Implementación de un Clúster Kubernetes con Alta Disponibilidad de DNS

CFGS Administració de Sistemes Informàtics i Xarxes

Autor: Camilo Colorado Escobar
Grupo: ASIX2A

Camilo Colorado Escobar
ASIX 2 A

Resumen del proyecto

Temática: Este proyecto se centra en la implementación de una infraestructura de alta disponibilidad mediante Kubernetes, junto con un sistema de resolución DNS llamado CoreDNS, orientado a distribuir las consultas y garantizar la continuidad del servicio incluso en escenarios de fallo.

Objetivo: Desplegar un clúster Kubernetes utilizando RKE2 con alta disponibilidad en los nodos workers, e integrar un servicio DNS basado en CoreDNS configurado específicamente para responder desde los nodos de trabajo.

Metodología: Se ha seguido una metodología práctica y progresiva, desplegando un entorno con cuatro máquinas virtuales un nodo servidor y tres workers.
CoreDNS se ha desplegado únicamente en los workers. La máquina cliente ha sido configurada para usar los workers como servidores DNS. Se ha simulado la caída de nodos para comprobar el comportamiento y la resistencia del sistema.

Conclusiones: La infraestructura permite la resolución de nombres a través de los workers, mejorando la disponibilidad y reduciendo puntos únicos de fallo. Los servicios DNS han demostrado ser tolerantes a fallos.

Palabras clave: Kubernetes, alta disponibilidad, RKE2, CoreDNS, balanceo de carga, resiliencia, Helm, kubectl, DaemonSet, ConfigMap.

Abstract

Topic: This project focuses on the implementation of a high-availability infrastructure using Kubernetes, along with a DNS resolution system called CoreDNS, aimed at distributing queries and ensuring service continuity even in failure scenarios.

Objective: To deploy a Kubernetes cluster using RKE2 with high availability on the worker nodes, and integrate a DNS service based on CoreDNS configured specifically to respond from the worker nodes.

Methodology: A practical and progressive methodology was followed, deploying an environment with four virtual machines—one server node and three workers.

CoreDNS was deployed only on the worker nodes. The client machine was configured to use the workers as DNS servers. Node failures were simulated to test the system's behavior and resilience.

Conclusions: The infrastructure enables name resolution through the workers, improving availability and reducing single points of failure. The DNS services have proven to be fault-tolerant.

Keywords: Kubernetes, high availability, RKE2, CoreDNS, load balancing, resilience, Helm, kubectl, DaemonSet, ConfigMap.

Índex

Introducción

1.1. Contexto

Las infraestructuras deben ser capaces de mantenerse activas incluso cuando algunos componentes fallan.

En este proyecto, se ha trabajado en montar un clúster Kubernetes que permita distribuir la carga y evitar interrupciones usando varias máquinas virtuales. Para que las aplicaciones se comuniquen y encuentren los servicios correctamente dentro del clúster, es clave tener un sistema DNS dinámico que actualice la información automáticamente. Por eso, se ha utilizado CoreDNS configurado en los nodos workers para que puedan resolver las consultas DNS y mantener la alta disponibilidad.

1.2. Justificación

En este proyecto es fundamental contar con un entorno que sea estable y que siga funcionando aunque alguno de los nodos falle. Para ello, se ha elegido RKE2 porque es una versión más segura y fácil de manejar de Kubernetes. Además, Rancher ayuda a gestionar el clúster de forma sencilla.

El uso de CoreDNS como DNS interno es clave para que los servicios dentro del clúster puedan comunicarse sin problemas y para garantizar que las consultas DNS se resuelvan rápidamente y de forma dinámica, mejorando así la disponibilidad del sistema.

1.3. Objetivos

- **General:** Montar un clúster de Kubernetes que sea estable, escalable y resistente a errores.
- **Específicos:**

- Crear un nodo de control y tres nodos worker con RKE2.
- Gestionar el clúster con Rancher.
- Configurar CoreDNS para que el DNS interno funcione correctamente.
- Verificar que el sistema puede redistribuir carga si un nodo falla.

1.4. Estrategia y planificación

- Preparar las máquinas virtuales.
- Instalar RKE2 en el nodo master y los workers.
- Configurar Rancher para controlar el clúster.
- Validar el funcionamiento de CoreDNS.
- Realizar pruebas de caída y verificar que todo sigue funcionando.

1.5. Metodología de trabajo

El entorno de trabajo se preparó usando VirtualBox en un equipo personal, con máquinas virtuales basadas en Ubuntu Desktop y Server. Para continuar el proyecto fuera de casa, se utilizó RustDesk para acceder remotamente a las máquinas.

El proceso de instalación comenzó con RKE2, seguido de la instalación de kubectl. Luego, Rancher se desplegó usando Helm, y finalmente CoreDNS se instaló a través de Rancher.

Los nodos workers fueron etiquetados con la etiqueta worker=true para identificarlos. Para la gestión y configuración del clúster, inicialmente se usaron kubectl y Helm para preinstalar componentes, pero el resto de la configuración se realizó directamente desde la interfaz de Rancher.

Para probar la alta disponibilidad, se apagaban nodos workers y se comprobaba que las consultas DNS continuaban resolviéndose correctamente. En el ConfigMap de CoreDNS se habilitó una sección para generar logs, los cuales se revisaban desde la terminal para verificar las peticiones del cliente.

El cliente se configuró para usar las tres IP de los nodos workers como servidores DNS, asegurando que siempre pudiera resolver las consultas mediante cualquiera de ellos.

Toda la configuración se gestionó principalmente desde Rancher, donde se accedía a los archivos YAML y a los ajustes del clúster para supervisar y modificar la instalación.

1.6. Presupuesto

El proyecto se ha realizado utilizando dos ordenadores físicos (el de clase y el de casa), sin recurrir a plataformas de pago ni servidores en la nube, por lo que no ha supuesto ningún coste económico directo.

Recursos utilizados:

- 2 ordenadores físicos con 8 GB de RAM y 2 CPUs cada uno.
- 5 máquinas virtuales creadas en total:
    - 1 para el servidor RKE2
    - 3 como nodos worker
    - 1 como cliente para pruebas
- Conexión de red entre las máquinas, tanto local como a Internet.
- Tiempo estimado de trabajo en instalación, configuración, pruebas y documentación: **X** horas.

2. Descripción del proyecto

2.1. Análisis de requisitos

- Mínimo dos nodos con 4 GB de RAM y 2 CPUs.
- Red interna entre las máquinas.
- Acceso SSH y permisos de usuario root.
- Ubuntu Server 24.04 .

2.2. Tecnologías

- **RKE2:** Proporciona una instalación sencilla y segura de Kubernetes.
- **Rancher:** Permite gestionar el clúster desde una interfaz web intuitiva.
- **CoreDNS:** Actúa como servidor DNS interno para los servicios dentro del clúster.

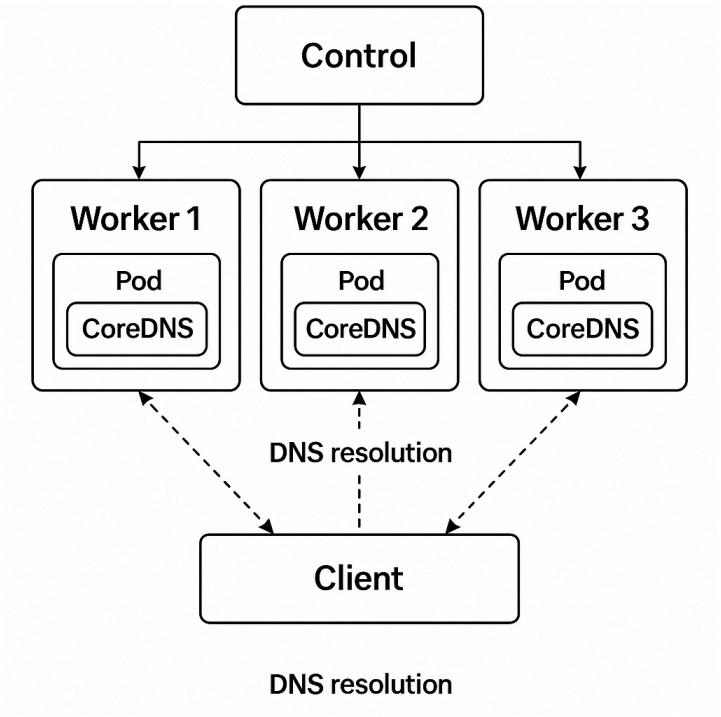
2.3. Estructura del clúster

El clúster está formado por un nodo master con RKE2 y tres nodos worker. Rancher se instala en el nodo master para gestionar todo el clúster desde una interfaz web.

CoreDNS se despliega sólo en los nodos worker mediante un DaemonSet (pod que se ejecuta automáticamente en todos los nodos del clúster), y es el encargado de resolver los nombres de los servicios dentro del clúster.

Además, se ha añadido una máquina cliente externa al clúster, que usa las IPs de los workers como servidores DNS para hacer pruebas de resolución.

Toda la red se ha configurado en modo adaptador interno para que las máquinas puedan comunicarse entre sí sin conexión a internet.



2.4. Funcionalidades

- **Balanceo de carga:** Kubernetes distribuye automáticamente las peticiones entre los pods disponibles, evitando que todo el tráfico se concentre en un solo pod y mejorando el rendimiento.

- **Alta disponibilidad:** Si un pod falla, Kubernetes lo reinicia automáticamente. Si un nodo worker se cae, los servicios se redistribuyen a otros nodos para que el sistema siga funcionando sin interrupciones.

- **Resolución DNS interna:** CoreDNS permite que los servicios dentro del clúster se comuniquen entre ellos utilizando nombres, sin necesidad de configuraciones manuales. Esto simplifica la conexión entre aplicaciones.

2.5. Definición de las funcionalidades

- **Balanceo de carga:**

El clúster de Kubernetes ejecuta el servicio DNS CoreDNS como un DaemonSet llamado coredns-worker, que se despliega en todos los nodos worker. Cada pod de CoreDNS escucha directamente en el puerto 53 del nodo usando hostPort (es decir, expone un puerto del contenedor directamente en el nodo para que se pueda acceder desde fuera del clúster). El cliente está configurado con las IPs de los tres workers en su fichero resolv.conf, en orden de prioridad (worker 1, worker 2, worker 3). El sistema de resolución DNS sigue ese orden, por lo que si uno falla, pasa al siguiente. La tolerancia a fallos se consigue gracias a esta configuración ordenada.

- **Redundancia de pods:**

Si un pod falla o se elimina, Kubernetes detecta esto y crea automáticamente otro pod nuevo para que el servicio siga funcionando sin interrupciones. Gracias a la alta disponibilidad, estos pods pueden regenerarse en cualquiera de los tres nodos worker disponibles.

- **Redundancia de nodos:**

El entorno tiene tres nodos worker conectados a un servidor central. Si uno de los nodos falla, Kubernetes redistribuye automáticamente los pods a los otros nodos activos, asegurando que el servicio no se interrumpa. Esta funcionalidad estará completamente implementada y probada.

- **Resolución DNS interna con CoreDNS:**

CoreDNS está desplegado únicamente en los nodos worker. Gracias al DaemonSet, cada worker tiene su propio pod DNS que responde directamente a las consultas. El cliente ha sido configurado para usar esos nodos como servidores DNS. Además, se ha modificado el ConfigMap (archivo de configuración que Kubernetes usa para ajustar servicios como CoreDNS) para activar los logs y poder verificar desde rancher o desde la terminal, qué worker ha resuelto cada consulta.

---

3. Paso a paso

La idea de este proyecto surgió durante las prácticas, donde pude ver cómo funcionaba un entorno real con Kubernetes y Rancher. Me llamó la atención cómo se podía gestionar todo de forma centralizada y cómo se evitaban las caídas de servicio. Esto me inspiró a montar mi propio entorno con alta disponibilidad y DNS interno. A continuación, explico las herramientas que he utilizado y cómo ha sido el proceso paso a paso, incluyendo algunos problemas que han ido surgiendo y cómo los he solucionado.

3.1. Herramientas utilizadas:

VirtualBox	Programa para crear y gestionar máquinas virtuales, usado para montar el entorno de pruebas.
RKE2	Instalador seguro y simple de Kubernetes hecho por Rancher.
Rancher	Plataforma web para gestionar clústeres Kubernetes fácilmente.
Kubectl	Comando para controlar y administrar Kubernetes desde la terminal.
Helm	Gestor de paquetes para instalar apps en Kubernetes con pocos comandos.
CoreDNS	Sistema de nombres (DNS) que permite que los servicios se encuentren por nombre dentro del clúster.

3.2. Preparación

Con el programa VirtualBox cree 4 Máquinas Virtuales:

- **1 nodo servidor**
- **Sistema operativo:** Ubuntu Desktop 24.04.2 LTS
- **Recursos asignados:** 8 GB de RAM, 100 GB de disco
- **Redes:**
    - Adaptador de red interna (IP: 192.168.10.10)
    - Adaptador en modo puente (para conexión con el exterior)
- **3 nodos workers**

- **Sistema operativo:** Ubuntu Desktop 24.04.2 LTS
- **Recursos asignados:** 8 GB de RAM, 100 GB de disco cada uno
- **Redes:**
- **Worker 1:** 192.168.10.20
- **Worker 2:** 192.168.10.30
- **Worker 3:** 192.168.10.40
- Todos con adaptador de red interna y adaptador en modo puente

Estas capacidades me permiten gestionar el proyecto sin problemas de rendimiento.

3.3. Dificultades para la preparación:

Durante la preparación encontré varias dificultades.

Los portátiles del aula no disponían de suficiente memoria RAM (solo 4 GB), lo cual hacía inviable ejecutar un clúster de Kubernetes con varios nodos.

Intenté utilizar IsarVDI como alternativa, ya que permite ampliar la memoria RAM, pero no ofrece suficiente almacenamiento para desplegar correctamente el entorno.

Finalmente, opté por usar mi ordenador personal, que sí dispone de los recursos necesarios, tanto a nivel de RAM como de capacidad de disco. Esto me permitió instalar y configurar todos los servicios sin problemas de rendimiento.

4. Instalación Nodo Servidor

4.1. Instalación y activación de RKE2:

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

Este comando descarga e instala automáticamente la última versión de RKE2.

```
systemctl enable rke2-server.service
```

Con este, el servicio de RKE2 se configura para iniciarse automáticamente cada vez que se encienda la máquina.

```
systemctl start rke2-server.service
```

Este comando arranca el servidor de Kubernetes en el nodo principal, dejando el clúster en funcionamiento.

Después de instalar RKE2, necesitaba instalar y configurar kubectl, que es la herramienta de línea de comandos para gestionar Kubernetes desde la terminal de comandos.

4.2. Instalación y configuración de kubectl:

```
curl -LO "https://dl.k8s.io/release/$(curl -sL
https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"
```

Este comando descarga automáticamente la última versión estable de kubectl para Linux.

```
chmod +x ./kubectl
```

Se le dan permisos de ejecución al archivo descargado.

```
mv ./kubectl /usr/local/bin/kubectl
```

Se mueve el binario a /usr/local/bin para poder usar kubectl desde cualquier parte del sistema sin tener que especificar la ruta.

4.3. Configurar acceso al clúster:

```
export KUBECONFIG=/etc/rancher/rke2/rke2.yaml

echo "export KUBECONFIG=/etc/rancher/rke2/rke2.yaml" >> ~/.bashrc

source ~/.bashrc
```

Se configura la variable KUBECONFIG para que kubectl sepa cómo conectarse al clúster. Además, se guarda en el archivo .bashrc para que quede permanente entre sesiones.

4.4. Comprobar conexión con el clúster:

```
kubectl get nodes
```

Este comando muestra el nodo servidor como Ready, lo que confirma que kubectl está correctamente conectado y funcionando.

Después de tener el clúster listo, instalé Rancher, la interfaz web de administración de Kubernetes, usando Helm, que es el gestor de paquetes para Kubernetes.

4.5. Instalación inicial de Rancher con Helm:

```
snap install helm --classic
```

Se instaló Helm con snap parecido a (apt). Esta herramienta permite instalar aplicaciones en Kubernetes de forma sencilla mediante "charts".

```
helm repo add rancher-latest https://releases.rancher.com/server-charts/latest
```

Se añadió el repositorio oficial de Rancher a Helm para poder descargar sus paquetes.

```
helm repo update
```

Este comando actualiza todos los repositorios de Helm para asegurarse de que se tenga la última versión disponible de los charts.

```
kubectl create namespace cattle-system
```

Creó un namespace llamado cattle-system, que es donde se alojarán todos los recursos de Rancher dentro del clúster.

5. Configuración de acceso a Rancher

Para poder acceder a la interfaz web de Rancher usando el nombre rancher.local, editamos el archivo /etc/hosts en el nodo servidor. Añadimos las direcciones IP de todos los nodos del clúster y asignamos el nombre **rancher.local** a la IP del nodo servidor (192.168.10.10). Esto permite abrir Rancher desde un navegador escribiendo:

```
https://rancher.local
```

De esta forma, el sistema reconoce el nombre sin necesidad de configurar un DNS externo.

5.1. Instalación de Rancher:

```
helm install rancher rancher-latest/rancher \
  --namespace cattle-system \
  --set hostname=rancher.local \
```

```
--set replicas=2 \
--set bootstrapPassword=usuario \
--set ingress.tls.source=auto
```

helm install rancher rancher-latest/rancher:

Este es el comando básico para instalar Rancher usando Helm.

rancher es el nombre que le damos a esta instalación.

rancher-latest/rancher indica el paquete (chart) oficial de Rancher que vamos a instalar desde el repositorio añadido previamente.

--namespace cattle-system:

Indica que Rancher se instalará dentro del namespace cattle-system. Esto sirve para organizar y aislar los recursos de Rancher dentro del clúster.

--set hostname=rancher.local:

Define el nombre de host que Rancher usará para ser accesible. En este caso, rancher.local es el dominio local configurado para acceder a Rancher desde el navegador.

--set replicas=2:

Establece que se creen dos réplicas del pod de Rancher. Esto ayuda a mejorar la disponibilidad, porque si uno falla, el otro sigue funcionando.

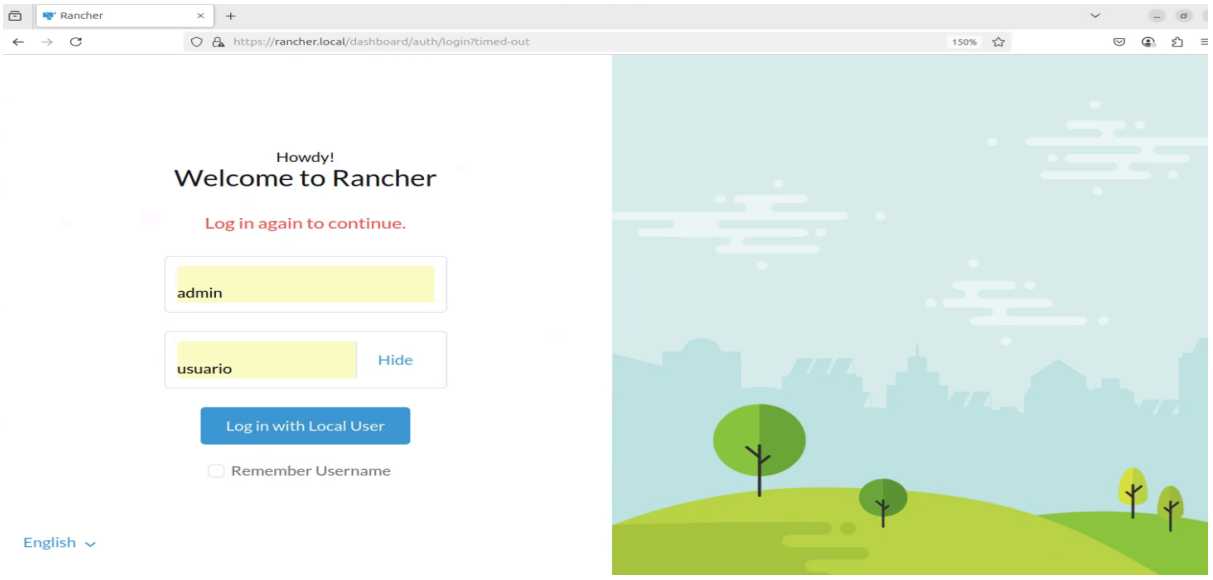
--set bootstrapPassword=usuario:

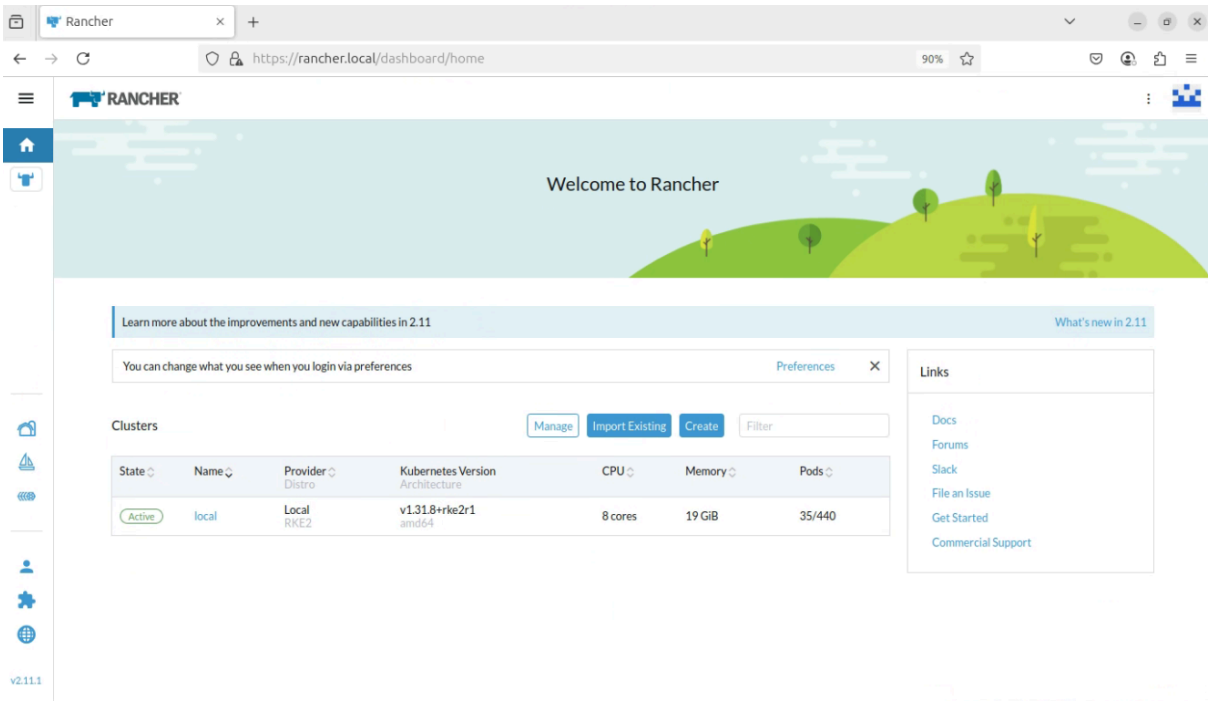
Define la contraseña inicial para el acceso a Rancher después de la instalación.

--set ingress.tls.source=auto:

Configura el sistema para que gestione automáticamente los certificados TLS (HTTPS) para el acceso seguro a Rancher, usando certificados generados automáticamente (con Let's Encrypt o certificados auto-firmados).

5.2. INTERFAZ WEB:





6. Instalacion Nodos Workers

6.1. Descargar e instalar el agente RKE2

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

Este comando descarga e instala el agente de RKE2, que es el componente necesario para que el nodo funcione como worker.

6.2. Crear el archivo de configuración del agente:

sudo mkdir -p /etc/rancher/rke2/

sudo vim /etc/rancher/rke2/config.yaml

Añadiremos esta configuración:

server: **https://192.168.10.10:9345**
token:
104faca013b69f9e002bcba7316c018ad161a63fe599de7ff58574cafa3a1efe60::server:696d557708cc35fd4641f2a2b49781ed
node-name: **worker1** # Aquí se indica el nombre del nodo
node-ip: **192.168.10.*** # Aquí se indica la IP correspondiente del nodo

6.3. Habilitar y arrancar el servicio del agente:

sudo systemctl enable rke2-agent.service

sudo systemctl start rke2-agent

Esto pone en marcha el servicio que conecta este nodo con el servidor principal y lo integra al clúster.

6.4. Configurar el archivo /etc/hosts

Para Worker1:

127.0.0.1 localhost
127.0.1.1 worker1
192.168.10.10 rancher.local
192.168.10.20 worker1
192.168.10.30 worker2
192.168.10.40 worker3

Para Worker2:

127.0.0.1 localhost
127.0.1.1 worker2
192.168.10.10 rancher.local
192.168.10.20 worker1

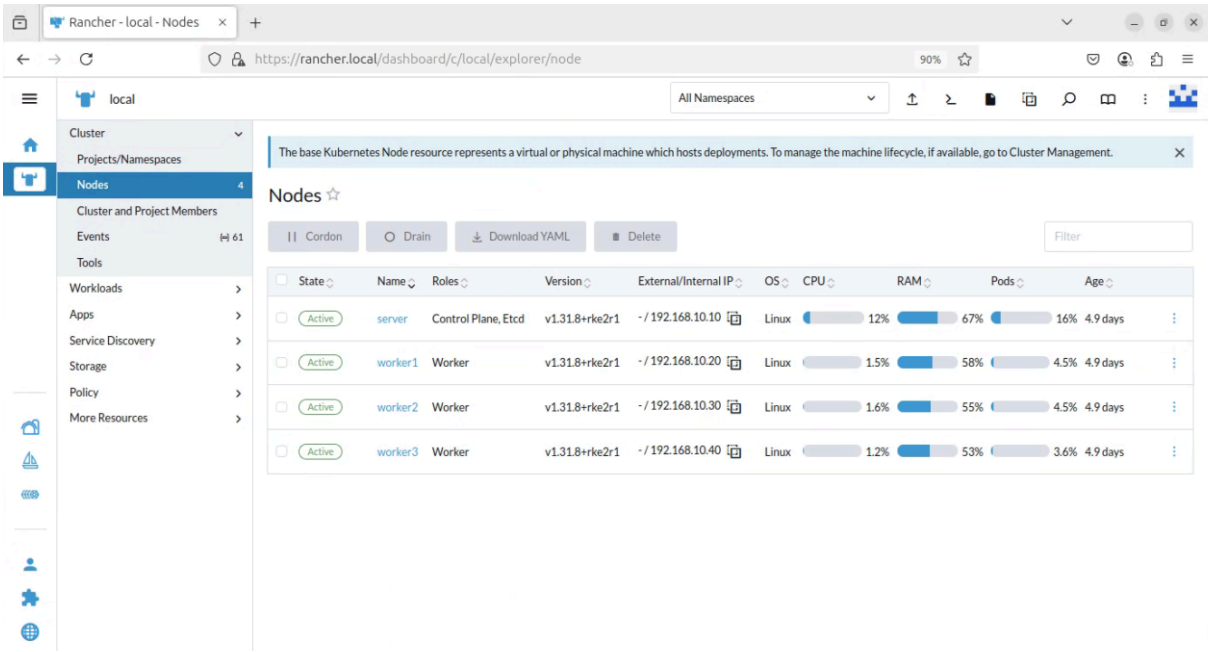
192.168.10.30 worker2
192.168.10.40 worker3

Para Worker3:

127.0.0.1 localhost
127.0.1.1 worker3
192.168.10.10 rancher.local
192.168.10.20 worker1
192.168.10.30 worker2
192.168.10.40 worker3

7. Verificación de RKE2 y Rancher:

```
root@server:/home/usuario# kubectl get nodes
NAME      STATUS   ROLES                       AGE     VERSION
server    Ready    control-plane,etcd,master   4d23h   v1.31.8+rke2r1
worker1   Ready    worker                      4d22h   v1.31.8+rke2r1
worker2   Ready    worker                      4d21h   v1.31.8+rke2r1
worker3   Ready    worker                      4d21h   v1.31.8+rke2r1
root@server:/home/usuario#
```

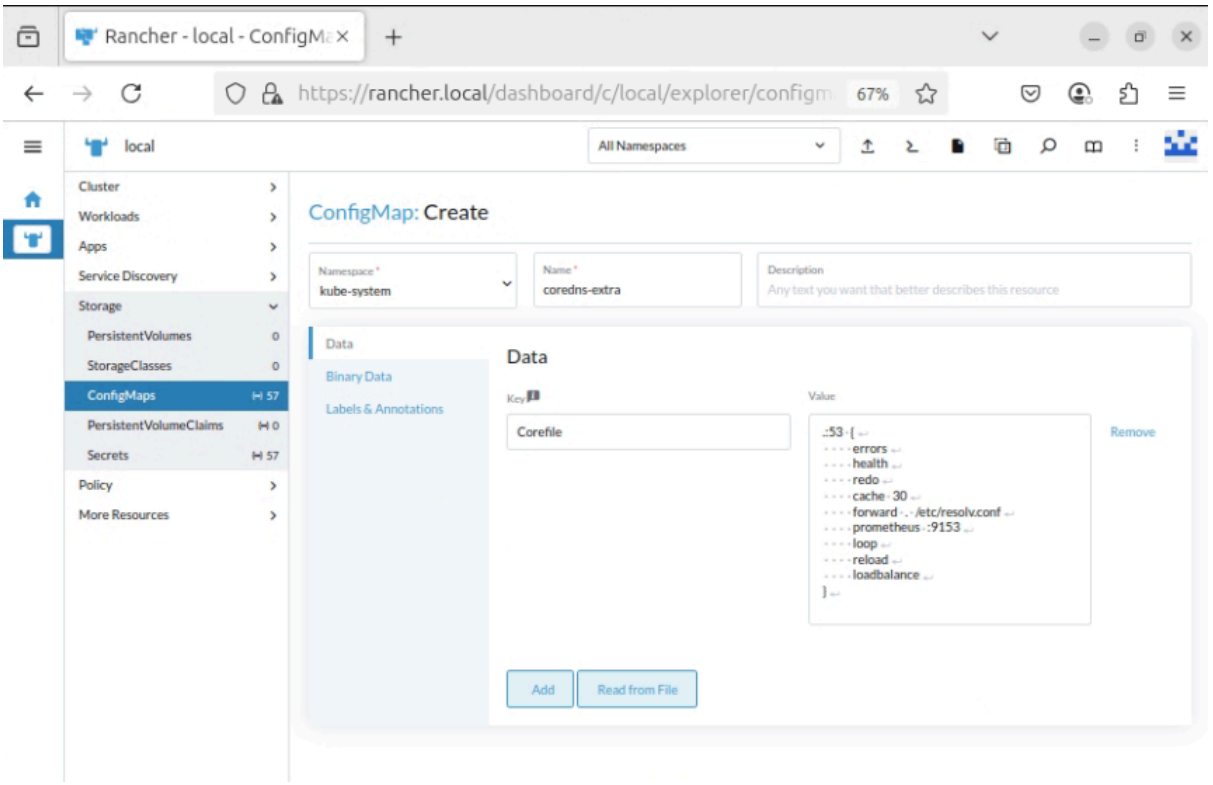


8. Instalación COREDNS-WORKER

8.1. ConfigMaps:

La pestaña ConfigMaps en Rancher permite gestionar configuraciones de recursos en Kubernetes sin tener que modificar directamente el código de los pods. En nuestro caso, se utiliza para definir el archivo Corefile, que es la configuración principal de CoreDNS, el servicio DNS interno del clúster.

Este Corefile indica cómo debe comportarse CoreDNS: cómo resolver nombres, a dónde enviar las consultas externas, cómo gestionar errores y cómo monitorizar el servicio. Gracias a los ConfigMaps, se puede editar esta configuración fácilmente desde la interfaz de Rancher y aplicar los cambios sin tener que recrear los pods manualmente.



log y errors: registran las consultas DNS y los errores para poder hacer diagnóstico.

health: permite que Kubernetes compruebe si CoreDNS está activo.

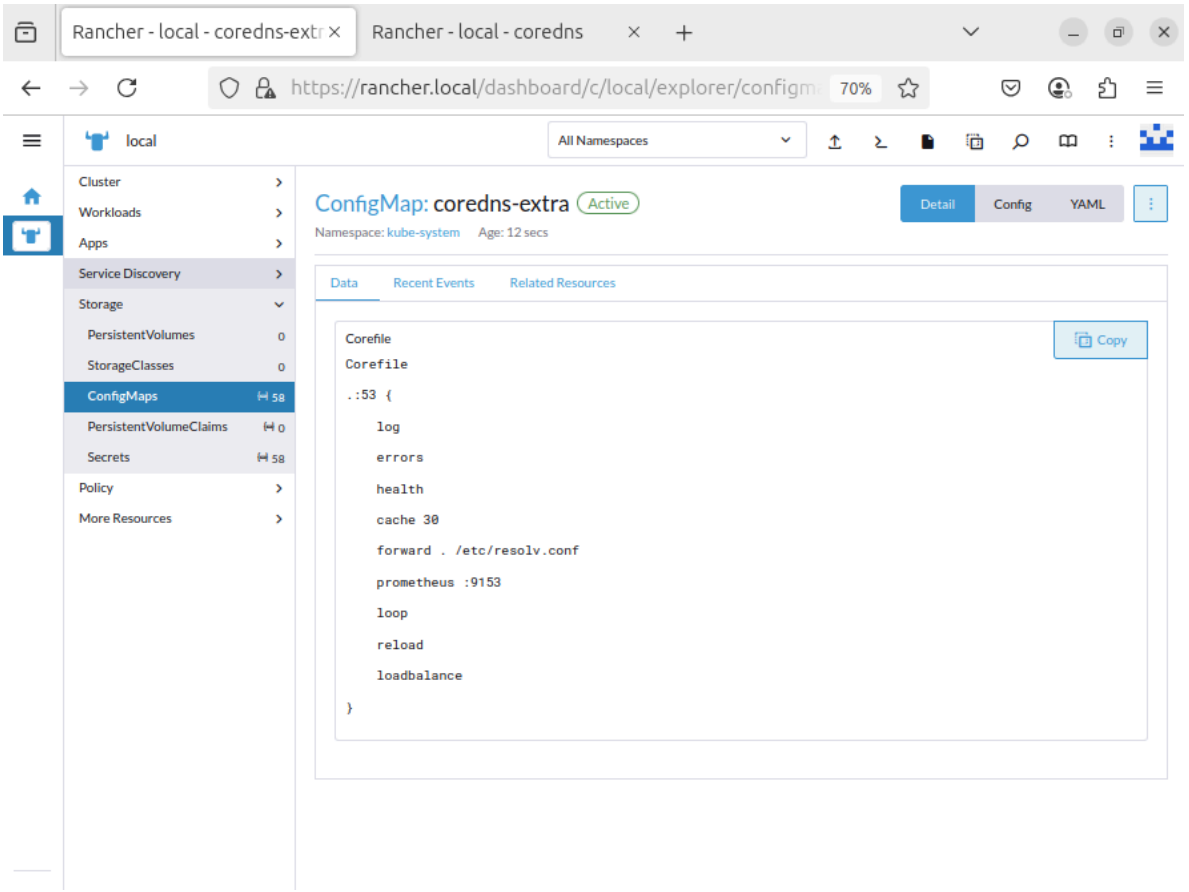
cache 30: guarda en memoria las respuestas DNS durante 30 segundos para mejorar el rendimiento.

forward . /etc/resolv.conf: envía las consultas DNS externas a los servidores definidos en /etc/resolv.conf del nodo (normalmente, los DNS del proveedor o del sistema host).

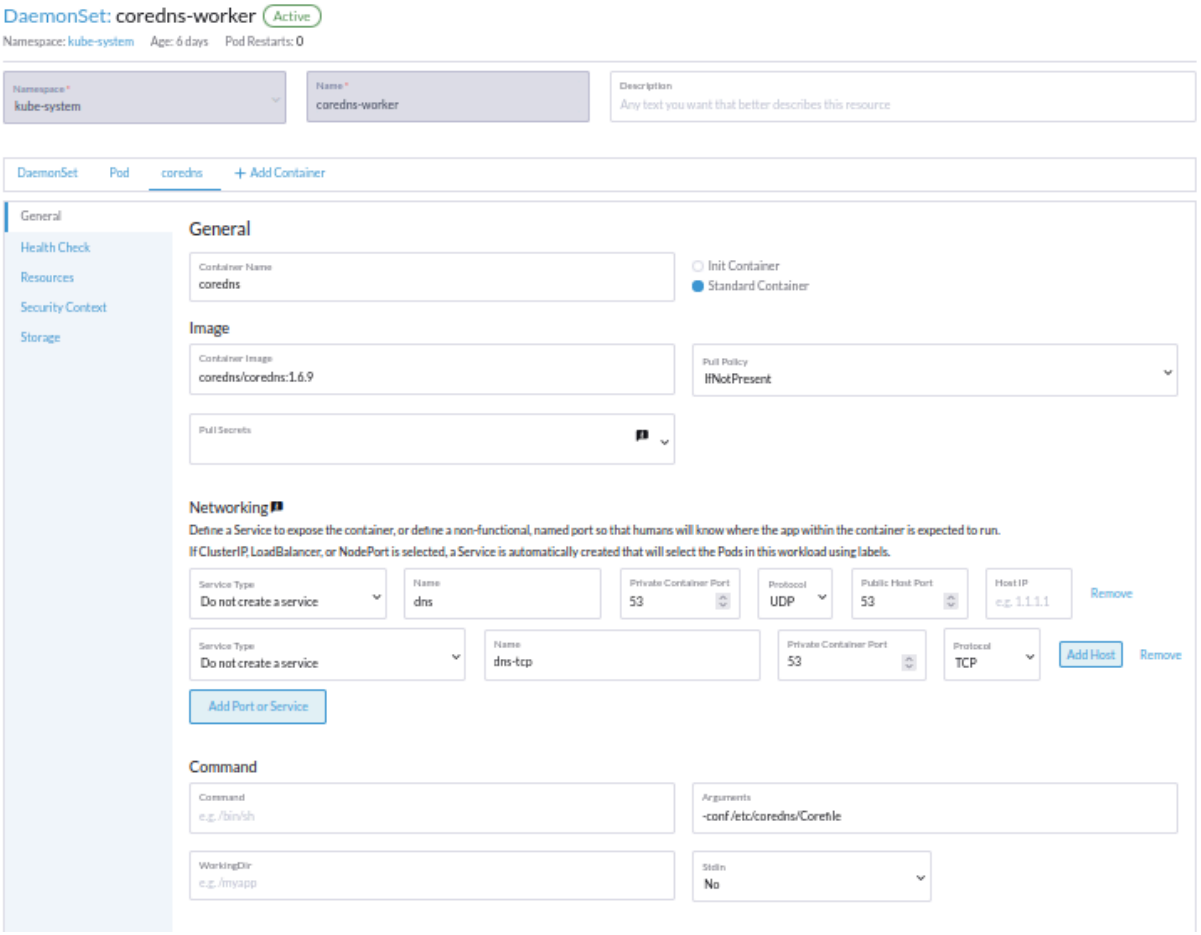
prometheus :9153: expone estadísticas para hacer monitorización con Prometheus.

loop, reload y loadbalance: mejoran la estabilidad y la eficiencia del servicio evitando bucles, permitiendo recargar la configuración automáticamente y equilibrando la carga de las consultas.

Esta configuración asegura que cada instancia de CoreDNS en los nodos worker pueda resolver nombres tanto dentro como fuera del clúster.



8.2. DaemonSets:



En la pestaña DaemonSets, se creó un nuevo recurso llamado coredns-worker, asociado al namespace kube-system

Se seleccionaron los nodos con el rol worker para que CoreDNS solo se desplegará en ellos.

Se usó la imagen coredns/coredns:1.6.9.

Se configuró el contenedor para que usara el fichero Corefile como configuración principal.

Se expuso el puerto 53 UDP y TCP tanto en el contenedor como en el host (gracias a hostPort), lo que permite recibir consultas DNS externas.

Se montó un ConfigMap llamado coredns como volumen, para que el pod tenga acceso al fichero de configuración Corefile.

---

9. Conclusiones

9.1. Conclusiones generales del proyecto

Este proyecto ha sido todo un desafío, pero al final he conseguido montar un sistema de alta disponibilidad con Kubernetes, usando Rancher para gestionarlo de forma más sencilla y CoreDNS para que el DNS funcione dentro del clúster sin problemas.

No ha sido fácil configurar todo bien, entender cómo conectar los nodos, instalar RKE2, y ajustar CoreDNS para que actúe como un servidor DNS completo ha requerido mucho tiempo y pruebas. Sin esta memoria, creo que sería complicado para alguien hacerlo por su cuenta, porque hay muchos detalles y pasos que no son tan evidentes a simple vista.

Gracias a este trabajo, he aprendido mucho sobre cómo funcionan los sistemas distribuidos y cómo mantenerlos funcionando aunque alguna parte falle. También he visto lo importante que es automatizar la gestión y usar herramientas visuales como Rancher para no perderse entre tantos comandos.

Estoy muy contento porque he logrado que el sistema funcione correctamente y pueda demostrar que es posible implementar este tipo de soluciones prácticas y reales.

10. Cumplimiento de los objetivos

Objetivo	Estado	Comentario
Crear un clúster con RKE2	Hecho	Cuatro máquinas virtuales configuradas: 1 servidor y 3 nodos worker
Gestionar el clúster con Rancher	Hecho	Rancher instalado en el servidor, gestionando correctamente el clúster
Configurar CoreDNS	Hecho	DNS interno funcional en todos los nodos, permite comunicación por nombre
Redistribuir carga si falla un nodo	Hecho	Con tres nodos worker, se puede redistribuir el tráfico y regenerar pods

Implementar alta disponibilidad con 3 nodos worker	Hecho	Redistribución funcional y testada

11. Valoración de la metodología y planificación

Se ha seguido una metodología práctica y progresiva: instalar, probar, validar y documentar. Rancher ha facilitado el control visual del estado del clúster. Aunque algunas fases han requerido más tiempo del previsto.

12. Problemas encontrados y soluciones

Durante el desarrollo del proyecto surgieron diferentes problemas, tanto técnicos como de recursos, que se resolvieron con distintas estrategias:

Limitaciones del hardware en clase: Los ordenadores del centro no disponían de la memoria RAM suficiente para ejecutar varias máquinas virtuales con Kubernetes. La mayoría apenas alcanzaban los 4 GB de RAM, lo que hacía inviable levantar un clúster.

Solución: Se decidió realizar el montaje completo en mi PC personal, con mejores especificaciones, y acceder remotamente desde clase usando RustDesk.

Falta de espacio con IsarVDI: Como alternativa, probé utilizar escritorios virtuales del instituto (IsarVDI), que permiten más RAM, pero su capacidad de almacenamiento era insuficiente para las imágenes del clúster.

Solución: También descartado. El entorno final se preparó y probó exclusivamente desde mi equipo local.

Configuración de red entre nodos: Al principio, algunas máquinas no se comunicaban correctamente entre sí, lo que provocaba errores al unir los nodos al clúster.

Solución: Se revisaron las configuraciones de adaptadores de red en VirtualBox, asegurando que todos los nodos tuvieran un adaptador en red interna con el mismo nombre y se comunicaran correctamente.

Acceso a Rancher desde navegador: Para evitar tener que acceder a Rancher desde la interfaz web usando directamente la IP del nodo servidor, se configuró el nombre rancher.local como alias.

Solución: Se editó el archivo /etc/hosts en las máquinas desde donde se accedería al navegador, añadiendo la línea 192.168.10.10 rancher.local. Además, en el comando de instalación de Rancher con Helm, se especificó el parámetro --set hostname=rancher.local para que coincidiera con ese alias y se generara el certificado TLS correctamente.

Logs de CoreDNS: En un principio no se veían los logs de las peticiones DNS realizadas por el cliente.

Solución: Se añadió configuración específica en el ConfigMap de CoreDNS para habilitar el registro de logs, y se usó la terminal para consultarlos.

Alta disponibilidad DNS: Para simular la tolerancia a fallos, se apagaron nodos workers, pero no siempre se resolvían los nombres desde el cliente.

Solución: Se configuró el archivo /etc/resolv.conf del cliente para incluir las IPs de los tres workers en orden, permitiendo que, si un nodo no respondía, la consulta pasará al siguiente.

13. Visión de futuro

Para seguir mejorando el entorno, se plantea:

- Añadir nodos adicionales para probar elasticidad a mayor escala.
- Exponer servicios al exterior usando un ingress controller.
- Integrar herramientas como Prometheus y Grafana para monitorización en tiempo real.
- Automatizar los despliegues usando Helm Charts.

---

14. Glosario

- **Kubernetes**: Plataforma de orquestación de contenedores que automatiza el despliegue y la gestión de aplicaciones.

- **Pod**: Unidad básica de ejecución en Kubernetes, que contiene uno o más contenedores.
- **Nodo**: Máquina virtual o física que forma parte del clúster y ejecuta pods.
- **Clúster**: Conjunto de nodos (máquinas) que trabajan juntos para ejecutar aplicaciones y servicios de forma coordinada dentro de un entorno de Kubernetes.
- **RKE2**: Versión simplificada y segura de Kubernetes, desarrollada por Rancher.
- **Rancher**: Plataforma de gestión de clústeres Kubernetes con interfaz web.
- **CoreDNS**: Servidor DNS utilizado dentro de Kubernetes para la resolución de servicios internos.
- **Alta disponibilidad**: Capacidad de un sistema para seguir funcionando correctamente aunque se produzcan fallos.
- **kubectl**: Herramienta de línea de comandos para gestionar y operar clústeres de Kubernetes.
- **Helm**: Gestor de paquetes para Kubernetes que facilita el despliegue y la gestión de aplicaciones complejas mediante "charts".
- **Charts**: Conjuntos de archivos que definen, configuran y despliegan una aplicación en Kubernetes usando Helm.
- **Namespace**: Espacio lógico dentro de un clúster de Kubernetes que permite aislar recursos y organizar mejor los entornos y aplicaciones.
- **ConfigMaps**: Objetos de Kubernetes que permiten almacenar datos de configuración en formato clave-valor para ser utilizados por los pods.
- **DaemonSets**: Controlador de Kubernetes que garantiza que una copia específica de un pod se ejecute en todos o en algunos nodos del clúster.

---

15. Bibliografía

- Kubernetes Official Documentation - https://kubernetes.io (consultado en abril de 2025)
- RKE2 Documentation - https://docs.rke2.io (consultado en abril de 2025)

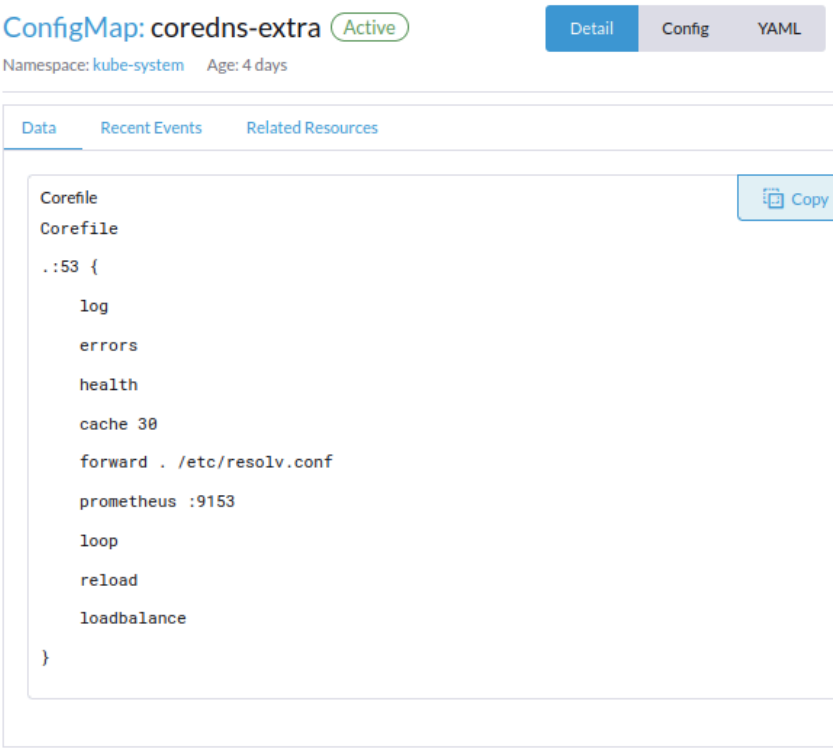
- Rancher Documentation - Enterprise Kubernetes Management Platform & Software | Rancher (consultado en abril de 2025)
- Ubuntu Server 24.04 LTS - Get Ubuntu Server | Download | Ubuntu (consultado en abril de 2025)
- Ubuntu Desktop 24.04.2 LTS Download Ubuntu Desktop | Ubuntu (consultado en abril de 2025)
- CoreDNS GitHub - https://github.com/coredns/coredns (consultado en abril de 2025)
- ChatGPT - https://chatgpt.com/ (consultado cada vez que fue necesario)

16. Anexos

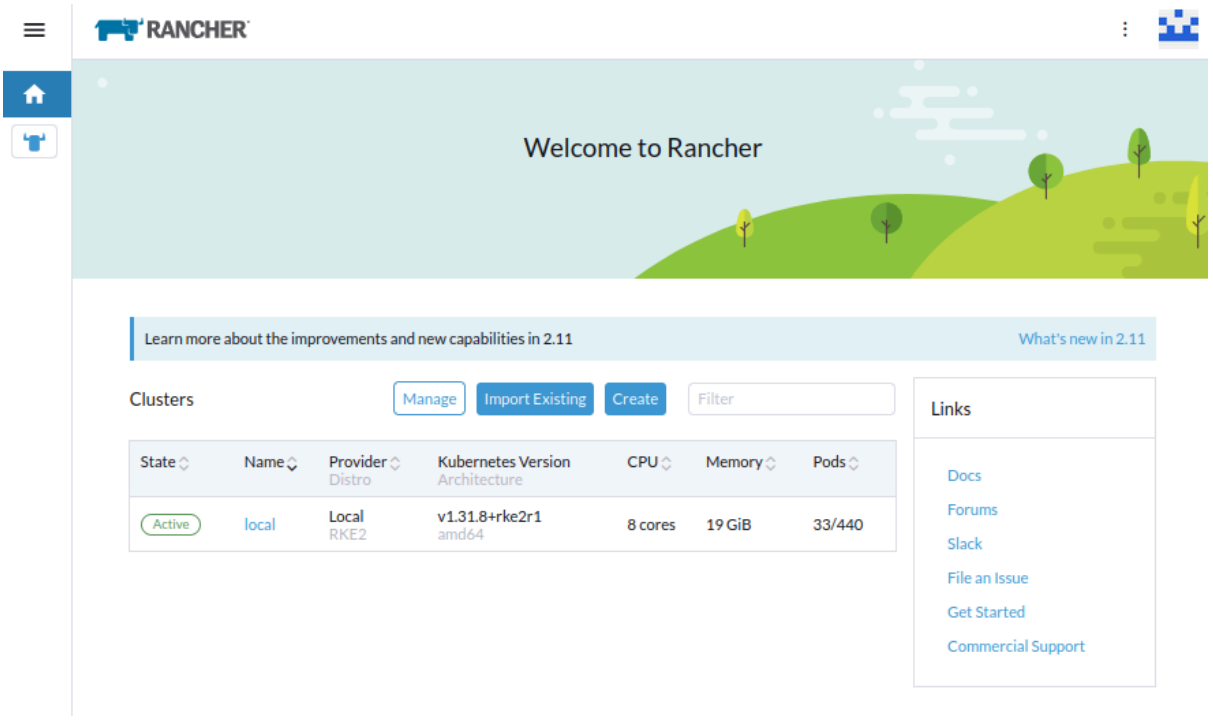
Anexo 1: Archivos YAML utilizados para desplegar servicios.

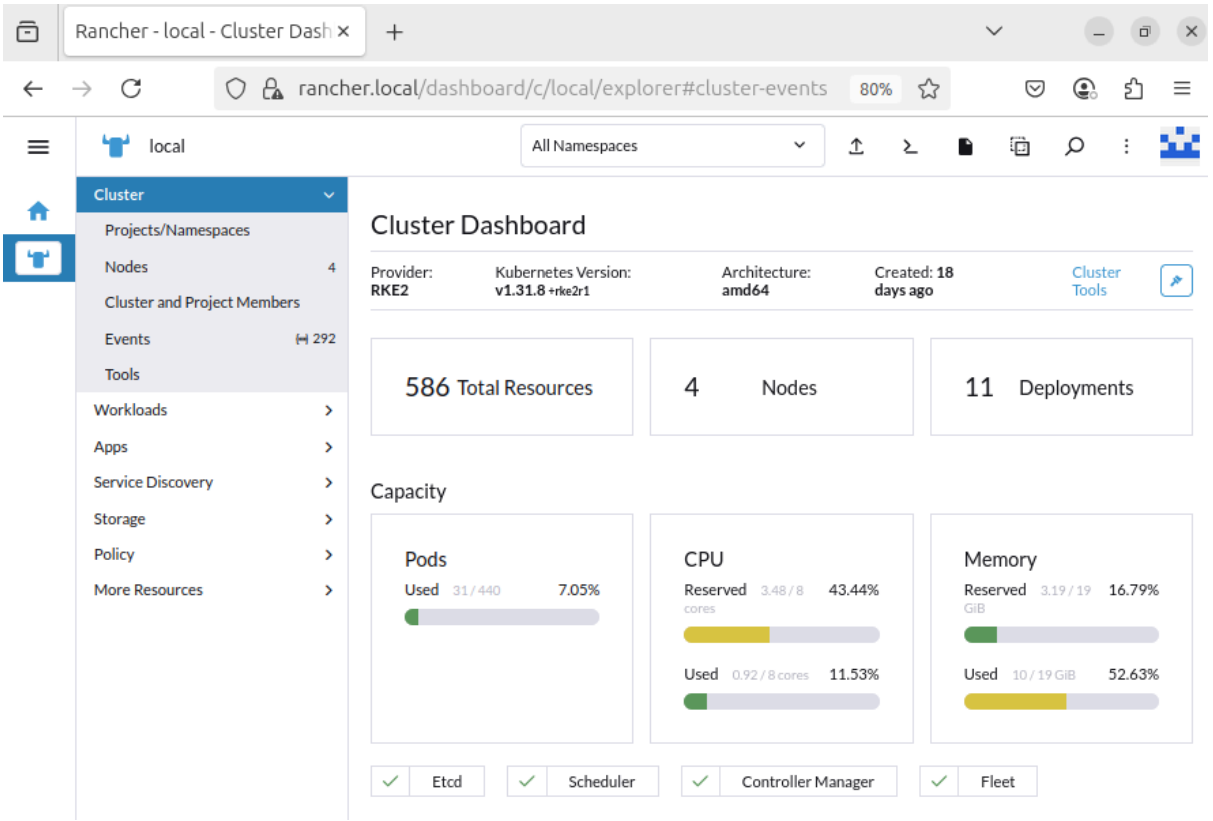
```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: coredns
  namespace: kube-system
data:
  Corefile: |
    .:53 {
        log
        errors
        health
        cache 30
        forward . /etc/resolv.conf
        prometheus :9153
        loop
        reload
        loadbalance
    }
```

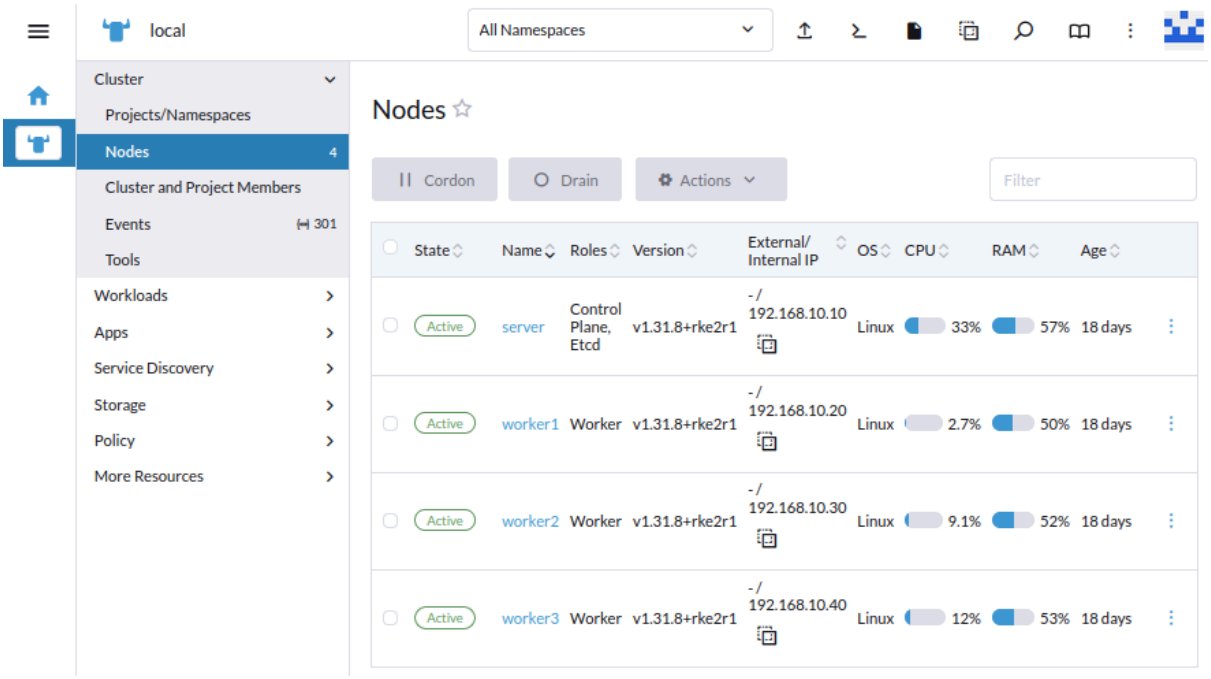
```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: coredns-worker
  namespace: kube-system
spec:
  selector:
    matchLabels:
      k8s-app: coredns-worker
  template:
    metadata:
      labels:
        k8s-app: coredns-worker
    spec:
      nodeSelector:
        node-role.kubernetes.io/worker: "true"
      containers:
        - name: coredns
          image: coredns/coredns:1.6.9
          args: [ "-conf", "/etc/coredns/Corefile" ]
          ports:
            - name: dns
              containerPort: 53
              hostPort: 53
              protocol: UDP
            - name: dns-tcp
              containerPort: 53
              hostPort: 53
              protocol: TCP
          volumeMounts:
            - name: config-volume
              mountPath: /etc/coredns
              readOnly: true
      volumes:
        - name: config-volume
          configMap:
            name: coredns
```

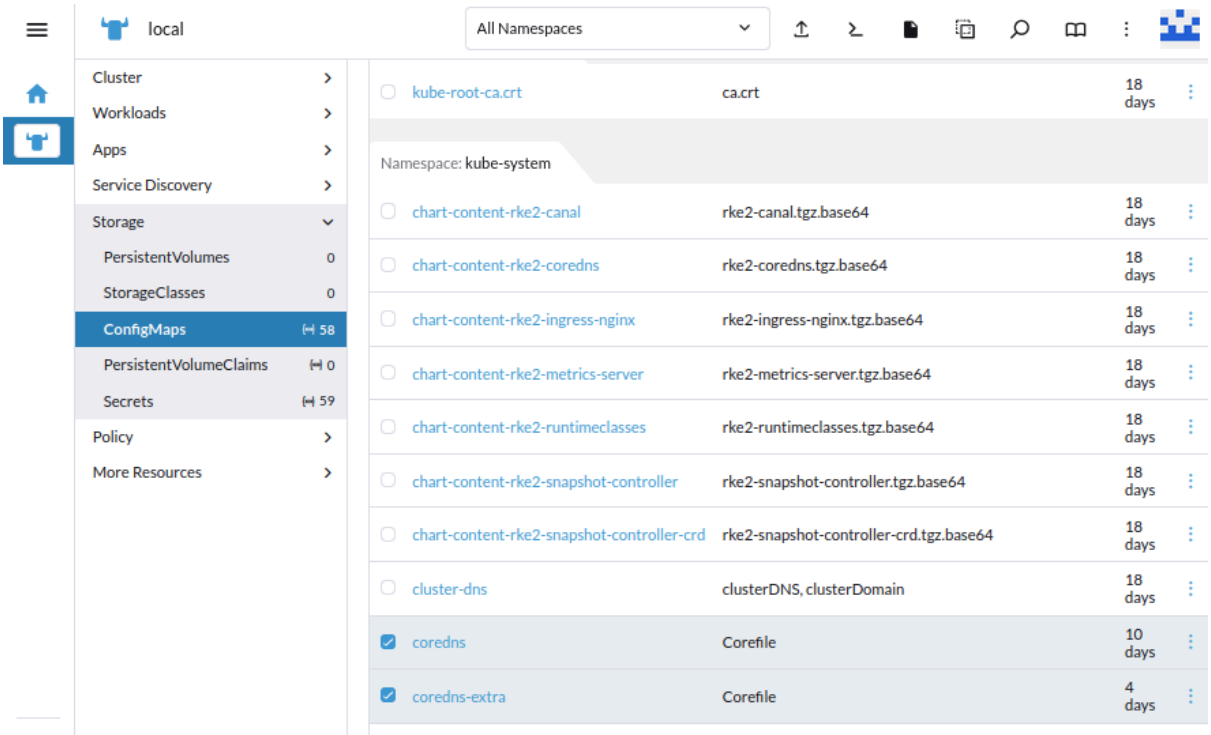

ConfigMap: coredns-extra
Active
Detail
Config
YAML
Namespace: kube-system
Age: 4 days
Data
Recent Events
Related Resources
Corefile
Copy
Corefile
.:53 {
log
errors
health
cache 30
forward . /etc/resolv.conf
prometheus :9153
loop
reload
loadbalance
}

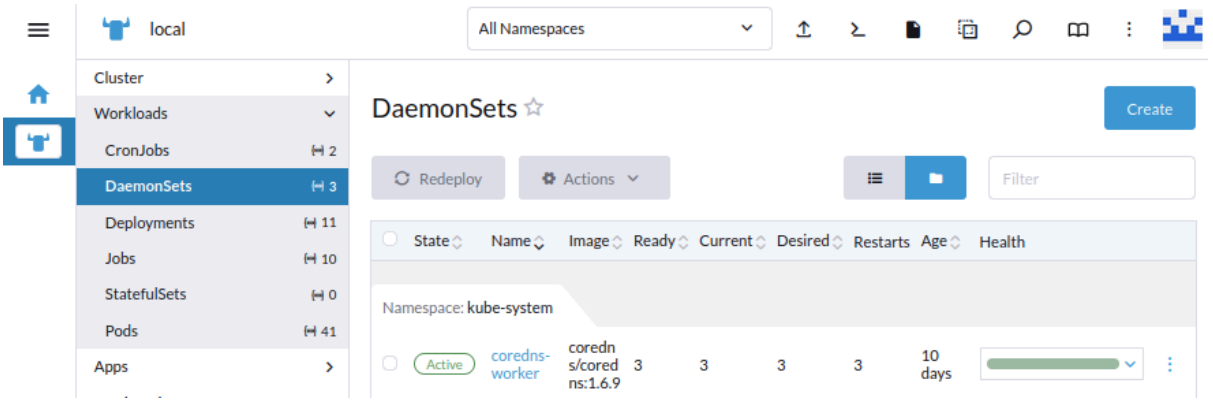
Anexo 2: Capturas de pantalla del panel de Rancher.

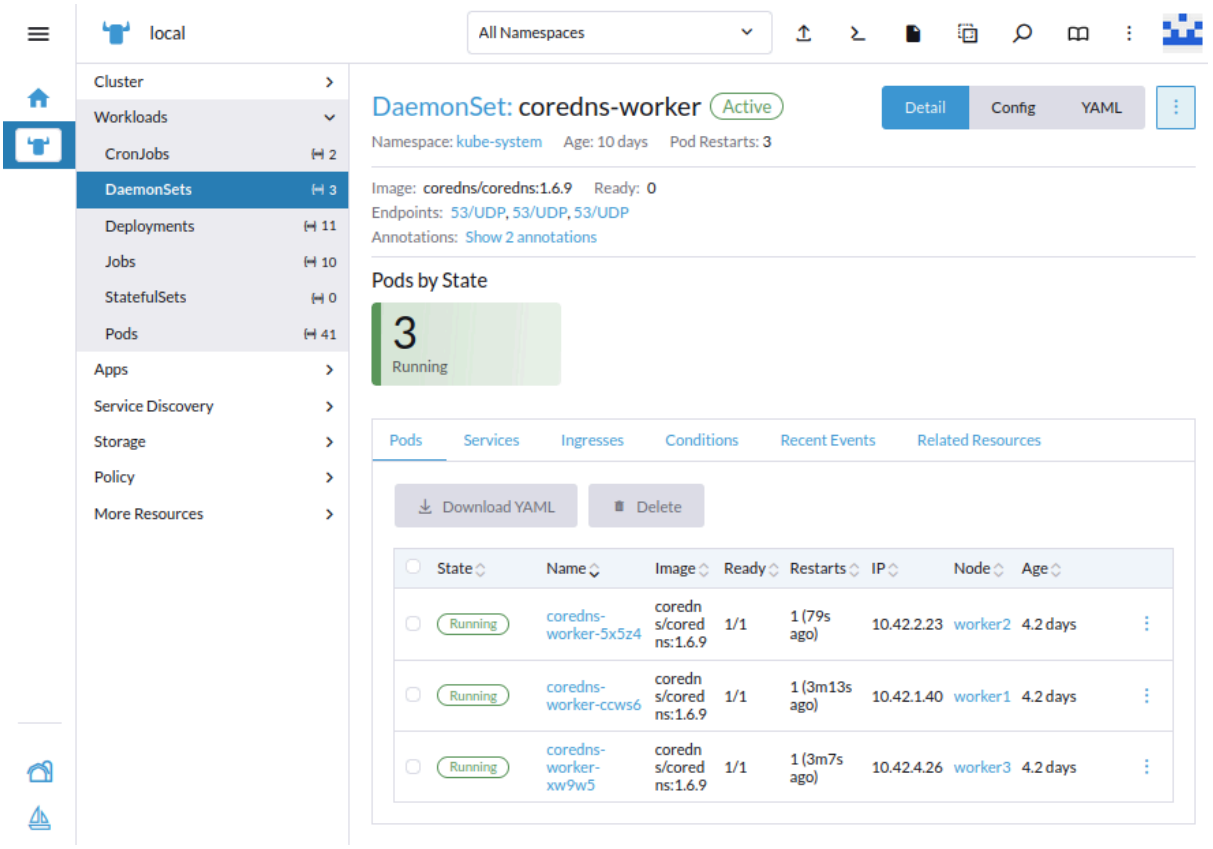

RANCHER
Welcome to Rancher
Learn more about the improvements and new capabilities in 2.11
What's new in 2.11
Clusters
Manage
Import Existing
Create
Filter
State
Name
Provider
Distro
Kubernetes Version
Architecture
CPU
Memory
Pods
Active
local
Local
RKE2
v1.31.8+rke2r1
amd64
8 cores
19 GiB
33/440
Links
Docs
Forums
Slack
File an Issue
Get Started
Commercial Support


Rancher - local - Cluster Dash
rancher.local/dashboard/c/local/explorer#cluster-events
80%
local
All Namespaces
Cluster
Projects/Namespaces
Nodes 4
Cluster and Project Members
Events 292
Tools
Workloads
Apps
Service Discovery
Storage
Policy
More Resources
Cluster Dashboard
Provider: RKE2
Kubernetes Version: v1.31.8 +rke2r1
Architecture: amd64
Created: 18 days ago
Cluster Tools
586 Total Resources
4 Nodes
11 Deployments
Capacity
Pods
Used 31 / 440 7.05%
CPU
Reserved 3.48 / 8 cores 43.44%
Used 0.92 / 8 cores 11.53%
Memory
Reserved 3.19 / 19 GiB 16.79%
Used 10 / 19 GiB 52.63%
Etcd
Scheduler
Controller Manager
Fleet

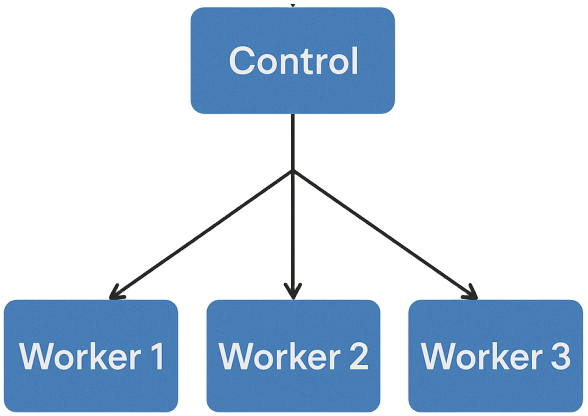

local
All Namespaces
Cluster
Projects/Namespaces
Nodes 4
Cluster and Project Members
Events 301
Tools
Workloads
Apps
Service Discovery
Storage
Policy
More Resources
Nodes
Cordon
Drain
Actions
Filter
State Name Roles Version External/ Internal IP OS CPU RAM Age
Active server Control Plane, Etcd v1.31.8+rke2r1 -/ 192.168.10.10 Linux 33% 57% 18 days
Active worker1 Worker v1.31.8+rke2r1 -/ 192.168.10.20 Linux 2.7% 50% 18 days
Active worker2 Worker v1.31.8+rke2r1 -/ 192.168.10.30 Linux 9.1% 52% 18 days
Active worker3 Worker v1.31.8+rke2r1 -/ 192.168.10.40 Linux 12% 53% 18 days

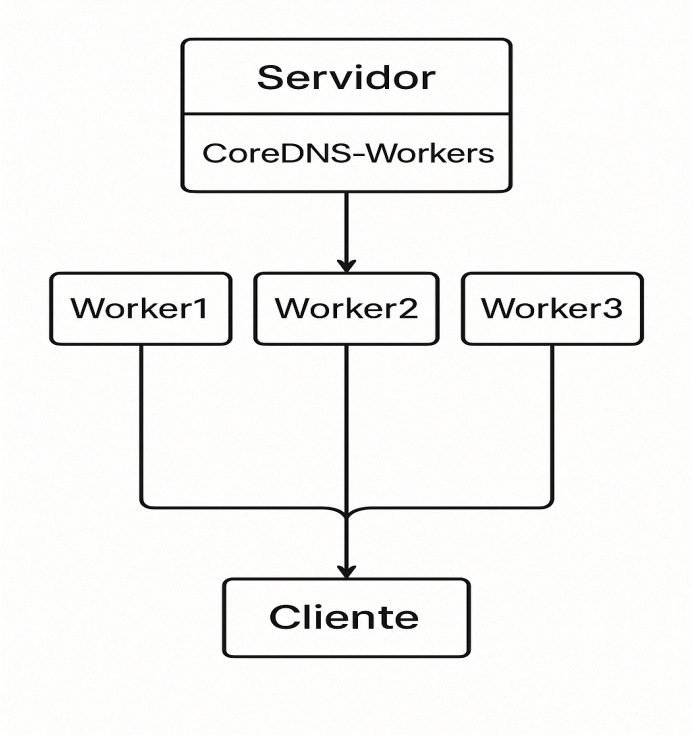

local
All Namespaces
Cluster
Workloads
Apps
Service Discovery
Storage
PersistentVolumes 0
StorageClasses 0
ConfigMaps 58
PersistentVolumeClaims 0
Secrets 59
Policy
More Resources
kube-root-ca.crt ca.crt 18 days
Namespace: kube-system
chart-content-rke2-canal rke2-canal.tgz.base64 18 days
chart-content-rke2-coredns rke2-coredns.tgz.base64 18 days
chart-content-rke2-ingress-nginx rke2-ingress-nginx.tgz.base64 18 days
chart-content-rke2-metrics-server rke2-metrics-server.tgz.base64 18 days
chart-content-rke2-runtimeclasses rke2-runtimeclasses.tgz.base64 18 days
chart-content-rke2-snapshot-controller rke2-snapshot-controller.tgz.base64 18 days
chart-content-rke2-snapshot-controller-crd rke2-snapshot-controller-crd.tgz.base64 18 days
cluster-dns clusterDNS, clusterDomain 18 days
coredns Corefile 10 days
coredns-extra Corefile 4 days

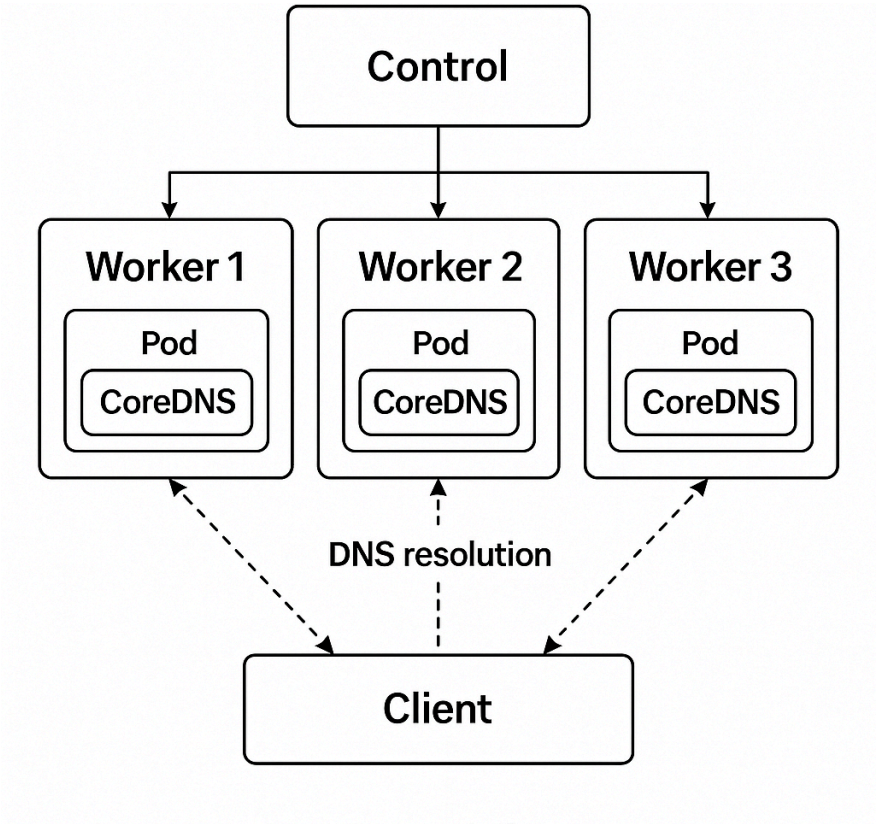




Anexo 3: Diagrama de la arquitectura del clúster.

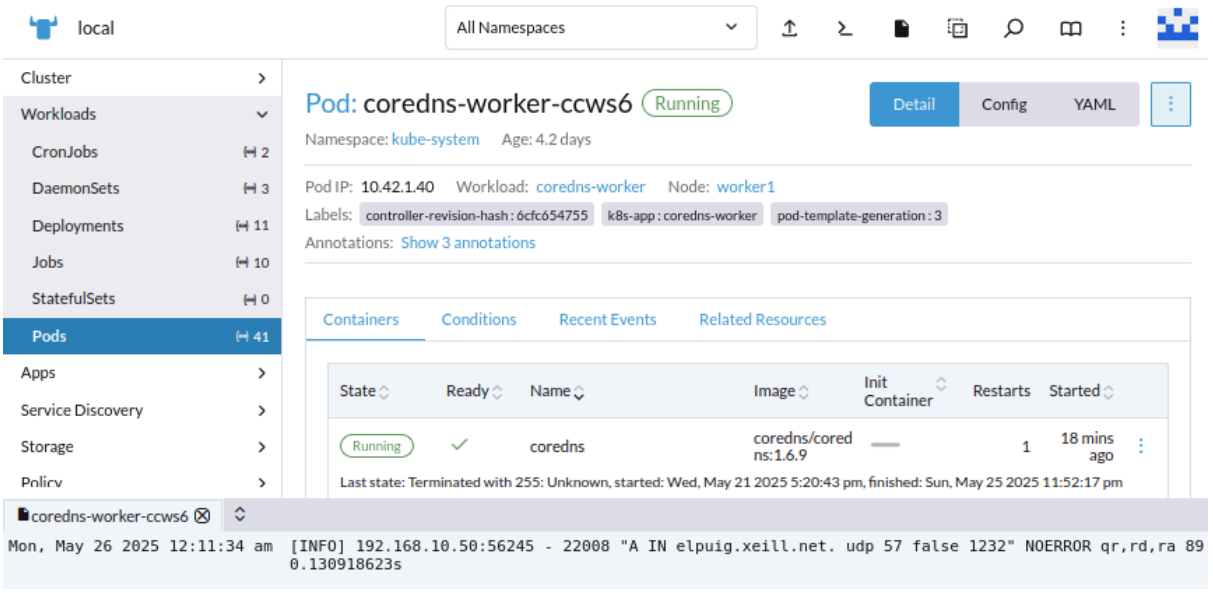

Control
Worker 1
Worker 2
Worker 3

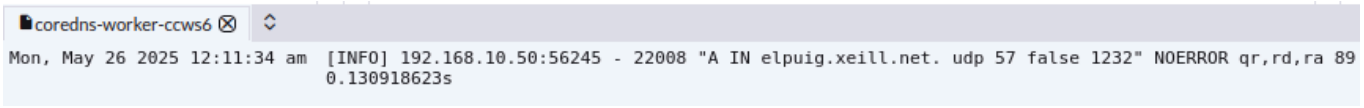

Servidor
CoreDNS-Workers
Worker1
Worker2
Worker3
Cliente



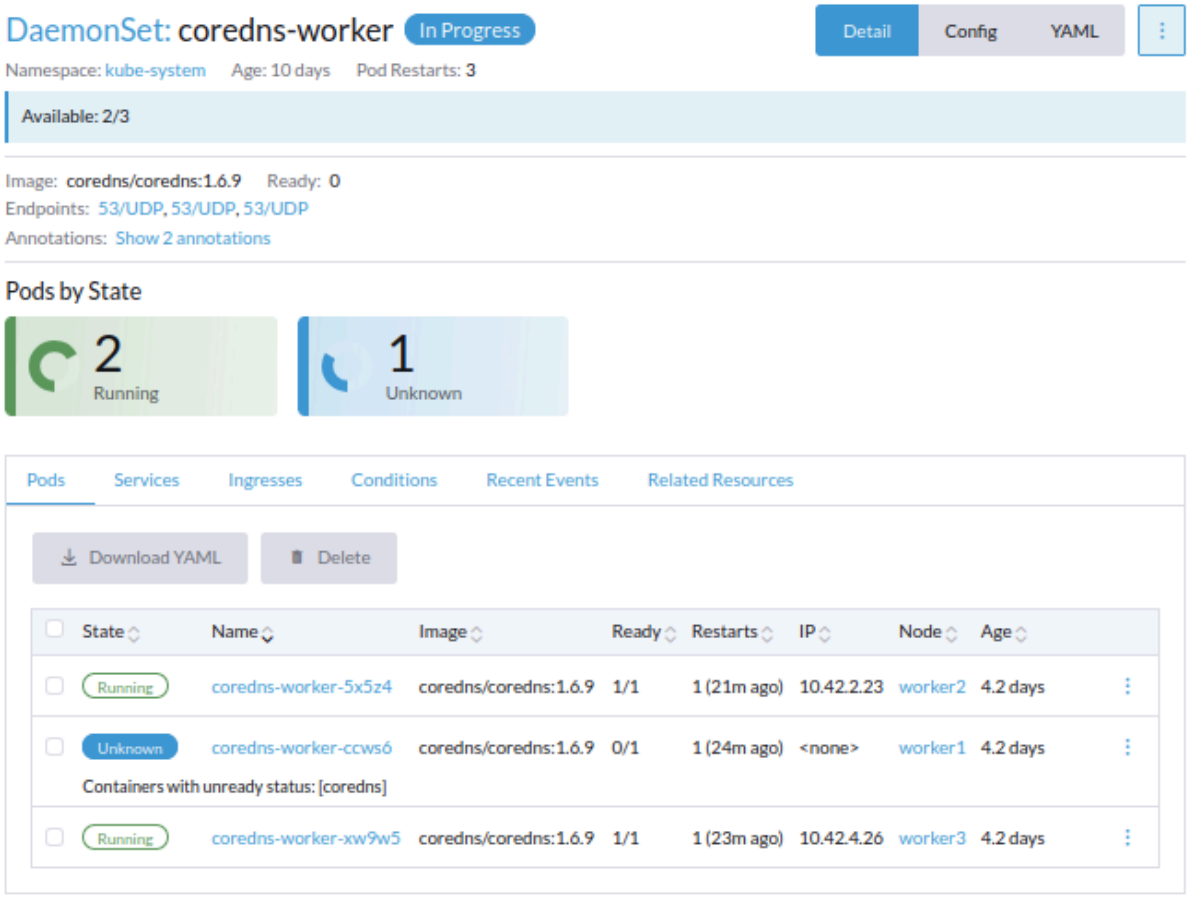
Anexo 4: Resultados de las pruebas ante caídas de nodo.

```
camilo@camilo:~$ dig elpuig.xeill.net

; <<>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <<>> elpuig.xeill.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22008
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 4414b9f08f3d2d99 (echoed)
;; QUESTION SECTION:
;elpuig.xeill.net.              IN      A

;; ANSWER SECTION:
elpuig.xeill.net.       30      IN      A       136.243.80.28

;; Query time: 132 msec
;; SERVER: 192.168.10.20#53(192.168.10.20) (UDP)
;; WHEN: Sun May 25 22:11:33 UTC 2025
;; MSG SIZE  rcvd: 89

camilo@camilo:~$
```





Apagamos el worker 1:


DaemonSet: coredns-worker
In Progress
Detail
Config
YAML
Namespace: kube-system
Age: 10 days
Pod Restarts: 3
Available: 2/3
Image: coredns/coredns:1.6.9
Ready: 0
Endpoints: 53/UDP, 53/UDP, 53/UDP
Annotations: Show 2 annotations
Pods by State
2
Running
1
Unknown
Pods
Services
Ingresses
Conditions
Recent Events
Related Resources
Download YAML
Delete
State
Name
Image
Ready
Restarts
IP
Node
Age
Running
coredns-worker-5x5z4
coredns/coredns:1.6.9
1/1
1 (21m ago)
10.42.2.23
worker2
4.2 days
Unknown
coredns-worker-ccws6
coredns/coredns:1.6.9
0/1
1 (24m ago)
<none>
worker1
4.2 days
Containers with unready status: [coredns]
Running
coredns-worker-xw9w5
coredns/coredns:1.6.9
1/1
1 (23m ago)
10.42.4.26
worker3
4.2 days

```
camilo@camilo:~$ dig elpuig.xeill.net
;; communications error to 192.168.10.20#53: timed out
;; communications error to 192.168.10.20#53: timed out
;; communications error to 192.168.10.20#53: timed out

; <<>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <<>> elpuig.xeill.net
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 46389
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL:

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: f227ef584f0592e4 (echoed)
;; QUESTION SECTION:
;elpuig.xeill.net.              IN      A

;; ANSWER SECTION:
elpuig.xeill.net.       30      IN      A       136.243.80.28

;; Query time: 88 msec
;; SERVER: 192.168.10.30#53(192.168.10.30) (UDP)
;; WHEN: Sun May 25 22:14:00 UTC 2025
;; MSG SIZE  rcvd: 89

camilo@camilo:~$
```