetes mediante Deployments y ReplicaSets.*

*Asimismo, se comprobó el funcionamiento del scheduler Kubernetes y la capacidad del clúster para redistribuir automáticamente cargas de trabajo ante la caída de nodos worker, garantizando la continuidad operativa de los servicios desplegados.*

**Redistribución automática de Pods mediante Deployment Kubernete**




## 5. PERSISTENCIA Y ALMACENAMIENTO

Con el objetivo de incorporar persistencia de datos dentro del clúster Kubernetes, se implementó una solución de almacenamiento compartido basada en NFS (Network File System). Esta infraestructura permite desacoplar los datos del ciclo de vida de los contenedores, garantizando la conservación de la información incluso tras la eliminación o recreación de Pods dentro del clúster.

La integración del almacenamiento persistente se realizó mediante:

- servidor NFS dedicado,
- Persistent Volumes (PV),
- Persistent Volume Claims (PVC),
- y montaje de volúmenes persistentes en Pods Kubernetes.

La arquitectura implementada permite que los datos se almacenen físicamente en el servidor kb360-nfs, mientras que Kubernetes gestiona el acceso mediante recursos lógicos de almacenamiento persistente.

**Arquitectura de persistencia implementada**

Pod Kubernetes

↓

Persistent Volume Claim (PVC)

↓

Persistent Volume (PV)

↓

Servidor NFS (kb360-nfs)

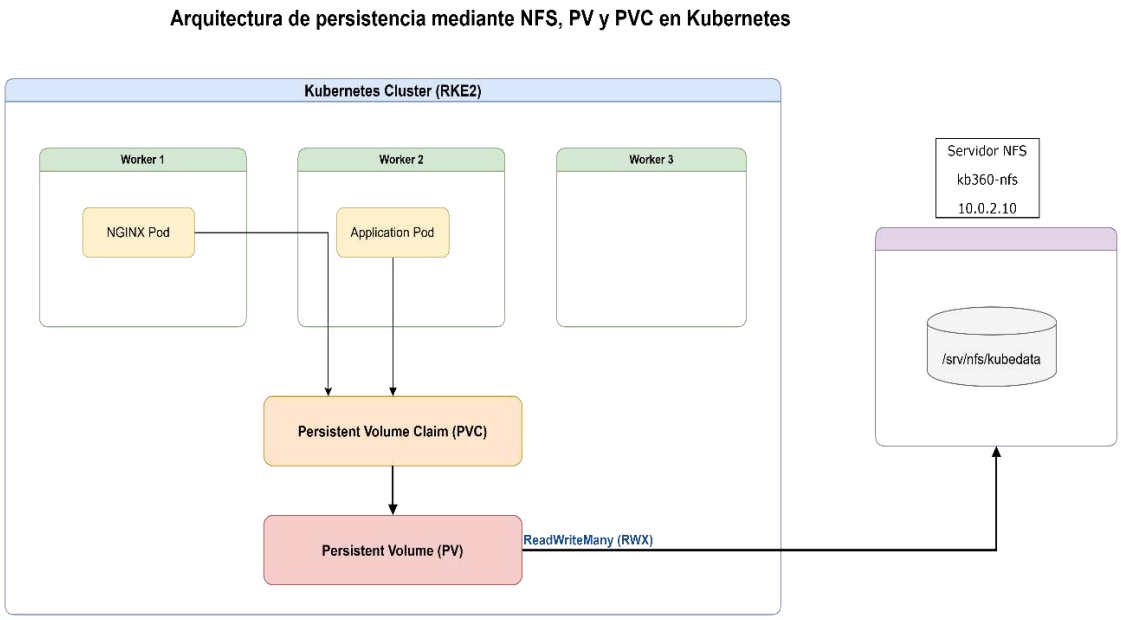
La utilización de NFS proporciona un sistema de almacenamiento compartido accesible desde todos los nodos del clúster, permitiendo el uso del modo de acceso ReadWriteMany (RWX) y facilitando escenarios de persistencia distribuidos dentro del entorno Kubernetes.

Durante esta fase se realizaron pruebas funcionales de persistencia, validando:

- Creación de volúmenes persistentes,
- Asociación automática PV/PVC,
- Escritura de datos desde contenedores,
- Eliminación de Pods,
- Recuperación posterior de la información almacenada.

Las validaciones realizadas confirmaron el correcto funcionamiento del sistema de almacenamiento persistente integrado dentro de la infraestructura Kubernetes del proyecto KubeBackup360.

**Arquitectura de persistencia mediante NFS, PV y PVC en Kubernetes**



### 5.1 Implementación del servidor NFS

Con el objetivo de proporcionar almacenamiento persistente al clúster Kubernetes, se implementó un servidor NFS dedicado sobre el nodo kb360-nfs. Esta solución permite compartir almacenamiento entre todos los nodos del entorno y facilita la integración de persistencia mediante Persistent Volumes (PV) y Persistent Volume Claims (PVC) dentro de Kubernetes.

La utilización de NFS proporciona un sistema de almacenamiento centralizado y accesible desde los distintos nodos worker del clúster, permitiendo conservar información incluso tras la eliminación o recreación de Pods.

#### 5.1.1 Validación de conectividad con el servidor NFS

Antes de iniciar la configuración del servicio NFS, se realizaron pruebas de conectividad y resolución DNS desde los nodos Kubernetes hacia el servidor kb360-nfs.

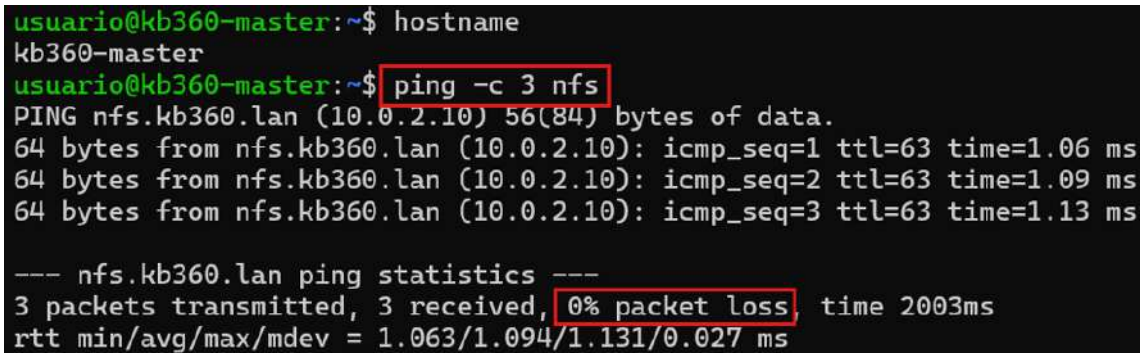
Validación realizada:

**ping kb360-nfs**

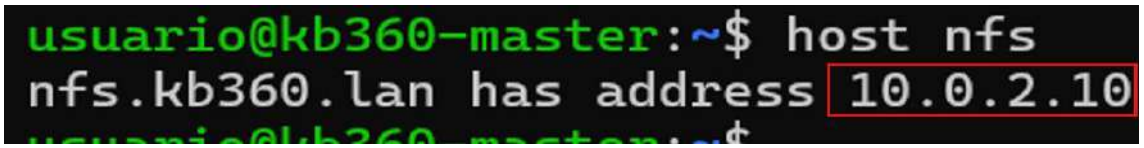
**host kb360-nfs**

Las comprobaciones confirmaron la correcta comunicación entre nodos y la resolución funcional del servidor NFS dentro del dominio interno kb360.lan.

**Validación de conectividad con el servidor NFS**




**Resolución DNS del servidor NFS**




#### 5.1.2 Instalación del servicio NFS

Una vez validada la conectividad, se procedió a instalar el servicio nfs-kernel-server sobre el nodo kb360-nfs.

Instalación realizada:

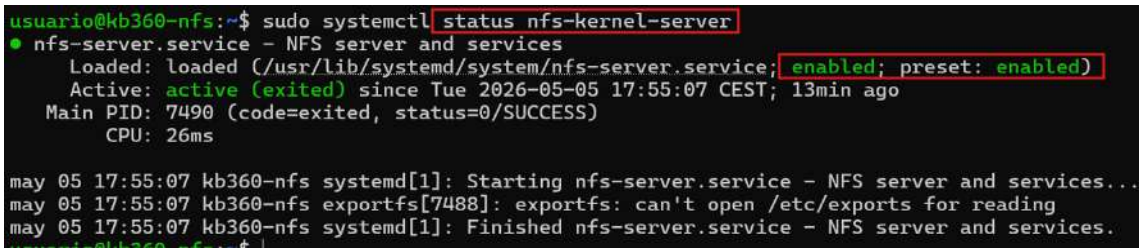
**sudo apt install nfs-kernel-server -y**

Posteriormente, se verificó el estado del servicio:

**sudo systemctl status nfs-kernel-server**

Las validaciones confirmaron que el servicio NFS se encontraba activo y preparado para exportar almacenamiento hacia el clúster Kubernetes.

**Servicio NFS en ejecución**




### 5.1.3 Configuración del directorio compartido

Para almacenar los datos persistentes del entorno Kubernetes, se creó un directorio compartido destinado a ser exportado mediante NFS.

Creación del directorio:

**sudo mkdir -p /srv/nfs/kubedata**

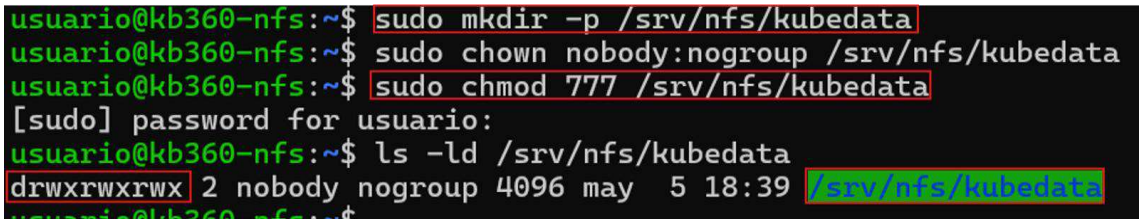
Configuración de permisos:

**sudo chown nobody:nogroup /srv/nfs/kubedata**

**sudo chmod 777 /srv/nfs/kubedata**

La configuración aplicada permite acceso compartido desde los nodos del clúster y facilita el uso de volúmenes persistentes compatibles con Kubernetes.

**Creación del directorio compartido NFS**
**Permisos aplicados al directorio compartido**




“Se crea el directorio /srv/nfs/kubedata y se asignan permisos amplios (chmod 777) y propiedad neutral (nobody:nogroup) para garantizar compatibilidad entre los nodos del clúster Kubernetes y evitar conflictos de permisos en el sistema NFS.”

### 5.1.4 Configuración de exportaciones NFS

Una vez creado el directorio compartido, se configuró el fichero /etc/exports para permitir el acceso desde la red interna del clúster Kubernetes.

Edición del fichero:

**sudo nano /etc/exports**

Configuración aplicada:

**/srv/nfs/kubedata 10.0.1.0/24(rw,sync,no_subtree_check,no_root_squash)**

Esta configuración permite:

- acceso lectura/escritura,

- sincronización de operaciones,
- acceso desde los nodos Kubernetes,
- compatibilidad con contenedores y volúmenes persistentes.

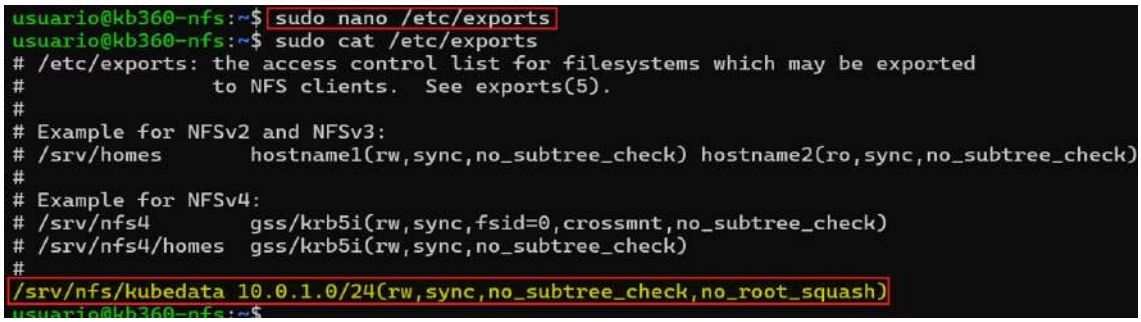
Posteriormente, se aplicaron los cambios:

**sudo exportfs -a**

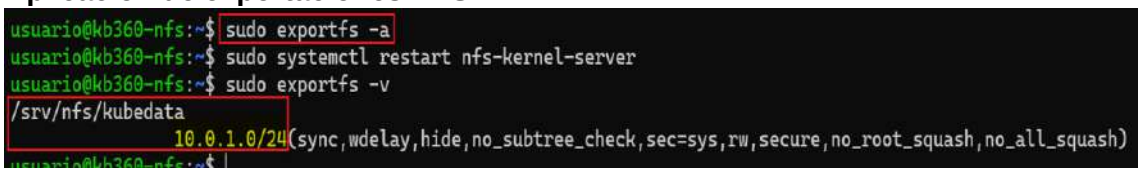
**sudo systemctl restart nfs-kernel-server**

**CAPTURAS OBLIGATORIAS**

**Configuración de exportaciones NFS**




**Aplicación de exportaciones NFS**




*Confirmando que el recurso se encuentra correctamente exportado.*
La exportación del directorio permite que los nodos Kubernetes accedan a almacenamiento compartido de forma centralizada. El uso de opciones como no_root_squash resulta fundamental en entornos de contenedores, ya que evita conflictos de permisos al permitir que procesos ejecutados como root dentro de los contenedores mantengan sus privilegios sobre el sistema de archivos NFS.

### 5.1.5 Validación del servidor NFS

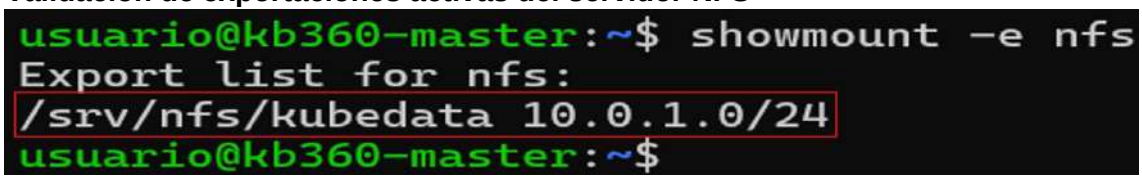
Finalmente, se verificó que el servidor exportaba correctamente el almacenamiento compartido configurado.

Validación realizada:

**showmount -e kb360-nfs**

Las comprobaciones confirmaron que el directorio compartido se encontraba accesible desde la red interna del clúster Kubernetes.

**Validación de exportaciones activas del servidor NFS**




**Resultado de la implementación NFS**

La implementación del servidor NFS permitió incorporar almacenamiento persistente compartido dentro de la infraestructura Kubernetes del proyecto KubeBackup360.

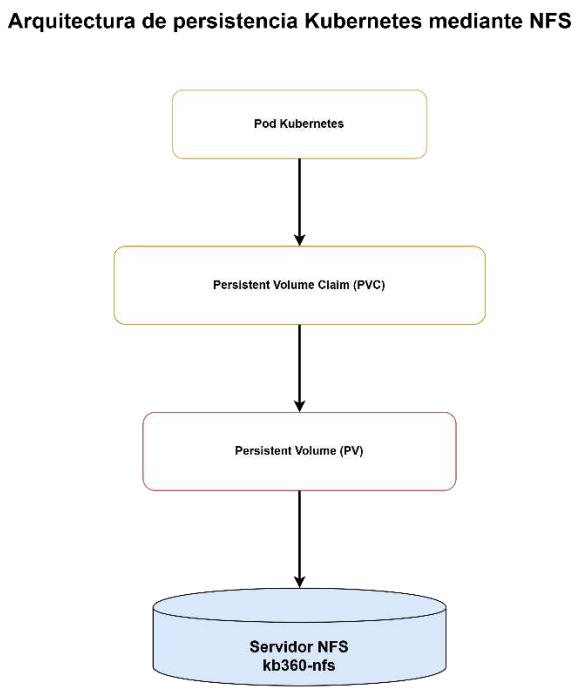
Las validaciones realizadas confirmaron la correcta exportación del almacenamiento, la conectividad entre nodos y la disponibilidad del recurso compartido para su posterior integración mediante volúmenes persistentes dentro del clúster Kubernetes.

### 5.2 Integración de NFS en Kubernetes

Una vez implementado y validado el servidor NFS, se procedió a integrar el almacenamiento compartido dentro del clúster Kubernetes mediante Persistent Volumes (PV) y Persistent Volume Claims (PVC). Esta integración permite que los Pods utilicen almacenamiento persistente desacoplado del ciclo de vida de los contenedores.

La arquitectura implementada permite que Kubernetes gestione el acceso al almacenamiento NFS utilizando recursos persistentes, garantizando la conservación de datos incluso tras la eliminación o recreación de Pods.

**Arquitectura de integración de almacenamiento**




#### 5.2.1 Instalación del cliente NFS en nodos Kubernetes

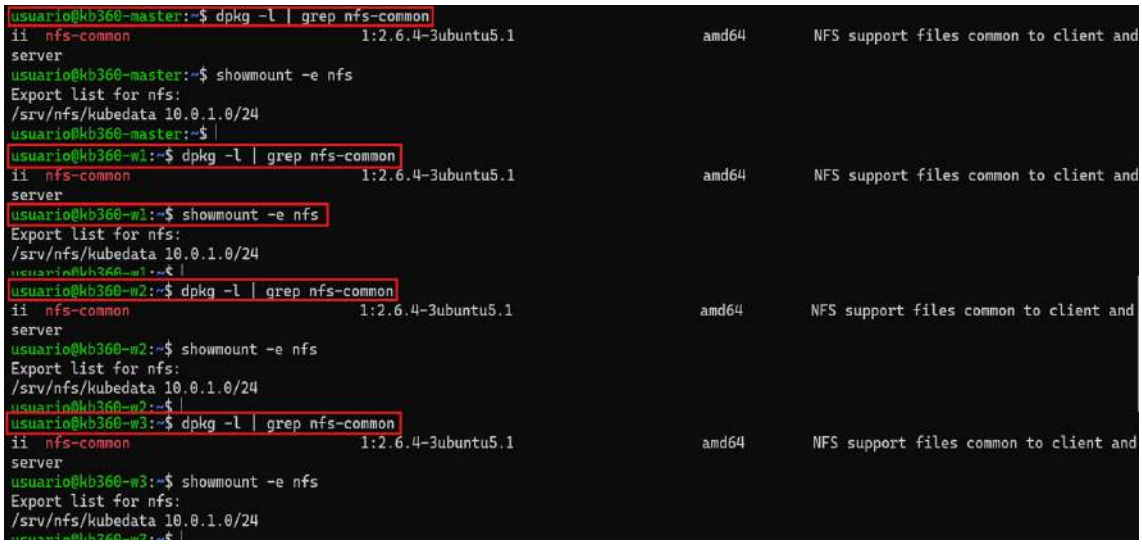
Antes de utilizar almacenamiento NFS dentro del clúster, fue *necesario instalar el paquete nfs-common en todos los nodos Kubernetes para habilitar el montaje de recursos NFS.*

Instalación realizada:

**sudo apt install nfs-common -y**

Las comprobaciones confirmaron que todos los nodos del clúster disponían de soporte para almacenamiento NFS.

**Instalación y comprobación del cliente NFS en nodos Kubernetes**




### 5.2.2 Creación del Persistent Volume (PV)

Una vez preparado el entorno, se creó un recurso Persistent Volume asociado al almacenamiento exportado por el servidor NFS.

El volumen persistente fue definido mediante un fichero YAML:

**nano pv-nfs.yaml**

Configuración utilizada:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    server: 10.0.2.10
    path: /srv/nfs/kubedata
```

Posteriormente, el recurso fue desplegado en Kubernetes:

**kubectl apply -f pv-nfs.yaml**

La validación confirmó que el volumen persistente fue creado correctamente dentro del clúster.

**Configuración del Persistent Volume NFS**

```
usuario@kb360-master:~$ nano pv-nfs.yaml
usuario@kb360-master:~$ cat pv-nfs.yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-nfs-kubedata
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Retain
  nfs:
    path: /srv/nfs/kubedata
    server: nfs
```

Explicación clave

- server: nfs → usa DNS (Bind9)
- path → directorio configurado
- ReadWriteMany → múltiples nodos pueden escribir
- Retain → no borra datos al eliminar PVC

**Persistent Volume creado correctamente**

```
usuario@kb360-master:~$ kubectl apply -f pv-nfs.yaml
persistentvolume/pv-nfs-kubedata created
usuario@kb360-master:~$ kubectl get pv
NAME             CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS      CLAIM   STORAGECLASS   VOLUMEATTRIBUTESCLASS   REASON   AGE
pv-nfs-kubedata  5Gi        RWX            Retain           Available                          <unset>                          62s
```

El volumen persistente quedó en estado Available, confirmando su correcta configuración y disponibilidad para ser utilizado por Kubernetes mediante un Persistent Volume Claim (PVC).

El uso de Persistent Volumes permite desacoplar el almacenamiento del ciclo de vida de los Pods, garantizando la persistencia de los datos dentro del clúster. Además, el modo de acceso ReadWriteMany (RWX) facilita el acceso compartido desde múltiples nodos Kubernetes.

***Es importante destacar que Kubernetes no almacena físicamente los datos en el nodo master, sino que únicamente gestiona los recursos de almacenamiento. En este proyecto, los datos se almacenan realmente en el servidor kb360-nfs, mientras Kubernetes administra el acceso mediante Persistent Volumes y Persistent Volume Claims.***

### 5.2.3 Creación del Persistent Volume Claim (PVC)

Una vez creado el volumen persistente, se configuró un Persistent Volume Claim para solicitar almacenamiento desde Kubernetes y asociarlo dinámicamente al PV disponible.

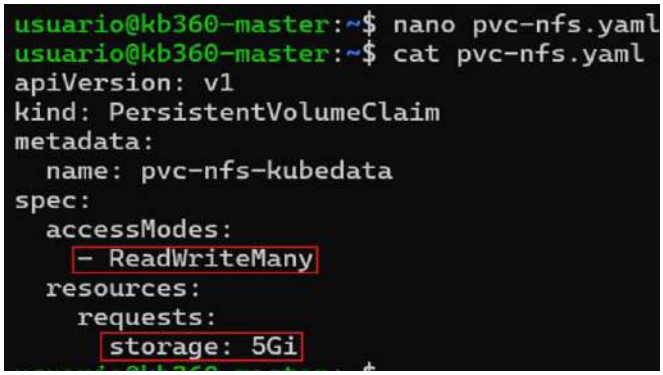
Configuración utilizada:

**nano pvc-nfs.yaml**

**apiVersion: v1**
**kind: PersistentVolumeClaim**
**metadata:**
**name: pvc-nfs**
**spec:**
**accessModes:**
**- ReadWriteMany**
**resources:**
**requests:**
**storage: 1Gi**
**Aplicación del recurso:**
**kubectl apply -f pvc-nfs.yaml**
**Posteriormente, se verificó el estado del PVC:**
**kubectl get pvc**

Las comprobaciones confirmaron que el PVC quedó correctamente asociado al volumen

**Configuración del Persistent Volume Claim**




storage: 5Gi → debe coincidir con el PV
ReadWriteMany → permite acceso desde varios nodos
*Kubernetes buscará automáticamente un PV compatible*

Persistent Volume Claim en estado Bound


El estado Bound confirma que el Persistent Volume Claim (PVC) fue asociado correctamente al Persistent Volume (PV) disponible.

*El uso de PVC permite que los Pods soliciten almacenamiento de forma dinámica sin depender de la ubicación física del sistema de almacenamiento. Kubernetes gestiona automáticamente la asociación entre PV y PVC, facilitando una administración transparente y flexible del almacenamiento persistente dentro del clúster*

### 5.2.4 Validación de persistencia en Kubernetes

Con el objetivo de validar el funcionamiento del almacenamiento persistente, se desplegó un Pod utilizando el PVC previamente configurado.

El Pod fue asociado al volumen persistente mediante la definición YAML correspondiente, permitiendo montar el almacenamiento NFS dentro del contenedor.

Una vez desplegado el Pod, se accedió al contenedor y se generó un archivo de prueba dentro del volumen persistente:

**kubectl exec -it nginx-pv -- bash**

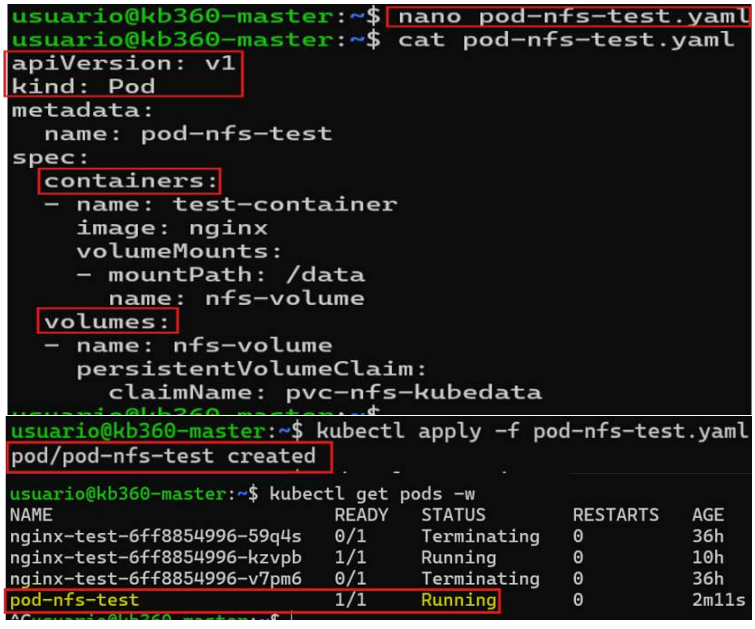
Creación del archivo:

**echo "KubeBackup360" > /data/test.txt**

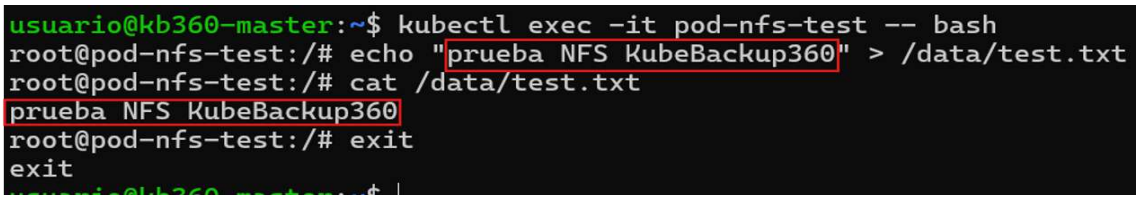
Posteriormente, el Pod fue eliminado y recreado nuevamente para validar que la información permanecía almacenada dentro del servidor NFS.

Las comprobaciones realizadas confirmaron que los datos persistían correctamente tras la recreación del contenedor, validando el funcionamiento del sistema de almacenamiento persistente integrado en Kubernetes.

**Pod utilizando almacenamiento persistente NFS**




**Escritura de datos en volumen persistente**



El volumen persistente fue montado en la ruta /data del contenedor, permitiendo la escritura de datos sobre el almacenamiento NFS compartido.

Para validar la persistencia, se accedió al Pod mediante
kubectl exec -it pod-nfs-test -- bash
Dentro del contenedor se creó un archivo de prueba sobre el volumen montado. Posteriormente, el Pod fue eliminado y recreado, comprobando que los datos permanecían disponibles gracias al almacenamiento persistente proporcionado por NFS y gestionado mediante PVC.

El resultado obtenido confirma que el archivo se ha creado correctamente y que los datos pueden ser escritos y leídos desde el volumen montado.

**Esta prueba válida que:**

- El pod puede acceder al volumen persistente
- El montaje del PVC es correcto
- La comunicación con el servidor NFS funciona
- Se pueden realizar operaciones de escritura y lectura

**Eliminación del Pod con almacenamiento persistente**

```
usuario@kb360-master:~$ kubectl delete pod pod-nfs-test
pod "pod-nfs-test" deleted from default namespace
```

**Creación y comprobación del Pod con almacenamiento persistente**

```
usuario@kb360-master:~$ kubectl delete pod pod-nfs-test
pod "pod-nfs-test" deleted from default namespace
usuario@kb360-master:~$
usuario@kb360-master:~$ kubectl apply -f pod-nfs-test.yaml
pod/pod-nfs-test created
usuario@kb360-master:~$ kubectl get pods -w
NAME                          READY   STATUS        RESTARTS   AGE
nginx-test-6ff8854996-59q4s   0/1     Terminating   0          36h
nginx-test-6ff8854996-kzvpb   1/1     Running       0          10h
nginx-test-6ff8854996-v7pm6   0/1     Terminating   0          36h
pod-nfs-test                  1/1     Running       0          11s
^Cusuario@kb360-master:~$
```

**Persistencia de datos tras recreación del Pod**

```
^Cusuario@kb360-master:~kubectl exec -it pod-nfs-test -- bashsh
root@pod-nfs-test:/# cat /data/test.txt
prueba NFS KubeBackup360
root@pod-nfs-test:/# exit
exit
```

**Esta prueba demuestra qu**e:

- El almacenamiento no depende del pod
- Los datos se almacenan en el servidor NFS
- Kubernetes monta correctamente el volumen en nuevos pods
- El sistema garantiza persistencia real de la información

**Resultado de la integración NFS en Kubernetes**

El sistema de almacenamiento persistente basado en NFS ha sido correctamente implementado y validado, cumpliendo el objetivo de mantener los datos independientemente del ciclo de vida de los contenedores.

La integración del servidor NFS dentro del clúster Kubernetes permitió implementar almacenamiento persistente funcional mediante Persistent Volumes y Persistent Volume Claims.

Las validaciones realizadas confirmaron la correcta asociación entre recursos persistentes y Pods Kubernetes, garantizando la conservación de datos independientemente del ciclo de vida de los contenedores desplegados dentro del entorno KubeBackup360.

## 6. SISTEMA DE BACKUPS Y RECUPERACIÓN

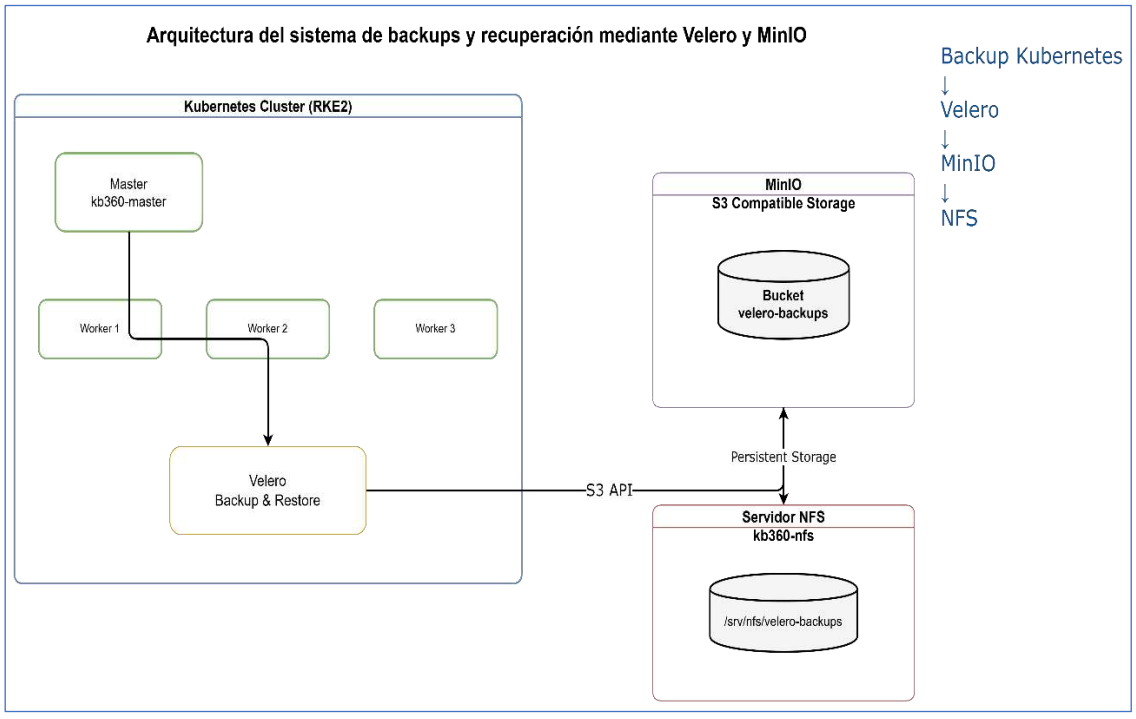
Con el objetivo de proteger la infraestructura Kubernetes frente a fallos o pérdida de recursos, se implementó un sistema de backups y recuperación basada en Velero y MinIO. La solución desarrollada permite realizar copias de seguridad de recursos Kubernetes, almacenar los backups de forma persistente y restaurar servicios dentro del entorno del proyecto KubeBackup360.

La arquitectura implementada integra:

- Velero como herramienta de backup y restore,
- MinIO como almacenamiento compatible con S3,
- y NFS como sistema de almacenamiento persistente.

Durante esta fase se validaron procesos de backup, restauración y automatización de copias de seguridad dentro del clúster Kubernetes.

**Arquitectura del sistema de backups y recuperación mediante Velero y MinIO**



| Elemento | Función |
|---|---|
| Velero | Gestiona backups |
| MinIO | Simula almacenamiento S3 |
| NFS | Guarda físicamente los datos |

### 6.1 Introducción al sistema DRP

Tras implementar la persistencia mediante NFS, se integró un sistema básico de Disaster Recovery (DRP) orientado a la protección y recuperación de recursos Kubernetes. Para ello, se utilizó Velero como herramienta de backup y restore, junto con MinIO como almacenamiento compatible con S3.

La arquitectura implementada permitió:

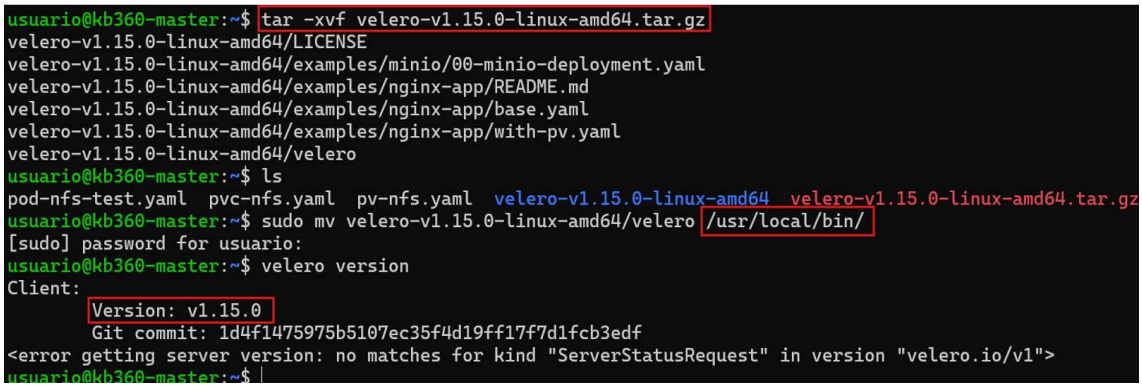
- realizar backups de recursos Kubernetes,
- almacenar copias de seguridad de forma persistente,
- restaurar Deployments y servicios eliminados,
- y automatizar tareas de backup dentro del clúster.

Las validaciones realizadas confirmaron el correcto funcionamiento del sistema de backups y recuperación implementado dentro del entorno Kubernetes del proyecto KubeBackup360.

**Objetivo de este paso**

Instalar la herramienta **Velero CLI** en el nodo kb360-master, la cual permitirá:

- Crear backups del clúster Kubernetes
- Restaurar recursos
- Gestionar copias de seguridad desde terminal
- Descargar e instalar Velero en el nodo master




**Verificación de instalación de Velero**

Tras mover el binario al directorio /usr/local/bin, se ha verificado el correcto funcionamiento de la herramienta mediante:

**velero version**

El resultado obtenido muestra correctamente la versión cliente instalada (v1.15.0), confirmando que la herramienta CLI se encuentra operativa en el nodo master.

Adicionalmente, se muestra un mensaje indicando que no es posible obtener la versión del servidor Velero. Este comportamiento es esperado, ya que en esta fase únicamente se ha instalado el cliente y todavía no se ha desplegado el servidor Velero dentro del clúster Kubernetes.

### 6.2 Preparación del almacenamiento de backups

Con el objetivo de almacenar las copias de seguridad generadas por Velero, se preparó un directorio específico dentro del servidor NFS destinado al almacenamiento persistente de backups del clúster Kubernetes.

La estructura configurada permitirá que MinIO almacene de forma persistente los backups generados por Velero utilizando almacenamiento compartido sobre NFS.

Creación del directorio de backups:

**sudo mkdir -p /srv/nfs/velero-backups**

Posteriormente, se configuraron los permisos y propietarios necesarios:

**sudo chown nobody:nogroup /srv/nfs/velero-backups**

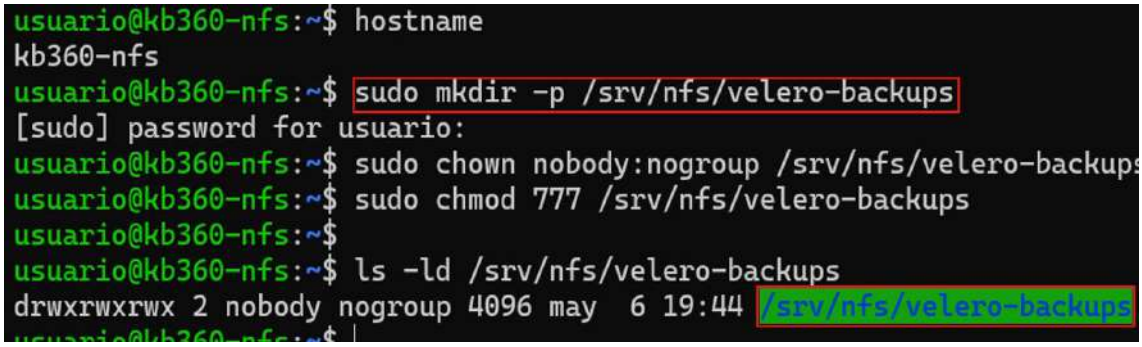
**sudo chmod 777 /srv/nfs/velero-backups**

Finalmente, se validó la correcta creación del directorio:

**ls -ld /srv/nfs/velero-backups**

Las comprobaciones confirmaron que el almacenamiento persistente quedó correctamente preparado para ser utilizado posteriormente por MinIO y Velero dentro del entorno Kubernetes.

**Creación del directorio de almacenamiento para backups**
**Validación de permisos del almacenamiento de backups**




**Resultado de la preparación del almacenamiento**

La preparación del almacenamiento persistente permitió disponer de una ubicación centralizada para el almacenamiento de backups dentro del entorno Kubernetes.

El directorio configurado será utilizado posteriormente por MinIO para almacenar las copias de seguridad generadas por Velero, garantizando persistencia y conservación de los datos dentro de la infraestructura KubeBackup360.

### 6.3.2 Arquitectura de almacenamiento MinIO

La arquitectura implementada para el sistema de backups se basa en el uso de MinIO como plataforma de almacenamiento compatible con S3 integrada dentro del clúster Kubernetes. Esta solución permite almacenar las copias de seguridad generadas por Velero utilizando almacenamiento persistente sobre NFS.

El funcionamiento general del sistema se basa en el siguiente flujo:

Velero
↓
MinIO Bucket
↓
Persistent Volume Claim (PVC)
↓
Persistent Volume (PV)
↓
Servidor NFS

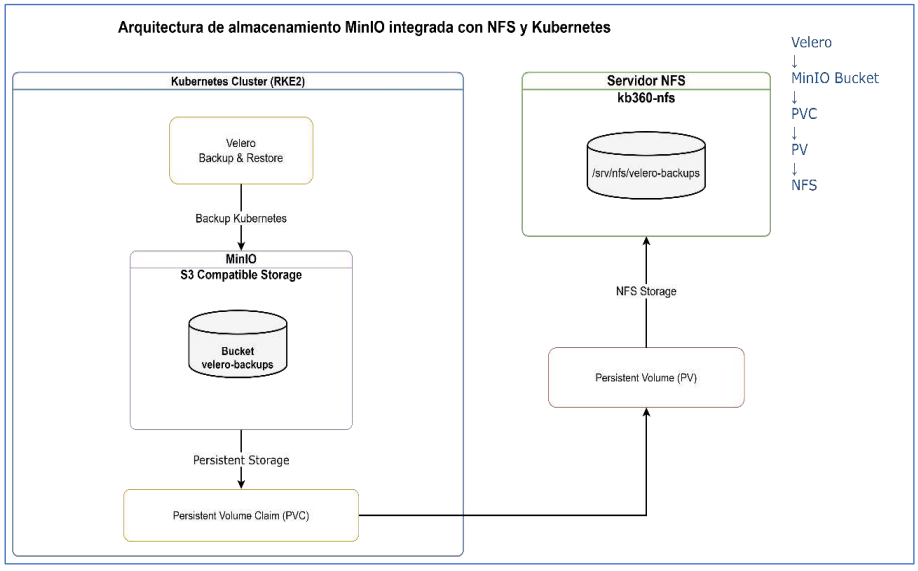
En esta arquitectura:

- Velero genera los backups de recursos Kubernetes,
- MinIO actúa como almacenamiento S3 compatible,
- Kubernetes gestiona el almacenamiento persistente mediante PVC y PV,
- y los datos se almacenan físicamente en el servidor kb360-nfs.

Esta integración permite desacoplar el almacenamiento del ciclo de vida de los contenedores y garantiza persistencia de los backups incluso ante reinicios o recreaciones de Pods dentro del clúster.

Asimismo, el uso de almacenamiento persistente sobre NFS permite conservar las copias de seguridad de forma centralizada y accesible desde la infraestructura Kubernetes.

**Arquitectura de almacenamiento MinIO integrada con NFS y Kubernetes**




**Resultado de la arquitectura implementada**

La arquitectura desplegada permitió integrar almacenamiento S3 compatible dentro del entorno Kubernetes utilizando MinIO y almacenamiento persistente NFS.

Las validaciones realizadas confirmaron la correcta integración entre Kubernetes, MinIO y el sistema de almacenamiento persistente utilizado para la gestión de backups dentro del proyecto KubeBackup360.

Perfecto. He revisado el estado actual de la memoria y toda la Fase 3 ya redactada para mantener coherencia técnica y estilo documental. La referencia principal efectivamente debe seguir la línea de la Fase 3 ya desarrollada, especialmente el tono técnico-práctico que aparece en fase_3_Final.pdf.

A partir de aquí, el siguiente bloque correcto es:

### 6.3.3 Despliegue de MinIO en Kubernetes

Una vez creado el namespace específico para MinIO y definida la arquitectura de almacenamiento persistente basada en NFS, se procedió al despliegue del servicio MinIO dentro del clúster Kubernetes.

El objetivo de esta implementación consiste en proporcionar un sistema de almacenamiento de objetos compatible con S3 que permita a Velero almacenar las copias de seguridad del clúster Kubernetes sin depender de proveedores cloud externos.

La arquitectura implementada queda estructurada de la siguiente forma:

**Velero → Bucket MinIO → PVC → PV → NFS**

De este modo, Kubernetes administra los recursos lógicos de almacenamiento mientras que los datos físicos permanecen almacenados de forma persistente en el servidor NFS del laboratorio.

Para el despliegue se creó un archivo de configuración denominado:

**nano minio.yaml**

El manifiesto YAML define los siguientes recursos Kubernetes:

| Recurso | Función |
|---|---|
| PersistentVolumeClaim | almacenamiento persistente para MinIO |
| Deployment | despliegue del contenedor MinIO |
| Service | acceso de red al servicio |

El PersistentVolumeClaim utilizado para MinIO se configuró con acceso

ReadWriteMany, permitiendo el uso del almacenamiento compartido basado en NFS:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-pvc
  namespace: minio
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
```

**storage: 2Gi**

Posteriormente, el despliegue fue aplicado al clúster mediante:

**kubectl apply -f minio.yaml**

El resultado obtenido confirmó la creación correcta de los recursos definidos dentro del namespace minio.

Finalmente, se verificó el estado operativo de MinIO utilizando:

kubectl get pods,pvc,svc -n minio

La validación mostró:

- pod MinIO en estado Running
- PVC correctamente asociado
- Service operativo
- almacenamiento persistente activo

Con esta implementación quedó desplegado el backend de almacenamiento compatible con S3 necesario para la integración posterior con Velero y el sistema de backups del proyecto KubeBackup360.

**Figuras recomendadas**

**Configuración YAML de MinIO y PersistentVolumeClaim**

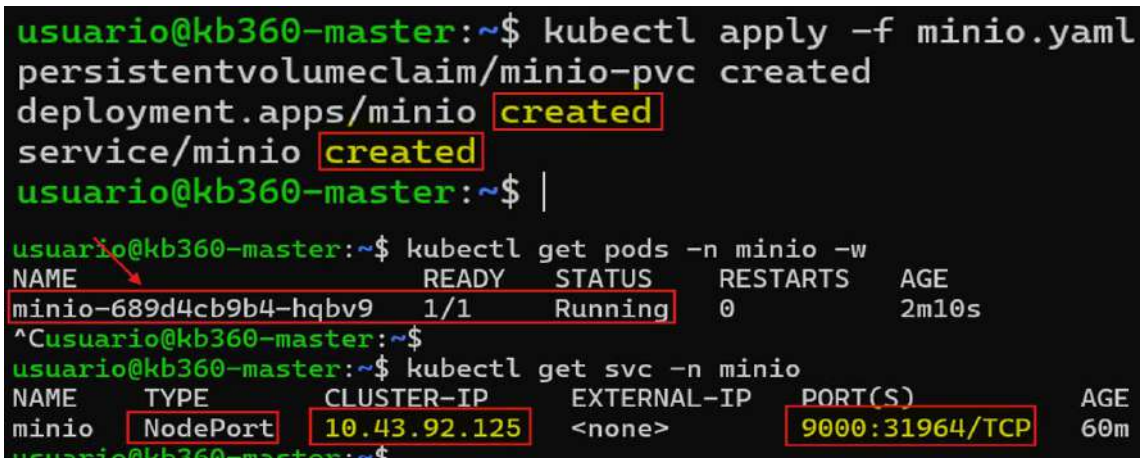
```
usuario@kb360-master:~$ nano minio.yaml
usuario@kb360-master:~$ cat minio.yaml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: minio-pvc
  namespace: minio
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 5Gi

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: minio
  namespace: minio
spec:
  replicas: 1
  selector:
    matchLabels:
      app: minio
  template:
    metadata:
      labels:
        app: minio
    spec:
      containers:
      - name: minio
        image: minio/minio
        args:
          - server
          - /data
        env:
        - name: MINIO_ROOT_USER
          value: admin
        - name: MINIO_ROOT_PASSWORD
          value: password123
        ports:
        - containerPort: 9000
        volumeMounts:
        - mountPath: /data
          name: minio-storage
      volumes:
      - name: minio-storage
        persistentVolumeClaim:
          claimName: minio-pvc

---
apiVersion: v1
kind: Service
metadata:
  name: minio
  namespace: minio
spec:
  selector:
    app: minio
  ports:
    - port: 9000
      targetPort: 9000
  type: NodePort
usuario@kb360-master:~$
```

**Este archivo crea:**

- Recurso Función
- PV almacenamiento persistente
- Deployment servidor MinIO
- Service acceso por red

**Aplicación del despliegue MinIO**

**Verificación de Pod, PVC y Service de MinIO**




En esta fase se ha procedido al despliegue del servicio de almacenamiento de objetos MinIO dentro del clúster Kubernetes mediante la aplicación de un archivo de configuración (minio.yaml). Este archivo define los recursos necesarios para su funcionamiento, incluyendo un PersistentVolumeClaim (PVC) para el almacenamiento persistente, un Deployment para la ejecución del contenedor de MinIO y un Service para permitir el acceso a través de la red.

### 6.3.4 Validación funcional de MinIO

Una vez completado el despliegue de MinIO dentro del clúster Kubernetes, se procedió a realizar la validación funcional del servicio con el objetivo de comprobar el correcto funcionamiento del backend de almacenamiento compatible con S3 utilizado posteriormente por Velero.

Inicialmente, se verificó el estado operativo de los recursos desplegados en el namespace minio mediante:

**kubectl get pods,pvc,svc -n minio**

La comprobación confirmó que:

- el pod de MinIO se encontraba en estado Running
- el PersistentVolumeClaim estaba asociado correctamente
- el Service Kubernetes permanecía operativo
- el almacenamiento persistente basado en NFS funcionaba correctamente

Posteriormente, se habilitó el acceso externo a la consola web de administración mediante un Service de tipo NodePort, permitiendo acceder a la interfaz gráfica desde la red interna del laboratorio.

El acceso se realizó utilizando la dirección:

http://10.0.1.10:30902

Las credenciales configuradas durante el despliegue fueron:
Usuario: admin
Contraseña: password123

Una vez validado el acceso a la consola web, se comprobó el correcto funcionamiento de la interfaz administrativa de MinIO y la disponibilidad del sistema de almacenamiento de objetos.

Finalmente, se creó el bucket denominado velero, el cual será utilizado como repositorio principal para almacenar las copias de seguridad generadas por Velero.

Con esta validación quedó confirmado que:

- MinIO funciona correctamente dentro de Kubernetes
- el almacenamiento persistente NFS está integrado correctamente
- el acceso web administrativo es funcional
- el backend compatible con S3 está operativo
- la infraestructura queda preparada para la instalación y configuración de Velero

La implementación de MinIO permite disponer de un entorno de almacenamiento completamente local y compatible con S3, facilitando la gestión de backups del clúster Kubernetes sin depender de servicios cloud externos.

**Despliegue y verificación del servicio MinIO en Kubernetes**

```
usuario@kb360-master:~$ kubectl apply -f minio.yaml
persistentvolumeclaim/minio-pvc created
deployment.apps/minio created
service/minio created
```

Como resultado, Kubernetes ha creado correctamente los recursos definidos, tal y como se observa en la salida del sistema, donde se confirma la creación del volumen persistente (minio-pvc), el despliegue de la aplicación (deployment.apps/minio) y el servicio asociado (service/minio).

```
usuario@kb360-master:~$ kubectl get pods -n minio -w
NAME                     READY   STATUS    RESTARTS   AGE
minio-689d4cb9b4-hqbv9   1/1     Running   0          2m10s
^Cusuario@kb360-master:~$
usuario@kb360-master:~$ kubectl get svc -n minio
NAME    TYPE       CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
minio   NodePort   10.43.92.125    <none>        9000:31964/TCP   60m
```

En la salida se observa que el pod de MinIO se encuentra en estado Running y completamente operativo (1/1), lo que confirma que el despliegue se ha realizado con éxito y que el servicio está funcionando correctamente dentro del clúster.

Este paso valida la correcta integración del almacenamiento persistente con Kubernetes y establece la base necesaria para su utilización como backend en la gestión de copias de seguridad mediante Velero

### 6.3.5 Exposición de la consola web de MinIO

Durante las pruebas de validación del servicio MinIO se detectó que inicialmente únicamente se encontraba expuesto el puerto correspondiente a la API S3 utilizada por

Velero. Sin embargo, la consola web administrativa requiere un puerto independiente para permitir el acceso gráfico al sistema de almacenamiento.

Para habilitar la interfaz web de administración, se modificó el recurso Service de Kubernetes añadiendo un segundo puerto de tipo NodePort destinado específicamente a la consola web de MinIO.

Posteriormente, se aplicó la nueva configuración mediante:

**kubectl apply -f minio-service.yaml**

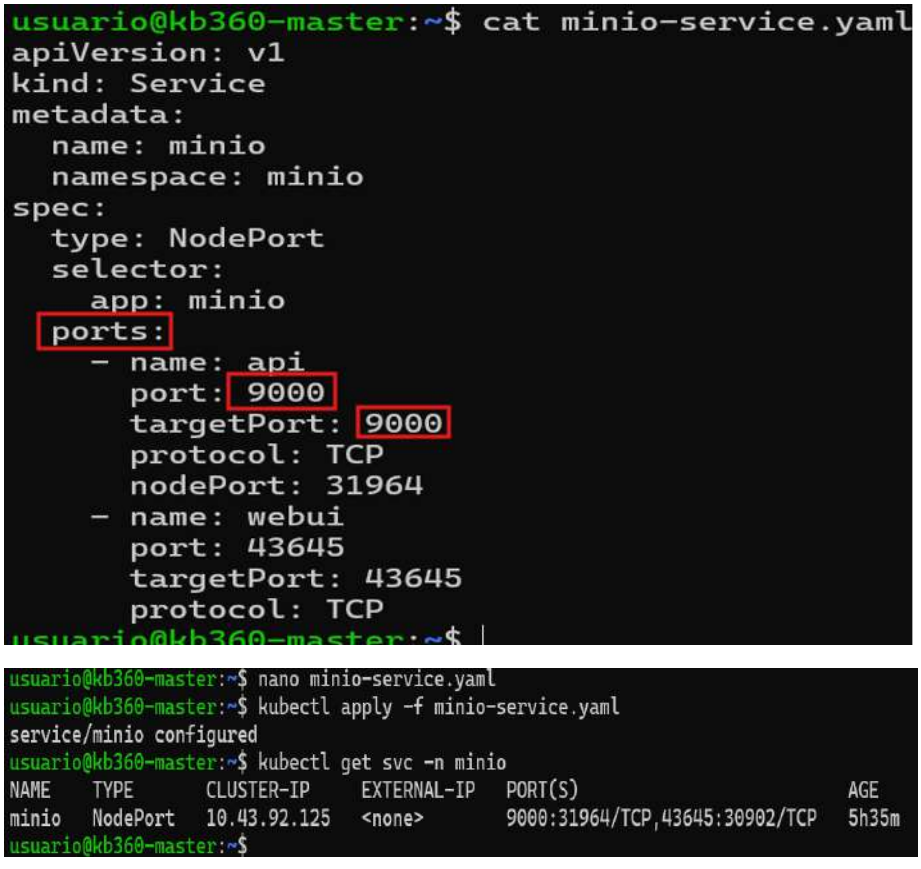
Finalmente, se verificó el estado del servicio utilizando:

**kubectl get svc -n minio**

El resultado obtenido muestra que Kubernetes asignó correctamente el puerto externo 30902, permitiendo acceder a la consola web de MinIO desde la red interna del laboratorio.

Esta configuración permitió completar la validación del acceso administrativo al sistema de almacenamiento compatible con S3 desplegado dentro del clúster Kubernetes.

**Configuración del Service NodePort y exposición web de MinIO**




La imagen muestra la configuración del recurso Service de Kubernetes utilizado para exponer externamente la consola web de MinIO mediante un puerto NodePort, así como la validación posterior del servicio dentro del namespace minio.

### 6.3.6 Acceso operativo a la consola web de MinIO

La imagen muestra el acceso funcional a la consola web administrativa de MinIO mediante el puerto NodePort configurado previamente en Kubernetes, validando el correcto funcionamiento del sistema de almacenamiento compatible con S3 dentro del clúster

**Exposición de la consola web de MinIO**

Durante las pruebas de acceso se detectó que el servicio inicialmente solo exponía el puerto correspondiente a la API S3 de MinIO. Sin embargo, la consola web administrativa utiliza un puerto independiente generado dinámicamente por la aplicación.

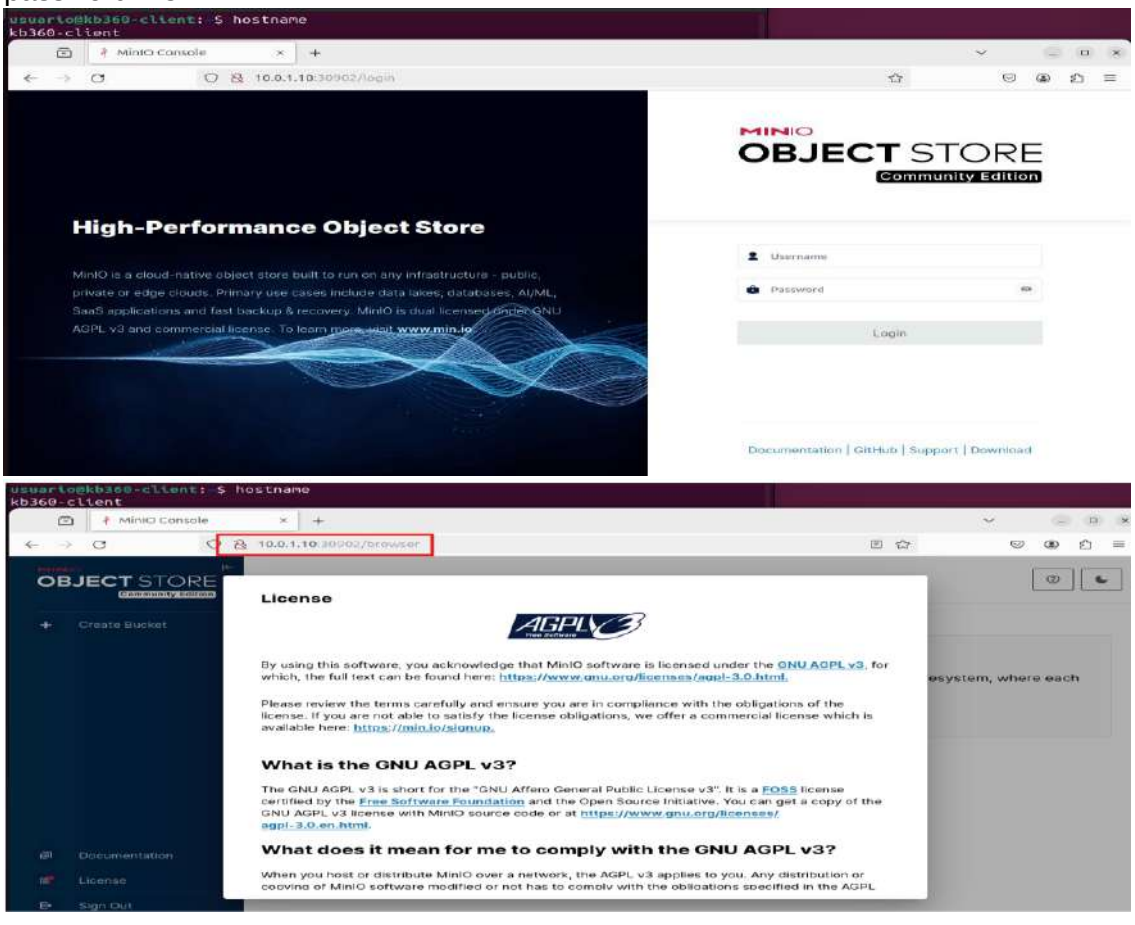
Para solucionar este problema, se modificó el recurso Service de Kubernetes añadiendo un segundo puerto NodePort destinado específicamente a la interfaz web de administración.

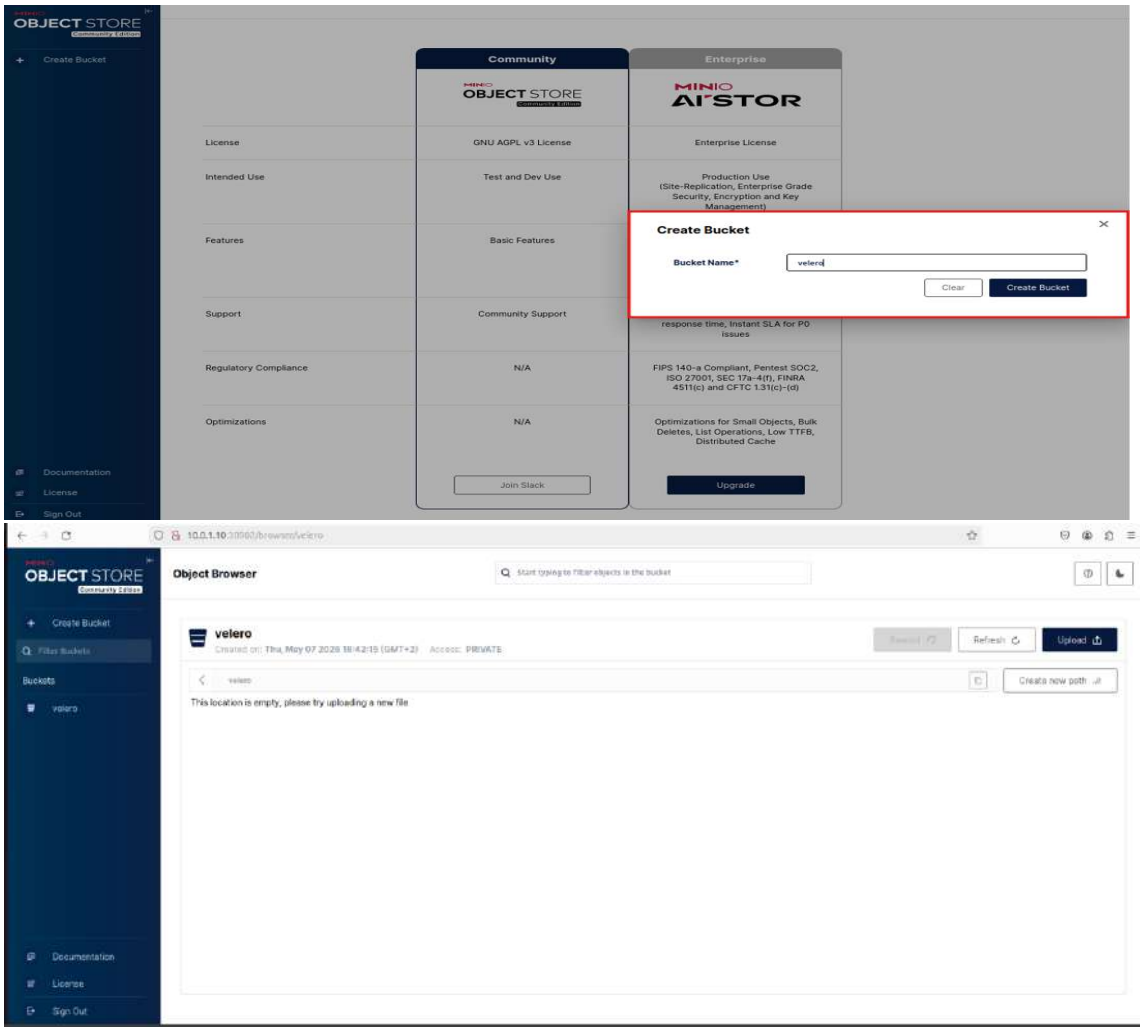
Tras aplicar la nueva configuración, Kubernetes asignó el puerto externo 30902, permitiendo acceder correctamente a la consola web de MinIO desde la red interna del laboratorio.

**Creación del bucket para Velero**

http://10.0.1.10:30902

admin
password123






Una vez desplegado y accesible el servicio MinIO, se ha procedido a crear el bucket denominado velero, el cual será utilizado como repositorio principal de almacenamiento para las copias de seguridad del clúster Kubernetes.

La creación se realizó desde la interfaz web de MinIO utilizando las credenciales configuradas previamente durante el despliegue.

Este bucket permitirá almacenar:

backups completos del clúster
metadatos de restauración
snapshots y configuraciones asociadas

### 6.4 Instalación de Velero en Kubernetes

Una vez preparado el backend de almacenamiento compatible con S3 mediante MinIO, se procedió a instalar Velero dentro del clúster Kubernetes con el objetivo de implementar un sistema de copias de seguridad y recuperación ante desastres.
La arquitectura final implementada queda estructurada de la siguiente forma:

**Velero → MinIO (bucket velero) → PVC → NFS**

De este modo, Velero almacenará las copias de seguridad del clúster Kubernetes dentro del bucket velero previamente creado en MinIO, mientras que los datos físicos permanecerán almacenados de forma persistente sobre el servidor NFS del laboratorio.

#### 6.4.1 Creación de credenciales S3

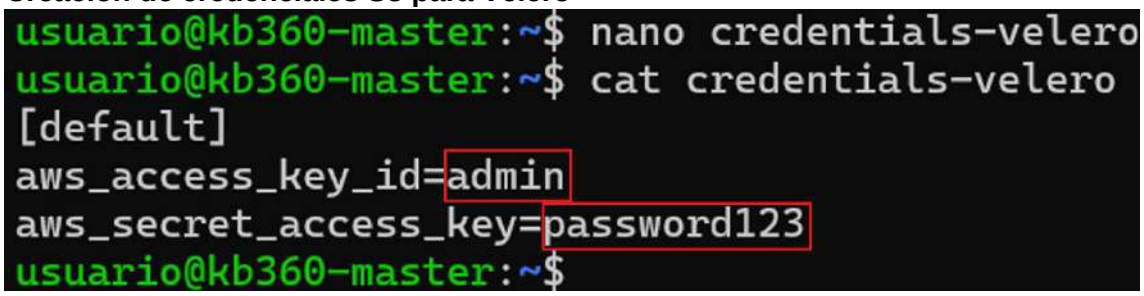
Debido a que Velero utiliza almacenamiento compatible con S3 para gestionar las copias de seguridad, fue necesario crear un archivo de credenciales para permitir la autenticación contra MinIO.

Para ello, se creó el siguiente archivo en el nodo kb360-master:

**nano credentials-velero**

Aunque durante la instalación el proveedor aparece como aws, Velero utiliza realmente MinIO mediante compatibilidad con la API S3 de Amazon.

**Creación de credenciales S3 para Velero**




La imagen muestra la creación del archivo credentials-velero, utilizado por Velero para autenticarse contra el backend de almacenamiento MinIO compatible con S3.

#### 6.4.2 Instalación de Velero

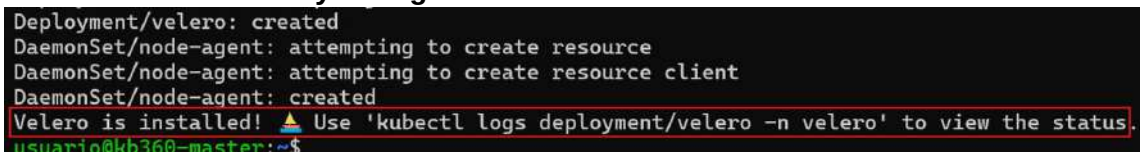
La instalación de Velero se realizó mediante el siguiente comando:

```
velero install \
--provider aws \
--plugins velero/velero-plugin-for-aws:v1.10.0 \
--bucket velero \
--secret-file ./credentials-velero \
--use-node-agent \
--backup-location-config
region=minio,s3ForcePathStyle="true",s3Url=http://minio.minio.svc.cluster.local:9000
```

Este proceso realizó automáticamente:

- creación del namespace velero
- despliegue de los componentes de Velero
- configuración del backend S3
- conexión con MinIO
- preparación del sistema de backups Kubernetes

**Instalación de Velero y configuración del backend S3**




La imagen muestra la ejecución del proceso de instalación de Velero dentro del clúster Kubernetes utilizando MinIO como backend de almacenamiento compatible con S3.

### 6.4.3 Verificación de la instalación

Una vez desplegado Velero dentro del clúster Kubernetes, se procedió a verificar el estado de los componentes instalados mediante:

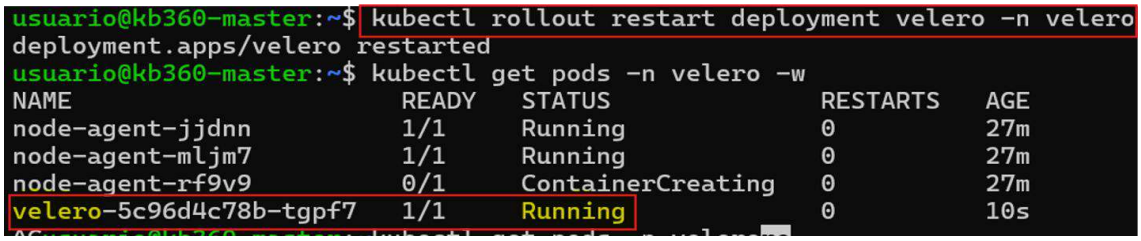
**kubectl get pods -n velero**

La validación confirmó que los pods del namespace velero se encontraban en estado Running, indicando que la instalación se había realizado correctamente.
Posteriormente, se verificó el funcionamiento del cliente y del repositorio de almacenamiento mediante:

**velero versión**
**velero backup-location get**

El resultado obtenido confirmó que el bucket velero aparecía en estado Available, validando la correcta comunicación entre Velero y MinIO mediante el backend compatible con S3.

**Verificación de pods y componentes de Velero**



```
usuario@kb360-master:~$ kubectl get pods -n velero
NAME                      READY   STATUS    RESTARTS   AGE
node-agent-bzxqh          1/1     Running   0          3m35s
node-agent-jjdnn          1/1     Running   0          44m
node-agent-mljm7          1/1     Running   0          44m
velero-5c96d4c78b-tgpf7   1/1     Running   0          17m
```

**Validación del backend de almacenamiento de Velero**

```
usuario@kb360-master:~$ velero version
Client:
        Version: v1.15.0
        Git commit: 1d4f1475975b5107ec35f4d19ff17f7d1fcb3edf
Server:
        Version: v1.15.0
usuario@kb360-master:~$ velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX   PHASE       LAST VALIDATED                  ACCESS MODE   DEFAULT
default   aws        velero          Available   2026-05-07 19:40:36 +0200 CEST   ReadWrite     true
```

Aunque el proveedor aparece como aws, en realidad Velero está utilizando MinIO mediante compatibilidad S3.

Esto es completamente normal, ya que:

**Velero utiliza el plugin AWS/S3**
**MinIO implementa la API S3 de forma compatible**

### 6.5 Gestión de copias de seguridad

#### 6.5.1 Creación de backup

Una vez finalizada la instalación y validación funcional de Velero, se procedió a realizar la primera copia de seguridad del clúster Kubernetes.

El objetivo de esta prueba consistió en validar el funcionamiento del sistema de backups y comprobar la correcta integración entre Velero, MinIO y el almacenamiento persistente basado en NFS.

La creación inicial del backup se realizó mediante:

velero backup create backup-test

Velero realiza copias de seguridad lógicas del clúster Kubernetes, incluyendo:

- **recursos Kubernetes**
- **configuraciones YAML**
- **metadatos**
- **estado de aplicaciones**
- **persistencia asociada**

Durante las primeras pruebas, el backup finalizó en estado PartiallyFailed debido a que Velero intentaba generar snapshots sobre volúmenes NFS, funcionalidad no disponible en este entorno de laboratorio.

Este comportamiento es esperado en infraestructuras basadas en NFS, ya que los snapshots nativos están orientados principalmente a proveedores cloud o sistemas CSI compatibles.

Para adaptar el proceso al entorno utilizado, se generó una nueva copia de seguridad desactivando los snapshots de volúmenes mediante:

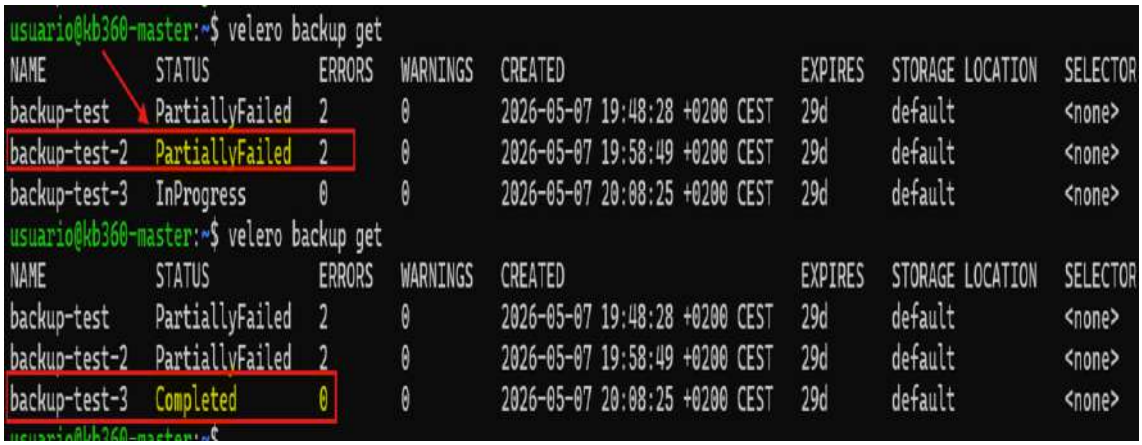
**velero backup create backup-test-3 --snapshot-volumes=false**

Posteriormente, se verificó el estado del backup utilizando:

**velero backup get**

El resultado final mostró el backup backup-test-3 en estado Completed, confirmando que la copia de seguridad se había realizado correctamente.

**Creación y validación inicial de backups con Velero**




El resultado final muestra el backup backup-test-3 en estado Completed, sin errores ni advertencias, confirmando que la copia de seguridad se ha realizado correctamente.

Durante la primera prueba de backup, Velero finalizó en estado PartiallyFailed debido a un intento de realizar snapshots sobre volúmenes NFS. Este comportamiento es esperado en un entorno de laboratorio basado en NFS, ya que los snapshots nativos están orientados a proveedores cloud o sistemas CSI compatibles. **Para adaptar el backup al entorno local, se generó una nueva copia desactivando los snapshots de volúmenes mediante --snapshot-volumes=false.**

### 6.5.2 Verificación del backup

Una vez generado el backup backup-test-3, se procedió a validar su contenido y estado mediante:

**velero backup describe backup-test-3 --details**

El resultado obtenido muestra el backup en estado Completed, sin errores ni advertencias, confirmando que Velero realizó correctamente la copia de seguridad del clúster Kubernetes.

Asimismo, se verificó que un total de 581 recursos Kubernetes fueron respaldados satisfactoriamente.

Durante esta validación también se confirmó que los snapshots nativos de volúmenes permanecían deshabilitados (Velero-Native Snapshot PVs: false), adaptación necesaria

debido al uso de almacenamiento persistente basado en NFS dentro del entorno del laboratorio.

**Objetivo de este paso**

Comprobar que el backup generado:

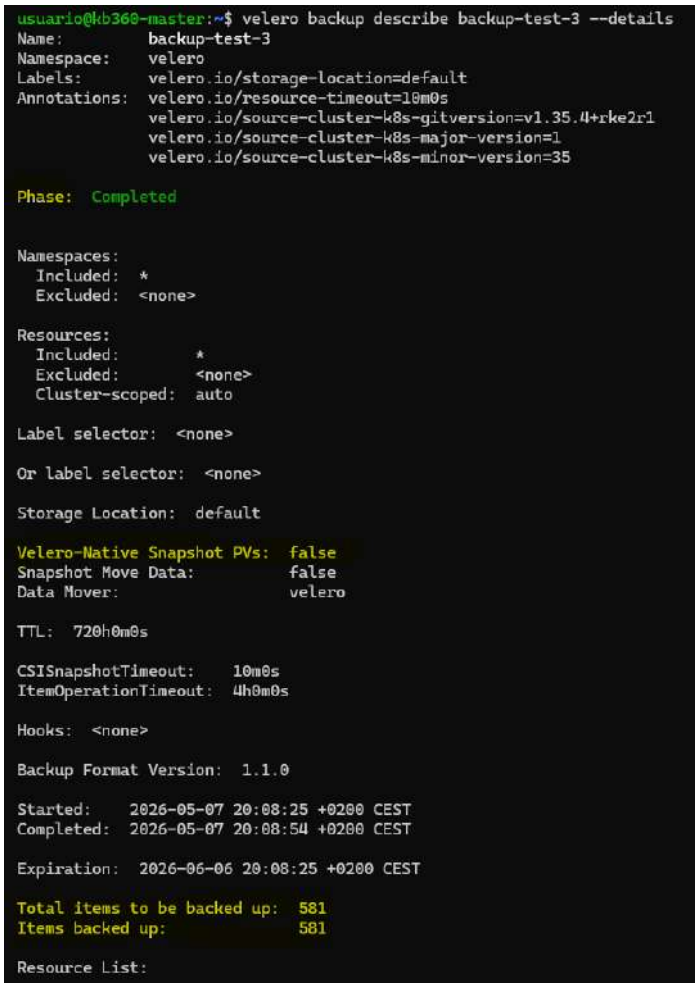
- existe correctamente

- contiene recursos Kubernetes

- está almacenado en MinIO

- puede utilizarse posteriormente para restauración

**Verificación detallada del backup backup-test-3**




La imagen muestra la validación detallada del backup backup-test-3, incluyendo el estado Completed, la cantidad total de recursos Kubernetes respaldados y la configuración utilizada para desactivar snapshots sobre volúmenes NFS.

### 6.6 Simulación de desastre y pérdida del servicio

#### 6.6.1 Verificación inicial del entorno

Antes de realizar la simulación de desastre, se procedió a verificar el estado operativo actual del clúster Kubernetes con el objetivo de comprobar el correcto funcionamiento de la aplicación desplegada previamente.

Durante esta validación se comprobó:

- existencia del Deployment
- pods activos
- servicio NodePort operativo
- estado funcional previo al fallo

Para ello, se ejecutaron los siguientes comandos:

**kubectl get deployments**

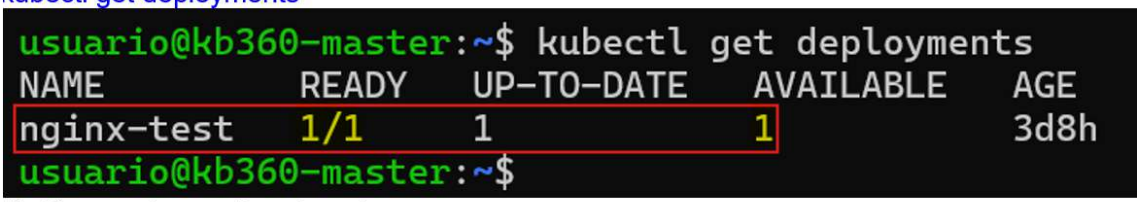
**kubectl get pods -o wide**

**kubectl get svc**

La validación confirmó que la aplicación nginx-test se encontraba desplegada y operativa dentro del clúster Kubernetes antes de iniciar la simulación del fallo controlado.

**Verificación inicial del entorno Kubernetes antes del desastre**

kubectl get deployments

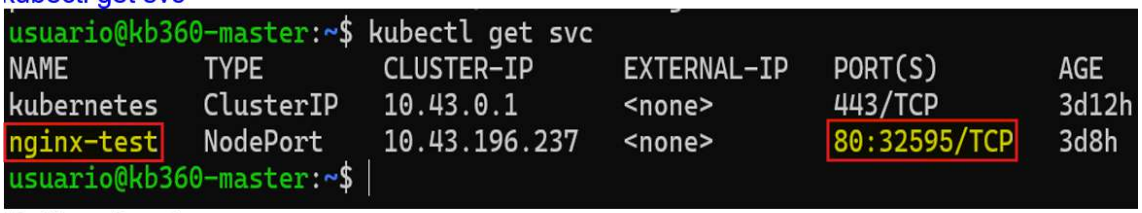



Verificar pods y nodo asignado

kubectl get pods -o wide


Verificar servicios

kubectl get svc




Verificar Puertos

La imagen muestra el estado inicial del clúster Kubernetes antes de realizar la simulación de desastre, incluyendo el Deployment activo, los pods en ejecución y el servicio NodePort operativo.

### 6.6.2 Simulación de desastre — Eliminación del Deployment

Una vez validado el entorno inicial, se procedió a simular un fallo real eliminando completamente el Deployment Kubernetes asociado a la aplicación desplegada.

El objetivo de esta prueba consistió en provocar la pérdida del servicio y validar posteriormente la capacidad de recuperación mediante Velero.

La eliminación del Deployment se realizó mediante:

**kubectl delete deployment nginx-test**

Posteriormente, se verificó el estado de los pods utilizando:

**kubectl get pods -o wide**

El resultado obtenido confirmó que Kubernetes eliminó correctamente el controlador Deployment y comenzó el proceso de finalización de los pods asociados.

Durante esta fase, el pod apareció temporalmente en estado Terminating, indicando que la carga estaba siendo destruida y que ya no existía ningún mecanismo de autorecuperación activo dentro del clúster.

**Eliminación del Deployment nginx-test**

kubectl delete deployment nginx-test

```
usuario@kb360-master:~$ kubectl delete deployment nginx-test
deployment.apps "nginx-test" deleted from default namespace
usuario@kb360-master:~$
```

**Verificar pods**

kubectl get pods -o wide

```
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                          READY   STATUS        RESTARTS   AGE    IP          NODE       NOMINATED NODE   READINESS GATE
S
nginx-test-6ff8854996-59q4s   0/1     Terminating   0          3d8h   10.42.3.6   kb360-w3   <none>           <none>
pod-nfs-test                  1/1     Running       0          43h    10.42.1.9   kb360-w1   <none>           <none>
```

La imagen muestra la eliminación del Deployment nginx-test y el proceso de finalización de los pods asociados durante la simulación de desastre controlado.

### 6.6.3 Validación de pérdida del servicio

Tras eliminar el Deployment, se procedió a validar la pérdida completa del servicio desplegado dentro del clúster Kubernetes.

Inicialmente, se verificó nuevamente el estado de los pods mediante:

**kubectl get pods -o wide**

Posteriormente, se comprobó el estado del Service NodePort:

**kubectl get svc**

Aunque el Service continuaba existiendo dentro de Kubernetes, este ya no disponía de pods backend asociados, por lo que la aplicación dejó de estar disponible.

Finalmente, se validó la pérdida real del servicio mediante una prueba HTTP utilizando:

**curl http://10.0.1.10:32595**

El resultado obtenido confirmó que la aplicación ya no respondía tráfico, demostrando que el servicio había perdido completamente sus endpoints internos.

**Validación de pérdida del servicio Kubernetes**

kubectl get pods -o wide

```
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                          READY   STATUS        RESTARTS   AGE    IP          NODE       NOMINATED NODE   READINESS GATE
S
nginx-test-6ff8854996-59q4s   0/1     Terminating   0          3d8h   10.42.3.6   kb360-w3   <none>           <none>
pod-nfs-test                  1/1     Running       0          43h    10.42.1.9   kb360-w1   <none>           <none>
usuario@kb360-master:~$
```

**El pod nginx-test debe desaparecer completamente.**

```
usuario@kb360-master:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
kubernetes   ClusterIP   10.43.0.1        <none>        443/TCP          3d12h
nginx-test   NodePort    10.43.196.237    <none>        80:32595/TCP     3d8h
usuario@kb360-master:~$
```

**Verificar que el servicio sigue existiendo**

**curl http://10.0.1.10:32595**

```
usuario@kb360-master:~$ curl http://10.0.1.10:32595
curl: (7) Failed to connect to 10.0.1.10 port 32595 after 0 ms: Couldn't connect to server
```

**Validar caída real del servicio**

La imagen muestra la pérdida completa de la aplicación tras eliminar el Deployment Kubernetes, validando que el servicio NodePort permanecía definido, pero sin pods backend disponibles.

**Qué demuestra esta prueba:**

- El Deployment era el responsable de mantener la aplicación
- El Service sin pods no puede funcionar
- La aplicación se ha perdido
- El sistema necesita recuperación mediante backup/restore

**Análisis técnico final de la prueba**

Estado del pod

Resultado:

**nginx-test-xxxxx  Terminating**

Esto confirma que:

- La carga ha sido destruida
- No existe Deployment que la recupere
- Kubernetes no recreará nuevos pods
- Estado del servicio

**CONCLUSIÓN TÉCNICA**

*La prueba de desastre controlado ha permitido validar el comportamiento del clúster Kubernetes ante la eliminación completa de una carga de trabajo gestionada mediante Deployment.*

*Esta situación demuestra la importancia de disponer de mecanismos de backup y recuperación, ya que la pérdida del controlador Deployment implica la desaparición completa de la aplicación dentro del clúster.*

*Asimismo, la prueba evidencia que un Service de Kubernetes no garantiza disponibilidad por sí mismo, dependiendo completamente de la existencia de pods backend operativos.*

*La simulación de desastre se ha completado correctamente, dejando el entorno preparado para la siguiente fase.*

## 6.7 Restauración mediante Velero

### 6.7.1 Verificación de backups disponibles

Antes de iniciar el proceso de restauración, se procedió a verificar la existencia de una copia de seguridad válida y disponible dentro de Velero.

Esta comprobación permite garantizar que:

- el backup existe correctamente
- el estado del backup es válido
- Velero puede acceder al almacenamiento MinIO
- la restauración podrá ejecutarse correctamente

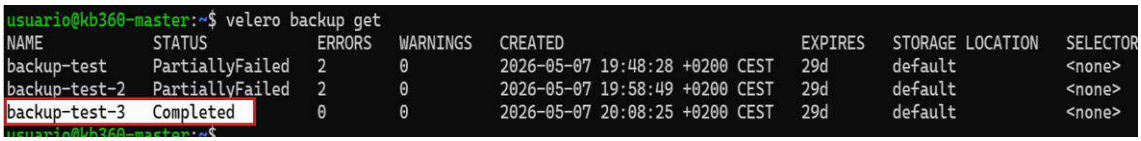
Para ello, se ejecutaron los siguientes comandos:

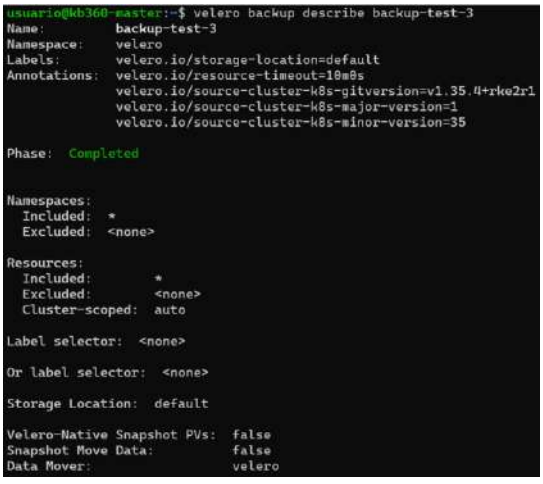
velero backup get

velero backup describe backup-test-3

El resultado obtenido confirmó que el backup backup-test-3 se encontraba en estado Completed, validando que la copia de seguridad estaba disponible para iniciar el proceso de recuperación.

**Verificación de backups disponibles en Velero**



La imagen muestra la validación del backup backup-test-3 almacenado en MinIO y disponible para ser utilizado durante el proceso de restauración del clúster Kubernetes.

**Esta comprobación es fundamental para garantizar que:**

- el backup existe
- el estado es correcto
- Velero puede acceder al almacenamiento
- la restauración podrá ejecutarse correctamente

### 6.7.2 Creación de la restauración

Una vez validado el backup disponible, se procedió a iniciar el proceso de restauración mediante Velero utilizando la copia de seguridad previamente generada.

La restauración se ejecutó mediante:

velero restore create restore-test-1 --from-backup backup-test-3

Durante este proceso, Velero recuperó automáticamente los recursos Kubernetes almacenados en el backup, incluyendo:

- Deployments
- Pods
- Services
- configuraciones asociadas
- recursos del clúster

Posteriormente, se verificó el estado de la restauración utilizando:

velero restore get

El resultado obtenido mostró la restauración en estado Completed, confirmando que el proceso se había ejecutado correctamente y sin errores.

**Ejecución del proceso de restauración con Velero**


La imagen muestra la ejecución del proceso de restauración mediante Velero y la validación posterior del estado Completed del restore generado.

**Esto significa que:**

Kubernetes recibió los recursos restaurados
Velero pudo recrear objetos del clúster
la recuperación se completó correctamente

ERRORS = 0

**la restauración es válida y funcional.**

### 6.7.3 Validación de recuperación del clúster

Una vez finalizada la restauración, se procedió a validar la recuperación funcional de los recursos eliminados previamente durante la simulación de desastre.

Inicialmente, se verificó la recuperación del Deployment mediante:

**kubectl get deployments**

Posteriormente, se comprobó la recreación automática de los pods:

**kubectl get pods -o wide**

También se validó el estado del Service NodePort asociado a la aplicación:

**kubectl get svc**

Finalmente, se realizó una prueba HTTP utilizando:

**curl http://10.0.1.10:32595**

El resultado obtenido confirmó que la aplicación volvió a estar completamente operativa tras la restauración realizada mediante Velero.

**Validación de recuperación de recursos Kubernetes**

```
usuario@kb360-master:~$ kubectl get deployments
NAME         READY   UP-TO-DATE   AVAILABLE   AGE
nginx-test   1/1     1            1           10m
```

**Verificar pods recuperados**

```
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                          READY   STATUS        RESTARTS   AGE    IP           NODE       NOMINATED NODE   READINESS GATES
nginx-test-6ff8854996-59g4s   0/1     Terminating   0          3d9h   10.42.3.6    kb360-w3   <none>           <none>
nginx-test-6ff8854996-kzvpb   1/1     Running       0          12m    10.42.1.14   kb360-w1   <none>           <none>
pod-nfs-test                  1/1     Running       0          44h    10.42.1.9    kb360-w1   <none>           <none>
```

**Verificar servicios restaurados**

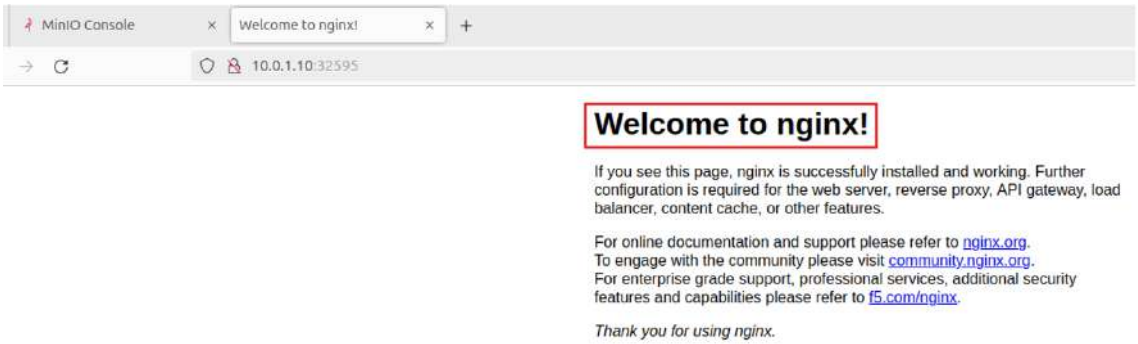
```
usuario@kb360-master:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubernetes   ClusterIP   10.43.0.1       <none>        443/TCP        3d13h
nginx-test   NodePort    10.43.196.237   <none>        80:32595/TCP   3d9h
```

**Validar recuperación real del servicio**

La imagen muestra la recuperación automática del Deployment, los pods y el Service NodePort tras ejecutar la restauración mediante Velero.

**Validación funcional de la aplicación restaurada**

```
usuario@kb360-master:~$ curl http://10.0.1.10:32595
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
```




La imagen muestra la validación funcional de la aplicación restaurada tras el proceso de recuperación, confirmando que el servicio volvió a responder correctamente mediante el Service NodePort.

- Velero recuperó correctamente el Deployment
- Kubernetes recreó la aplicación
- el Service volvió a tener endpoints activos
- la aplicación volvió a estar disponible
- el sistema DRP funciona correctamente

### 6.7.4 Conclusión de la restauración

La restauración realizada mediante Velero permitió recuperar correctamente los recursos eliminados del clúster Kubernetes utilizando la copia de seguridad almacenada previamente en MinIO.

Durante el proceso, Kubernetes recreó automáticamente los pods asociados al Deployment restaurado y recuperó el acceso funcional a la aplicación mediante el Service NodePort configurado previamente.

La validación final confirmó que:

- Velero recuperó correctamente los recursos Kubernetes
- El Deployment volvió a estar operativo
- Los pods fueron recreados automáticamente
- El servicio NodePort recuperó conectividad
- La aplicación volvió a responder correctamente

***Esta prueba valida el correcto funcionamiento del sistema de recuperación ante desastres (Disaster Recovery) implementado en el proyecto KubeBackup360, demostrando que es posible restaurar aplicaciones y servicios críticos del clúster***

***a partir de copias de seguridad almacenadas externamente mediante Velero y MinIO.***

### 6.7.5 Validación final del restore

#### 6.7.5.1 Verificación final del estado del clúster

Una vez completado el proceso de restauración mediante Velero, se procedió a realizar una validación final del estado general del clúster Kubernetes con el objetivo de comprobar la estabilidad del entorno tras la recuperación ante desastres.

Durante esta fase se verificó que:

- los recursos restaurados permanecían operativos
- los pods funcionaban correctamente
- los servicios respondían correctamente
- el almacenamiento persistente seguía disponible
- el clúster mantenía estabilidad tras el proceso DRP

Inicialmente, se comprobó el estado de los nodos Kubernetes mediante:

**kubectl get nodes**

Posteriormente, se verificó el estado general de los pods del clúster:

**kubectl get pods -A**

A continuación, se validó la persistencia del almacenamiento NFS comprobando la disponibilidad de los datos almacenados previamente:

**kubectl exec -it pod-nfs-test -- cat /data/test.txt**

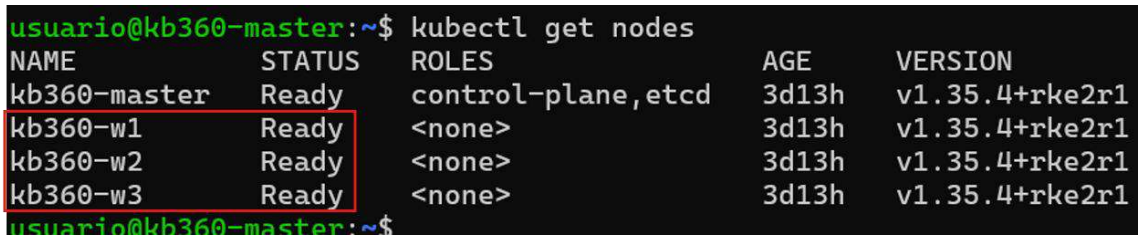
Finalmente, se verificó el estado operativo de Velero y las restauraciones realizadas mediante:

**velero backup get**

**velero restore get**

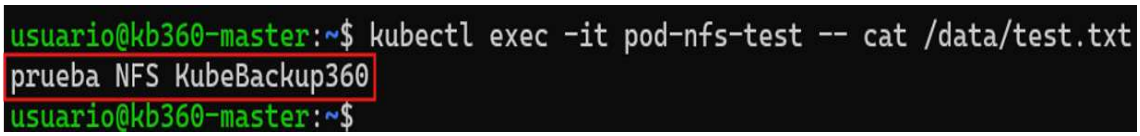
Los resultados obtenidos confirmaron que tanto los backups como las restauraciones permanecían disponibles y en estado funcional dentro del entorno Kubernetes.

**Verificación final del estado del clúster Kubernetes**




La imagen muestra la validación final del estado operativo del clúster Kubernetes tras completar el proceso de recuperación mediante Velero.

**Validación de persistencia NFS tras la restauración**




La imagen muestra la validación de los datos almacenados sobre el sistema NFS, confirmando que la persistencia permaneció intacta tras el proceso de desastre y recuperación.

**Verificación final de backups y restores en Velero**

La imagen muestra la validación final de los backups y restauraciones gestionadas mediante Velero dentro del clúster Kubernetes.

**Conclusión técnica de la validación final**

La validación final del proceso de restauración confirma que el sistema de recuperación ante desastres implementado en el proyecto KubeBackup360 funciona correctamente y permite recuperar aplicaciones y servicios del clúster Kubernetes tras una pérdida simulada de recursos.

Durante las pruebas realizadas se verificó que:

- todos los nodos del clúster permanecen operativos
- los componentes críticos de Kubernetes funcionan correctamente
- la red interna del clúster continúa disponible
- el sistema de persistencia NFS mantiene la integridad de los datos
- Velero y MinIO continúan operativos tras la restauración

Asimismo, se confirmó que el Deployment restaurado volvió a desplegar correctamente la aplicación NGINX, recuperando el acceso al servicio mediante NodePort.

La persistencia de datos también fue validada satisfactoriamente, demostrando que los volúmenes persistentes asociados al almacenamiento NFS conservaron la información tras el proceso de desastre y recuperación.

*En conjunto, las pruebas realizadas demuestran que la infraestructura implementada proporciona mecanismos funcionales de backup, restauración y recuperación ante desastres, cumpliendo correctamente los objetivos definidos para el proyecto KubeBackup360.*

### 6.8 Automatización de backups

#### 6.8.1 Programación de copias de seguridad

Una vez validado el funcionamiento general del sistema de backup y restauración, se procedió a configurar copias de seguridad automáticas mediante Velero.

El objetivo de esta fase consiste en automatizar la generación periódica de backups del clúster Kubernetes, permitiendo mantener mecanismos de recuperación ante desastres sin necesidad de intervención manual.

La programación automática de backups se realizó mediante:

**velero schedule create backup-diario --schedule="0 0 * * *"**

Esta configuración genera automáticamente una copia de seguridad diaria utilizando la sintaxis cron integrada en Velero.

Antes de activar la automatización, se verificó el estado operativo de los componentes de Velero mediante:

**kubectl get pods -n velero**

La validación confirmó que el deployment de Velero y los componentes node-agent permanecían en estado Running.

Posteriormente, se verificó el estado del backend de almacenamiento utilizando:

**velero backup-location get**

El resultado obtenido confirmó que el bucket velero continuaba disponible y accesible mediante MinIO.

Finalmente, se comprobó la creación correcta de la programación automática mediante:

**velero schedule get**

La validación mostró el schedule backup-diario correctamente registrado dentro de Velero.

**Verificación de componentes de Velero antes de la automatización**

```
usuario@kb360-master:~$ kubectl get pods -n velero
NAME                      READY   STATUS              RESTARTS   AGE
node-agent-bzxqh          1/1     Running             0          5h58m
node-agent-jjdnn          1/1     Running             0          6h39m
node-agent-mljm7          1/1     Running             0          6h39m
node-agent-q8p9p          0/1     ContainerCreating   0          14m
velero-5c96d4c78b-tgpf7   1/1     Running             0          6h11m
```

```
usuario@kb360-master:~$ velero schedule get
usuario@kb360-master:~$ kubectl get pods -n velero
NAME                      READY   STATUS    RESTARTS   AGE
node-agent-bzxqh          1/1     Running   0          6h12m
node-agent-jjdnn          1/1     Running   0          6h53m
node-agent-mljm7          1/1     Running   0          6h53m
node-agent-q8p9p          1/1     Running   0          29m
velero-5c96d4c78b-tgpf7   1/1     Running   0          6h26m
```

La imagen muestra los componentes operativos de Velero dentro del namespace velero, verificando que el sistema de backups se encuentra funcional antes de activar las copias automáticas.

**Validación del backend de almacenamiento MinIO**

```
usuario@kb360-master:~$ velero backup-location get
NAME      PROVIDER   BUCKET/PREFIX   PHASE       LAST VALIDATED                  ACCESS MODE   DEFAULT
default   aws        velero          Available   2026-05-08 01:34:23 +0200 CEST   ReadWrite     true
```

La imagen muestra la validación del backend de almacenamiento utilizado por Velero, confirmando la disponibilidad del bucket velero dentro de MinIO.

*Esta validación garantiza que el sistema puede ejecutar backups programados de forma automática.*

### 6.8.2 Creación del primer backup automático programado

Una vez validado el funcionamiento general de Velero, se procedió a configurar un sistema de copias de seguridad automáticas mediante Velero Schedule.

El objetivo de esta configuración consiste en automatizar la generación periódica de backups del clúster Kubernetes, reduciendo la intervención manual y mejorando los mecanismos de recuperación ante desastres.

Velero utiliza expresiones cron para programar la ejecución automática de copias de seguridad.

En este caso, se utilizó la expresión:

**0 */6 * * ***

Esta configuración indica que el backup se ejecutará automáticamente cada 6 horas.

Dentro del entorno de laboratorio, esta frecuencia permite:

- validar rápidamente el funcionamiento automático
- generar evidencias de ejecución periódica
- comprobar la integración con MinIO
- simular un entorno DRP real

La creación del schedule se realizó mediante:

**velero schedule create daily-backup **

**--schedule="0 */6 * * *" **

**--snapshot-volumes=false**

Durante la configuración se deshabilitaron los snapshots de volúmenes mediante la opción:

**--snapshot-volumes=false**

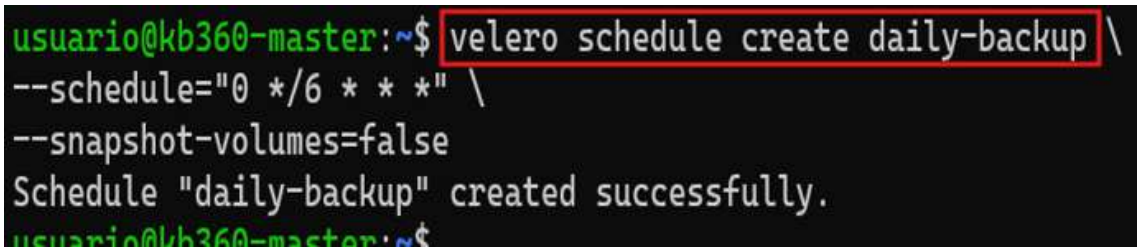
Esta configuración evita los errores detectados anteriormente relacionados con snapshots incompatibles sobre almacenamiento NFS.

Posteriormente, se verificó el estado de los schedules configurados utilizando:

**velero schedule get**

El resultado confirmó que el schedule daily-backup había sido registrado correctamente dentro de Velero.

**Creación del schedule automático daily-backup**




Nombre del schedule**: daily-backup**
Frecuencia: **cada 6 horas**
Snapshot desactivado: --snapshot-volumes=**false Esto evita el error anterior relacionado con NFS y AWS snapshots**
La imagen muestra la creación del schedule automático daily-backup, configurado para ejecutar copias de seguridad periódicas cada 6 horas.

**Verificación de schedules configurados en Velero**


La imagen muestra la validación del schedule daily-backup dentro de Velero tras completar la configuración automática de backups.

### 6.8.3 Validación de ejecución automática de backups

Una vez configurado el sistema de automatización, se procedió a validar el funcionamiento del schedule generado mediante Velero.

El objetivo de esta prueba consiste en comprobar que:

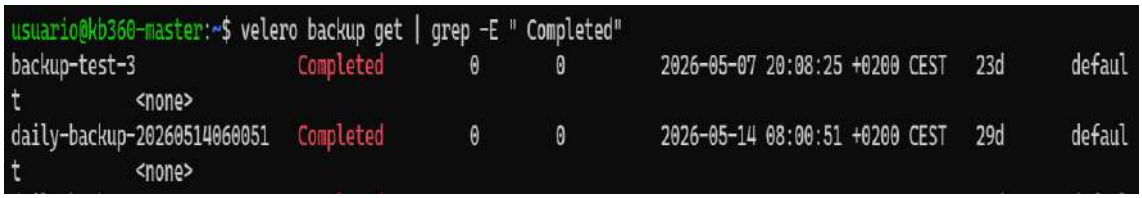
- Velero ejecuta correctamente el schedule
- los backups se generan automáticamente
- las copias aparecen registradas
- MinIO almacena correctamente los backups
- el sistema DRP automatizado funciona correctamente

Para ello, se ejecutó manualmente un backup basado en el schedule configurado mediante:

**velero backup create --from-schedule daily-backup**

Posteriormente, se verificó el estado de los backups generados utilizando:

velero backup get




El resultado obtenido confirmó que el backup fue completado correctamente sin errores ni advertencias:

**ERRORS: 0**

**WARNINGS: 0**

Esto valida que:

- la comunicación con MinIO funciona correctamente
- Velero permanece estable
- los backups generados son consistentes
- el sistema automático de copias funciona correctamente

La imagen muestra la ejecución automática de backups mediante el schedule daily-backup y la validación posterior del estado correcto de las copias generadas.

**Conclusión técnica de la automatización de backups**

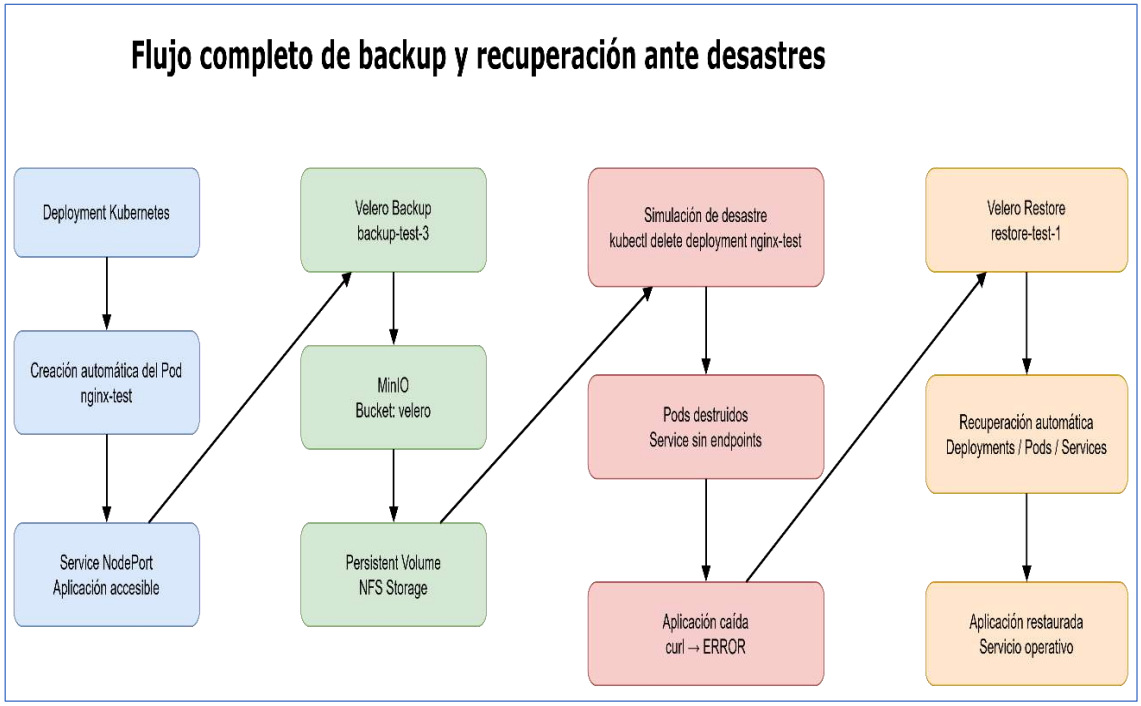
Se ha implementado correctamente un sistema de automatización de copias de seguridad mediante Velero Schedule, permitiendo la ejecución periódica de backups del clúster Kubernetes sin intervención manual.

Para ello, se configuró un schedule denominado daily-backup, programado mediante una expresión cron para ejecutarse automáticamente cada 6 horas.

Posteriormente, se validó el correcto funcionamiento del sistema observando la creación automática de nuevos backups generados por Velero, almacenados correctamente en MinIO y completados sin errores ni advertencias.

Esta configuración proporciona un mecanismo continuo de protección y recuperación ante desastres, mejorando significativamente la resiliencia del clúster Kubernetes y reduciendo el riesgo de pérdida de información ante fallos del sistema.

*Asimismo, la automatización implementada reproduce prácticas habituales de entornos empresariales reales, donde las copias de seguridad programadas constituyen un elemento esencial dentro de cualquier estrategia de continuidad de servicio y Disaster Recovery*.




***El diagrama muestra el flujo completo del sistema DRP implementado en KubeBackup360, incluyendo el despliegue inicial de la aplicación, la generación de backups mediante Velero, el almacenamiento persistente en MinIO y NFS, la simulación de desastre y la posterior recuperación automática de los recursos Kubernetes.***

### 6.9 Monitorización del clúster (Prometheus y Grafana)

Con el objetivo de incorporar capacidades de observabilidad y monitorización dentro del entorno Kubernetes del proyecto KubeBackup360, se implementó una plataforma de monitorización basada en Prometheus y Grafana.

La arquitectura desplegada permite supervisar en tiempo real el estado del clúster Kubernetes, incluyendo métricas relacionadas con:

- utilización de CPU y memoria
- estado de nodos y Pods
- consumo de recursos
- disponibilidad de servicios
- funcionamiento interno del clúster

Para simplificar el despliegue de los distintos componentes de monitorización se utilizó Helm, herramienta ampliamente utilizada en entornos Kubernetes para automatizar instalaciones complejas mediante charts oficiales.

La plataforma implementada integra:

- Prometheus como sistema de recopilación de métricas
- Grafana como plataforma de visualización y dashboards
- exporters Kubernetes para monitorización de nodos y servicios
- dashboards gráficos para observabilidad en tiempo real

La incorporación de esta infraestructura permite disponer de mecanismos de supervisión similares a los utilizados en entornos cloud-native y plataformas Kubernetes empresariales.

### 6.9.1 Verificación inicial del estado del clúster antes de monitorización

Antes de desplegar la plataforma de monitorización basada en Prometheus y Grafana, se realizó una validación inicial del estado operativo del clúster Kubernetes con el objetivo de comprobar:

- estado de los nodos
- funcionamiento general de los Pods
- disponibilidad de métricas Kubernetes
- estabilidad general del entorno

Esta validación resulta importante para garantizar que la infraestructura se encuentra completamente estable antes de incorporar nuevos componentes de observabilidad y monitorización.

Inicialmente, se verificó el estado de los nodos Kubernetes mediante:

**kubectl get nodes**

La validación confirmó que todos los nodos del clúster aparecían en estado Ready, indicando el correcto funcionamiento del plano de control Kubernetes y de los nodos worker desplegados dentro del entorno RKE2.

**Verificación inicial de nodos Kubernetes**

```
usuario@kb360-master:~$ kubectl get nodes
NAME          STATUS   ROLES                  AGE    VERSION
kb360-master  Ready    control-plane,etcd     4d7h   v1.35.4+rke2r1
kb360-w1      Ready    <none>                 4d7h   v1.35.4+rke2r1
kb360-w2      Ready    <none>                 4d7h   v1.35.4+rke2r1
kb360-w3      Ready    <none>                 4d7h   v1.35.4+rke2r1
usuario@kb360-master:~$
```

Posteriormente, se comprobó el estado general de los Pods desplegados dentro del clúster mediante:
**kubectl get pods -A**

La salida permitió verificar que los componentes principales del entorno Kubernetes se encontraban en estado Running o Completed, incluyendo:

- persistencia NFS
- MinIO
- Velero

- monitorización Kubernetes
- componentes internos de RKE2

Asimismo, esta validación confirmó que los distintos namespaces y servicios desplegados durante fases anteriores permanecían operativos y correctamente integrados dentro del clúster.

**Estado general de Pods del clúster Kubernetes**

```
usuario@kb360-master:~$ kubectl get pods -A
NAMESPACE     NAME                                                READY   STATUS      RESTARTS      AGE
default       nginx-test-6ff8854996-kzvpb                         1/1     Running     0             18h
default       pod-nfs-test                                        1/1     Running     0             2d15h
kube-system   cloud-controller-manager-kb360-master               1/1     Running     0             4d7h
kube-system   etcd-kb360-master                                   1/1     Running     0             4d7h
kube-system   helm-install-rke2-canal-sg4ph                       0/1     Completed   0             4d7h
kube-system   helm-install-rke2-coredns-kksrj                     0/1     Completed   0             4d7h
kube-system   helm-install-rke2-ingress-nginx-cs5sw               0/1     Completed   0             4d7h
kube-system   helm-install-rke2-metrics-server-bnsbn              0/1     Completed   0             4d7h
kube-system   helm-install-rke2-runtimeclasses-87wn4              0/1     Completed   0             4d7h
kube-system   helm-install-rke2-snapshot-controller-457nh         0/1     Completed   2             4d7h
kube-system   helm-install-rke2-snapshot-controller-crd-wptss     0/1     Completed   0             4d7h
kube-system   kube-apiserver-kb360-master                         1/1     Running     0             4d7h
kube-system   kube-controller-manager-kb360-master                1/1     Running     0             4d7h
kube-system   kube-proxy-kb360-master                             1/1     Running     0             4d7h
kube-system   kube-proxy-kb360-w1                                 1/1     Running     0             2d15h
kube-system   kube-proxy-kb360-w2                                 1/1     Running     0             4d7h
kube-system   kube-proxy-kb360-w3                                 1/1     Running     0             4d7h
kube-system   kube-scheduler-kb360-master                         1/1     Running     0             4d7h
kube-system   rke2-canal-khc4f                                    2/2     Running     0             23h
kube-system   rke2-canal-n9fzh                                    2/2     Running     0             17h
kube-system   rke2-canal-nvcx4                                    2/2     Running     0             29h
kube-system   rke2-canal-vtdlv                                    2/2     Running     0             4d7h
kube-system   rke2-coredns-rke2-coredns-75bc4f545d-794d9          1/1     Running     0             4d7h
kube-system   rke2-coredns-rke2-coredns-75bc4f545d-nghlq          1/1     Running     0             31h
kube-system   rke2-coredns-rke2-coredns-autoscaler-79774bc964-8fhdp   1/1     Running     0             4d7h
kube-system   rke2-ingress-nginx-controller-59cqr                 1/1     Running     0             4d7h
kube-system   rke2-ingress-nginx-controller-6cp52                 1/1     Running     0             4d7h
kube-system   rke2-ingress-nginx-controller-dsv4n                 1/1     Running     0             4d7h
kube-system   rke2-ingress-nginx-controller-fn7dc                 1/1     Running     0             4d7h
kube-system   rke2-metrics-server-685458d954-l7jcz                1/1     Running     0             4d7h
kube-system   rke2-snapshot-controller-7df49fb687-nxlp6           1/1     Running     1 (31h ago)   4d7h
minio         minio-689d4cb9b4-hqbv9                              1/1     Running     0             29h
monitoring    monitoring-grafana-55966d7c6f-n9hhn                 3/3     Running     3 (87m ago)   5h18m
monitoring    monitoring-kube-prometheus-operator-54f68d65b4-47dqn   1/1     Running     0             4h57m
monitoring    monitoring-kube-state-metrics-5957bd45bc-nz26j      1/1     Running     0             4h57m
monitoring    monitoring-prometheus-node-exporter-29gmp           1/1     Running     0             5h18m
monitoring    monitoring-prometheus-node-exporter-frphl           1/1     Running     0             5h18m
monitoring    monitoring-prometheus-node-exporter-m5txm           1/1     Running     0             5h18m
monitoring    monitoring-prometheus-node-exporter-swm2s           1/1     Running     0             5h18m
monitoring    prometheus-monitoring-kube-prometheus-prometheus-0  2/2     Running     0             5h12m
velero        node-agent-bzxqh                                    1/1     Running     0             23h
velero        node-agent-jjdnn                                    1/1     Running     0             24h
velero        node-agent-mljm7                                    1/1     Running     0             24h
```

Entre los componentes más importantes validados durante esta comprobación destacan:

**Persistencia**
- pod-nfs-test

**MinIO**
- minio

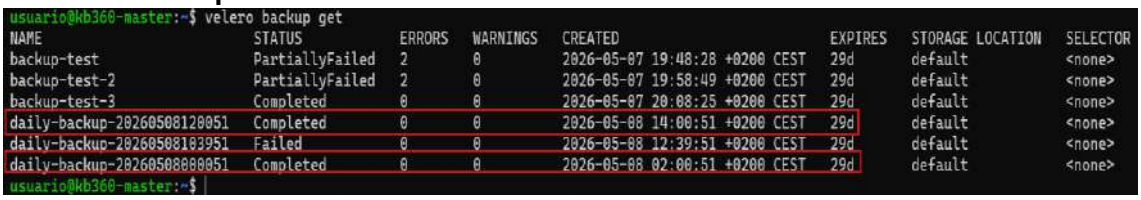
**Velero**
- velero
- node-agent

**Monitorización**
- grafana
- prometheus
- node-exporter

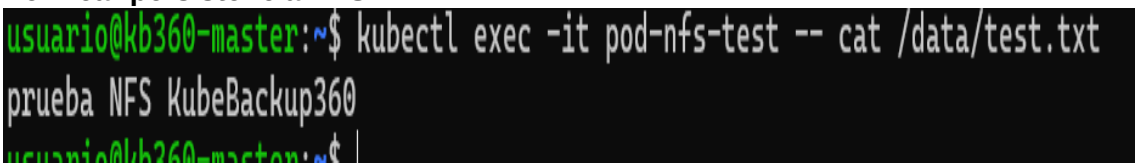
Finalmente, se verificó la disponibilidad de métricas Kubernetes mediante:
**kubectl top nodes**

Esta comprobación permitió validar que el servicio Metrics Server funcionaba correctamente y que Kubernetes era capaz de recopilar métricas de CPU y memoria de los nodos del clúster.
Asimismo, esta validación resulta fundamental para garantizar que Prometheus podrá recopilar métricas correctamente una vez desplegada la plataforma de monitorización.

**Verificar backups automáticos**




**Verificar persistencia NFS**




Las validaciones realizadas confirmaron que el entorno Kubernetes se encontraba completamente operativo, estable y preparado para desplegar la plataforma de observabilidad basada en Prometheus y Grafana dentro del proyecto KubeBackup360.

### 6.9.2 Acceso web y validación gráfica de Prometheus y Grafana

**Objetivo:**

Validar el funcionamiento real de la plataforma de monitorización desplegada en Kubernetes mediante el acceso web a Grafana y Prometheus desde un equipo cliente externo.
Esta validación permitió comprobar:

- conectividad de red entre cliente y clúster Kubernetes
- funcionamiento de servicios NodePort
- integración entre Grafana y Prometheus
- recopilación de métricas Kubernetes
- visualización gráfica en tiempo real

**Configuración de acceso remoto mediante túneles SSH**

Para acceder a las interfaces web de Grafana y Prometheus desde el equipo cliente se utilizaron túneles SSH hacia el nodo master Kubernetes.

La conexión se realizó mediante:

**ssh -L 30080:10.0.1.10:30080 -L 30090:10.0.1.10:30090 usuario@192.168.0.75**

Esta configuración permitió redirigir localmente los puertos NodePort desplegados en Kubernetes:

- **Grafana → puerto 30080**

• **Prometheus → puerto 30090**

De esta forma, el navegador del equipo cliente pudo acceder directamente a los servicios de monitorización utilizando:

**Grafana:**
**http://localhost:30080**

**Prometheus:**
**http://localhost:30090**

La utilización de túneles SSH permitió establecer un puente seguro entre la red cliente y la red interna Kubernetes, evitando exponer directamente los servicios de monitorización hacia el exterior.

**Configuración de acceso remoto a Grafana y Prometheus mediante túneles SSH**

ssh -L 30080:10.0.1.10:30080 -L 30090:10.0.1.10:30090 usuario@192.168.0.75
**navegador:**

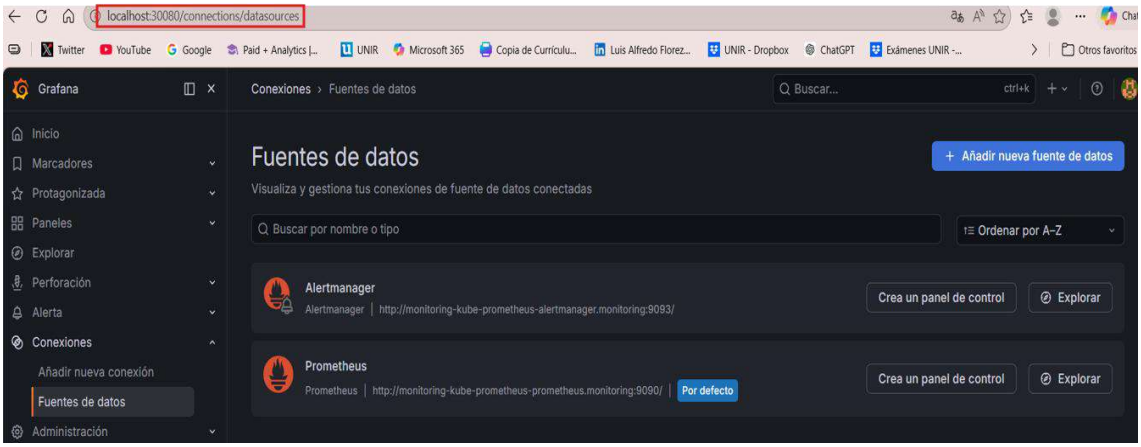

**Acceso web a Grafana**

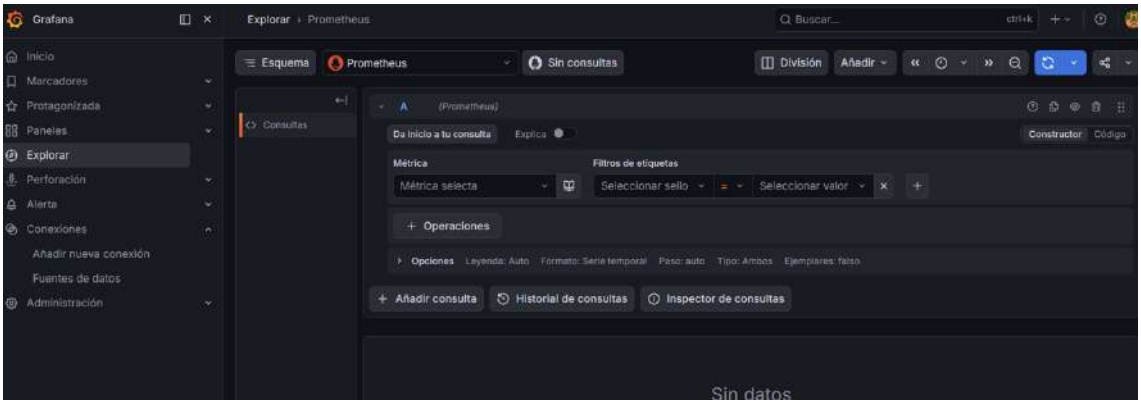
Posteriormente, se accedió a la interfaz web de Grafana desde el navegador del equipo cliente verificando el correcto funcionamiento del servicio.

Durante esta validación se comprobó:

• acceso web funcional
• autenticación correcta
• disponibilidad de la interfaz gráfica
• funcionamiento del servicio NodePort Kubernetes

Grafana: http://localhost:30080



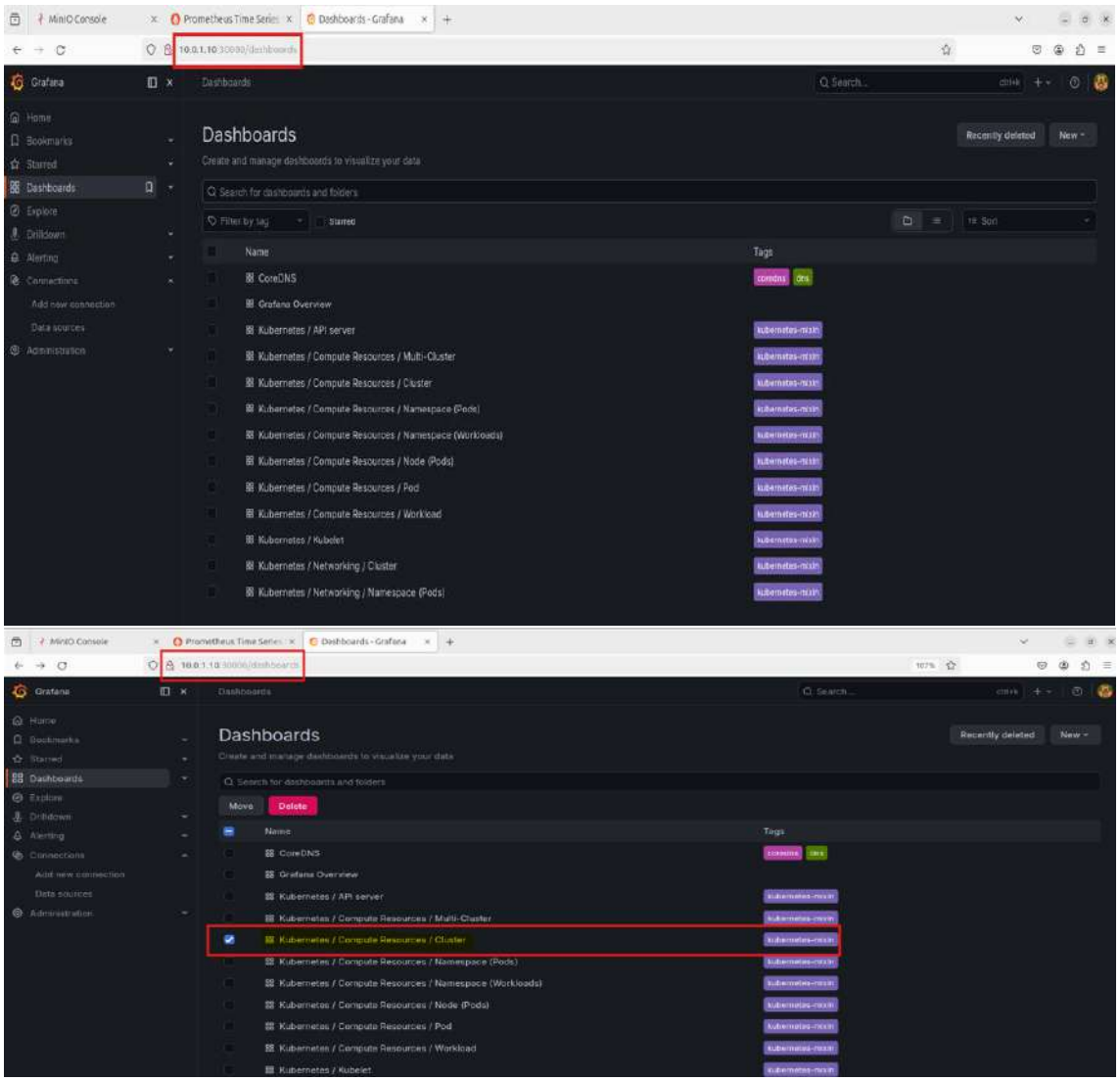

**Validación de datasource Prometheus**

Una vez iniciada sesión en Grafana, se verificó la integración entre Grafana y Prometheus desde el apartado Data Sources.

Durante la validación se comprobó que Grafana podía comunicarse correctamente con Prometheus y consultar métricas procedentes del clúster Kubernetes.

La comprobación confirmó el correcto funcionamiento de la datasource Prometheus utilizada por Grafana.

Prometheus: http://localhost:30090




**Esto valida que:**

• Grafana se comunica correctamente con Prometheus
• Prometheus recopila métricas del clúster Kubernetes
• la plataforma de monitorización permanece operativa

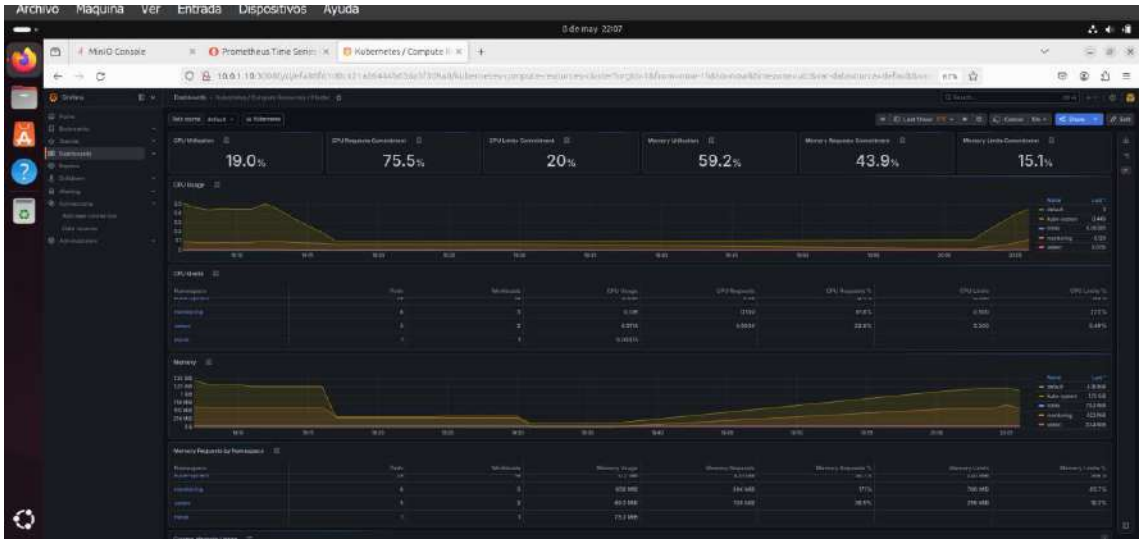
**Validación de dashboards Kubernetes**

Posteriormente, se verificó el funcionamiento de los dashboards Kubernetes desplegados automáticamente dentro de Grafana.

Durante esta validación se visualizaron métricas relacionadas con:

• utilización de CPU
• consumo de memoria RAM
• estado de Pods Kubernetes
• utilización de recursos del clúster
• actividad general de los nodos

Inicialmente, se comprobó la disponibilidad de dashboards Kubernetes desde el apartado Dashboards de Grafana.




### 6.10 Validación técnica final

Una vez completadas todas las tareas correspondientes a la Fase 3, se realizó una validación global de la infraestructura Kubernetes con el objetivo de comprobar el funcionamiento integrado de todos los componentes desplegados dentro del entorno KubeBackup360.

Durante esta validación se verificó:

- estado operativo del clúster Kubernetes
- persistencia NFS
- funcionamiento de MinIO y Velero
- automatización de backups
- recuperación ante desastres
- monitorización mediante Prometheus y Grafana

Inicialmente, se comprobó el estado general del clúster Kubernetes verificando que todos los nodos permanecían en estado Ready, garantizando la estabilidad y disponibilidad de la infraestructura RKE2.

Posteriormente, se validó el estado de los Pods desplegados dentro del clúster, confirmando el correcto funcionamiento de los componentes críticos implementados durante la fase:

- persistencia NFS
- MinIO
- Velero
- Prometheus
- Grafana
- node-exporters
- servicios internos Kubernetes

Asimismo, se verificó el correcto funcionamiento del sistema automatizado de backups mediante Velero, observando la generación periódica de copias de seguridad almacenadas correctamente en MinIO.

Esto valida que:

• la comunicación con MinIO funciona correctamente
• Velero permanece estable
• los backups generados son consistentes
• el sistema automático de copias funciona correctamente

Del mismo modo, se confirmó la persistencia de datos mediante NFS, validando la conservación de información previamente almacenada sobre volúmenes persistentes Kubernetes.

Esto confirma:

• persistencia funcional
• almacenamiento NFS operativo
• conservación correcta de datos

Finalmente, se comprobó el funcionamiento de la plataforma de monitorización y observabilidad basada en Prometheus y Grafana, verificando la recopilación activa de métricas y la monitorización correcta de múltiples endpoints Kubernetes.

Esto confirma:

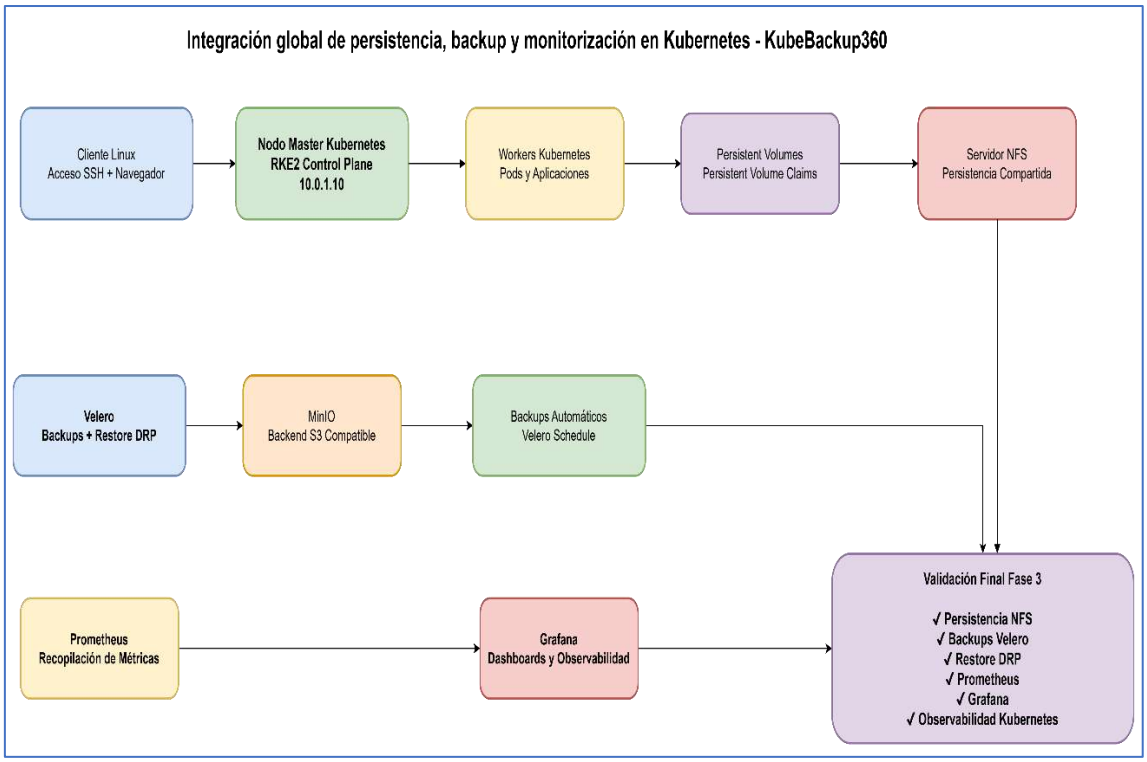
• Prometheus recopila métricas correctamente
• Grafana visualiza métricas en tiempo real
• los endpoints Kubernetes permanecen monitorizados
• la plataforma de observabilidad funciona correctamente

Con todo ello, quedó validado el funcionamiento conjunto de toda la infraestructura desplegada durante la Fase 3, incluyendo persistencia, backup, recuperación, automatización y monitorización del clúster Kubernetes.

**Arquitectura global de persistencia, DRP y observabilidad en Kubernetes - KubeBackup360**

La siguiente figura representa la integración global de todos los componentes desplegados durante la Fase 3 del proyecto KubeBackup360, incluyendo persistencia NFS, almacenamiento S3 mediante MinIO, backups y restauración con Velero, así como la plataforma de monitorización y observabilidad basada en Prometheus y Grafana.

El diagrama resume el flujo funcional completo de la infraestructura Kubernetes implementada, mostrando la relación entre persistencia, automatización de backups, recuperación ante desastres y monitorización del clúster.




***El diagrama representa la integración conjunta de almacenamiento persistente, Disaster Recovery y observabilidad dentro del clúster Kubernetes desplegado mediante RKE2 durante la Fase 3 del proyecto KubeBackup360.***

## 7. CONTENERIZACIÓN Y GESTIÓN DE IMÁGENES DOCKER

Tras completar la implementación de persistencia, backup, recuperación ante desastres y monitorización dentro del clúster Kubernetes, se inició la fase de contenerización del proyecto KubeBackup360 mediante Docker.

El objetivo principal de esta fase consiste en preparar aplicaciones contenerizadas para su posterior despliegue dentro de Kubernetes utilizando imágenes Docker y repositorios compatibles con entornos cloud-native.

La contenerización permite empaquetar aplicaciones junto con todas sus dependencias dentro de imágenes portables y reproducibles, facilitando su distribución y despliegue tanto en entornos locales como en infraestructuras Kubernetes.

Durante esta fase se trabajará principalmente con:

• Docker Engine

• imágenes Docker

• contenedores

• Docker Hub

• Dockerfile

• aplicaciones web contenerizadas

Docker será utilizado para:

• construir imágenes de aplicaciones

• ejecutar contenedores localmente

• validar el funcionamiento de servicios web

• publicar imágenes en Docker Hub

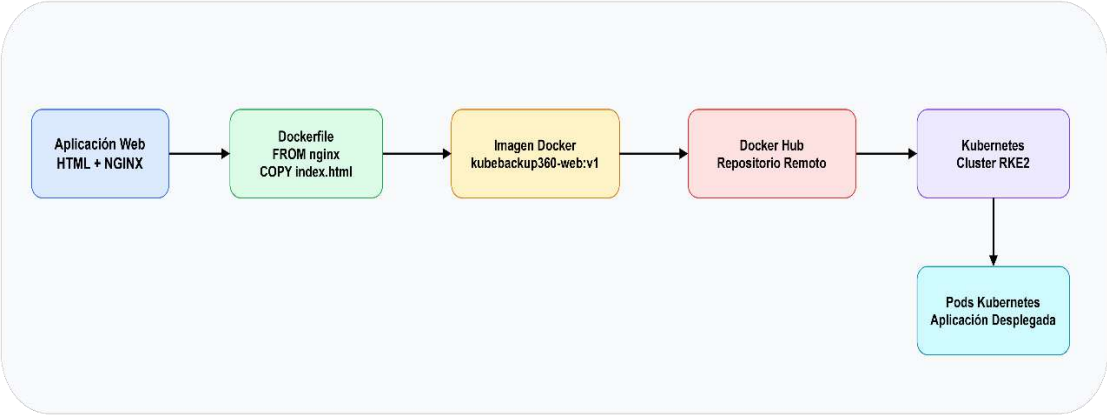
• preparar despliegues posteriores en Kubernetes

Aunque el clúster RKE2 utiliza internamente containerd como runtime de contenedores, Docker continúa siendo una herramienta ampliamente utilizada para tareas de desarrollo, construcción y distribución de imágenes compatibles con Kubernetes.

La integración entre Docker, Docker Hub y Kubernetes permite reproducir un flujo de trabajo similar al utilizado en entornos DevOps y plataformas cloud-native profesionales.

El flujo general utilizado durante esta fase queda representado en la siguiente arquitectura:

**Flujo general Docker y Kubernetes utilizado en KubeBackup360**




Construcción, almacenamiento y despliegue de aplicaciones contenerizadas dentro del entorno Kubernetes KubeBackup360

*El diagrama representa el proceso completo de construcción, almacenamiento y despliegue de aplicaciones contenerizadas dentro del entorno Kubernetes del proyecto KubeBackup360*.

### 7.1 Introducción a Docker y contenedores

**Objetivo**

Introducir los conceptos básicos relacionados con Docker y la contenerización de aplicaciones dentro del entorno Kubernetes utilizado en el proyecto KubeBackup360.

Docker es una plataforma de virtualización ligera que permite ejecutar aplicaciones dentro de contenedores aislados y portables.

A diferencia de las máquinas virtuales tradicionales, los contenedores comparten el kernel del sistema operativo anfitrión, reduciendo considerablemente el consumo de recursos y mejorando la velocidad de despliegue.

Cada contenedor incluye:

• aplicación

• librerías

• dependencias

• configuraciones necesarias para su ejecución

Esto permite que una misma aplicación pueda ejecutarse de forma consistente en distintos entornos sin necesidad de modificaciones adicionales.

Dentro del ecosistema Kubernetes, las aplicaciones desplegadas en Pods utilizan imágenes de contenedor como base de ejecución.

Por este motivo, Docker se convierte en una herramienta fundamental para:

• construir imágenes

• distribuir aplicaciones

• gestionar contenedores

• integrar despliegues cloud-native

Durante esta fase se trabajará con imágenes Docker basadas en NGINX para desplegar una aplicación web simple que posteriormente será utilizada dentro del clúster Kubernetes.

Asimismo, se utilizará Docker Hub como repositorio remoto para almacenar y distribuir imágenes contenerizadas utilizadas posteriormente por Kubernetes.

La integración entre Docker, Docker Hub y Kubernetes permite reproducir un flujo de trabajo similar al utilizado en entornos DevOps y plataformas cloud-native profesionales.

El flujo general utilizado durante esta fase queda representado en la siguiente arquitectura:

Aplicación Web

↓

Dockerfile

↓

Imagen Docker

↓

Docker Hub

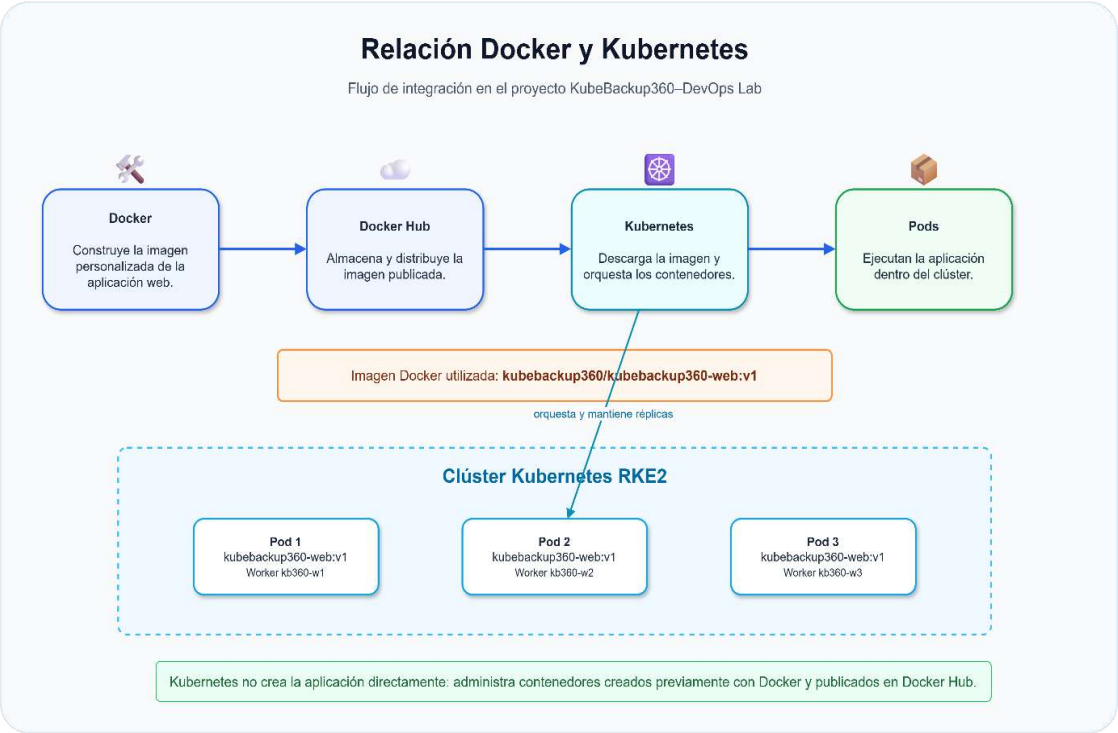
↓

Kubernetes

↓

Pods

**Relación entre Docker, Docker Hub y Kubernetes en KubeBackup360**




*El diagrama representa el flujo general utilizado en el proyecto KubeBackup360 para construir imágenes Docker, almacenarlas en Docker Hub y desplegarlas posteriormente dentro del clúster Kubernetes RKE2 mediante Pods.*

### 7.2 Instalación de Docker Engine
### Objetivo de este paso

Instalar Docker Engine dentro del entorno KubeBackup360 con el objetivo de disponer de una plataforma de creación, ejecución y gestión de contenedores compatible con Kubernetes.

Docker será utilizado durante esta fase para:

• construir imágenes Docker

• ejecutar contenedores localmente

• validar aplicaciones web

• publicar imágenes en Docker Hub

• preparar despliegues posteriores en Kubernetes

***Aunque el clúster Kubernetes RKE2 utiliza internamente containerd como runtime de contenedores, Docker puede coexistir perfectamente dentro del entorno sin interferir con el funcionamiento del clúster*.**

La instalación se realizó sobre el nodo master Kubernetes utilizando el gestor de paquetes APT:

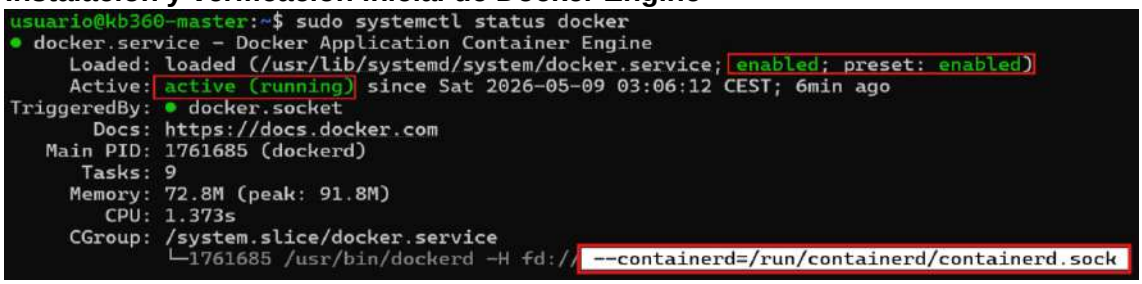
**sudo apt install docker.io -y**

Una vez completada la instalación, se verificó el estado del servicio Docker mediante:

**systemctl status docker**

La validación confirmó que el servicio Docker quedó iniciado correctamente en estado activo (active/running).

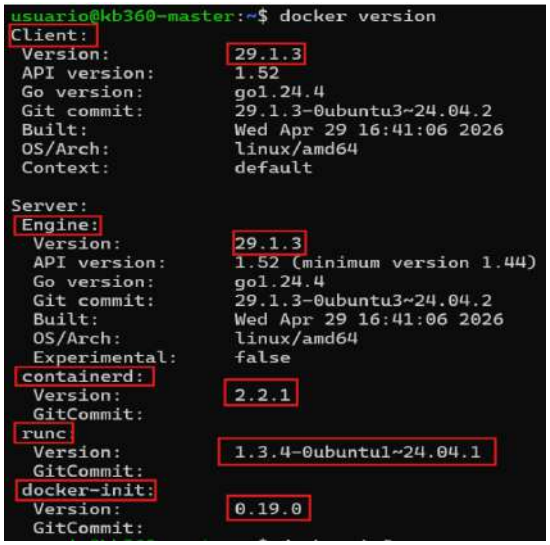
**Instalación y verificación inicial de Docker Engine**



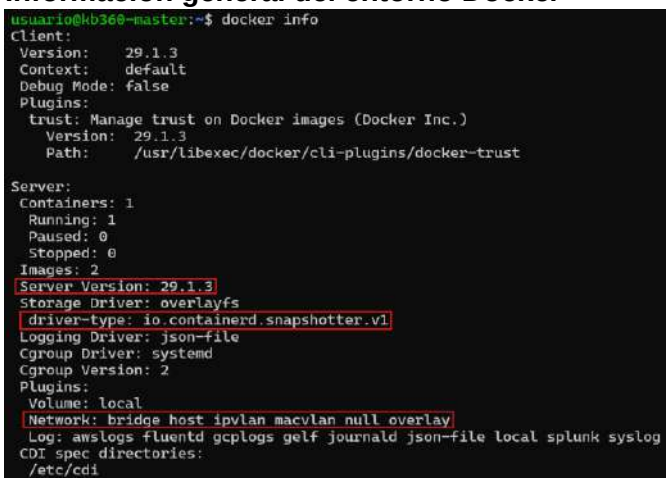

Posteriormente, se verificó la versión instalada de Docker utilizando:

Esta comprobación permitió confirmar el correcto funcionamiento del cliente y daemon Docker dentro del sistema.

**Verificación de versión Docker instalada**

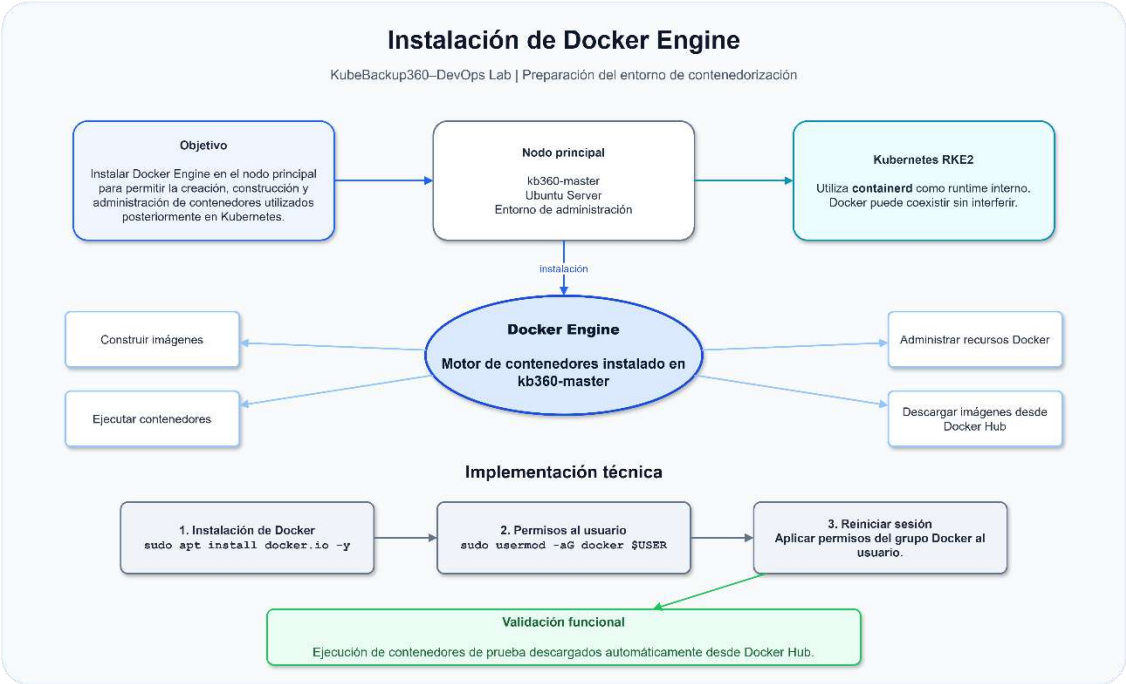



**Información general del entorno Docker**



La instalación y validación realizadas confirmaron que Docker Engine quedó completamente operativo y preparado para la creación y ejecución de contenedores dentro del proyecto KubeBackup360.

**Arquitectura e instalación de Docker Engine en**




*El diagrama representa la instalación de Docker Engine sobre el nodo principal Kubernetes, así como su integración dentro del entorno RKE2 utilizado en el proyecto KubeBackup360.*

### 7.3 Verificación funcional de Docker

Validar el correcto funcionamiento de Docker Engine mediante la ejecución de un primer contenedor de prueba dentro del entorno KubeBackup360.

Esta validación permitió comprobar:

• funcionamiento del daemon Docker

• descarga automática de imágenes desde Docker Hub

• creación y ejecución de contenedores

• conectividad con el repositorio Docker Hub

• operatividad general del entorno Docker

Para realizar la validación inicial se ejecutó el contenedor de prueba oficial hello-world mediante:

**docker run hello-world**

Durante la ejecución, Docker descargó automáticamente la imagen desde Docker Hub y creó un contenedor temporal mostrando un mensaje de funcionamiento correcto.

*El resultado confirmó que Docker podía:*

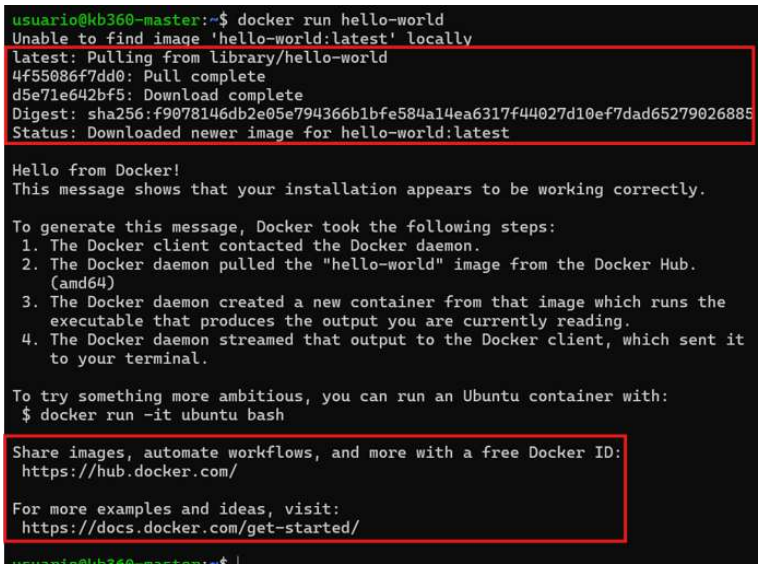
• *descargar imágenes remotas*

• *crear contenedores correctamente*

• *ejecutar procesos dentro del contenedor*

• *comunicarse con Docker Hub*

**Ejecución inicial del contenedor hello-world**




Posteriormente, se verificó el listado de imágenes Docker disponibles en el sistema mediante:

**docker images**

La salida mostró la imagen hello-world descargada correctamente desde Docker Hub.

**Verificación de imágenes Docker descargadas**


Finalmente, se comprobó el historial de contenedores ejecutados mediante:

**docker ps -a**
La validación confirmó que el contenedor hello-world había sido creado y ejecutado correctamente dentro del entorno Docker.

*Las validaciones realizadas confirmaron que Docker Engine funciona correctamente dentro del entorno KubeBackup360, permitiendo descargar imágenes, crear contenedores y ejecutar aplicaciones contenerizadas de forma operativa*.

### 7.4 Gestión básica de contenedores Docker

Realizar operaciones básicas de administración y gestión de contenedores Docker dentro del entorno KubeBackup360, permitiendo comprender el ciclo de vida de los contenedores y las tareas habituales de mantenimiento utilizadas en entornos Docker.

Durante esta fase se trabajó principalmente con:

• inicio y parada de contenedores

• eliminación de contenedores

• eliminación de imágenes Docker

• validación del estado de ejecución

Inicialmente, se verificó el listado de contenedores activos mediante:

**docker ps**

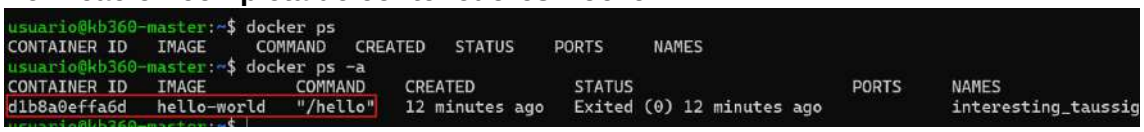
Este comando permite visualizar únicamente los contenedores que se encuentran actualmente en ejecución dentro del sistema Docker.

Posteriormente, se comprobó el listado completo de contenedores creados dentro del entorno Docker mediante:

**docker ps -a**

La salida permitió visualizar tanto contenedores activos como detenidos.

**Verificación completa de contenedores Docker**

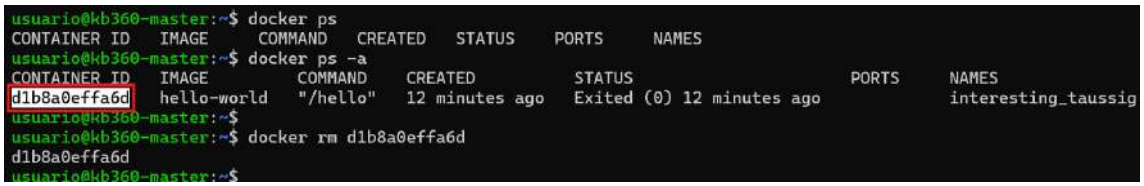



A continuación, se realizó la eliminación de un contenedor Docker mediante:

**docker rm <container_id>**

Esta operación permite eliminar contenedores detenidos que ya no son necesarios dentro del sistema.

**Eliminación de contenedores Docker**
**docker rm d1b8a0effa6d**

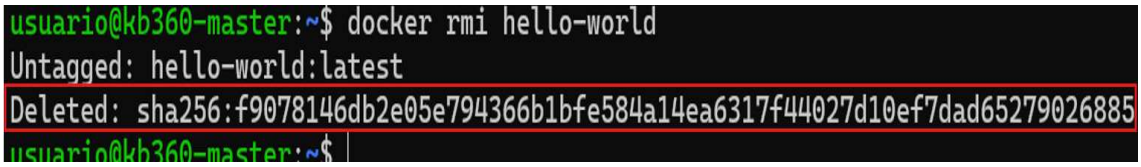



Posteriormente, se procedió a eliminar imágenes Docker almacenadas localmente utilizando:

**docker rmi <image_id>**

La eliminación de imágenes permite liberar espacio de almacenamiento y gestionar correctamente los recursos del entorno Docker.

**Eliminación de imágenes Docker locales**




Explicación técnica: **docker rmi**, elimina imágenes Docker locales almacenadas en: /var/lib/docker.
Las operaciones realizadas permitieron validar el funcionamiento de las tareas básicas de administración Docker, incluyendo gestión de contenedores, mantenimiento de imágenes y control del ciclo de vida de aplicaciones contenerizadas dentro del proyecto KubeBackup360.

**7.5 Creación de aplicación web para contenedor Docker**
Preparar una aplicación web sencilla que será utilizada posteriormente para construir una imagen Docker y desplegar un contenedor web dentro del entorno KubeBackup360.

La aplicación desarrollada servirá como ejemplo práctico para validar:

• construcción de imágenes Docker

• despliegue de contenedores web

• publicación en Docker Hub

• integración posterior con Kubernetes

Para ello, se creó una estructura básica de proyecto web compuesta principalmente por un archivo index.html que será servido posteriormente mediante NGINX dentro del contenedor Docker.

kubebackup360-web/

└── web/

└── index.html

Inicialmente, se creó el directorio del proyecto mediante:

**mkdir -p ~/kubebackup360-web**

**cd ~/kubebackup360-web**

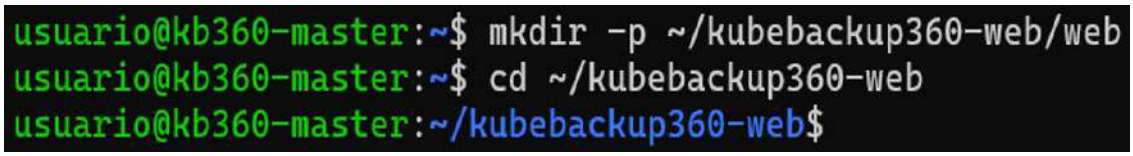
Posteriormente, se generó el archivo principal index.html:

**nano index.html**

Dentro del archivo se añadió una página web simple personalizada para el proyecto KubeBackup360.

La aplicación web será utilizada posteriormente como contenido principal del contenedor Docker basado en NGINX.

**Creación de la estructura básica de la aplicación web**



```
usuario@kb360-master:~/kubebackup360-web$ nano web/index.html
usuario@kb360-master:~/kubebackup360-web$ cat web/index.html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>KubeBackup360</title>
    <style>
        body {
            background-color: #0f172a;
            color: white;
            font-family: Arial, sans-serif;
            text-align: center;
            padding-top: 100px;
        }

        h1 {
            color: #38bdf8;
            font-size: 48px;
        }

        p {
            font-size: 22px;
        }

        .box {
            background-color: #1e293b;
            padding: 40px;
            border-radius: 12px;
            width: 60%;
            margin: auto;
            box-shadow: 0px 0px 15px rgba(0,0,0,0.5);
        }
    </style>
</head>
<body>

<div class="box">
    <h1>KubeBackup360</h1>
    <p>Proyecto Kubernetes FP</p>
    <p>Docker + Kubernetes + Backup + Monitorización</p>
</div>

</body>
</html>
usuario@kb360-master:~/kubebackup360-web$
```

Posteriormente, se verificó la estructura general del proyecto mediante:

**tree ~/kubebackup360-web**

La validación confirmó que la aplicación web quedó correctamente preparada para ser utilizada posteriormente durante la construcción de la imagen Docker.

**Verificación de estructura del proyecto web**

```
usuario@kb360-master:~/kubebackup360-web$ tree ~/kubebackup360-web
/home/usuario/kubebackup360-web
└── web
    └── index.html

2 directories, 1 file
usuario@kb360-master:~/kubebackup360-web$
```

• directorio kubebackup360-web

• archivo index.html

• estructura general del proyecto

La aplicación web creada servirá como contenido principal del contenedor Docker desplegado posteriormente dentro del entorno Kubernetes del proyecto KubeBackup360.

### 7.6 Creación del Dockerfile

Crear el archivo Dockerfile necesario para construir una imagen Docker personalizada basada en NGINX, permitiendo contenerizar la aplicación web desarrollada previamente dentro del proyecto KubeBackup360.

El Dockerfile define las instrucciones utilizadas por Docker para generar automáticamente imágenes contenerizadas.

Durante este proceso se configurará:

• imagen base NGINX

• copia de archivos web

• estructura interna del contenedor

• contenido servido por el servidor web

Inicialmente, se creó el archivo Dockerfile dentro del directorio del proyecto:

**nano Dockerfile**

Dentro del fichero se definió una configuración básica utilizando la imagen oficial NGINX como base del contenedor.

La configuración utilizada fue similar a la siguiente:

**FROM nginx:latest**
**COPY index.html /usr/share/nginx/html/index.html**

La instrucción:

**FROM nginx:latest**

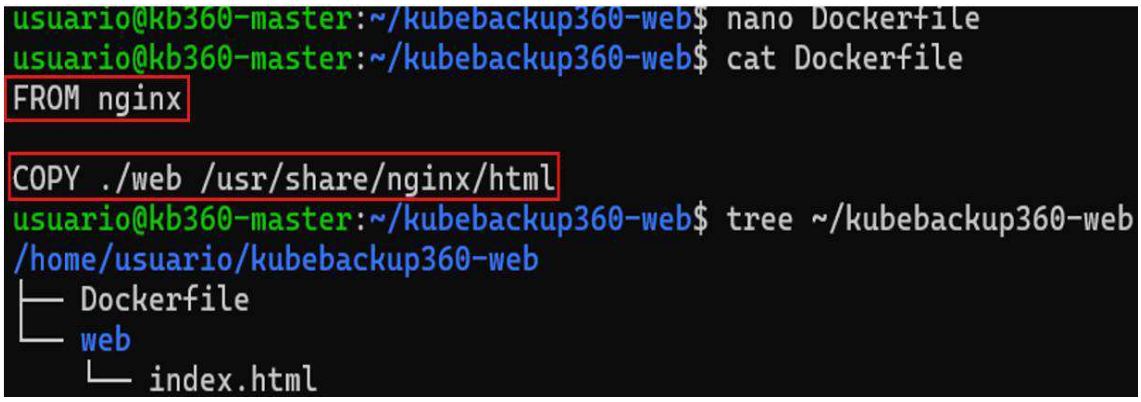
permite utilizar la imagen oficial NGINX descargada desde Docker Hub como base del contenedor.

Posteriormente, la instrucción:

**COPY index.html /usr/share/nginx/html/index.html**

copia la aplicación web personalizada al directorio utilizado por NGINX para servir contenido web dentro del contenedor.

**Creación y configuración del Dockerfile**




• contenido final del Dockerfile

• instrucciones Docker correctamente definidas

El Dockerfile creado permitirá generar posteriormente una imagen Docker personalizada basada en NGINX que contendrá la aplicación web desarrollada para el proyecto KubeBackup360.

### 7.7 Construcción de imagen Docker

Construir una imagen Docker personalizada a partir del Dockerfile creado previamente, incorporando la aplicación web desarrollada dentro de un contenedor basado en NGINX.

La construcción de imágenes Docker permite generar paquetes portables y reutilizables que posteriormente podrán ser desplegados tanto localmente como dentro del clúster Kubernetes.

La construcción de la imagen se realizó desde el directorio del proyecto utilizando:

**docker build -t kubebackup360-web:v1 .**

La opción:

**-t** permite asignar un nombre y etiqueta a la imagen generada.

En este caso:

**kubebackup360-web:v1** representa:

• nombre de la imagen

• versión inicial del contenedor

Durante el proceso de construcción, Docker ejecutó automáticamente las instrucciones definidas dentro del Dockerfile:

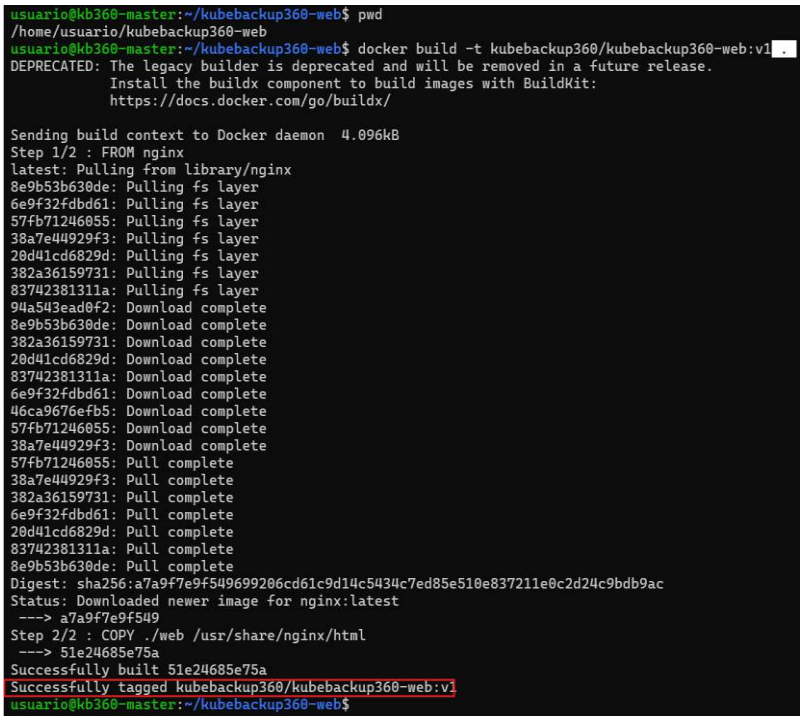
• descarga de imagen base NGINX

• creación de capas Docker

• copia del archivo index.html

• generación final de la imagen personalizada

**Construcción de imagen Docker personalizada**



Una vez finalizada la construcción, se verificó el listado de imágenes Docker disponibles mediante:

**docker images**

La salida confirmó la creación correcta de la imagen:

kubebackup360-web:v1

**Verificación de imágenes Docker creadas**

```
usuario@kb360-master:~/kubebackup360-web$ docker images

IMAGE                                   ID             DISK USAGE   CONTENT SIZE   EXTRA
kubebackup360/kubebackup360-web:v1      51e24685e75a       237MB             63MB
nginx:latest                            a7a9f7e9f549       240MB           65.8MB
usuario@kb360-master:~/kubebackup360-web$
```

La validación realizada confirmó que la imagen Docker personalizada fue construida correctamente y quedó preparada para ejecutar contenedores web dentro del entorno KubeBackup360.

### 7.8 Ejecución local del contenedor Docker

Ejecutar localmente un contenedor Docker basado en la imagen personalizada construida previamente, validando el funcionamiento correcto de la aplicación web dentro del entorno KubeBackup360.

La ejecución del contenedor se realizó mediante:

**docker run -d -p 8080:80 --name kubebackup360-web kubebackup360-web:v1**

La configuración utilizada permite:

• ejecutar el contenedor en segundo plano (-d)

• redirigir el puerto local 8080 hacia el puerto 80 interno del contenedor

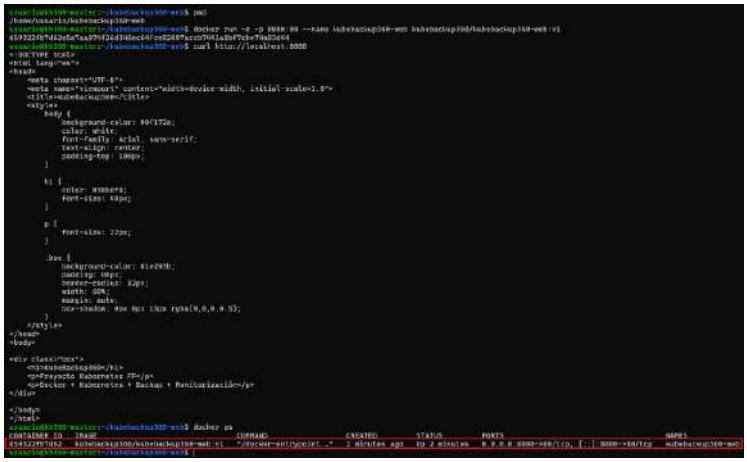
• asignar el nombre kubebackup360-web al contenedor

Durante la ejecución, Docker inició automáticamente el servidor NGINX contenido dentro de la imagen personalizada.

**Ejecución del contenedor Docker basado en NGINX**

Comando utilizado:

docker run -d -p 8080:80 --name kubebackup360-web kubebackup360-web:v1



Posteriormente, se verificó el estado del contenedor mediante:

**docker ps**

La salida confirmó que el contenedor permanecía en estado Up, indicando el correcto funcionamiento del servicio web NGINX.

**Verificación del contenedor Docker en ejecución**


• contenedor kubebackup360-web

• estado Up

• puertos 8080:80

• imagen utilizada

A continuación, se validó el funcionamiento del servicio web accediendo localmente mediante:

**curl http://localhost:8080**

**http://localhost:8080**

*La respuesta obtenida mostró correctamente el contenido HTML personalizado de la aplicación KubeBackup360, confirmando que el servidor NGINX integrado en el contenedor funcionaba correctamente y que la aplicación web podía ser accedida satisfactoriamente*

Las pruebas realizadas confirmaron el correcto funcionamiento del contenedor Docker y de la aplicación web contenerizada dentro del entorno KubeBackup360.

**7.9 Gestión básica del contenedor Docker**

Realizar operaciones básicas de administración sobre el contenedor Docker desplegado previamente, permitiendo validar el ciclo de vida de los contenedores dentro del entorno KubeBackup360.

Durante este proceso se trabajó principalmente con:

• parada de contenedores

• inicio de contenedores

• reinicio de servicios

• visualización de logs

• eliminación de contenedores

Estas tareas forman parte de la administración habitual de entornos Docker y permiten gestionar el estado operativo de aplicaciones contenerizadas.

Inicialmente, se verificó el estado actual del contenedor mediante:

**docker ps**

La salida permitió comprobar que el contenedor kubebackup360-web permanecía en ejecución.

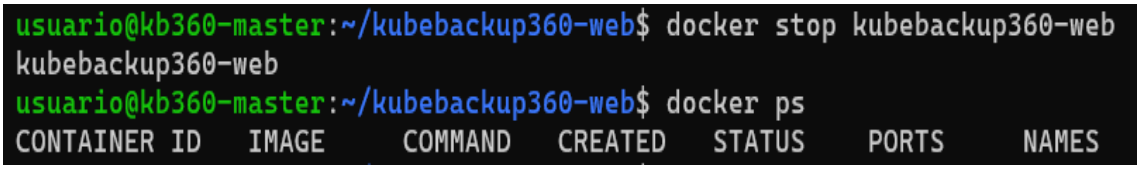
**Verificación inicial del contenedor Docker en ejecución**


Posteriormente, se detuvo el contenedor mediante:

**docker stop kubebackup360-web**

La operación finalizó correctamente deteniendo el servicio NGINX ejecutado dentro del contenedor.

**Detención del contenedor Docker**




• parada correcta del contenedor

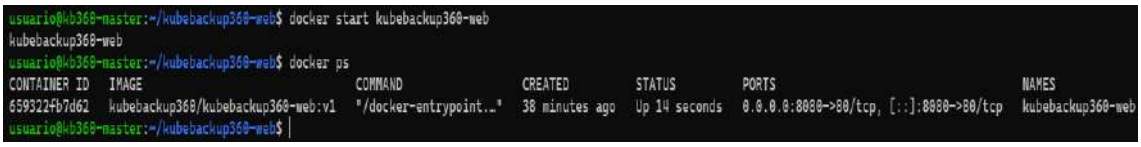
• nombre del contenedor detenido

A continuación, se inició nuevamente el contenedor utilizando:

**docker start kubebackup360-web**

La validación confirmó que el contenedor podía reiniciarse correctamente manteniendo la configuración previamente definida.

**Inicio del contenedor Docker**




• arranque correcto del contenedor

• nombre del contenedor iniciado

Posteriormente, se visualizaron los logs generados por el contenedor mediante:

**docker logs kubebackup360-web**

La salida permitió comprobar el funcionamiento interno del servidor NGINX ejecutado dentro del contenedor Docker.

• logs del contenedor

• información del servidor NGINX

Finalmente, se eliminó el contenedor Docker utilizando:

**docker rm -f kubebackup360-web**

La operación permitió eliminar completamente el contenedor del sistema Docker.

Detener nuevamente

docker stop kubebackup360-web

**Eliminación final del contenedor Docker**

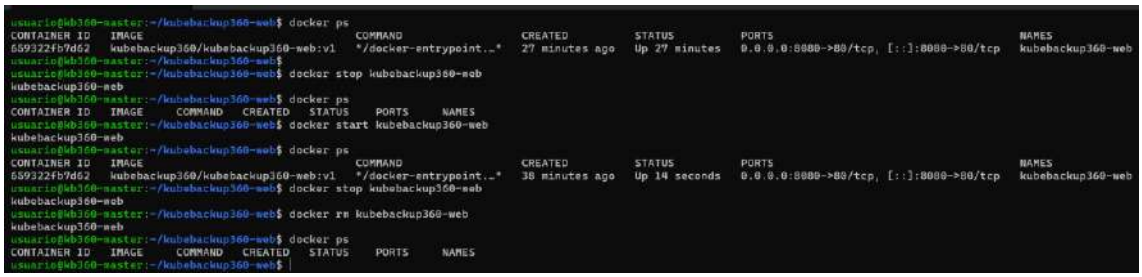
```
usuario@kb360-master:~/kubebackup360-web$ docker stop kubebackup360-web
kubebackup360-web
usuario@kb360-master:~/kubebackup360-web$ docker rm kubebackup360-web
kubebackup360-web
usuario@kb360-master:~/kubebackup360-web$ docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
usuario@kb360-master:~/kubebackup360-web$
```

• eliminación correcta del contenedor

• nombre del contenedor eliminado

Las operaciones realizadas permitieron validar correctamente la administración básica de contenedores Docker, incluyendo control del ciclo de vida, visualización de logs y eliminación de servicios contenerizados dentro del proyecto KubeBackup360.

**Proceso de eliminación General**




**7.10 Registro en Docker Hub**
**Objetivo de este paso**

Configurar el acceso al repositorio Docker Hub con el objetivo de permitir el almacenamiento y distribución remota de imágenes Docker utilizadas posteriormente dentro del entorno Kubernetes del proyecto KubeBackup360.

Docker Hub es una plataforma cloud utilizada para:

• almacenar imágenes Docker

• distribuir contenedores

• compartir aplicaciones contenerizadas

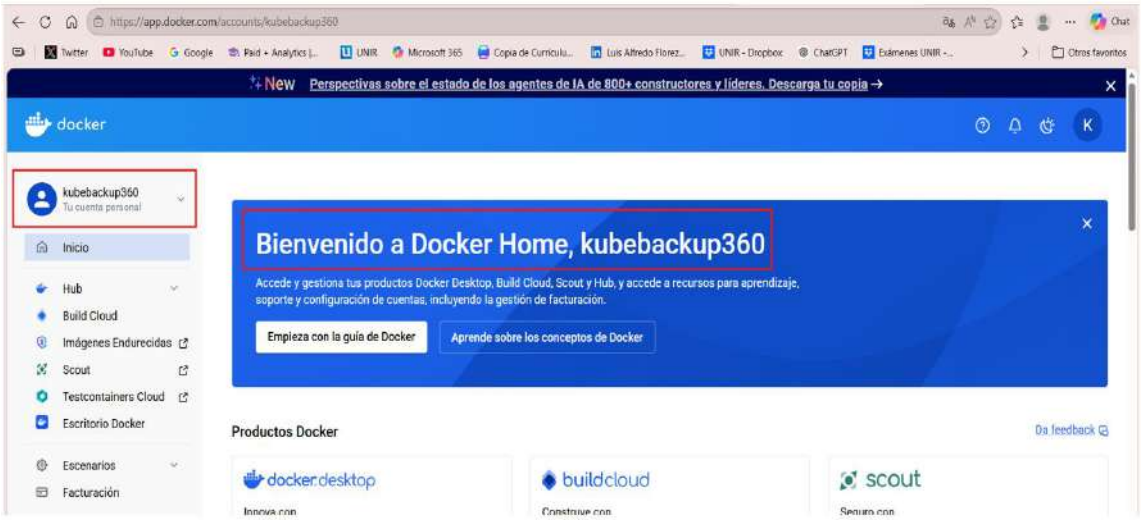
• facilitar despliegues Kubernetes

La utilización de Docker Hub permite reproducir un flujo de trabajo similar al utilizado en entornos DevOps y plataformas cloud-native reales.

Inicialmente, se accedió al repositorio Docker Hub desde navegador web para crear una cuenta de usuario destinada al proyecto.

Posteriormente, se verificó el acceso correcto al repositorio remoto y la disponibilidad del perfil Docker Hub utilizado para almacenar imágenes Docker personalizadas.

**Acceso web al repositorio Docker Hub**




Una vez creada la cuenta, se realizó la autenticación desde el entorno Linux mediante:

docker login

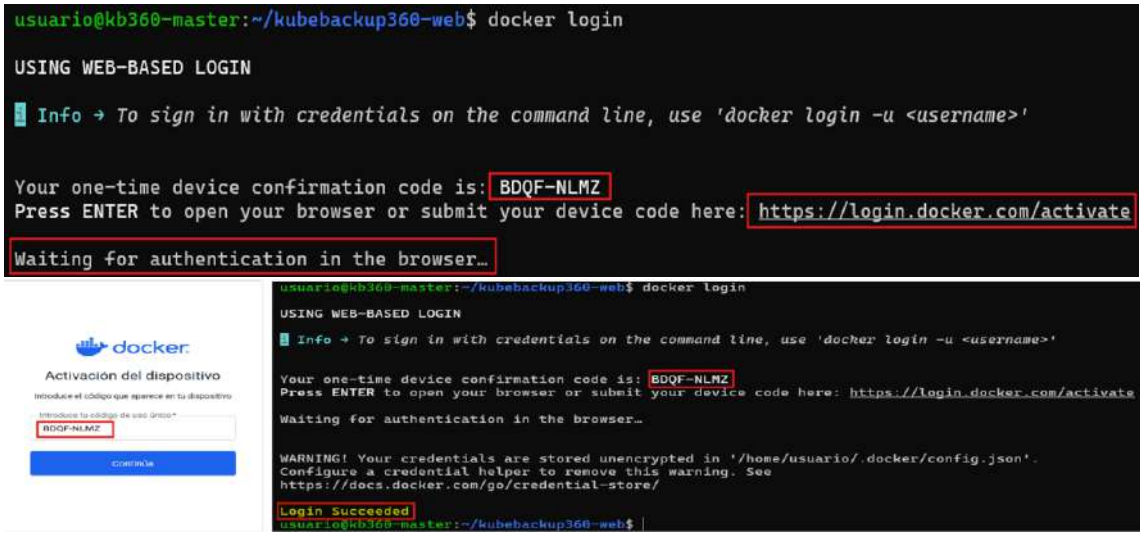
Durante el proceso se introdujeron:

• nombre de usuario Docker Hub

• contraseña de acceso

Tras completar la autenticación, Docker mostró un mensaje de inicio de sesión correcto.

**Autenticación Docker Hub desde terminal Linux**




• ejecución docker login

• autenticación correcta

• mensaje Login Succeeded

La autenticación realizada permitió conectar correctamente el entorno Docker local con el repositorio remoto Docker Hub, dejando preparado el sistema para publicar imágenes Docker utilizadas posteriormente dentro del clúster Kubernetes.

### 7.11 Publicación de imagen Docker en Docker Hub

Publicar la imagen Docker personalizada creada previamente dentro del repositorio remoto Docker Hub, permitiendo almacenar y distribuir la aplicación contenerizada para su posterior utilización en Kubernetes.

La publicación de imágenes Docker permite:

• compartir aplicaciones contenerizadas

• descargar imágenes desde otros sistemas

• centralizar almacenamiento de contenedores

• facilitar despliegues Kubernetes

• reproducir flujos DevOps reales

Antes de publicar la imagen, fue necesario etiquetarla utilizando el nombre del repositorio Docker Hub asociado a la cuenta creada previamente.

La etiqueta se generó mediante:

**docker tag kubebackup360-web:v1 usuario-dockerhub/kubebackup360-web:v1**

Esta operación permitió asociar la imagen local al repositorio remoto Docker Hub.

• comando docker tag

• nombre completo de la imagen

• repositorio Docker Hub utilizado

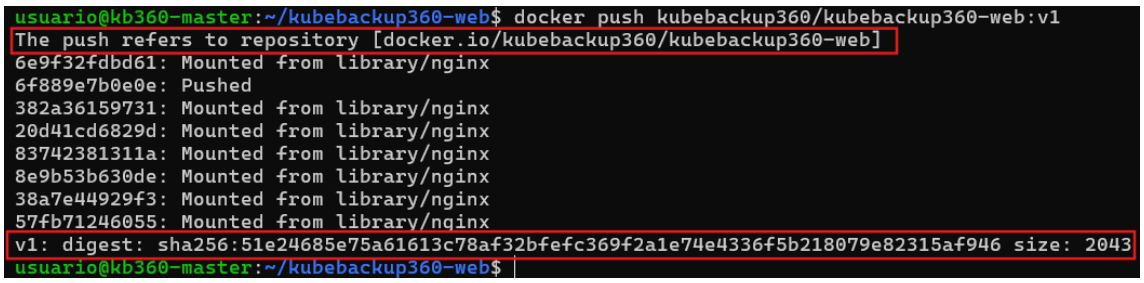
Posteriormente, se publicó la imagen Docker mediante:

docker push usuario-dockerhub/kubebackup360-web:v1

Durante el proceso, Docker transfirió automáticamente las distintas capas de la imagen hacia Docker Hub.

La salida mostró el progreso de subida de capas y la publicación correcta de la imagen remota.

**Publicación de imagen Docker en Docker Hub**




• subida de capas Docker

• progreso push

• mensaje push successful

Una vez completada la publicación, se verificó desde la interfaz web de Docker Hub que la imagen aparecía correctamente almacenada dentro del repositorio remoto.

• repositorio Docker Hub

• imagen kubebackup360-web

La validación realizada confirmó que la imagen Docker personalizada quedó correctamente publicada en Docker Hub y preparada para ser utilizada posteriormente dentro del clúster Kubernetes del proyecto KubeBackup360.

**7.12 Descarga remota de imagen desde Docker Hub**
Validar la disponibilidad remota de la imagen Docker publicada previamente en Docker Hub, comprobando que puede descargarse correctamente desde el repositorio remoto y reutilizarse posteriormente dentro del entorno Kubernetes.

Esta validación permitió comprobar:

• disponibilidad pública de la imagen Docker

• funcionamiento del repositorio Docker Hub

• descarga remota de imágenes

• reutilización de contenedores

• preparación para despliegues Kubernetes

Inicialmente, se eliminó la imagen Docker almacenada localmente con el objetivo de forzar una descarga remota real desde Docker Hub.

La eliminación se realizó mediante:

**docker rmi usuario-dockerhub/kubebackup360-web:v1**

La operación eliminó la imagen local del sistema Docker.

**Eliminación local de imagen Docker**

• eliminación correcta de la imagen

• image ID eliminado

• mensaje de borrado exitoso

Posteriormente, se descargó nuevamente la imagen desde Docker Hub utilizando:

**docker pull usuario-dockerhub/kubebackup360-web:v1**

Durante la descarga, Docker recuperó automáticamente las capas de la imagen desde el repositorio remoto Docker Hub.

La salida mostró:

• descarga de capas

• verificación de integridad

• recuperación correcta de la imagen

**Descarga remota de imagen desde Docker Hub**
Comando utilizado:

docker pull usuario-dockerhub/kubebackup360-web:v1

```
usuario@kb360-master:~/kubebackup360-web$ docker pull kubebackup360/kubebackup360-web:v1
v1: Pulling from kubebackup360/kubebackup360-web
6f889e7b0e0e: Pull complete
Digest: sha256:51e24685e75a61613c78af32bfefc369f2a1e74e4336f5b218079e82315af946
Status: Downloaded newer image for kubebackup360/kubebackup360-web:v1
docker.io/kubebackup360/kubebackup360-web:v1
usuario@kb360-master:~/kubebackup360-web$ docker images

IMAGE                                  ID             DISK USAGE   CONTENT SIZE   EXTRA
kubebackup360/kubebackup360-web:v1     51e24685e75a       237MB           63MB
nginx:latest                           a7a9f7e9f549       240MB         65.8MB
usuario@kb360-master:~/kubebackup360-web$
```

• descarga de capas Docker

• progreso pull

• mensaje Download complete

Una vez finalizada la descarga, se verificó nuevamente el listado de imágenes Docker disponibles en el sistema mediante:

**docker images**

La validación confirmó que la imagen había sido recuperada correctamente desde Docker Hub.

**Verificación de imagen Docker descargada remotamente**
Comando utilizado:

```
usuario@kb360-master:~/kubebackup360-web$ #Verificar imágenes Docker
usuario@kb360-master:~/kubebackup360-web$ docker images

IMAGE                                  ID             DISK USAGE   CONTENT SIZE   EXTRA
kubebackup360/kubebackup360-web:v1     51e24685e75a       237MB           63MB
nginx:latest                           a7a9f7e9f549       240MB         65.8MB
usuario@kb360-master:~/kubebackup360-web$ #Verificar acceso Docker Hub
usuario@kb360-master:~/kubebackup360-web$ docker info
Client:
 Version:    29.1.3
 Context:    default
 Debug Mode: false
 Plugins:
  trust: Manage trust on Docker images (Docker Inc.)
    Version:  29.1.3
    Path:     /usr/libexec/docker/cli-plugins/docker-trust

Server:
 Containers: 0
  Running: 0
  Paused: 0
  Stopped: 0
 Images: 2
 Server Version: 29.1.3
 Storage Driver: overlayfs
  driver-type: io.containerd.snapshotter.v1
 Logging Driver: json-file
 Cgroup Driver: systemd
 Cgroup Version: 2
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local splunk syslog
 CDI spec directories:
  /etc/cdi
  /var/run/cdi
```

La validación realizada confirmó que la imagen Docker podía almacenarse y recuperarse correctamente desde Docker Hub, quedando preparada para ser utilizada posteriormente dentro del clúster Kubernetes del proyecto KubeBackup360.

### 7.13 Preparación de imágenes Docker para Kubernetes

Preparar la integración entre Docker y Kubernetes utilizando imágenes Docker publicadas previamente en Docker Hub, permitiendo su posterior despliegue dentro del clúster Kubernetes del proyecto KubeBackup360.

En Kubernetes, las aplicaciones desplegadas mediante Pods utilizan imágenes de contenedor almacenadas en repositorios remotos compatibles con OCI (Open Container Initiative), como Docker Hub.

Por este motivo, la publicación previa de imágenes Docker resulta fundamental para permitir que Kubernetes pueda:

• descargar imágenes automáticamente

• crear Pods a partir de contenedores

• desplegar aplicaciones escalables

• gestionar servicios cloud-native

Durante esta fase, la imagen utilizada fue:

usuario-dockerhub/kubebackup360-web:v1

Esta imagen contiene:

• servidor NGINX

• aplicación web personalizada KubeBackup360

• configuración lista para Kubernetes

Dentro de Kubernetes, los Deployments utilizan el parámetro:

**image:**

para indicar la imagen Docker que será descargada automáticamente desde Docker Hub durante la creación de Pods.

Un ejemplo simplificado de definición Kubernetes sería:

```
containers:
  - name: kubebackup360-web
    image: usuario-dockerhub/kubebackup360-web:v1
```

Cuando Kubernetes detecta esta configuración:

• descarga automáticamente la imagen desde Docker Hub

• crea contenedores basados en dicha imagen

• genera Pods dentro del clúster

• mantiene las réplicas configuradas

Este flujo permite separar completamente:

• construcción de aplicaciones

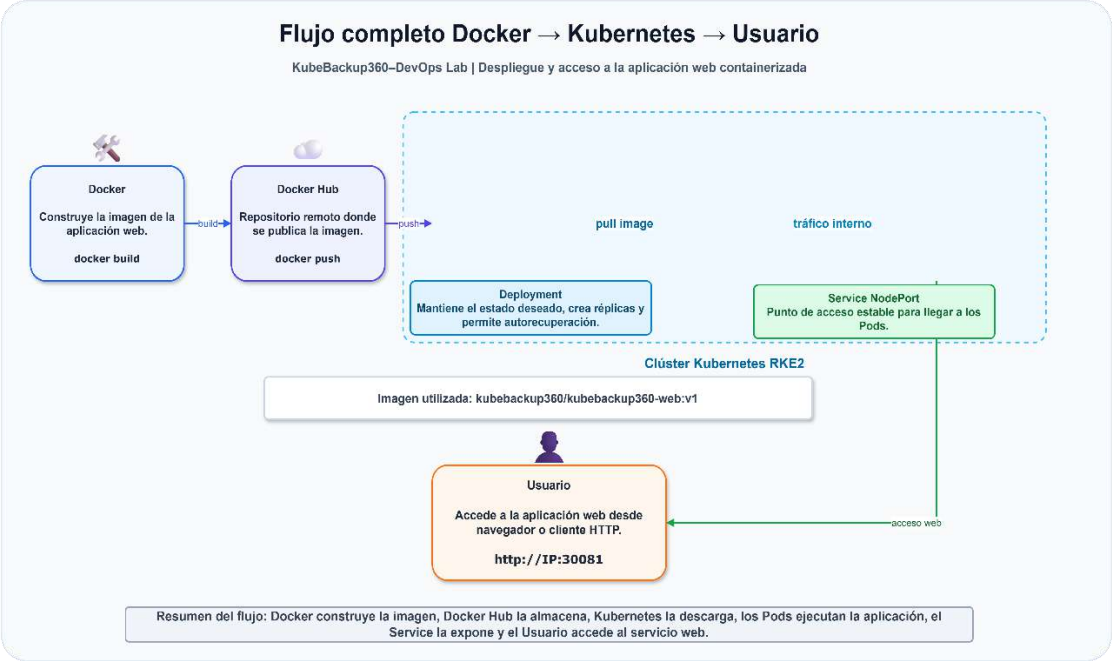
• almacenamiento de imágenes

• despliegue Kubernetes

reproduciendo una arquitectura similar a la utilizada en entornos DevOps profesionales.

El flujo completo de integración Docker y Kubernetes queda representado en la siguiente arquitectura:

**Flujo completo de despliegue Docker y Kubernetes en KubeBackup360**




El diagrama representa el flujo completo utilizado para construir imágenes Docker, almacenarlas en Docker Hub y desplegarlas posteriormente dentro del clúster Kubernetes mediante Deployments, Pods y Services accesibles por el usuario final.

Asimismo, este modelo de integración facilita:

• automatización de despliegues

• actualización de aplicaciones

• escalabilidad horizontal

• administración centralizada de contenedores

• integración con entornos Kubernetes cloud-native

La preparación realizada durante esta fase dejó completamente lista la infraestructura para comenzar posteriormente el despliegue real de aplicaciones Docker dentro del clúster Kubernetes del proyecto KubeBackup360.

### 7.14 Validación final del entorno Docker
### Objetivo de este paso

Realizar una validación global del entorno Docker implementado durante la Fase 7, comprobando el correcto funcionamiento de todos los componentes relacionados con la creación, gestión y distribución de contenedores dentro del proyecto KubeBackup360.

Durante esta validación se comprobó:

• instalación correcta de Docker Engine

• ejecución funcional de contenedores

• construcción de imágenes Docker

• funcionamiento de Docker Hub

• publicación y descarga remota de imágenes

• preparación para despliegues Kubernetes

Inicialmente, se verificó nuevamente el estado operativo del servicio Docker mediante:

**systemctl status docker**

La validación confirmó que Docker Engine permanecía activo y funcionando correctamente dentro del nodo principal Kubernetes.

**Verificación final del servicio Docker**

```
usuario@kb360-master:~/kubebackup360-web$ #Verificar servicio Docker
usuario@kb360-master:~/kubebackup360-web$ systemctl status docker --no-pager
● docker.service - Docker Application Container Engine
     Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: enabled)
     Active: active (running) since Sat 2026-05-09 03:06:12 CEST; 15h ago
TriggeredBy: ● docker.socket
       Docs: https://docs.docker.com
   Main PID: 1761685 (dockerd)
      Tasks: 11
     Memory: 56.3M (peak: 98.2M)
        CPU: 10.610s
     CGroup: /system.slice/docker.service
             └─1761685 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock
```

• servicio docker activo

• estado active (running)

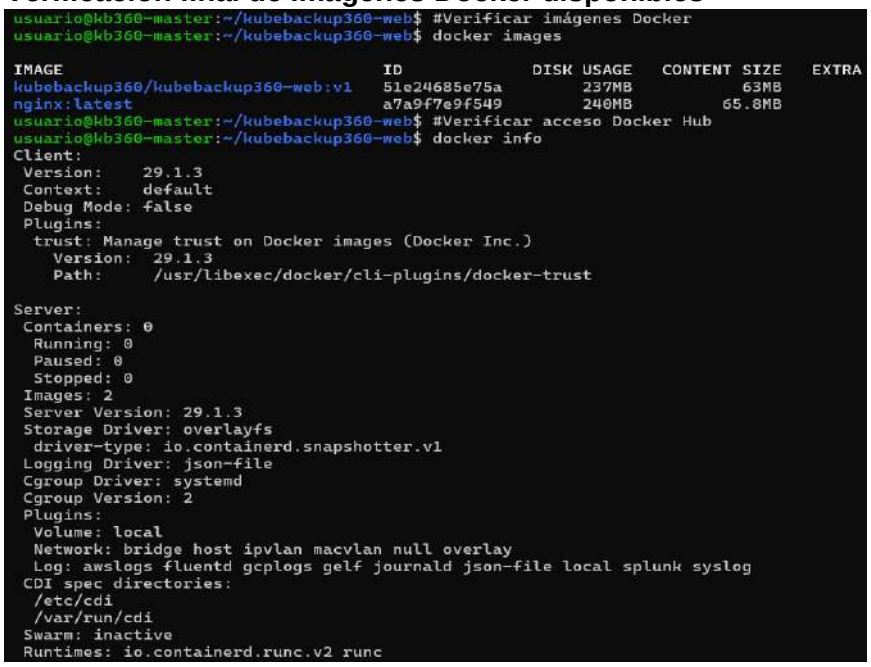
• daemon operativo

Posteriormente, se verificó el listado final de imágenes Docker disponibles en el sistema mediante:

**docker images**

La salida confirmó la disponibilidad de la imagen personalizada utilizada durante el proyecto.

**Verificación final de imágenes Docker disponibles**




• imagen kubebackup360-web

• tag v1

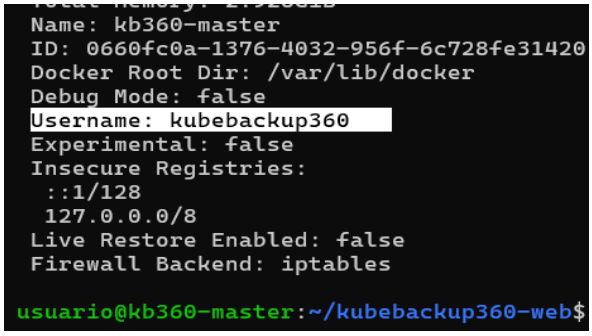
• repository Docker Hub

A continuación, se validó el funcionamiento general de los contenedores Docker mediante:

**docker ps -a**

La validación permitió comprobar el historial de contenedores ejecutados durante las pruebas realizadas.

**Verificación final de contenedores Docker ejecutados**

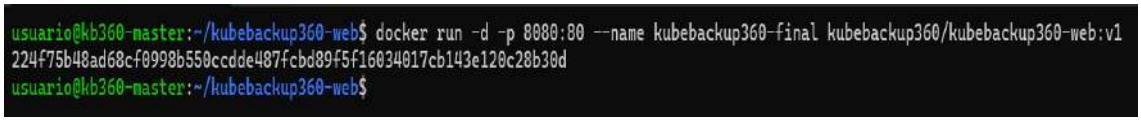



• contenedores creados

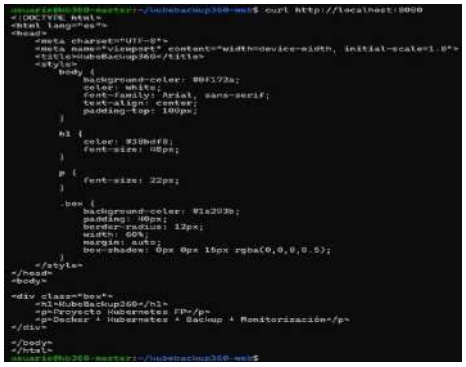
• estado Running o Exited

• imágenes utilizadas

**Verificar ejecución final del contenedor**



### Validar funcionamiento web



Asimismo, se verificó la disponibilidad remota de la imagen Docker publicada en Docker Hub, confirmando que podía descargarse correctamente desde el repositorio remoto.

Esta validación garantiza que Kubernetes podrá utilizar posteriormente dicha imagen durante el despliegue de Pods dentro del clúster.

Durante toda la fase se validó correctamente el flujo completo:

• construcción de imágenes Docker

• ejecución de contenedores

• publicación en Docker Hub

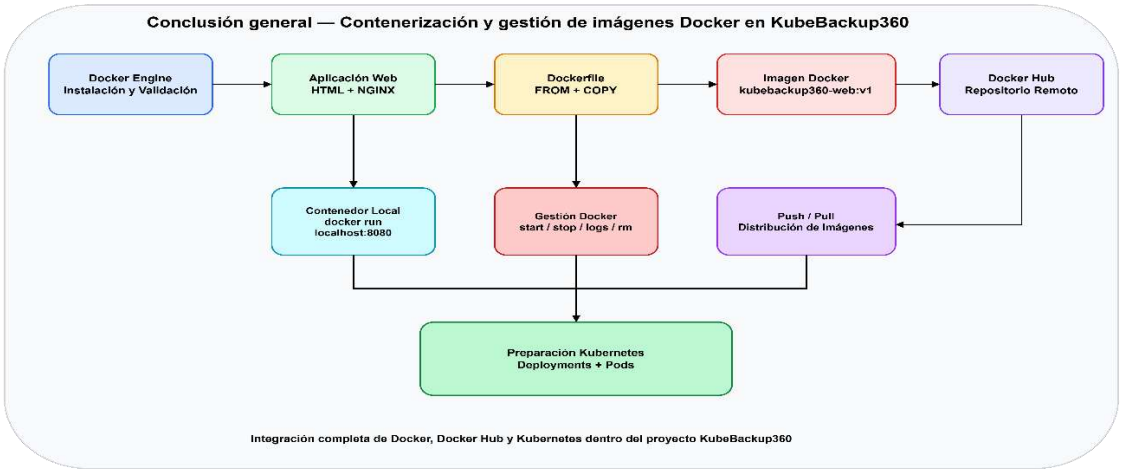
• recuperación remota de imágenes

• preparación para Kubernetes

El flujo implementado reproduce un entorno similar al utilizado en arquitecturas DevOps y plataformas cloud-native profesionales.

Las validaciones realizadas confirmaron que el entorno Docker quedó completamente operativo y preparado para integrarse posteriormente con Kubernetes dentro del proyecto KubeBackup360.

### Integración general de Docker y Kubernetes en KubeBackup360




*El diagrama resume el flujo completo de creación, gestión y distribución de aplicaciones contenerizadas mediante Docker, Docker Hub y Kubernetes dentro del entorno KubeBackup360.*

## 8. DESPLIEGUE DE APLICACIONES EN KUBERNETES

El objetivo principal de esta fase consiste en desplegar aplicaciones contenerizadas dentro del clúster Kubernetes del proyecto KubeBackup360 utilizando imágenes Docker publicadas previamente en Docker Hub.

Durante esta etapa se validará el funcionamiento real de distintos componentes fundamentales de Kubernetes relacionados con:

• Pods

• Deployments

• ReplicaSets

• Services NodePort

• networking Kubernetes

• escalabilidad horizontal

• autorecuperación automática

• rolling updates

• rollback

Kubernetes permite automatizar el despliegue y administración de aplicaciones contenerizadas mediante una arquitectura distribuida basada en múltiples nodos interconectados.

A diferencia de Docker, que administra contenedores individualmente, Kubernetes trabaja mediante recursos declarativos capaces de gestionar aplicaciones completas de forma automática, escalable y tolerante a fallos.

Los Pods representan la unidad básica de ejecución dentro del clúster Kubernetes y son administrados mediante Deployments y ReplicaSets, permitiendo mantener automáticamente el número de réplicas deseadas y recrear contenedores ante fallos o eliminaciones.

Asimismo, los Services Kubernetes permiten proporcionar acceso estable a las aplicaciones desplegadas y exponer servicios hacia el exterior mediante NodePort.

**Networking Kubernetes y comunicación interna**

La comunicación interna del clúster Kubernetes se realiza mediante un sistema de networking basado en plugins CNI (Container Network Interface).

En el entorno KubeBackup360 se utilizó:

Canal/Calico

como solución de red Kubernetes.

Canal combina tecnologías como:

• Flannel

• Calico

**Permitiendo proporcionar:**

• comunicación entre Pods

• asignación automática de IPs internas

• conectividad entre nodos Kubernetes

• comunicación distribuida dentro del clúster

Cada Pod desplegado recibe automáticamente una dirección IP interna perteneciente a la red Kubernetes, permitiendo la comunicación entre aplicaciones ejecutadas sobre distintos nodos worker.

Asimismo, Kubernetes utiliza CoreDNS para proporcionar resolución DNS interna dentro del clúster, facilitando la comunicación entre Pods y Services mediante nombres lógicos.

**Validación inicial del entorno Kubernetes**

Antes de comenzar el despliegue de aplicaciones, se realizó una comprobación inicial del estado general del clúster verificando la disponibilidad de nodos y el correcto funcionamiento de los principales componentes Kubernetes.

Para ello, se ejecutó:

**kubectl get nodes**

La validación confirmó que todos los nodos permanecían en estado:

**Ready**

Posteriormente, se verificó el estado de los principales componentes desplegados previamente dentro del entorno Kubernetes:

**kubectl get pods -A | grep -E 'velero|minio|grafana|prometheus|coredns'**

Esta comprobación permitió validar el funcionamiento correcto de:

• persistencia NFS

• MinIO

• Velero

• Prometheus

• Grafana

• CoreDNS

• networking Canal/Calico

antes de comenzar el despliegue de nuevas aplicaciones dentro del clúster.

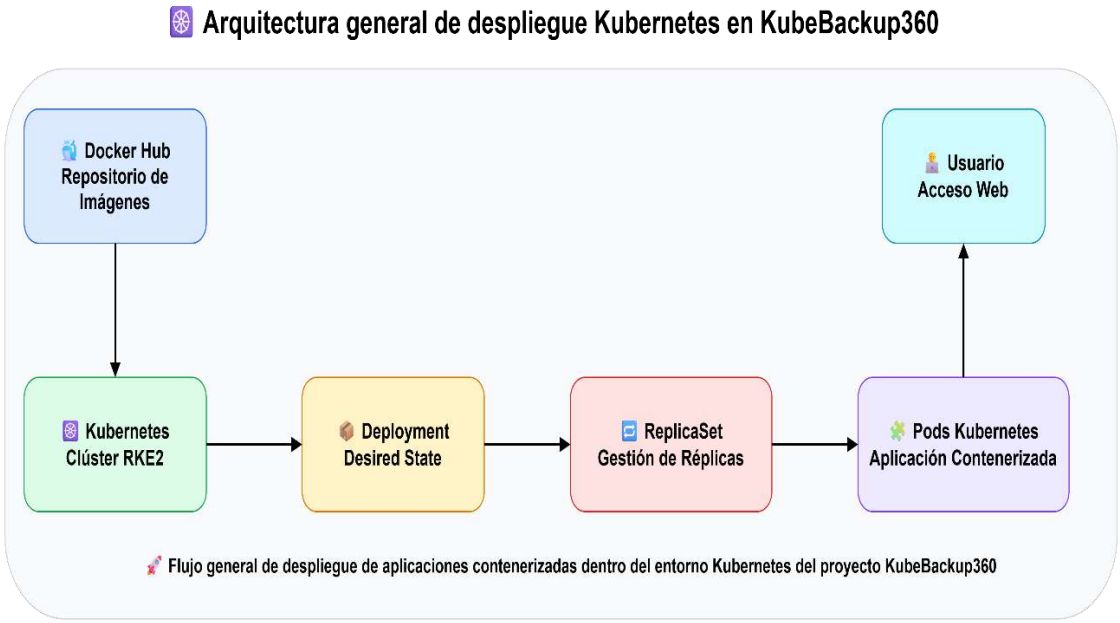
**Verificación inicial del estado del clúster Kubernetes**

```
usuario@kb360-master:~$ kubectl get nodes
NAME           STATUS   ROLES                  AGE    VERSION
kb360-master   Ready    control-plane,etcd     5d8h   v1.35.4+rke2r1
kb360-w1       Ready    <none>                 5d8h   v1.35.4+rke2r1
kb360-w2       Ready    <none>                 5d8h   v1.35.4+rke2r1
kb360-w3       Ready    <none>                 5d8h   v1.35.4+rke2r1
```

```
usuario@kb360-master:~$ kubectl get pods -A | grep -E 'velero|minio|grafana|prometheus|coredns'
kube-system   helm-install-rke2-coredns-kksrj                          0/1     Completed   0              5d9h
kube-system   rke2-coredns-rke2-coredns-75bc4f545d-794d9               1/1     Running     0              5d9h
kube-system   rke2-coredns-rke2-coredns-75bc4f545d-nghlq               1/1     Running     0              2d8h
kube-system   rke2-coredns-rke2-coredns-autoscaler-79774bc964-8fhdp    1/1     Running     0              5d9h
minio         minio-689d4cb9b4-hqbv9                                   1/1     Running     0              2d6h
monitoring    monitoring-grafana-55966d7c6f-n9hhn                      3/3     Running     7 (22h ago)    30h
monitoring    monitoring-kube-prometheus-operator-54f68d65b4-47dqn     1/1     Running     0              30h
monitoring    monitoring-prometheus-node-exporter-29gmp                1/1     Running     0              30h
monitoring    monitoring-prometheus-node-exporter-frphl                1/1     Running     0              30h
monitoring    monitoring-prometheus-node-exporter-m5txm                1/1     Running     0              30h
monitoring    monitoring-prometheus-node-exporter-swm2s                1/1     Running     0              30h
monitoring    prometheus-monitoring-kube-prometheus-prometheus-0       2/2     Running     0              30h
velero        node-agent-bzxqh                                         1/1     Running     0              2d1h
velero        node-agent-jjdnn                                         1/1     Running     0              2d1h
velero        node-agent-mljm7                                         1/1     Running     0              2d1h
velero        node-agent-q8p9p                                         1/1     Running     0              43h
velero        velero-5c96d4c78b-tgpf7                                  1/1     Running     0              2d1h
usuario@kb360-master:~$
```

- nodos Ready
- componentes críticos Running
- CoreDNS operativo
- entorno Kubernetes estable

*El flujo general utilizado durante esta fase queda representado en la siguiente arquitectura:*




El diagrama representa el flujo completo utilizado para desplegar aplicaciones contenerizadas dentro del clúster Kubernetes del proyecto KubeBackup360.

Durante esta fase también se validarán distintos mecanismos avanzados de Kubernetes relacionados con:

- balanceo de carga
- distribución automática entre workers
- alta disponibilidad
- autorecuperación de Pods
- actualización progresiva de aplicaciones

Todo ello permitirá reproducir una arquitectura Kubernetes funcional similar a la utilizada en infraestructuras cloud-native y entornos DevOps profesionales.

**8.1 Primer despliegue de aplicación en Kubernetes**
Desplegar por primera vez una aplicación contenerizada dentro del clúster Kubernetes KubeBackup360 utilizando una imagen Docker almacenada en Docker Hub.

Durante este proceso se validará:

• descarga automática de imágenes Docker

• creación de Pods Kubernetes

• ejecución de contenedores dentro del clúster

• asignación automática de IPs internas

• conectividad interna Kubernetes

La aplicación desplegada utilizará la imagen Docker creada previamente durante la fase de contenerización:

usuario-dockerhub/kubebackup360-web:v1

Esta imagen contiene:

• servidor NGINX

• aplicación web personalizada KubeBackup360

• contenido HTML desplegable dentro de Kubernetes

El despliegue inicial se realizó directamente mediante el comando:

**kubectl run kubebackup360-web **

**--image=usuario-dockerhub/kubebackup360-web:v1**

Durante la ejecución, Kubernetes realizó automáticamente:

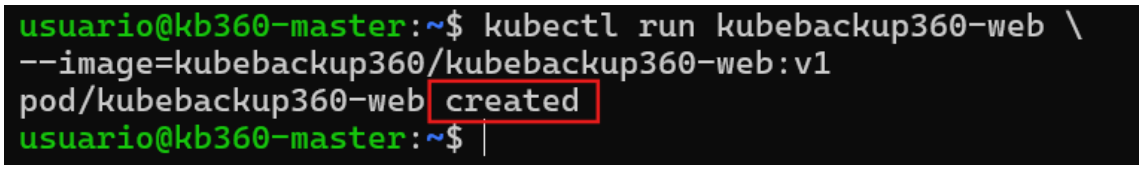
• descarga de la imagen desde Docker Hub

• creación del Pod Kubernetes

• asignación del Pod a un nodo worker

• inicialización del contenedor NGINX

**Creación del primer Pod Kubernetes desde Docker Hub**




• creación correcta del Pod

• imagen Docker utilizada

• ejecución del comando kubectl run

Posteriormente, se verificó el estado del Pod desplegado mediante:

**kubectl get pods -o wide**

La validación permitió comprobar:

• estado Running

• IP interna asignada

• nodo worker utilizado

• tiempo de ejecución del Pod

**Verificación del Pod desplegado en Kubernetes**

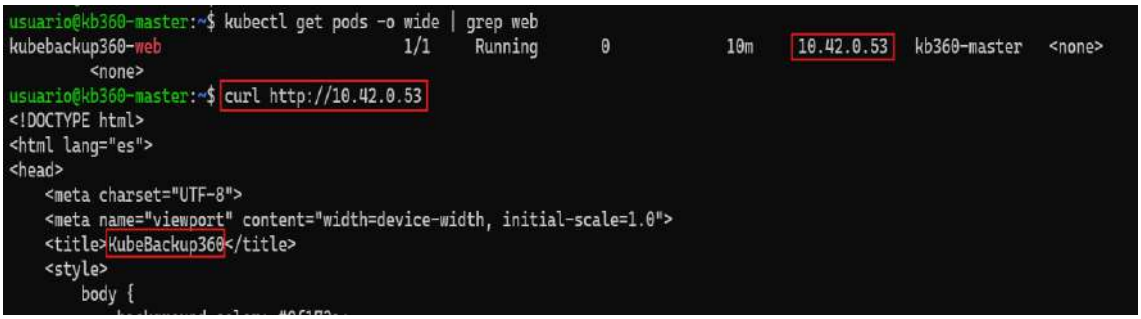

La salida confirmó que Kubernetes descargó correctamente la imagen Docker desde Docker Hub y desplegó el contenedor dentro del clúster Kubernetes.

Posteriormente, se validó la conectividad interna del Pod mediante:

**curl http://IP_DEL_POD**

La respuesta obtenida mostró correctamente el contenido HTML de la aplicación KubeBackup360, confirmando el funcionamiento operativo del contenedor NGINX desplegado dentro del Pod Kubernetes.

**Validación HTTP interna del Pod Kubernetes**




• respuesta HTML de la aplicación

• conectividad interna Kubernetes

• funcionamiento correcto del Pod

Durante las pruebas también se observaron incidencias relacionadas con Canal/Calico sobre uno de los nodos worker del clúster, afectando temporalmente la conectividad de algunos Pods.

Estas incidencias fueron diagnosticadas y corregidas posteriormente durante las validaciones de networking Kubernetes.

La validación realizada confirmó el correcto funcionamiento del primer despliegue de aplicaciones contenerizadas dentro del clúster Kubernetes del proyecto KubeBackup360.

### 8.2 Creación de Deployments

Implementar un Deployment Kubernetes para administrar de forma automatizada la aplicación desplegada anteriormente mediante Pods manuales.

El uso de Deployments permite que Kubernetes gestione automáticamente:

• creación de Pods

• autorecuperación

• escalabilidad horizontal

• actualizaciones de aplicaciones

• mantenimiento del estado deseado

*A diferencia del despliegue manual realizado anteriormente mediante **kubectl run** los Deployments permiten administrar aplicaciones de forma persistente y automatizada dentro del clúster Kubernetes.*

#### Eliminación del Pod manual

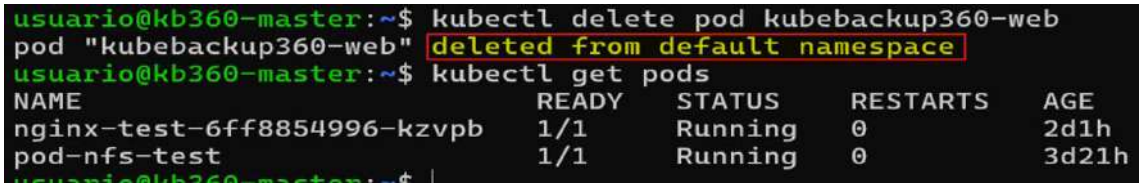
Antes de crear el Deployment, se eliminó el Pod creado manualmente durante el apartado anterior para evitar conflictos y duplicidad de recursos dentro del clúster.

Para ello, se ejecutó:

**kubectl delete pod kubebackup360-web**

La validación confirmó la eliminación correcta del Pod manual desplegado previamente.

**Eliminación del Pod manual Kubernetes**




#### Creación del archivo Deployment YAML

Posteriormente, se creó un archivo YAML declarativo para definir el Deployment Kubernetes encargado de administrar automáticamente la aplicación desplegada.

Para ello, se ejecutó:

**nano kubebackup360-deployment.yaml**

El Deployment configurado define:

• imagen Docker utilizada

• nombre del Deployment

• selector de Pods

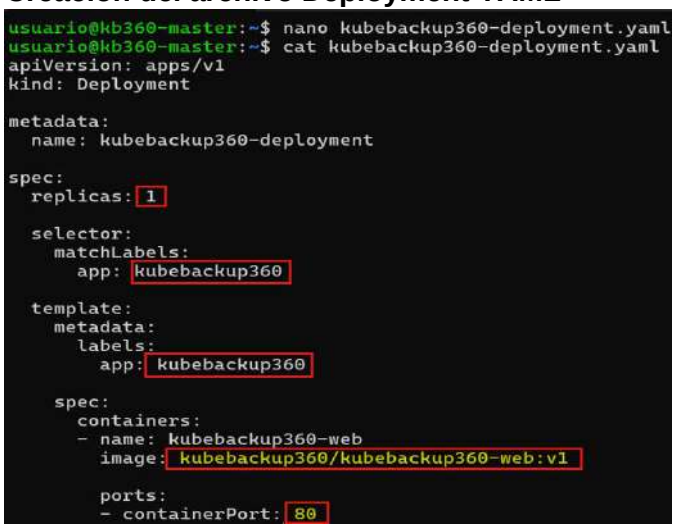
• número de réplicas

• contenedor NGINX

• etiquetas Kubernetes

*El uso de archivos YAML permite administrar recursos Kubernetes mediante configuraciones reutilizables y fácilmente modificables.*

**Creación del archivo Deployment YAML**




• contenido parcial del YAML

• imagen Docker utilizada

• replicas definidas

**Aplicación del Deployment Kubernetes**

Una vez creado el archivo YAML, el Deployment fue desplegado dentro del clúster Kubernetes mediante:

**kubectl apply -f kubebackup360-deployment.yaml**

Durante este proceso, Kubernetes creó automáticamente:

• Deployment

• ReplicaSet

• Pod administrado automáticamente

**Despliegue del Deployment Kubernetes**


• deployment created

• aplicación correcta del YAML

**Verificación del Deployment**

Finalmente, se verificó el estado del Deployment y de los Pods generados automáticamente mediante:

**kubectl get deployments**
**kubectl get pods -o wide**

La validación permitió comprobar:

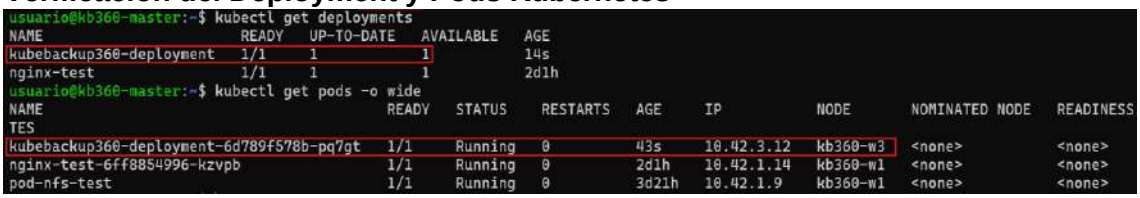
• Deployment operativo

• ReplicaSet activo

• Pod Running

• asignación automática a nodo worker

**Verificación del Deployment y Pods Kubernetes**




• Deployment disponible

• réplicas activas

• Pods Running

• nodo worker asignado

La implementación del Deployment permitió automatizar la administración de la aplicación contenerizada dentro del clúster Kubernetes del proyecto KubeBackup360, preparando el entorno para futuras pruebas de escalabilidad, autorecuperación y balanceo de carga.

**8.3 Exposición de aplicaciones mediante Services NodePort**

Exponer externamente la aplicación desplegada dentro del clúster Kubernetes utilizando un Service de tipo NodePort, permitiendo el acceso web desde equipos externos al entorno Kubernetes.

Los Services Kubernetes permiten proporcionar conectividad estable a los Pods, independientemente de los cambios dinámicos de IP generados dentro del clúster.

Durante este apartado se validará:

• comunicación entre Pods y Services

• exposición externa mediante NodePort

• acceso web desde navegador

• balanceo interno Kubernetes

• conectividad distribuida del clúster

**Verificación inicial del Deployment**

Antes de crear el Service Kubernetes, se verificó el estado del Deployment desplegado anteriormente mediante:

**kubectl get deployments**

**kubectl get pods -o wide**

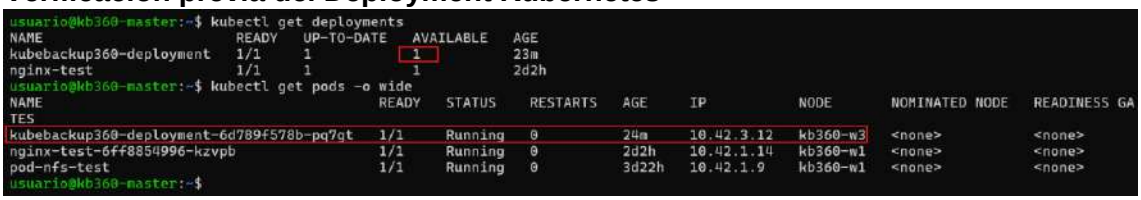
La validación confirmó:

• Deployment disponible

• Pods Running

• distribución correcta sobre nodos worker

**Verificación previa del Deployment Kubernetes**




**Creación del Service NodePort**

Posteriormente, se creó un archivo YAML para definir el Service Kubernetes encargado de exponer externamente la aplicación desplegada.

Para ello, se ejecutó:

**nano kubebackup360-service.yaml**

El Service configurado utiliza el tipo:

**NodePort**

permitiendo acceder a la aplicación desde fuera del clúster mediante:

**IP_WORKER:PUERTO_NODEPORT**

El archivo YAML define:

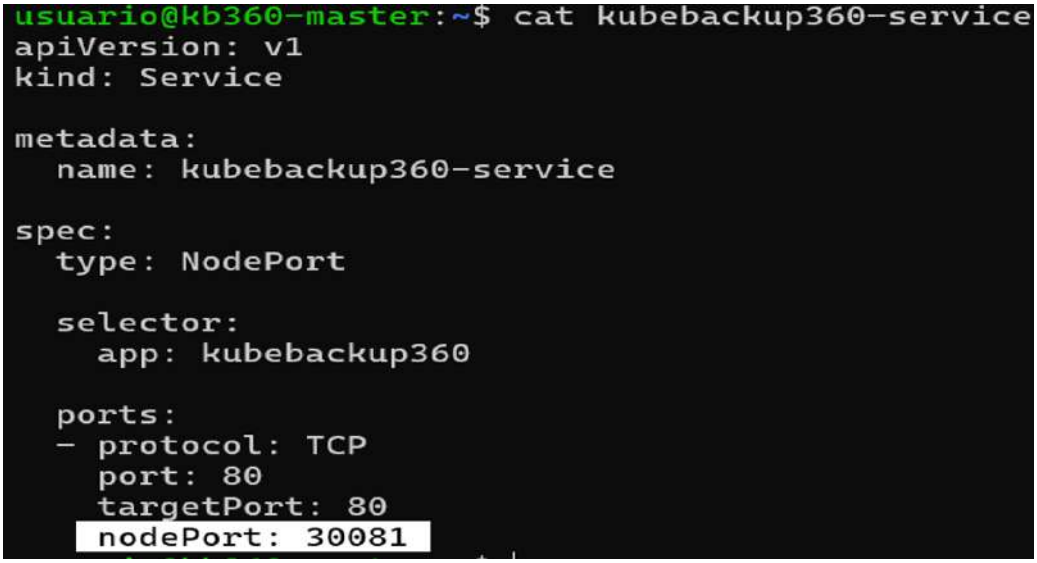
• selector de Pods

• puerto interno del contenedor

• puerto Service Kubernetes

• puerto NodePort externo

**Creación del archivo Service NodePort YAML**



**Explicación:**
selector
app: kubebackup360
*Relaciona el Service con los Pods del Deployment.*
port
*Puerto interno del Service.*
targetPort
Puerto del contenedor NGINX.
nodePort : 30081
*Puerto externo accesible desde la red*

**Aplicación del Service Kubernetes**

Una vez creado el archivo YAML, el Service fue desplegado dentro del clúster Kubernetes mediante:

**kubectl apply -f kubebackup360-service.yaml**

La validación confirmó la creación correcta del Service NodePort encargado de exponer externamente la aplicación Kubernetes.

**Despliegue del Service NodePort Kubernetes**

```
usuario@kb360-master:~$ nano kubebackup360-service.yaml
usuario@kb360-master:~$ kubectl apply -f kubebackup360-service.yaml
service/kubebackup360-service created
usuario@kb360-master:~$ kubectl get services
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubebackup360-service   NodePort    10.43.22.150    <none>        80:30081/TCP   86s
kubernetes              ClusterIP   10.43.0.1       <none>        443/TCP        5d15h
nginx-test              NodePort    10.43.196.237   <none>        80:32595/TCP   5d11h
```

• service created

• despliegue correcto del Service

**Verificación del Service NodePort**

Posteriormente, se verificó el estado del Service Kubernetes mediante:

**kubectl get svc**

La validación permitió comprobar:

• Service activo

• dirección interna Kubernetes

• puerto NodePort asignado

• exposición externa del servicio

**Verificación del Service NodePort Kubernetes**

```
usuario@kb360-master:~$ kubectl get svc
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubebackup360-service   NodePort    10.43.22.150    <none>        80:30081/TCP   6d10h
kubernetes              ClusterIP   10.43.0.1       <none>        443/TCP        12d
nginx-test              NodePort    10.43.196.237   <none>        80:32595/TCP   11d
usuario@kb360-master:~$
```

**Validación de acceso web externo**

Finalmente, se validó el acceso externo a la aplicación Kubernetes desde un navegador web utilizando:

**http://IP_WORKER:30081**

La respuesta obtenida mostró correctamente la aplicación web KubeBackup360 desplegada dentro del clúster Kubernetes.

Asimismo, se realizó una validación adicional mediante:

**curl http://IP_WORKER:30081**

confirmando:

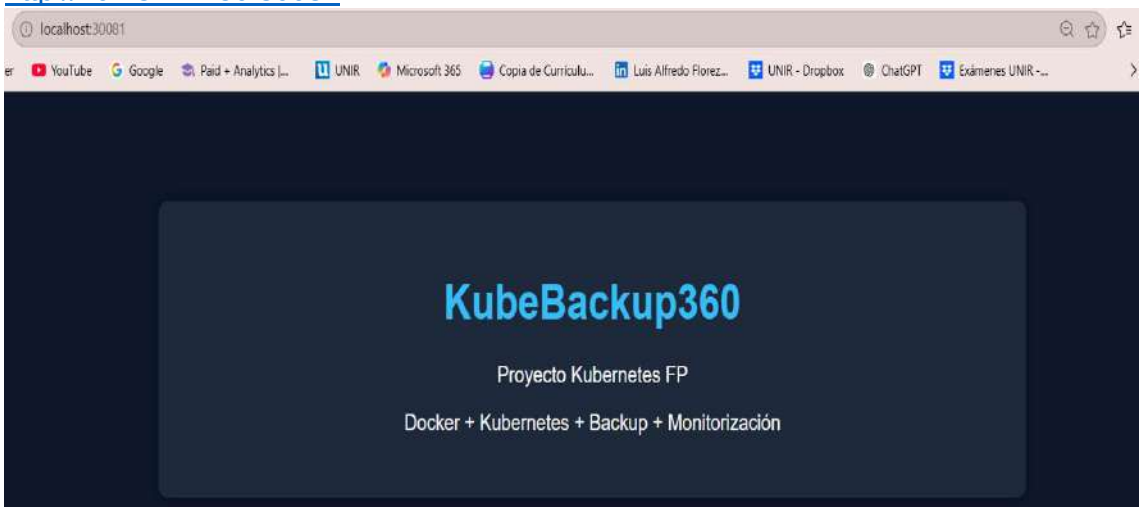
• conectividad externa funcional

• balanceo correcto del Service

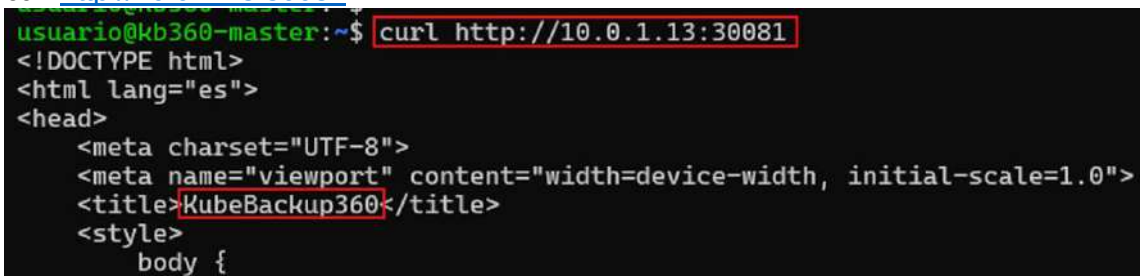
• acceso operativo a la aplicación

**Validación web del Service NodePort Kubernetes**

http://10.43.22.150:30081




curl http://10.0.1.13:30081




***Nota:*** *Durante las pruebas realizadas también se detectó inicialmente un conflicto con el puerto* ***30080*** *el cual ya estaba siendo utilizado previamente por* ***Grafana****. Por este motivo, se seleccionó posteriormente el puerto:* ***30081*** *evitando conflictos entre servicios desplegados dentro del entorno KubeBackup360.*

La implementación del Service NodePort permitió exponer correctamente la aplicación contenerizada hacia el exterior del clúster Kubernetes, validando el funcionamiento del networking y acceso distribuido dentro de la infraestructura KubeBackup360.

### 8.4 Escalabilidad horizontal en Kubernetes

Validar la capacidad de Kubernetes para escalar aplicaciones automáticamente mediante múltiples réplicas distribuidas entre distintos nodos worker del clúster.

Durante este apartado se comprobará:

• creación de múltiples Pods

• distribución automática entre workers

• balanceo de carga Kubernetes

• alta disponibilidad básica

• funcionamiento distribuido del clúster

La escalabilidad horizontal permite aumentar el número de Pods asociados a una aplicación para mejorar:

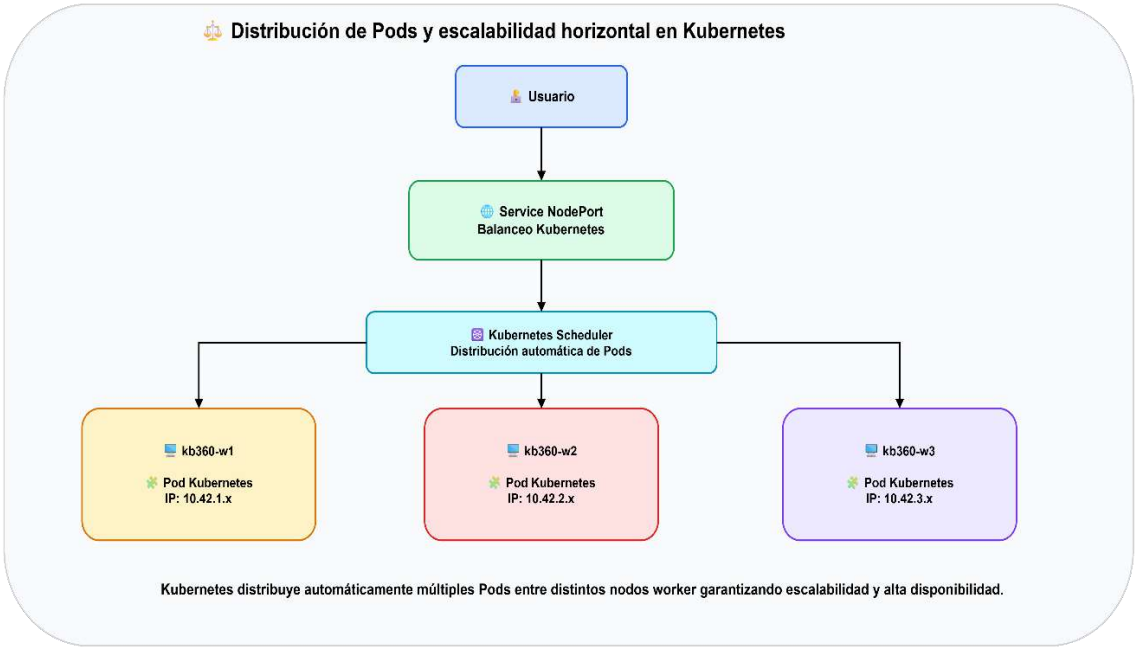
• disponibilidad

• tolerancia a fallos

• capacidad de procesamiento

• distribución de carga

**Distribución de Pods y escalabilidad horizontal en Kubernetes**




**Kubernetes administra automáticamente estas réplicas mediante Deployments y ReplicaSets.**

**Verificación inicial del Deployment**

Antes de realizar el escalado horizontal, se verificó el estado actual del Deployment y de los Pods desplegados mediante:

**kubectl get deployments**

**kubectl get pods -o wide**

La validación confirmó la existencia de una única réplica en estado operativo antes de iniciar el escalado.

**Estado inicial del Deployment Kubernetes**


• Deployment operativo

• una réplica activa

• Pod Running

**Escalado horizontal del Deployment**

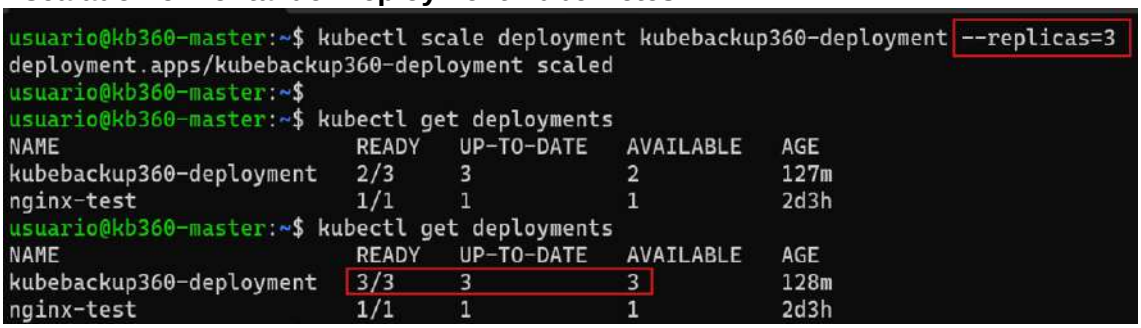
Posteriormente, se realizó el escalado horizontal del Deployment aumentando el número de réplicas a: **3**

Para ello, se ejecutó:

**kubectl scale deployment kubebackup360-deployment --replicas=3**

Durante este proceso, Kubernetes creó automáticamente nuevos Pods distribuyéndolos entre distintos nodos worker del clúster.

**Escalado horizontal del Deployment Kubernetes**




• escalado correcto del Deployment

• nuevas réplicas creadas

**Verificación de Pods distribuidos**

Una vez completado el escalado, se verificó el estado de los Pods mediante:

**kubectl get pods -o wide**

La validación permitió comprobar:

• múltiples Pods Running

• distribución automática entre workers

• direcciones IP internas distintas

• funcionamiento del scheduler Kubernetes

**Distribución de Pods entre nodos worker**

```
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                                      READY   STATUS    RESTARTS   AGE    IP           NODE
ES
kubebackup360-deployment-6d789f578b-9bwd8 1/1     Running   0          73s    10.42.1.18   kb360-w1
kubebackup360-deployment-6d789f578b-pq7gt 1/1     Running   0          128m   10.42.3.12   kb360-w3
kubebackup360-deployment-6d789f578b-qb9hd 1/1     Running   0          73s    10.42.2.14   kb360-w2
nginx-test-6ff8854996-kzvpb               1/1     Running   0          2d3h   10.42.1.14   kb360-w1
pod-nfs-test                              1/1     Running   0          4d     10.42.1.9    kb360-w1
```

Esta validación confirmó que Kubernetes distribuye automáticamente la carga de trabajo entre distintos nodos del clúster, mejorando la disponibilidad y tolerancia a fallos de la aplicación desplegada.

**Validación del acceso al Service**

Posteriormente, se validó nuevamente el acceso externo a la aplicación mediante:

**curl http://IP_WORKER:30081**

La respuesta obtenida confirmó que el Service Kubernetes continuaba funcionando correctamente tras el escalado horizontal del Deployment.

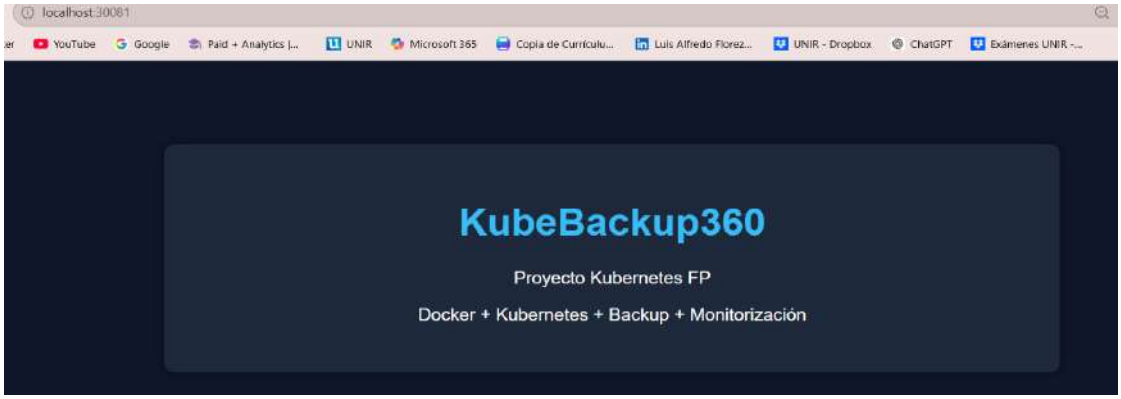
Asimismo, el Service mantuvo el balanceo de tráfico entre las distintas réplicas desplegadas automáticamente dentro del clúster.

**Validación del Service tras el escalado horizontal**

curl http://10.0.1.13:30081

```
usuario@kb360-master:~$ curl http://10.0.1.13:30081
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>KubeBackup360</title>
    <style>
```

http://localhost:30081



• respuesta HTML correcta

• acceso operativo mediante NodePort

• funcionamiento del Service Kubernetes

La validación realizada confirmó el correcto funcionamiento de la escalabilidad horizontal dentro del entorno Kubernetes KubeBackup360, demostrando la capacidad del clúster para distribuir aplicaciones contenerizadas entre múltiples nodos worker de forma automática.

### 8.5 Alta disponibilidad y autorecuperación de Pods

Validar los mecanismos de alta disponibilidad y autorecuperación automática proporcionados por Kubernetes mediante Deployments y ReplicaSets.

Durante este apartado se comprobará la capacidad del clúster Kubernetes para:

• detectar fallos en Pods

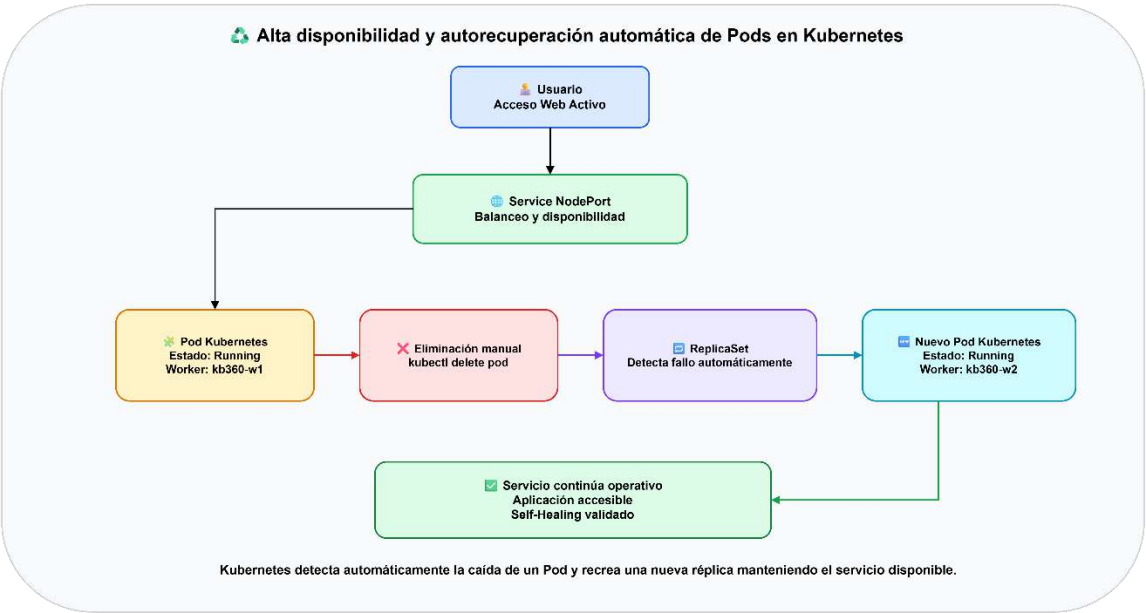
• recrear contenedores automáticamente

• mantener la disponibilidad del servicio

• garantizar el estado deseado definido por el Deployment

Uno de los principales beneficios de Kubernetes es su capacidad de self-healing, permitiendo recuperar automáticamente aplicaciones tras fallos o eliminaciones inesperadas de contenedores.

**Autorecuperación automática de Pods en Kubernetes**




**Verificación inicial del estado de los Pods**

Antes de realizar las pruebas de autorecuperación, se verificó el estado actual de los Pods desplegados dentro del clúster Kubernetes mediante:

**kubectl get pods -o wide**

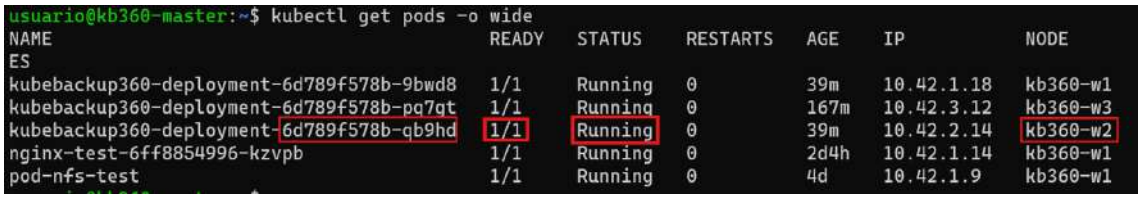
La validación confirmó:

• múltiples Pods Running

• distribución entre nodos worker

• funcionamiento correcto del Deployment

**Estado inicial de los Pods Kubernetes**




**Eliminación manual de un Pod Kubernetes**

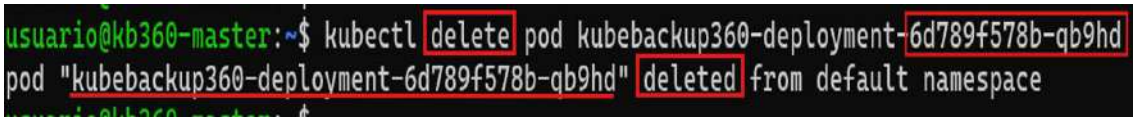
Para validar el mecanismo de autorecuperación, se eliminó manualmente uno de los Pods gestionados por el Deployment.

Para ello, se ejecutó:

**kubectl delete pod NOMBRE_DEL_POD**

Durante este proceso, Kubernetes detectó automáticamente la pérdida del Pod y procedió a crear una nueva réplica para mantener el número deseado definido dentro del Deployment.

**Eliminación manual de un Pod Kubernetes**




• Pod eliminado

• estado Terminating

• eliminación correcta del contenedor

**Autorecuperación automática**

Posteriormente, se monitorizó en tiempo real el comportamiento del clúster Kubernetes mediante:

**kubectl get pods -o wide -w**

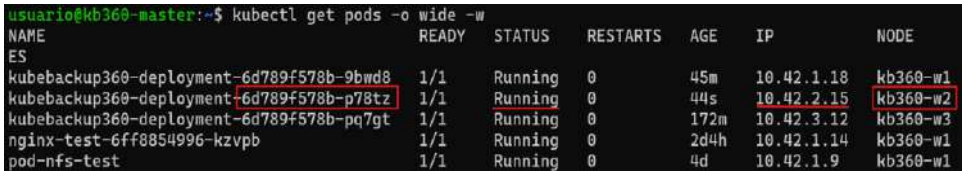
La validación permitió observar:
• desaparición del Pod eliminado

• creación automática de un nuevo Pod

• asignación automática a un worker

• nuevo Pod en estado Running

**Autorecuperación automática del Pod Kubernetes**



*Esta validación confirmó el correcto funcionamiento del mecanismo de self-healing proporcionado por Kubernetes mediante Deployments y ReplicaSets.*

**Validación del acceso al servicio tras la recuperación**

Finalmente, se verificó que la aplicación continuaba accesible externamente tras la recreación automática del Pod mediante:

**curl http://IP_WORKER:30081**

```
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                                      READY   STATUS    RESTARTS   AGE    IP           NODE       NOMINATED NODE
ES
kubebackup360-deployment-6d789f578b-9bwd8   1/1     Running   0          39m    10.42.1.18   kb360-w1   <none>
kubebackup360-deployment-6d789f578b-pq7gt   1/1     Running   0          167m   10.42.3.12   kb360-w3   <none>
kubebackup360-deployment-6d789f578b-qb9hd   1/1     Running   0          39m    10.42.2.14   kb360-w2   <none>
nginx-test-6ff8854996-kzvpb               1/1     Running   0          2d4h   10.42.1.14   kb360-w1   <none>
pod-nfs-test                              1/1     Running   0          4d     10.42.1.9    kb360-w1   <none>
usuario@kb360-master:~$
usuario@kb360-master:~$ kubectl delete pod kubebackup360-deployment-6d789f578b-qb9hd
pod "kubebackup360-deployment-6d789f578b-qb9hd" deleted from default namespace
usuario@kb360-master:~$
usuario@kb360-master:~$ kubectl get pods -o wide -w
NAME                                      READY   STATUS    RESTARTS   AGE    IP           NODE       NOMINATED NODE
ES
kubebackup360-deployment-6d789f578b-9bwd8   1/1     Running   0          45m    10.42.1.18   kb360-w1   <none>
kubebackup360-deployment-6d789f578b-p78tz   1/1     Running   0          44s    10.42.2.15   kb360-w2   <none>
kubebackup360-deployment-6d789f578b-pq7gt   1/1     Running   0          172m   10.42.3.12   kb360-w3   <none>
nginx-test-6ff8854996-kzvpb               1/1     Running   0          2d4h   10.42.1.14   kb360-w1   <none>
pod-nfs-test                              1/1     Running   0          4d     10.42.1.9    kb360-w1   <none>
^Cusuario@kb360-master:~curl http://10.0.1.13:30081

<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>KubeBackup360</title>
    <style>
        body {
```

*La respuesta obtenida confirmó que el Service Kubernetes continuó funcionando correctamente durante todo el proceso de recuperación automática.*

• respuesta HTML correcta

• aplicación accesible mediante NodePort

• servicio operativo tras la recreación automática

*La validación realizada permitió comprobar la capacidad del clúster Kubernetes KubeBackup360 para detectar fallos y recuperar automáticamente aplicaciones contenerizadas sin intervención manual, garantizando así mecanismos básicos de alta disponibilidad dentro de la infraestructura desplegada.*

**8.6 Rolling Update y Rollback en Kubernetes**

Validar los mecanismos de actualización progresiva y restauración automática proporcionados por Kubernetes mediante Deployments.

Durante este apartado se comprobará la capacidad del clúster Kubernetes para:

• actualizar aplicaciones sin interrupción del servicio

• reemplazar Pods progresivamente

• mantener disponibilidad durante actualizaciones

• restaurar versiones anteriores mediante rollback

Kubernetes permite realizar actualizaciones controladas de aplicaciones contenerizadas utilizando el mecanismo denominado:

**Rolling Update**

Este sistema reemplaza progresivamente los Pods antiguos por nuevas versiones de la aplicación sin detener completamente el servicio.

Asimismo, Kubernetes permite restaurar automáticamente versiones anteriores mediante:

**Rollback**

facilitando la recuperación rápida ante errores o fallos durante una actualización.

**Verificación inicial del Deployment**

Antes de realizar la actualización, se verificó el estado actual del Deployment y de los Pods desplegados dentro del clúster mediante:

**kubectl get deployments**
**kubectl get pods -o wide**

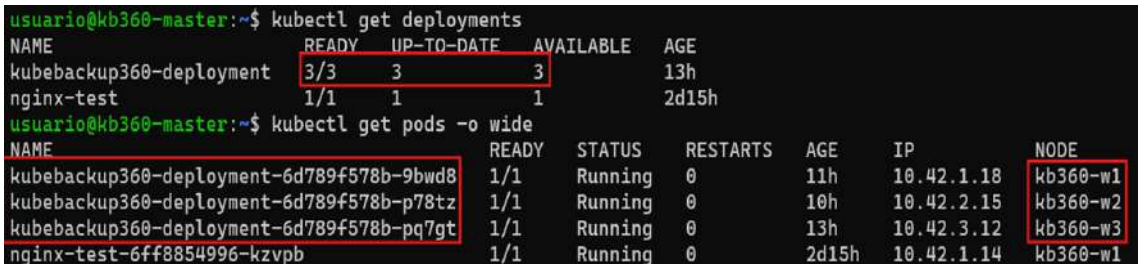
La validación confirmó:

• Deployment operativo

• Pods Running

• Service funcionando correctamente

**Estado inicial del Deployment antes del Rolling Update**




• Deployment activo

• Pods Running

• distribución sobre workers

**Ejecución del Rolling Update**

Posteriormente, se realizó una actualización progresiva del Deployment modificando la imagen utilizada por la aplicación.

Para ello, se ejecutó:

kubectl set image deployment/kubebackup360-deployment \

kubebackup360-web=nginx:latest

Durante este proceso, Kubernetes reemplazó automáticamente los Pods antiguos por nuevos Pods utilizando la nueva imagen definida.

**Ejecución del Rolling Update Kubernetes**

```
usuario@kb360-master:~$ kubectl set image deployment/kubebackup360-deployment \
kubebackup360-web=nginx:latest
deployment.apps/kubebackup360-deployment image updated
```

• actualización iniciada

• nueva imagen aplicada

**Monitorización de la actualización progresiva**

La evolución del Rolling Update se monitorizó en tiempo real mediante:

kubectl get pods -o wide -w

La validación permitió observar:

• Pods antiguos en estado Terminating

• creación progresiva de nuevos Pods

• mantenimiento del servicio operativo

• distribución automática entre workers

**Monitorización del Rolling Update en tiempo real**

```
usuario@kb360-master:~$ kubectl get pods -o wide -w
NAME                                    READY   STATUS              RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
kubebackup360-deployment-6d789f578b-9bwd8   1/1     Terminating         0          11h     10.42.1.18   kb360-w1   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   1/1     Running             0          11h     10.42.2.15   kb360-w2   <none>           <none>
kubebackup360-deployment-df5899996-h4q9f    0/1     ContainerCreating   0          0s      <none>       kb360-w2   <none>           <none>
kubebackup360-deployment-df5899996-scdsm    1/1     Running             0          2s      10.42.1.19   kb360-w1   <none>           <none>
kubebackup360-deployment-df5899996-x9fp6    1/1     Running             0          4s      10.42.3.13   kb360-w3   <none>           <none>
nginx-test-6ff8854996-kzvpb             1/1     Running             0          2d15h   10.42.1.14   kb360-w1   <none>           <none>
pod-nfs-test                            1/1     Running             0          4d11h   10.42.1.9    kb360-w1   <none>           <none>
kubebackup360-deployment-6d789f578b-9bwd8   1/1     Terminating         0          11h     10.42.1.18   kb360-w1   <none>           <none>
kubebackup360-deployment-df5899996-h4q9f    0/1     ContainerCreating   0          1s      <none>       kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-9bwd8   0/1     Completed           0          11h     10.42.1.18   kb360-w1   <none>           <none>
kubebackup360-deployment-6d789f578b-9bwd8   0/1     Completed           0          11h     10.42.1.18   kb360-w1   <none>           <none>
kubebackup360-deployment-6d789f578b-9bwd8   0/1     Completed           0          11h     10.42.1.18   kb360-w1   <none>           <none>
kubebackup360-deployment-df5899996-h4q9f    1/1     Running             0          3s      10.42.2.16   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   1/1     Terminating         0          11h     10.42.2.15   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   1/1     Terminating         0          11h     10.42.2.15   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   1/1     Terminating         0          11h     10.42.2.15   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   0/1     Completed           0          11h     10.42.2.15   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   0/1     Completed           0          11h     10.42.2.15   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-p78tz   0/1     Completed           0          11h     10.42.2.15   kb360-w2   <none>           <none>
^Cusuario@kb360-master:~$ kubectl get pods -o wide
NAME                                    READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GATES
kubebackup360-deployment-df5899996-h4q9f    1/1     Running   0          55s     10.42.2.16   kb360-w2   <none>           <none>
kubebackup360-deployment-df5899996-scdsm    1/1     Running   0          57s     10.42.1.19   kb360-w1   <none>           <none>
kubebackup360-deployment-df5899996-x9fp6    1/1     Running   0          59s     10.42.3.13   kb360-w3   <none>           <none>
nginx-test-6ff8854996-kzvpb             1/1     Running   0          2d15h   10.42.1.14   kb360-w1   <none>           <none>
pod-nfs-test                            1/1     Running   0          4d11h   10.42.1.9    kb360-w1   <none>           <none>
usuario@kb360-master:~$
```

• Pods antiguos eliminándose

• nuevos Pods Running

• actualización progresiva Kubernetes

La validación confirmó el correcto funcionamiento del mecanismo Rolling Update dentro del clúster Kubernetes KubeBackup360.

**Ejecución del Rollback**

Posteriormente, se realizó una restauración automática de la versión anterior mediante:

kubectl rollout undo deployment/kubebackup360-deployment

Kubernetes restauró automáticamente la configuración previa del Deployment, recuperando la versión estable utilizada anteriormente.

**Restauración automática mediante Rollback**

```
usuario@kb360-master:~$ kubectl rollout undo deployment/kubebackup360-deployment
deployment.apps/kubebackup360-deployment rolled back
usuario@kb360-master:~$ kubectl get pods -o wide -w
NAME                                    READY   STATUS              RESTARTS   AGE     IP           NODE
TES
kubebackup360-deployment-6d789f578b-cgcl5   0/1     ContainerCreating   0          2s      <none>       kb360-w1
kubebackup360-deployment-6d789f578b-w5rrs   1/1     Running             0          3s      10.42.3.14   kb360-w3
kubebackup360-deployment-df5899996-h4q9f    1/1     Running             0          9m46s   10.42.2.16   kb360-w2
kubebackup360-deployment-df5899996-scdsm    1/1     Running             0          9m48s   10.42.1.19   kb360-w1
kubebackup360-deployment-df5899996-x9fp6    0/1     Completed           0          9m50s   10.42.3.13   kb360-w3
nginx-test-6ff8854996-kzvpb             1/1     Running             0          2d15h   10.42.1.14   kb360-w1
pod-nfs-test                            1/1     Running             0          4d12h   10.42.1.9    kb360-w1
kubebackup360-deployment-6d789f578b-cgcl5   1/1     Running             0          2s      10.42.1.20   kb360-w1
kubebackup360-deployment-df5899996-scdsm    1/1     Terminating         0          9m48s   10.42.1.19   kb360-w1
kubebackup360-deployment-6d789f578b-jh4v8   0/1     Pending             0          0s      <none>       <none>
kubebackup360-deployment-df5899996-scdsm    1/1     Terminating         0          9m48s   10.42.1.19   kb360-w1
kubebackup360-deployment-6d789f578b-jh4v8   0/1     Pending             0          0s      <none>       kb360-w2
kubebackup360-deployment-6d789f578b-jh4v8   0/1     ContainerCreating   0          0s      <none>       kb360-w2
kubebackup360-deployment-df5899996-x9fp6    0/1     Completed           0          9m50s   10.42.3.13   kb360-w3
kubebackup360-deployment-df5899996-x9fp6    0/1     Completed           0          9m50s   10.42.3.13   kb360-w3
kubebackup360-deployment-df5899996-scdsm    1/1     Terminating         0          9m49s   10.42.1.19   kb360-w1
kubebackup360-deployment-6d789f578b-jh4v8   0/1     ContainerCreating   0          1s      <none>       kb360-w2
kubebackup360-deployment-df5899996-scdsm    0/1     Completed           0          9m49s   10.42.1.19   kb360-w1
kubebackup360-deployment-df5899996-scdsm    0/1     Completed           0          9m49s   10.42.1.19   kb360-w1
kubebackup360-deployment-df5899996-scdsm    0/1     Completed           0          9m49s   10.42.1.19   kb360-w1
kubebackup360-deployment-6d789f578b-jh4v8   1/1     Running             0          3s      10.42.2.17   kb360-w2
kubebackup360-deployment-df5899996-h4q9f    1/1     Terminating         0          9m49s   10.42.2.16   kb360-w2
kubebackup360-deployment-df5899996-h4q9f    1/1     Terminating         0          9m49s   10.42.2.16   kb360-w2
kubebackup360-deployment-df5899996-h4q9f    1/1     Terminating         0          9m49s   10.42.2.16   kb360-w2
kubebackup360-deployment-df5899996-h4q9f    0/1     Completed           0          9m50s   10.42.2.16   kb360-w2
kubebackup360-deployment-df5899996-h4q9f    0/1     Completed           0          9m50s   10.42.2.16   kb360-w2
kubebackup360-deployment-df5899996-h4q9f    0/1     Completed           0          9m50s   10.42.2.16   kb360-w2
```

• rollback ejecutado correctamente

• restauración del Deployment

**Verificación final del Deployment**

Finalmente, se verificó nuevamente el estado del Deployment y de los Pods desplegados mediante:

**kubectl get deployments**
**kubectl get pods -o wide**

Asimismo, se validó el acceso web a la aplicación utilizando:

**curl http://IP_WORKER:30081**

La validación confirmó:

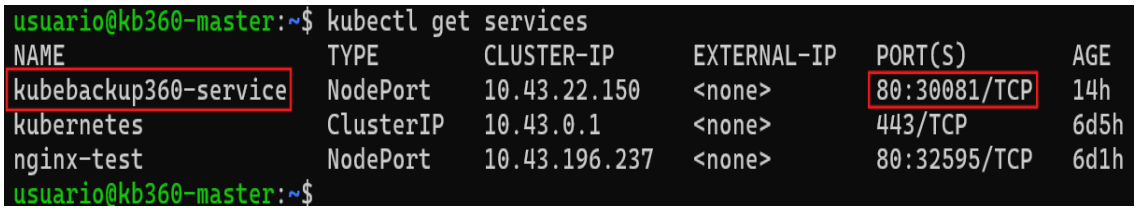
• Deployment operativo

• Pods Running

• restauración correcta de la aplicación

• disponibilidad continua del servicio

**Validación final tras Rolling Update y Rollback**




- exposición externa
- networking Kubernetes
- acceso correcto a la aplicación web

*La validación realizada permitió comprobar el correcto funcionamiento de los mecanismos de actualización progresiva y recuperación automática proporcionados por Kubernetes, garantizando así continuidad de servicio y administración avanzada de aplicaciones dentro del entorno KubeBackup360.*

**8.7 Validación final y conclusión técnica de la Fase 4**

Realizar una validación global de toda la infraestructura desplegada durante la Fase 4 comprobando el correcto funcionamiento integrado de los componentes Kubernetes utilizados para el despliegue de aplicaciones contenerizadas.

Durante esta validación se comprobará:

• estado general del clúster Kubernetes

• funcionamiento de Deployments y ReplicaSets

• distribución de Pods entre workers

• Services NodePort

• conectividad web de la aplicación

• escalabilidad horizontal

• autorecuperación automática

• Rolling Update y Rollback

La validación final permitirá confirmar que la infraestructura Kubernetes desplegada dentro del entorno KubeBackup360 funciona correctamente de forma distribuida y tolerante a fallos.

**Verificación global del clúster Kubernetes**

Inicialmente, se verificó el estado general de los nodos Kubernetes mediante:

**kubectl get nodes**

La validación confirmó que todos los nodos del clúster permanecían en estado:

Ready

garantizando la estabilidad y disponibilidad de la infraestructura Kubernetes RKE2.

**Estado final de nodos Kubernetes**

```
usuario@kb360-master:~$ kubectl get nodes
NAME           STATUS   ROLES                  AGE    VERSION
kb360-master   Ready    control-plane,etcd     6d5h   v1.35.4+rke2r1
kb360-w1       Ready    <none>                 6d4h   v1.35.4+rke2r1
kb360-w2       Ready    <none>                 6d4h   v1.35.4+rke2r1
kb360-w3       Ready    <none>                 6d4h   v1.35.4+rke2r1
```

• nodo master • nodos worker • estado Ready

**Verificación de Deployments y Pods**

Posteriormente, se verificó el estado de los Deployments y Pods desplegados dentro del clúster mediante:

**kubectl get deployments**

**kubectl get pods -o wide**

La validación permitió comprobar:

• Deployments disponibles

• Pods Running

• distribución automática entre workers

• funcionamiento del scheduler Kubernetes

**Estado final de Deployments y Pods Kubernetes**

```
usuario@kb360-master:~$ kubectl get deployments
NAME                     READY   UP-TO-DATE   AVAILABLE   AGE
kubebackup360-deployment   3/3     3            3           14h
nginx-test               1/1     1            1           2d16h
usuario@kb360-master:~$ kubectl get pods -o wide
NAME                                      READY   STATUS    RESTARTS   AGE     IP           NODE       NOMINATED NODE   READINESS GA
TES
kubebackup360-deployment-6d789f578b-cgcl5   1/1     Running   0          26m     10.42.1.20   kb360-w1   <none>           <none>
kubebackup360-deployment-6d789f578b-jh4v8   1/1     Running   0          26m     10.42.2.17   kb360-w2   <none>           <none>
kubebackup360-deployment-6d789f578b-w5rrs   1/1     Running   0          26m     10.42.3.14   kb360-w3   <none>           <none>
nginx-test-6ff8854996-kzvpb               1/1     Running   0          2d16h   10.42.1.14   kb360-w1   <none>           <none>
pod-nfs-test                              1/1     Running   0          4d12h   10.42.1.9    kb360-w1   <none>           <none>
```

- 3 Pods Running
- distribución entre workers
- IPs internas distintas
- alta disponibilidad activa

**Verificación de Services Kubernetes**

Asimismo, se verificó el funcionamiento del Service NodePort desplegado durante la fase mediante:

kubectl get svc

La validación confirmó:

• Service operativo

• puerto NodePort activo

• conectividad externa disponible

**Verificación final del Service NodePort**
Comando utilizado:

```
usuario@kb360-master:~$ kubectl get svc
NAME                    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
kubebackup360-service   NodePort    10.43.22.150    <none>        80:30081/TCP   6d18h
kubernetes              ClusterIP   10.43.0.1       <none>        443/TCP        12d
nginx-test              NodePort    10.43.196.237   <none>        80:32595/TCP   12d
usuario@kb360-master:~$
```

• Service kubebackup360-service

• TYPE NodePort

• puerto 30081

**Validación final de acceso web**

Finalmente, se validó nuevamente el acceso a la aplicación desplegada dentro del clúster Kubernetes mediante:

**curl http://IP_WORKER:30081**

```
usuario@kb360-master:~$ curl http://10.0.1.13:30081
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>KubeBackup360</title>
    <style>
```

• acceso correcto a la aplicación

• funcionamiento del Service NodePort

• conectividad operativa del clúster Kubernetes

• disponibilidad continua del servicio

• URL NodePort utilizada

• aplicación KubeBackup360 operativa

**Conclusión técnica de la Fase 4**

Durante esta fase se implementó correctamente una plataforma de despliegue de aplicaciones contenerizadas basada en Kubernetes RKE2 dentro del entorno KubeBackup360.

Las pruebas realizadas permitieron validar satisfactoriamente:

• despliegue de Pods Kubernetes

• administración mediante Deployments

• funcionamiento de ReplicaSets

• exposición de aplicaciones mediante Services NodePort

• distribución automática entre nodos worker

• escalabilidad horizontal

• autorecuperación automática de Pods

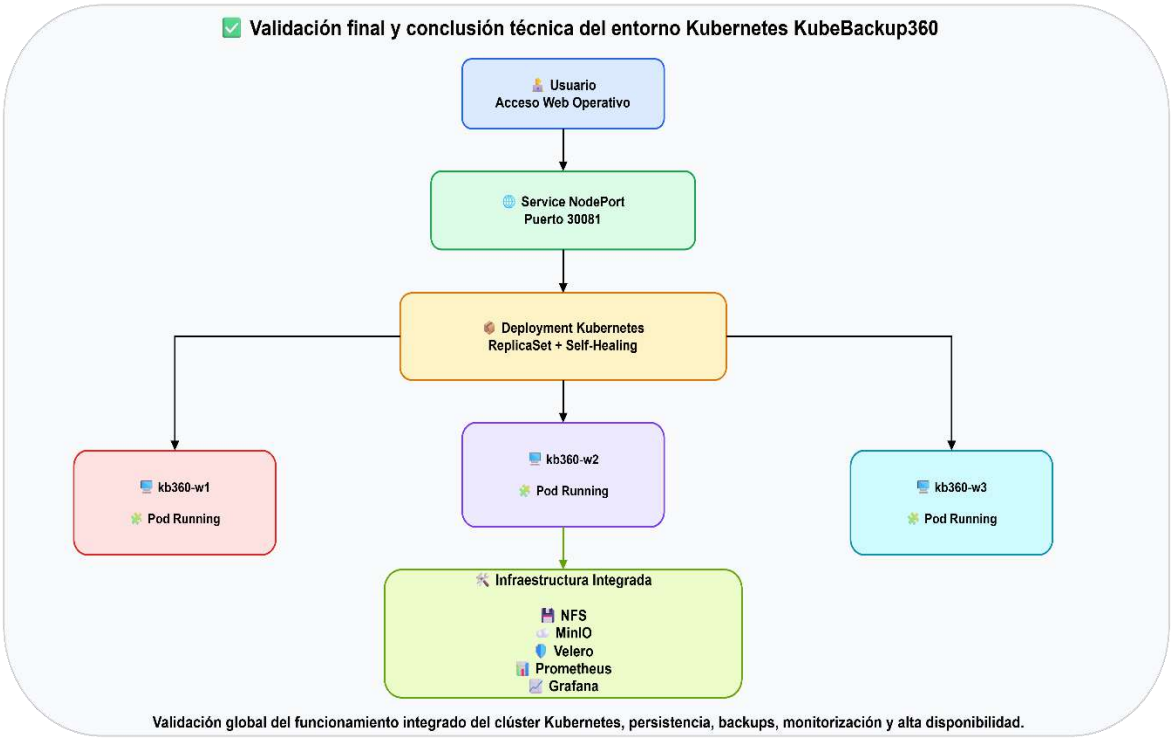
• actualizaciones progresivas mediante Rolling Update

• restauración automática mediante Rollback

Asimismo, se validó el correcto funcionamiento del networking Kubernetes mediante Canal/Calico y la resolución DNS interna proporcionada por CoreDNS.

La infraestructura desplegada permitió reproducir un entorno Kubernetes funcional similar al utilizado en plataformas cloud-native y entornos DevOps profesionales, consolidando conocimientos relacionados con administración avanzada de aplicaciones contenerizadas y orquestación distribuida dentro del proyecto KubeBackup360.

**Arquitectura global y validación funcional del proyecto KubeBackup360**



# 9. BACKUPS, RESTAURACIÓN Y VALIDACIÓN FINAL DEL SISTEMA

El objetivo de esta fase consiste en validar el funcionamiento completo del sistema de protección y recuperación implementado en el proyecto KubeBackup360, comprobando que la infraestructura Kubernetes es capaz de:

- generar copias de seguridad funcionales
- almacenar backups de forma persistente
- restaurar recursos eliminados
- mantener la persistencia de datos
- garantizar la continuidad operativa del clúster
- automatizar procesos de recuperación ante desastres
- supervisar el estado del entorno mediante monitorización

Durante esta fase se validará el correcto funcionamiento conjunto de las tecnologías implementadas previamente:

- Kubernetes RKE2
- NFS
- Velero
- MinIO
- Docker
- Prometheus
- Grafana

La validación final permitirá comprobar que la infraestructura desarrollada cumple correctamente los objetivos definidos para el proyecto KubeBackup360, incluyendo persistencia, backups, Disaster Recovery y observabilidad del entorno.

## 9.1 Restauración mediante Velero

### Objetivo del apartado

Validar el funcionamiento del sistema de recuperación ante desastres implementado mediante Velero y MinIO dentro del clúster Kubernetes.

El proceso de restauración permitirá comprobar que la infraestructura puede recuperar automáticamente aplicaciones y recursos Kubernetes tras una pérdida simulada de servicios.

### Arquitectura DRP implementada

La arquitectura de Disaster Recovery desplegada en el proyecto se basa en la siguiente estructura:

Velero
↓
Bucket MinIO
↓
Persistent Volume Claim
↓
Persistent Volume
↓
Servidor NFS

En esta arquitectura:

- Velero administra los backups y restauraciones
- MinIO proporciona almacenamiento compatible con S3
- Kubernetes administra los recursos persistentes
- NFS almacena físicamente las copias de seguridad

Esta integración permite desacoplar el almacenamiento del ciclo de vida de los contenedores y garantiza persistencia de los backups incluso ante reinicios o recreaciones de Pods dentro del clúster Kubernetes.

**Verificación de backups disponibles**

Antes de iniciar el proceso de restauración, se verificó la existencia de una copia de seguridad válida dentro de Velero.

Esta comprobación resulta fundamental para garantizar que:

- el backup existe correctamente
- el estado del backup es válido
- Velero puede acceder al almacenamiento MinIO
- la restauración puede ejecutarse sin errores

Para ello, se validó el estado de los backups almacenados previamente dentro del bucket velero configurado en MinIO. Las comprobaciones realizadas confirmaron que el backup utilizado para la recuperación se encontraba en estado Completed, garantizando así la disponibilidad del sistema de recuperación ante desastres.

**Simulación de desastre**

Con el objetivo de validar el funcionamiento real del sistema DRP, se realizó una simulación controlada de pérdida de servicio dentro del clúster Kubernetes.

Durante esta prueba se eliminó completamente una aplicación desplegada mediante Deployment, provocando:

- pérdida de los Pods asociados
- desaparición del servicio

- interrupción del acceso mediante NodePort
- indisponibilidad total de la aplicación

La simulación permitió comprobar el comportamiento del clúster ante un fallo real y validar la necesidad de disponer de mecanismos de backup y recuperación.

**Proceso de restauración**

Una vez validado el backup disponible, se procedió a iniciar el proceso de recuperación utilizando Velero.

Durante la restauración, Velero recuperó automáticamente:

- Deployments
- Pods
- Services
- configuraciones Kubernetes
- recursos asociados al clúster

El proceso confirmó que Kubernetes podía reconstruir automáticamente los objetos eliminados previamente durante la simulación de desastre, restaurando el estado funcional de la aplicación dentro del entorno.

**Validación funcional de la restauración**

Tras completar el proceso de recuperación, se realizó una validación funcional del entorno restaurado.

Las comprobaciones permitieron verificar que:

- el Deployment volvió a estar operativo
- Kubernetes recreó automáticamente los Pods
- el Service NodePort recuperó conectividad
- la aplicación volvió a responder correctamente
- el sistema DRP funcionó correctamente

Asimismo, se validó el acceso funcional a la aplicación restaurada mediante pruebas HTTP, confirmando que el servicio volvió a responder correctamente tras la recuperación.

**9.2 Automatización de backups**
**Objetivo del apartado**

Automatizar la generación periódica de copias de seguridad dentro del clúster Kubernetes utilizando Velero Schedule.

La automatización de backups permite:

- reducir intervención manual
- mejorar la protección del entorno
- mantener mecanismos DRP permanentes
- garantizar copias de seguridad periódicas
- aumentar la resiliencia operativa del clúster

**Programación automática de backups**

Una vez validado el funcionamiento general del sistema DRP, se configuraron copias de seguridad automáticas mediante Velero.

Velero utiliza expresiones cron para ejecutar backups periódicos automáticamente. Dentro del entorno del laboratorio se configuró una política de backup periódico orientada a facilitar:

- validación rápida del sistema
- generación automática de evidencias
- comprobación continua del backend MinIO
- simulación de un entorno DRP real

Asimismo, se deshabilitaron los snapshots de volúmenes incompatibles con NFS para adaptar correctamente el sistema al entorno académico implementado.

**Validación del sistema automático**

Tras configurar el sistema de automatización, se verificó que:

- Velero ejecutaba correctamente los schedules
- los backups automáticos se generaban correctamente
- MinIO almacenaba las copias de seguridad
- el backend S3 permanecía operativo
- el sistema automatizado funcionaba correctamente

Las pruebas realizadas confirmaron el correcto funcionamiento de los mecanismos automáticos de backup implementados dentro del proyecto KubeBackup360.

## 9.3 Validación final del clúster

**Objetivo del apartado**

Comprobar el estado general de la infraestructura Kubernetes tras completar las pruebas de backup, restauración y automatización.

La validación final permitió verificar:

- estabilidad del clúster

- funcionamiento de los nodos
- operatividad de Pods y Services
- persistencia del almacenamiento NFS
- disponibilidad de Velero y MinIO
- funcionamiento de Prometheus y Grafana
- integridad general del entorno

**Validación de persistencia NFS**

Se verificó que los datos almacenados sobre NFS permanecían disponibles tras el proceso de restauración.

Las pruebas realizadas confirmaron que:

- el almacenamiento persistente seguía operativo
- los datos permanecían intactos
- Kubernetes mantenía correctamente los volúmenes persistentes
- los Persistent Volumes continuaban funcionales tras el DRP

Esta validación confirmó que el almacenamiento persistente implementado mediante NFS funciona correctamente integrado con Kubernetes.

**Validación de monitorización**

También se verificó el correcto funcionamiento de la plataforma de observabilidad desplegada mediante Prometheus y Grafana.

La monitorización permitió supervisar:

- estado de nodos Kubernetes
- consumo de CPU y memoria
- estado de Pods y Deployments
- métricas del clúster
- disponibilidad de servicios
- comportamiento del sistema tras backups y restores

La validación confirmó que el sistema de observabilidad permanecía completamente operativo tras las pruebas de Disaster Recovery.

## 9.4 Conclusiones del proyecto

### 1. Lo que se esperaba hacer

El objetivo inicial del proyecto KubeBackup360 era diseñar e implementar una infraestructura Kubernetes funcional dentro de un entorno de laboratorio virtualizado, orientada a la gestión de aplicaciones containerizadas, almacenamiento persistente, copias de seguridad, monitorización y recuperación ante desastres.

Desde el planteamiento inicial, se buscaba construir un entorno lo más cercano posible a una infraestructura real, utilizando máquinas virtuales Ubuntu Server sobre VirtualBox, redes segmentadas, servicios internos de routing, DHCP y DNS, y un clúster Kubernetes desplegado mediante RKE2. La finalidad principal era comprender cómo se integran distintas tecnologías DevOps y cloud-native dentro de un mismo sistema, aplicando conceptos como alta disponibilidad, escalabilidad, persistencia, observabilidad y recuperación de servicios.

También se esperaba que el proyecto permitiera validar de forma práctica el ciclo completo de una aplicación containerizada: creación de la aplicación, construcción de la imagen Docker, publicación en Docker Hub, despliegue en Kubernetes mediante Pods, Deployments y Services, exposición mediante NodePort y validación del acceso desde la red del laboratorio.

Otro objetivo importante era implementar un sistema de copias de seguridad y restauración mediante Velero, MinIO y NFS, de forma que el clúster pudiera recuperarse ante una pérdida simulada de recursos. Esta parte resultaba fundamental, ya que el proyecto no se limitaba únicamente al despliegue de Kubernetes, sino que pretendía demostrar mecanismos reales de protección, continuidad y recuperación de servicios.

En conjunto, se esperaba obtener una infraestructura técnica completa, documentada y defendible, capaz de demostrar conocimientos de administración de sistemas, redes, virtualización, contenedores, Kubernetes, almacenamiento, backups y monitorización.

### 2. Lo que se ha hecho

Durante el desarrollo del proyecto se ha conseguido implementar una infraestructura Kubernetes funcional y coherente con los objetivos definidos. Se ha creado un entorno virtualizado compuesto por varios nodos, incluyendo un router, un nodo master, varios workers, un servidor NFS y un cliente de pruebas. Esta infraestructura ha permitido simular un entorno distribuido similar al utilizado en sistemas empresariales reales, manteniendo una red segmentada y controlada.

En la parte de red, se configuró un nodo router encargado de proporcionar conectividad entre subredes, acceso a Internet, DHCP mediante KEA y resolución DNS mediante Bind9. Esta base permitió que los distintos nodos del laboratorio pudieran comunicarse correctamente por IP y por nombre, facilitando el despliegue posterior del clúster Kubernetes.

Posteriormente, se desplegó un clúster Kubernetes mediante RKE2, formado por un nodo master y varios nodos worker. Se validó el estado de los nodos, la conectividad interna, el funcionamiento de los componentes del sistema y la capacidad del scheduler para distribuir cargas de trabajo. Esta fase permitió disponer de una base Kubernetes estable sobre la que desarrollar el resto del proyecto.

También se implementó almacenamiento persistente mediante NFS, integrándolo con Kubernetes a través de Persistent Volumes y Persistent Volume Claims. Esta parte permitió comprobar que los datos no dependían del ciclo de vida de los Pods, sino que permanecían almacenados en el servidor NFS incluso tras eliminar y recrear contenedores.

En la parte de contenedores, se instaló Docker Engine, se creó una aplicación web propia, se construyó una imagen Docker personalizada y se publicó en Docker Hub con la etiqueta:

kubebackup360/kubebackup360-web:v1

Posteriormente, esta imagen fue desplegada dentro del clúster Kubernetes, validando correctamente el flujo completo entre Docker, Docker Hub y Kubernetes. Esto permitió demostrar que el proyecto no solo ejecutaba contenedores de prueba, sino también una aplicación propia desarrollada específicamente para KubeBackup360.

Además, se desplegó la aplicación mediante Deployments y Services, se configuró el acceso mediante NodePort y se realizaron pruebas de escalabilidad horizontal, aumentando el número de réplicas y comprobando la distribución de Pods entre distintos nodos worker. También se validó el comportamiento de Kubernetes ante fallos, demostrando la capacidad de recuperación y redistribución de cargas gestionadas mediante Deployments.

Una de las partes más importantes del proyecto fue la implementación del sistema de backups y recuperación ante desastres. Para ello se desplegó MinIO como almacenamiento compatible con S3, utilizando almacenamiento persistente basado en NFS, y posteriormente se configuró Velero para realizar copias de seguridad del clúster. Se validaron backups manuales, backups programados y restauraciones tras la eliminación controlada de recursos. Esta fase permitió demostrar el funcionamiento real de un proceso de Disaster Recovery dentro del entorno Kubernetes.

También se incorporó monitorización mediante Prometheus y Grafana, permitiendo visualizar el estado del clúster, los nodos, los Pods y el consumo de recursos. Esto aportó una visión más completa del sistema y permitió relacionar el despliegue técnico con la observabilidad del entorno.

Como resultado final, se ha conseguido una infraestructura funcional capaz de desplegar aplicaciones reales, mantener persistencia de datos, realizar backups, restaurar recursos eliminados, monitorizar el sistema y validar escenarios básicos de alta disponibilidad. Todo ello se ha documentado de forma progresiva, siguiendo una metodología por fases y dejando evidencias técnicas de cada parte del proceso.

**3. Lo que ha faltado por hacer**

Aunque el proyecto ha alcanzado los objetivos principales, algunas partes no se han podido desarrollar con la misma profundidad debido a limitaciones de tiempo, recursos físicos y complejidad técnica. El entorno se ha ejecutado sobre un laboratorio local con máquinas virtuales, lo que ha condicionado la capacidad disponible de CPU, memoria RAM y almacenamiento.

Una de las partes que quedó limitada fue la seguridad avanzada del clúster. Aunque se han aplicado buenas prácticas básicas y se ha trabajado en un entorno controlado, no se llegó a implementar de forma completa una configuración avanzada de RBAC, NetworkPolicies, gestión de usuarios, certificados TLS o políticas de acceso detalladas.

Estos elementos habrían permitido reforzar todavía más el proyecto desde el punto de vista de la seguridad.

También quedó pendiente una automatización más completa del despliegue. Aunque se utilizaron manifiestos YAML, comandos y configuraciones documentadas, no se llegó a integrar una solución completa de Infrastructure as Code mediante herramientas como Ansible o Terraform. Esto habría permitido repetir el despliegue del laboratorio de forma más automática y estructurada.

Otra parte que no se desarrolló completamente fue la integración de un pipeline CI/CD. El flujo Docker → Docker Hub → Kubernetes se realizó correctamente, pero de forma manual. Como mejora, habría sido interesante automatizar la construcción de la imagen, su publicación en Docker Hub y su despliegue posterior en Kubernetes mediante GitHub Actions, GitLab CI u otra herramienta similar.

También se podría haber profundizado más en las pruebas de carga y rendimiento. Aunque se validó la escalabilidad horizontal y la monitorización, no se realizaron pruebas intensivas de estrés para medir el comportamiento del clúster ante un aumento elevado de tráfico o consumo de recursos.

Por último, algunas funcionalidades avanzadas de alta disponibilidad quedaron planteadas como ampliación. El entorno actual permite validar alta disponibilidad a nivel de Pods y Deployments, pero no se llegó a implementar una alta disponibilidad completa del plano de control con varios nodos master, debido a la mayor complejidad técnica y a las limitaciones del laboratorio.

A pesar de ello, estas limitaciones no afectan al resultado principal del proyecto, ya que las funcionalidades esenciales fueron implementadas y validadas correctamente: clúster Kubernetes, despliegue de aplicaciones, persistencia, backups, restauración y monitorización.

**4. Mejoras y líneas futuras**

Como mejora principal, el proyecto podría evolucionar hacia una infraestructura más automatizada y reproducible. Para ello, se podría incorporar Ansible para automatizar la instalación de paquetes, configuración de red, despliegue de servicios y preparación de nodos. También sería posible utilizar Terraform para definir parte de la infraestructura de laboratorio, especialmente si en el futuro se migrara a un entorno cloud.

Otra mejora importante sería implementar un pipeline CI/CD completo. De esta forma, cada cambio realizado en la aplicación web podría activar automáticamente la construcción de una nueva imagen Docker, su publicación en Docker Hub y su despliegue en Kubernetes. Esto permitiría acercar el proyecto a un flujo DevOps más profesional y continuo.

En cuanto a seguridad, sería recomendable ampliar la configuración de RBAC, definir usuarios con permisos diferenciados, aplicar NetworkPolicies entre namespaces y proteger los accesos mediante certificados TLS. Estas mejoras permitirían reforzar el control de acceso y limitar la comunicación entre componentes del clúster.

También se podría mejorar la monitorización mediante dashboards personalizados en Grafana, alertas automáticas y reglas de Prometheus para detectar caídas de nodos, errores en Pods, consumo excesivo de CPU o memoria, fallos en backups o indisponibilidad de servicios. Esto aportaría una gestión más proactiva del entorno.

A nivel de backups, se podría ampliar el sistema de recuperación incorporando políticas de retención más avanzadas, pruebas periódicas de restauración y separación de copias según namespaces o criticidad de las aplicaciones. También sería interesante probar otros backends de almacenamiento compatibles con S3 o sistemas CSI con soporte de snapshots.

Finalmente, el proyecto podría evolucionar hacia una arquitectura Kubernetes más cercana a producción, incorporando varios nodos master, balanceador de acceso, Ingress Controller, certificados reales, almacenamiento más robusto y despliegues mediante Helm Charts.

**5. Conclusión final**

El desarrollo de KubeBackup360 ha permitido cumplir de forma satisfactoria los objetivos principales del proyecto. Se ha diseñado, implementado y validado una infraestructura Kubernetes funcional dentro de un entorno virtualizado, integrando tecnologías actuales del ámbito DevOps y cloud-native.

El resultado obtenido demuestra que es posible construir un laboratorio completo capaz de desplegar aplicaciones containerizadas, gestionar almacenamiento persistente, automatizar copias de seguridad, restaurar servicios ante fallos y monitorizar el estado del sistema. Además, el proyecto ha permitido aplicar de forma práctica conocimientos relacionados con redes, sistemas Linux, virtualización, Docker, Kubernetes, NFS, Velero, MinIO, Prometheus y Grafana.

Desde el punto de vista de la gestión, el proyecto se ha desarrollado de forma progresiva, dividiendo el trabajo en fases y validando cada componente antes de pasar al siguiente. Esta metodología ha permitido resolver problemas técnicos de forma ordenada, documentar evidencias y mantener una estructura clara en la memoria técnica.

Desde el punto de vista práctico, el proyecto ha sido positivo porque no se ha limitado a una explicación teórica, sino que ha demostrado el funcionamiento real de una infraestructura distribuida. Se han realizado despliegues, pruebas de fallo, restauraciones, escalabilidad, acceso externo y monitorización, obteniendo resultados funcionales y verificables.

En conclusión, KubeBackup360 constituye una base sólida para comprender cómo se construyen y administran infraestructuras Kubernetes modernas. Aunque existen mejoras posibles, el resultado alcanzado es coherente con el alcance académico del proyecto y demuestra una ejecución técnica positiva, completa y orientada a escenarios reales de administración de sistemas.