# CI/CD con Gitlab en Rancher + K3s



Jose Carlos Rodríguez Cañas

IES Gonzalo Nazareno

# Índice



# 1. Introducción

Esta memoria constituye una visión detallada del proyecto integrado, el cual está dirigido hacia **la creación de una plataforma** robusta de **gestión de clústeres de contenedores de código abierto**. En este proyecto se abordarán los aspectos esenciales relacionados con el **despliegue continuo, monitorización, implementación y creación de un cluster efectivo de producción con Rancher y K3s**.

## 1.1. Objetivos que se quieren conseguir

Los objetivos principales de este proyecto son los siguientes:

- **Implementación de un Cluster de K3s y Rancher:** El objetivo es entender sobre el despliegue de un cluster de K3s complementado con un orquestador de aplicaciones y contenedores como Rancher.

- **Investigación y Selección de Tecnologías Relevantes:** Ésto implica identificar herramientas, frameworks y plataformas que optimicen el despliegue, monitoreo y seguridad del clúster de contenedores (implementado en las métricas de Grafana y Prometheus), manteniéndolo en consonancia con las mejores prácticas de la industria.

- **Priorización de la Disponibilidad y Eficiencia:** La importancia de garantizar la disponibilidad y eficiencia tanto para los administradores de sistemas como para los desarrolladores. Ésto nos servirá para la adopción de prácticas y herramientas que faciliten la gestión del clúster, minimizando tiempos de inactividad y optimizando recursos.

- **Desarrollo de Herramientas de Supervisión y Seguridad**: Ésto implica la implementación de soluciones de monitoreo continuo, detección de amenazas y políticas de acceso, que garanticen la integridad y confidencialidad de los datos alojados en el clúster.

- **Facilitar la Implementación para Desarrolladores:** Ésto se logrará mediante la creación de herramientas y procesos automatizados que simplifiquen el despliegue y actualización de aplicaciones en el clúster.

- **Promoción de la Eficiencia y Seguridad en el Desarrollo y Despliegue de Aplicaciones:** Finalmente, se aspira a promover la eficiencia y seguridad en el ciclo de vida completo de las aplicaciones, desde su desarrollo hasta su despliegue en entornos de producción de manera continua.

## 1.2. ¿Por qué esta estructura y es necesario Rancher?

En **esta estructura** se basa en **crear un entorno de desarrollo y despliegue de aplicaciones que sea robusto**, escalable y fácil de gestionar para el administrador de sistemas. Por ello, utilizando herramientas como Gitlab, Rancher, K3s... Buscamos establecer un ambiente que permita la administración eficiente de contenedores y la implementación continua de aplicaciones.

**Rancher es una pieza clave en nuestra estructura por varias razones.** Lo más importante es que **nos facilita la administración del clúster de contenedores**, proporcionando una interfaz gráfica intuitiva y funcionalidades avanzadas para la configuración, monitoreo y escalado de aplicaciones. Ésto **nos ayuda a la gestión del clúster**, especialmente en **entornos complejos con múltiples nodos y servicios.**

Además, **Rancher ofrece características como el catálogo de aplicaciones**, que permite desplegar aplicaciones preconfiguradas con solo unos pocos clics. **También proporciona herramientas para el registro y la autenticación de usuarios**, lo que garantiza un acceso seguro al clúster y sus recursos.

En resumen, **la estructura propuesta con Rancher es totalmente necesaria** porque proporciona una plataforma que nos permite integrar de manera fácil de usar para la administración de clústeres de contenedores.

# 2. Fundamentos y conceptos básicos

Para poder proceder al montaje del entorno de producción, tendremos que entender cada uno de los servicios y aplicaciones que vamos a instalar en nuestro clúster.

## 2.1. K8s + K3s como clúster

Antes de comenzar las explicación de las tecnologías, tenemos que explicar que Kubernetes y K3s ya que son las bases de este proyecto.

**Kubernetes**, o más bien conocido como **K8s**, es una plataforma portable y extensible de código abierto que nos permite administrar las cargas de trabajo y servicios en contenedores. Además, **Kubernetes** facilita la automatización y permite la configuración declarativa.

K8s se puede entender como:

- **Una plataforma de contenedores**
- **Una plataforma de microservicios**
- **Una plataforma portable de nube**

y de muchas más maneras.


Con esto comentado, podemos ver que Kubernetes es una plataforma más que válida para la orquestación de clústers y contenedores

pero... **y ¿si llegamos más lejos?** Por ello, implementamos **K3s** con la ayuda de **Rancher** (lo comentaremos en el siguiente apartado).
**K3s** es una **distribución** de **Kubernetes** certificada y de alta disponibilidad diseñada para cargas de trabajo de producción en ubicaciones remotas, desatendidas y con recursos limitados o dentro de dispositivos IoT.

Una de las ventajas de utilizar **K3s** es que está empaquetado en **un único binario** (reconocido como Single binary), el cuál tiene un almacenamiento en **menos** de **70 MB de almacenamiento**. Ésto reduce las dependencias y los pasos necesarios para instalar, ejecutar y actualizar un clúster de Kubernetes de producción.


Un dato curioso de K3s es cómo está optimizado tanto para **ARM64** como para **ARMv7**, son compatibles con los binarios e imágenes de múltiples arquitecturas. Por ello, trabaja muy bien con algo pequeño como una **Raspberry PI** o como una instancia en **AWS** con un almacenamiento mínimo de 32GiB.

Como hemos comentado anteriormente, tendremos que integrar K3s con Rancher pero... ¿Qué es rancher y para que se utiliza? Lo veremos en el siguiente apartado.

## 2.2. ¿Qué es Rancher? Funcionamiento y compatibilidades

**Rancher** es un software de administración de contenedores de **código abierto** que nos sirve para gestionar aplicaciones de contenedores en entornos virtuales. Además, está perfectamente diseñado para integrarlo con **Kubernetes** o por ende, en nuestro caso utilizaremos **K3s**.

Aborda los desafíos operativos y de seguridad de administrar múltiples clústeres de **Kubernetes**, al tiempo que proporciona a los equipos de **DevOps** herramientas integradas para ejecutar cargas de trabajo en contenedores.

Otras características relevantes son la posibilidad de orquestación de servicios de almacenamiento persistente para **Docker**, compatibilidad con **Docker Machine** y la creación e importación de **clusters** como **Amazon EKS, Azure AKS y Google GKE**.


Por último, tenemos que entender que es el **Edge Computing** es el esquema que vamos a montar en Rancher. Este proporciona K3 para poder soportar la ejecución de Kubernetes dentro de los nodos edge.

## 2.3. Grafana + Prometheus para monitoreo y observabilidad

Antes de sumergirnos en el entorno de integración continua como **Gitlab**, es esencial comprender el funcionamiento de herramientas de monitoreo y observabilidad como **Grafana y Prometheus**.

**Grafana** es una herramienta poderosa para visualizar datos de series temporales, con una amplia integración con diversas fuentes de datos como **InfluxDB, PostgreSQL,** entre otras.

Además, su comunidad activa proporciona información valiosa y una variedad de plugins que permiten conectar con distintos sistemas. También ofrece la capacidad de enviar alertas de manera rápida y sencilla a través de correo electrónico.

Por otro lado, **Prometheus** es un software especializado en monitoreo y alertas, escrito en el lenguaje de programación **Go**. Éste recopila **métricas** de infraestructuras y aplicaciones, tales como datos sobre la CPU, RAM, rendimiento, aplicaciones en **Docker** o **K8s**, entre otros, almacenandose en su base de datos en forma de series temporales.

Algunas de sus características principales incluyen **un modelo de datos multidimensional, un lenguaje de consultas propio**, nodos autónomos de servidor único y configuración estática.

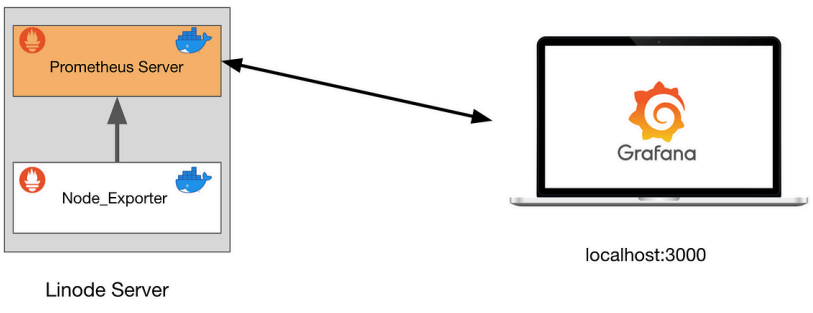
Una pregunta común que surge es cómo conectar Grafana y Prometheus para una monitorización y observabilidad eficientes. Ésto se logra mediante métricas, para lo cual Prometheus ofrece una herramienta llamada **node exporters**. Éstos pueden ser scripts o servicios que recolectan **métricas** específicas del sistema y las proporcionan en un formato adecuado para Prometheus.

Los **node exporters** son esenciales para el funcionamiento de Prometheus, ya que recopilan datos sobre el sistema y las aplicaciones. Sin embargo, dentro de los node exporters, los **collectors** desempeñan un papel fundamental al recopilar información específica, como el uso de la CPU o la memoria, y proporcionar a los node exporters.

Los usuarios tienen la flexibilidad de activar o desactivar estos **collectors** según sus necesidades y preferencias. Por ejemplo, si un usuario está interesado únicamente en monitorear el rendimiento de la CPU, puede activar el **collector** correspondiente y desactivar los demás.

En resumen, los collectors son componentes clave del node exporter que permiten la **recopilación de datos de métricas específicas** para su análisis posterior en Prometheus.

La integración de Prometheus con Grafana proporciona una solución completa para la **monitorización** y **observabilidad** de sistemas, permitiendo a los usuarios visualizar métricas en tiempo real, crear paneles personalizados y configurar alertas para mantener la estabilidad y el rendimiento del sistema.



## 2.4. Gitlab + Gitlab Runner como CI/CD

**GitLab**, al igual que **GitHub**, es un gestor de repositorios Git que facilita la colaboración en proyectos de desarrollo de software. Escrito en **Ruby** y **Go**, esta plataforma de código abierto es gratuita para uso personal y permite que equipos trabajen juntos en un mismo proyecto, proponiendo cambios y revirtiéndolos si es necesario.

Desde su versión 10.0, GitLab ha evolucionado más allá de ser solo un repositorio Git, convirtiéndose en una plataforma que abarca todo el ciclo de vida del desarrollo de software, ofreciendo una visión completa de DevOps.

Esta integración abarca desde la planificación y gestión del código fuente hasta el monitoreo y la seguridad, proporcionando una experiencia unificada para los usuarios.


Una función nativa de GitLab es la integración continua y la entrega continua (**CI/CD**), que permite el desarrollo, las pruebas y el despliegue continuos de aplicaciones.

Esta característica es impulsada por **GitLab CI/CD**, una herramienta que ejecuta trabajos de manera continua en una canalización de desarrollo.

El componente clave de GitLab CI/CD es el **GitLab Runner**, una aplicación que ejecuta trabajos definidos en la canalización de CI/CD.

El GitLab Runner puede ejecutarse en diferentes entornos, como máquinas virtuales, contenedores **Docker** o incluso dispositivos físicos, y se encarga de llevar a cabo las tareas de construcción, prueba y despliegue definidas en la configuración de CI/CD.

El proceso que sigue este disparador o **trigger** es el siguiente. Primero, el administrador o desarrollador en un entorno de desarrollo realiza un **commit** en el repositorio correspondiente. Tras ello, Gitlab avisa a este GitLab Runner y lee un fichero que contiene las fases de la implantación de la aplicación, incluyendo instalación de las dependencias, los **test jobs**...


Una vez termine dicho proceso, podrá dar **success** cuando se haya completado correctamente y ya tendríamos en correcto funcionamiento la aplicación en un almacenador de imágenes como **Docker**, **Podman**... y **error** si nos falta algo en nuestro código o cualquier error que deberemos consultar en sus respectivos logs.

Por ello, **GitLab** proporciona una plataforma completa para la gestión de proyectos de desarrollo de software, desde la planificación hasta la entrega, con una amplia gama de herramientas integradas como **GitLab CI/CD** y el **GitLab Runner**, que permiten la automatización y la mejora continua del proceso de desarrollo.

# 3. Escenario que se ha realizado

Antes de presentar el escenario, es necesario explicar los tipos de estructuras que se encuentran en Rancher. Una vez comprendidos, procederemos a analizar el esquema que he desarrollado.

## 3.1. Tipos de estructuras en Rancher

Antes de explorar los distintos tipos de arquitecturas en Rancher, es importante entender una recomendación clave de la plataforma: **al instalar Rancher en un solo nodo, este debe estar separado de los clústeres descendientes**. Ésto garantiza un despliegue sin conflictos y optimizado del clúster.

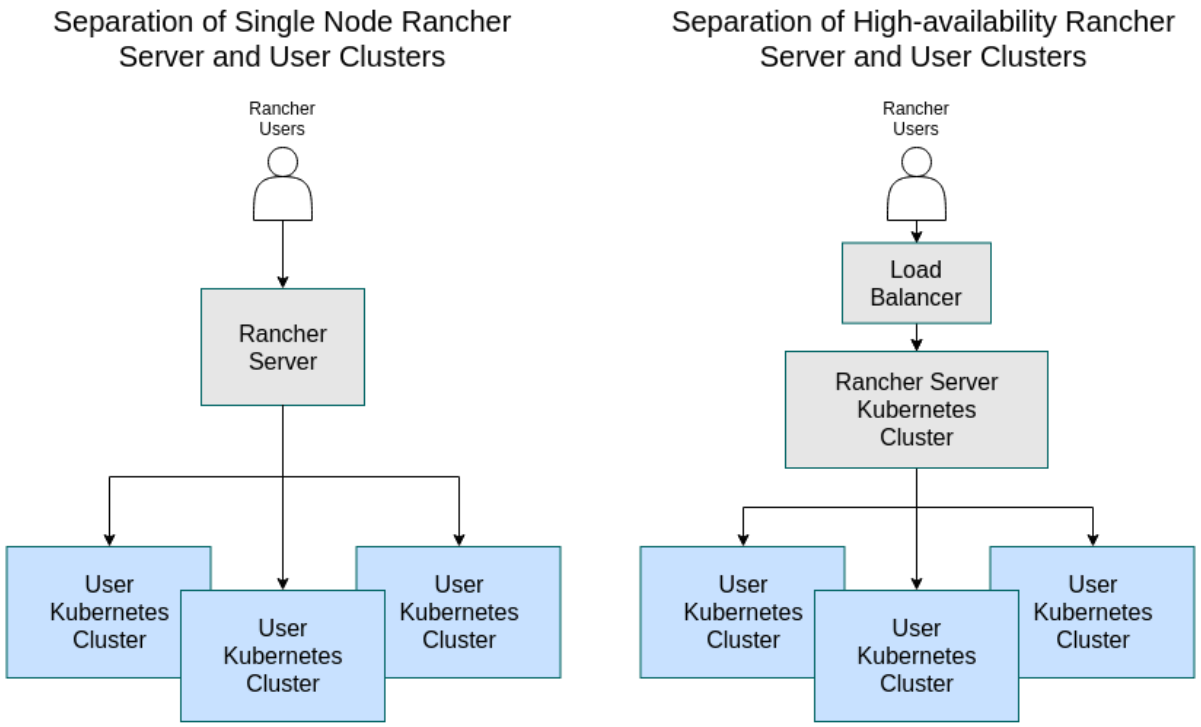
Los **clústeres de usuario** en Rancher representan entornos descendientes de Kubernetes donde se ejecutan aplicaciones y servicios específicos del usuario. Esta separación de clústeres habilita que podamos tener una gestión eficiente y permite una distribución adecuada de las cargas de trabajo.

Además, si se utiliza una instalación de **Docker** en Rancher, es un factor importante que el nodo servidor de Rancher esté también apartado de los clusters descendientes. Esta separación contribuye a la **estabilidad** y **eficiencia** del entorno Docker en Rancher. En definitiva, seguir estas pautas de separación entre el nodo servidor, los clusters descendientes y los clusters de usuario es clave para una gestión fluida y sin conflictos en el entorno de Rancher.

En Rancher, pueden coexistir 2 tipos de arquitecturas de instalación que son las siguientes:

- **Separación del servidor Rancher de nodo único y los clústeres de usuarios:** el cual se compone de los usuarios de K8s que se manejan de manera escalable desde el servidor de Rancher. A este servidor se accede desde la misma red local directamente.

- **Separación de servidores Rancher de alta disponibilidad y clústeres de usuarios:** éste es similar al anterior, la principal diferencia es la implementación de un balanceador de carga para controlar las peticiones a este mismo servidor.




Normalmente, **Rancher nos recomienda instalar un balanceador de carga** para nuestro servidor para que goce de la alta disponibilidad de su clúster de K8s. Ésto es prioritariamente por los datos de este clúster orquestado por Rancher.

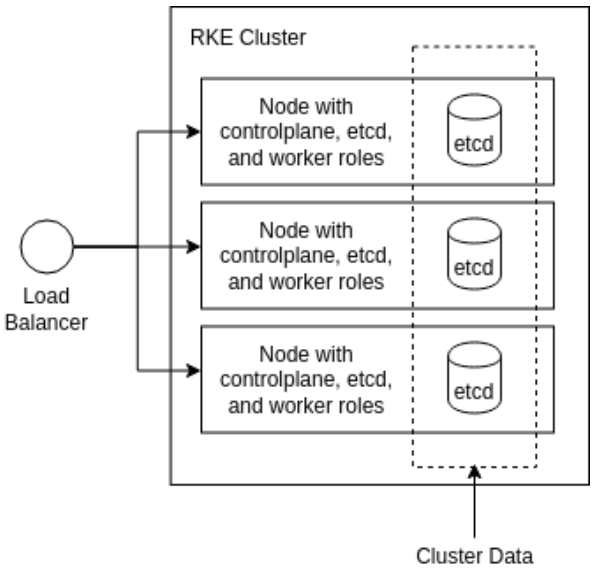
Por ello, **una instalación con alta disponibilidad** nos proporciona que el balanceador sirve como único punto de contacto para los clientes, distribuyendo el tráfico de red entre múltiples servidores en el clúster y ayudando a evitar que cualquier servidor se convierta en un punto de falla del mismo.

**Rancher** no recomiendo instalar Rancher en un solo contenedor Docker porque si el nodo deja de funcionar, no habrá copia de los datos del clúster disponibles en otros nodos pero... existe una **excepción**.

A la hora de crear el contenedor Docker, le añadimos un **volumen** el cual guarda simultáneamente todo el contenido sobre nuestra aplicación Rancher garantizando persistencia y seguridad en los datos eliminando por completo la idea de contenedor con **almacenamiento efímero**. Con esto dicho, ya sabemos con antelación cuál va a ser la intención de mi proyecto integrado.

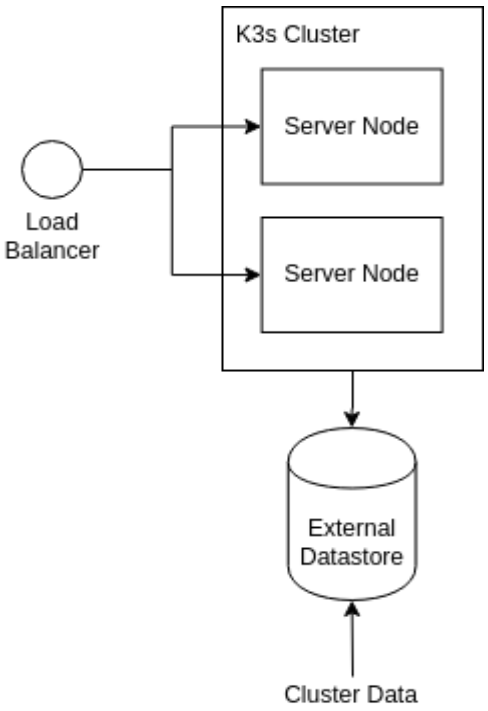
Ya visto los 2 tipos de arquitecturas de instalación en Rancher, pasamos a ver los **2 tipos de instalación de clusters de K8s** que se pueden montar. Estos son los siguientes:

- **Instalación de un clúster de RKE K8s:** En una instalación de RKE, los datos del clúster se replican en cada uno de los tres nodos etcd del clúster, lo que proporciona **redundancia y duplicación de datos** en caso de que uno de los nodos falle. Una imagen de cómo sería un clúster RKE Kubernetes que ejecuta **Rancher Management Server**:




Nota: **Rancher Kubernetes Engine** (RKE) es una distribución de Kubernetes certificada por CNCF que se ejecuta completamente dentro de contenedores Docker.

- **Instalación de un clúster de K3s K8s**: Una opción para el clúster de Kubernetes subyacente es utilizar un clúster montado previamente o no (ya que lo podemos crear desde interfaz gráfica de Rancher) de K3s Kubernetes. Con los conocimientos adquiridos sobre K3s anteriormente, pasamos a ver la arquitectura de un clúster de Kubernetes K3s que ejecuta **Rancher Management Server**:




Cada uno de estos clusters, mantienen unas características en cuanto a funciones y roles distintas entre ellos.

En los clusters de RKE se deben tener tres nodos y cada nodo debe tener las tres funciones de Kubernetes: **etcd, plano de control (controlplane) y trabajador (worker)**.

En los clusters de K3s hay dos tipos de nodos: **nodos de servidor y nodos de agente**. Tanto los servidores como los agentes pueden tener cargas de trabajo programadas en ellos. Los nodos del servidor ejecutan el maestro de K8s.

Conociendo las diferencias de estructuras de instalación y de clusters, voy a presentar en que se basará mi proyecto.

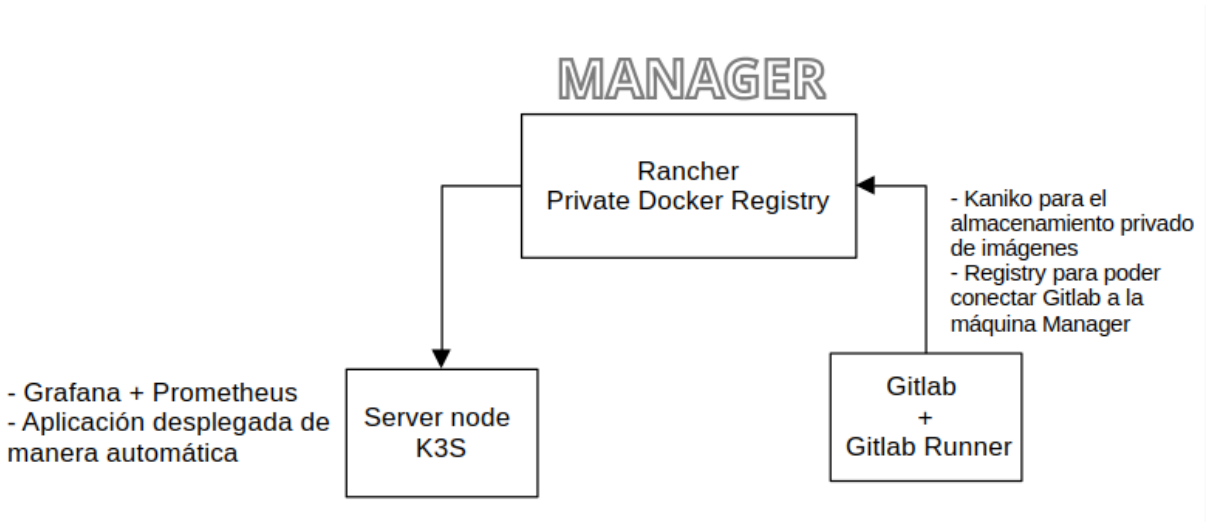
## 3.2. Escenario elegido con todas sus características

Con respecto al escenario que quiero desplegar, me basaré en una estructura con las. Por ello, se montará un cluster de **K3s** ya que nos proporciona un alto rendimiento consumiendo una cantidad de recursos mínima.

Se necesitará un nodo manager que tendrá instalado Rancher y nos controlará a nuestro nodo master de **K3s** que tendrá instalada las demás aplicaciones. Estos nodos se instalarán en KVM.

Seguido de esto, **Rancher** (implementado en Docker) con un balanceador de carga (Nginx) para poder hacer peticiones a la aplicación desplegada mediante **GitLab** (desplegado en Rancher) con su respectivo **Runner** y controlado mediante las métricas de **Grafana** y **Prometheus** (desplegado en Rancher).

Con estos componentes, el esquema se verá de la siguiente manera:




Con dichas características, podemos **ver** la **funcionalidad y el desempeño de cada una de estas herramientas de producción** como la integración continua mediante GitLab y la observabilidad y monitorización mediante Grafana y Prometheus **en un clúster local con alta disponibilidad** con K3s. Tras esto, pasamos a la instalación.

## 3.3. Instalaciones

En los siguientes apartados, realizaremos las instalaciones de todos los servicios y sistemas que utilizaremos en el proyecto.

### 3.3.1. Nodos mediante KVM

Para poder montar nuestro clúster de alta disponibilidad en Rancher con K3s, nos hace falta tener nuestros nodos montados correctamente.

Destacar que las máquinas de este esquema como la máquina host donde hemos instalado el gestor de máquinas virtuales, serán máquinas **Ubuntu 22.04 Jammy Jellyfish** pero estos comandos también son válidos para distribuciones como **Debian**.

Para desplegar el esquema, actualizaremos la máquina para poder tener los repositorios actualizados. Tras esto, ejecutaremos el 2 comando para ver que nuestro sistema cumpla con los requisitos de **KVM** y es necesario para evitar errores que puedan resultar en una instalación incompleta.

```
$ sudo apt update -y && sudo apt upgrade
$ sudo egrep -c '(vmx|svm)' /proc/cpuinfo
```

Cuando ya lo tengamos instalado, pasamos a la creación del escenario. Para poder desplegar nuestro esquema, deberemos crear una máquina en nuestro gestor de máquinas virtuales con la versión de **Ubuntu** correspondiente.

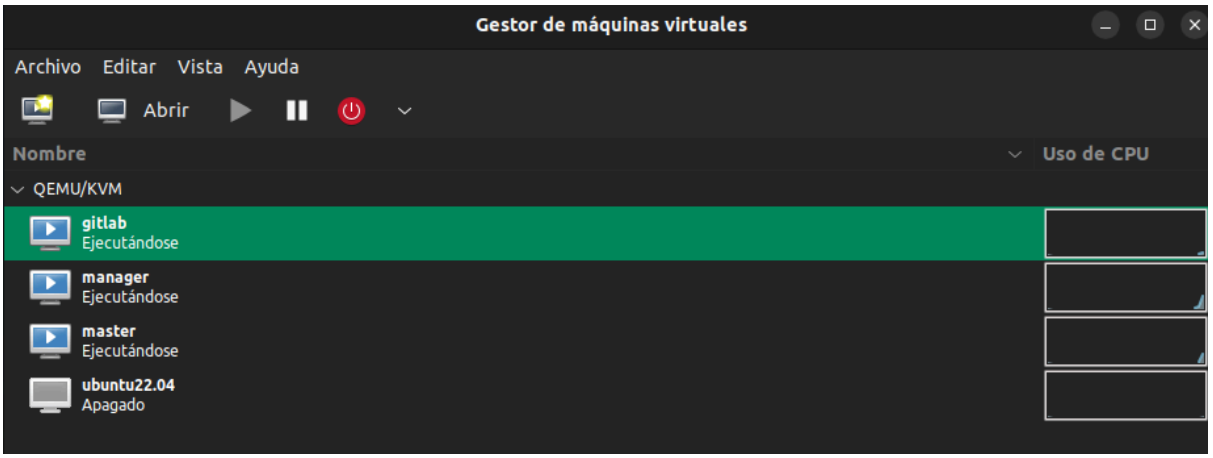
Si no tenemos instalado **KVM**, podemos instalarlo de la siguiente manera:

```
$ sudo apt-get install qemu-kvm libvirt-daemon-system libvirt-clients
bridge-utils virt-manager
$ sudo usermod -aG libvirt $USERNAME
$ sudo usermod -aG kvm $USERNAME
```

Además de instalarlo, tenemos que darle permisos para que el usuario sin privilegios pueda tener acceso a cualquier máquina virtual, volumen lógico... etc

Con esto comentado, pasamos a la instalación de una máquina **Ubuntu 22.04** y cuando la tengamos la clonamos 3 veces para poder realizar el proyecto.

Subrayar que la máquina principal, **tendrá un almacenamiento de 25GB y un uso de memoria máximo de 4GB de RAM.**




Algunos retoques pueden ser deshabilitar el uso de contraseña para el acceso de permisos de root, cambiar el nombre del hostname en la terminal para reconocer las máquinas y actualizar los repositorios para poder salir a internet ya que la versión de Ubuntu 22.04 está desactualizada (actualmente por la **24.04 Noble Numbat**).


Desplegamos las máquinas y accedemos a ellas para ver que se han creado correctamente. Con las máquinas creadas y operativas, pasamos al despliegue de **Rancher** y el clúster de **K3s**.

### 3.3.2. Rancher y cluster local de K3s

En primer lugar, instalamos Rancher en la máquina **manager**, en la máquina donde desplegamos **Gitlab** y en la máquina de Gitlab para poder hacer el login al **Registry privado** con mediante **Kaniko**.Acto seguido, conectamos el servidor **master** mediante un nodo de K3s.

Para instalar Rancher, utilizaremos Docker como herramienta de despliegue de la aplicación. La instalación de Docker se habilitará añadiendo unos repositorios que instalan **docker-ce**. Esto nos servirá para poder hacer uso de los comandos de Docker.

```
$ sudo apt-get update
$ sudo apt install apt-transport-https ca-certificates curl
software-properties-common

$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg
--dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg

$ echo "deb [arch=$(dpkg --print-architecture)
signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

$ sudo apt-get update
$ sudo apt install docker-ce
$ sudo usermod -aG docker ${USER}
```

A continuación, tendremos que utilizar un dominio propio de este modo podremos tener una clave certificada por una **CA**. Si quieres crear una propia y firmar con una Entidad Certificadora, también se puede realizar.

Para poder copiar las claves de mi host a mi máquina manager, las pasamos mediante **scp** desde mi host a esta misma. Un ejemplo:

```
$ scp /etc/ssl/rancher/* pepe@[ip_ext_máquina_manager]:
```

Seguidamente, creamos en la máquina manager el siguiente directorio y copiamos los archivos mediante **scp** a la ruta indicada:

```
$ sudo mkdir -p /etc/ssl/rancher
$ sudo cp [nombredominio]_ssl_certificate.cer /etc/ssl/rancher/
$ sudo cp _.[nombredominio]_private_key.key /etc/ssl/rancher/
```

Con los certificados ubicados en mi máquina manager, pasamos a la instalación de Rancher. Así podremos unificar las claves que tenemos en nuestra máquina manager y el contenedor **Docker**, tenemos que crear un volumen con el comando de **Docker Engine** para que se guarde automáticamente. Para ello, utilizaremos el siguiente comando:

```
$ sudo docker run -d --name rancher --restart=unless-stopped \
  -p 8080:80 -p 8443:443 \
  -v
/etc/ssl/rancher/pepepfoter15.es_ssl_certificate.cer:/etc/rancher/ssl/pepepf
oter15.es_ssl_certificate.cer \
  -v
/etc/ssl/rancher/_.pepepfoter15.es_private_key.key:/etc/rancher/ssl/_.pepepf
oter15.es_private_key.key \
  -v /var/lib/docker/rancher:/var/lib/rancher \
  --privileged \
  rancher/rancher:stable
```

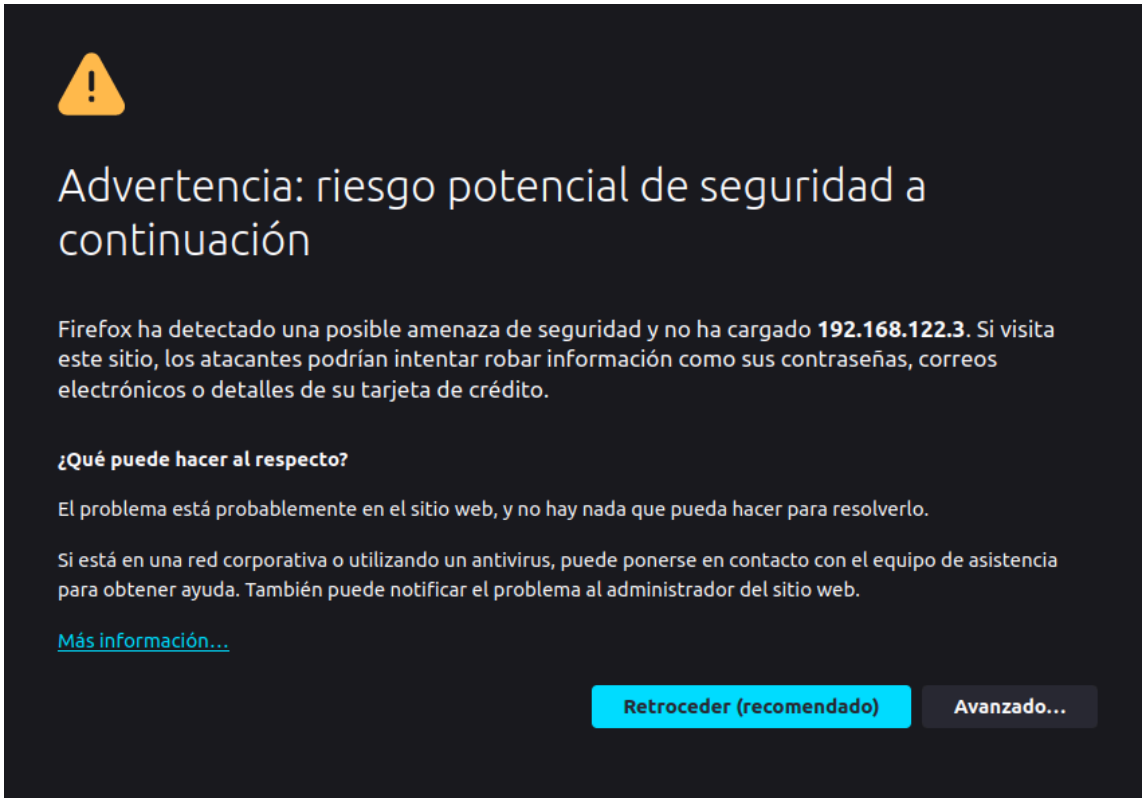
Automáticamente Docker, extrae la imagen de Rancher ubicada en **Docker Hub**, la descarga y crea el contenedor con la misma.

```
pepe@manager:~$ sudo docker run -d --name rancher --restart=unless-stopped \
  -p 8080:80 -p 8443:443 \
  -v /etc/ssl/rancher/pepepfoter15.es_ssl_certificate.cer:/etc/rancher/ssl/pepepfoter15.es_ssl_certif
icate.cer \
  -v /etc/ssl/rancher/_.pepepfoter15.es_private_key.key:/etc/rancher/ssl/_.pepepfoter15.es_private_ke
y.key \
  -v /var/lib/docker/rancher:/var/lib/rancher \
  --privileged \
  rancher/rancher:stable
Unable to find image 'rancher/rancher:stable' locally
stable: Pulling from rancher/rancher
53198d61f590: Pull complete
21251ce9e27c: Pull complete
11eb5e81edd1: Pull complete
75e3489ac5ae: Pull complete
1563d5833068: Pull complete
c4c5e72c3608: Pull complete
664c371e3589: Pull complete
37a2a72f797f: Pull complete
44e69243627d: Pull complete
9d4fe59c24b9: Pull complete
38b50947c238: Pull complete
676e3bff064c: Pull complete
1c038cbe2800: Pull complete
9f2fae27c7fc: Pull complete
fea0e93002f2: Pull complete
9ca5e34b294c: Pull complete
4cb5d07e072e: Pull complete
4ff9191461b1: Pull complete
616e7747783a: Pull complete
c672d755205c: Pull complete
Digest: sha256:6e3ea2fdc11c98de7fa71ef2dc434396c9e0255f8fbbc56de1f53752a5174458
Status: Downloaded newer image for rancher/rancher:stable
f2dc95660bd661234702badeddbded831bbb2774ce2e94363cb05ed8ca798047
pepe@manager:~$
```

Tras éllo, ya tendríamos instalado Rancher como contenedor. Para verificar que la máquina está funcionando correctamente, abrimos el navegador e ingresamos esta URL con la IP externa que utilizamos para pasar los certificados a nuestra máquina manager:

```
https://[ip_ext_máquina_manager]:8443/
```

Cuando accedemos, aparecerá un mensaje indicando que no se reconoce el certificado. Para continuar, seleccionamos **Avanzado** y luego **Aceptar el riesgo y continuar**.




Seguido de este proceso, aparecerá la interfaz de Rancher, la cual nos pedirá que ejecutemos algunos comandos para poder acceder mediante la contraseña de **Bootstrap**.

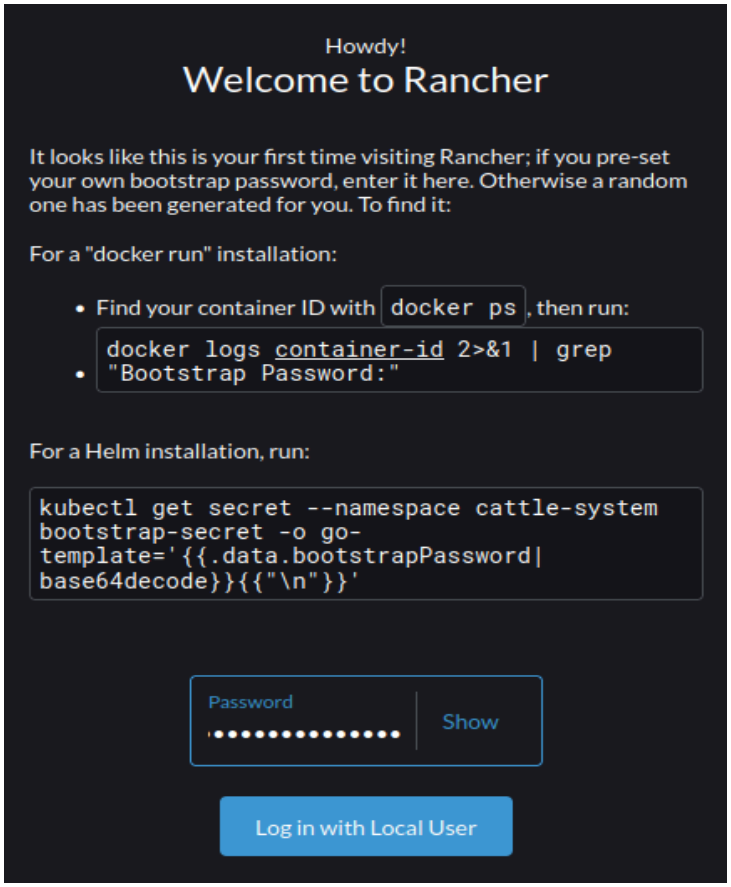
Para comenzar, nos pide que veamos el **ID** de nuestro contenedor y, seguido de esto, consultemos la contraseña con los siguientes comandos:

```
$ sudo docker ps
$ sudo docker logs [id-contenedor] 2>&1 | grep "Bootstrap Password:"
```

Todo ésto en nuestra máquina **manager**.

```
pepe@manager:~$ sudo docker ps
CONTAINER ID   IMAGE                  COMMAND           CREATED         STATUS         PORTS
                                                                      NAMES
f2dc95660bd6   rancher/rancher:stable   "entrypoint.sh"   6 minutes ago   Up 6 minutes   0.0.0.0:8080
->80/tcp, :::8080->80/tcp, 0.0.0.0:8443->443/tcp, :::8443->443/tcp   rancher
pepe@manager:~$ sudo docker logs  f2dc95660bd6  2>&1 | grep "Bootstrap Password:"
2024/06/11 11:49:26 [INFO] Bootstrap Password: tkj2nllvvgqqgvkmfmb2xrkf8vgmks8wh5tnggdpjv8jqm8z6vfb5m
pepe@manager:~$
```

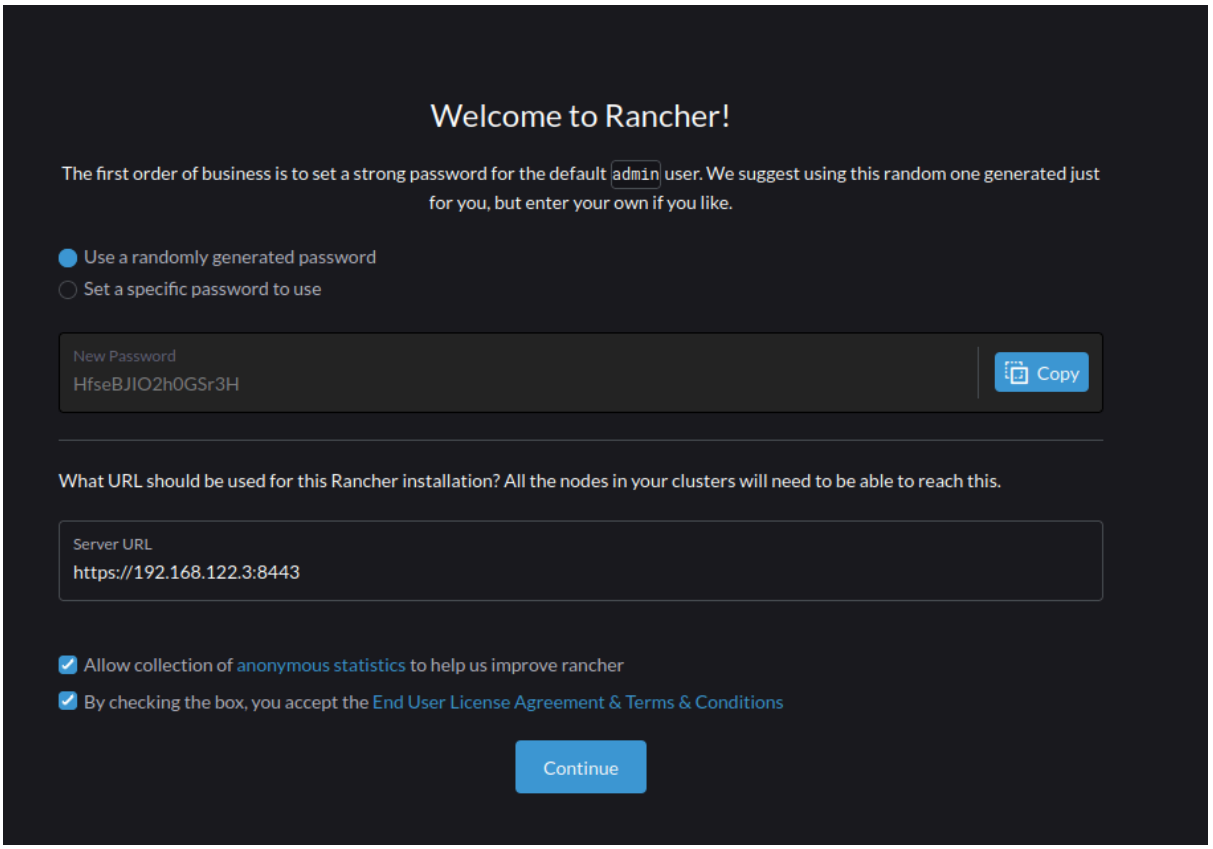
Como podemos observar, tenemos que copiar la clave y pegarla en la página desplegada en nuestro navegador.



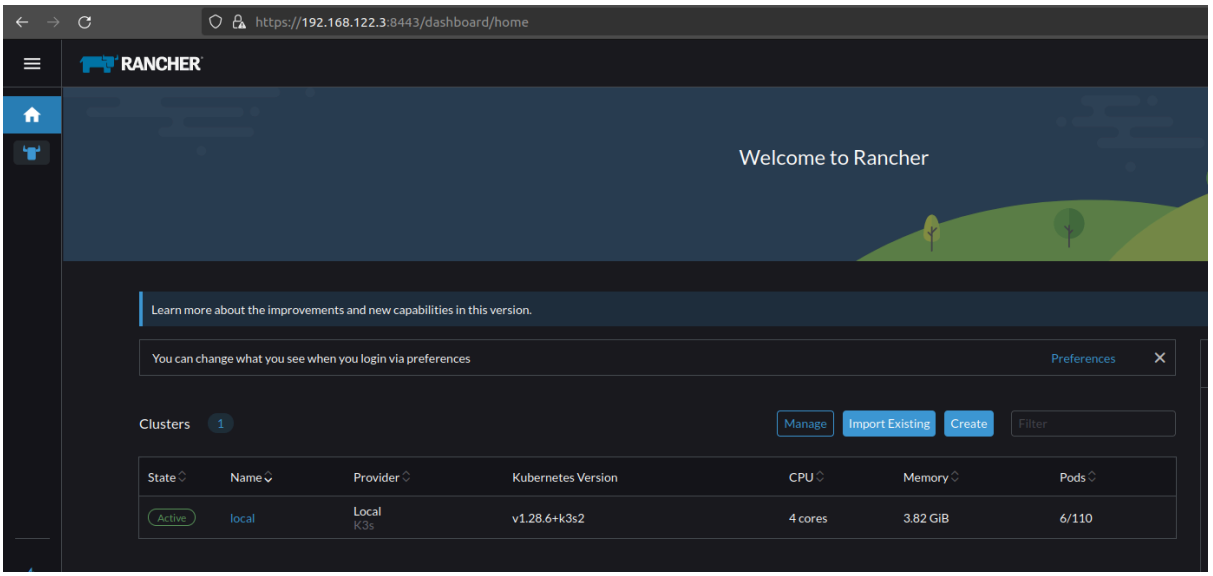

Seguido de ésto, podemos cambiar la contraseña del usuario administrador si queremos o la podemos dejar por defecto y podemos cambiar la URL a la cual debería usar Rancher en la instalación.

Un dato muy importante, **es que todos los nodos deben de ser accesibles a esta URL,** por ello, nos conviene no modificarlo.

Por último, aceptamos los términos y condiciones de esta aplicación para el usuario y continuamos.




Y con ésto, ya tendríamos Rancher instalado.

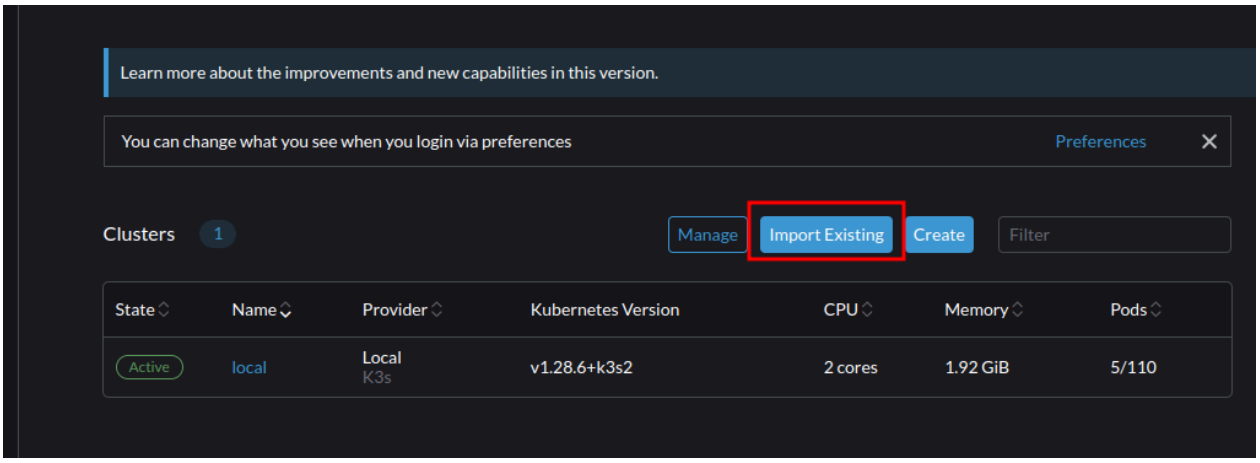


Al tener Rancher instalado, ya tenemos 1 paso, nos falta la instalación del nodo de **K3s**. Para ello, instalamos en la máquina **master** con el siguiente comando:

```
$ sudo apt-get update
$ curl -sfL https://get.k3s.io | sh -
```

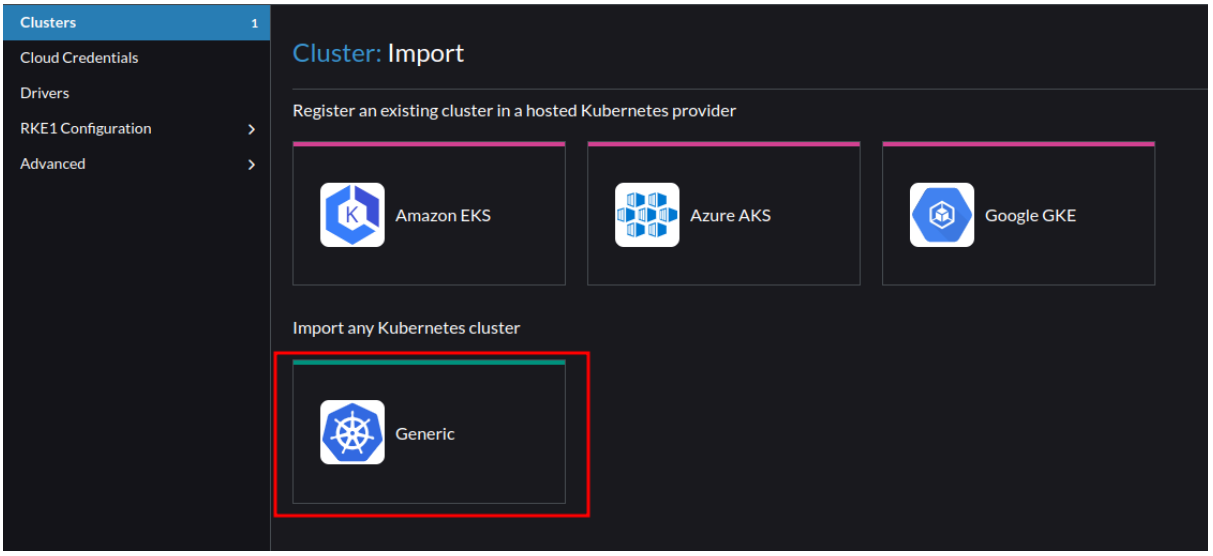
```
pepe@master:~$ curl -sfL https://get.k3s.io | sh -
[INFO]  Finding release for channel stable
[INFO]  Using v1.29.5+k3s1 as release
[INFO]  Downloading hash https://github.com/k3s-io/k3s/releases/download/v1.29.5+k3s1/sha256sum-amd64
.txt
[INFO]  Downloading binary https://github.com/k3s-io/k3s/releases/download/v1.29.5+k3s1/k3s
[INFO]  Verifying binary download
[INFO]  Installing k3s to /usr/local/bin/k3s
[INFO]  Skipping installation of SELinux RPM
[INFO]  Creating /usr/local/bin/kubectl symlink to k3s
[INFO]  Creating /usr/local/bin/crictl symlink to k3s
[INFO]  Creating /usr/local/bin/ctr symlink to k3s
[INFO]  Creating killall script /usr/local/bin/k3s-killall.sh
[INFO]  Creating uninstall script /usr/local/bin/k3s-uninstall.sh
[INFO]  env: Creating environment file /etc/systemd/system/k3s.service.env
[INFO]  systemd: Creating service file /etc/systemd/system/k3s.service
[INFO]  systemd: Enabling k3s unit
Created symlink /etc/systemd/system/multi-user.target.wants/k3s.service → /etc/systemd/system/k3s.ser
vice.
[INFO]  systemd: Starting k3s
pepe@master:~$
```

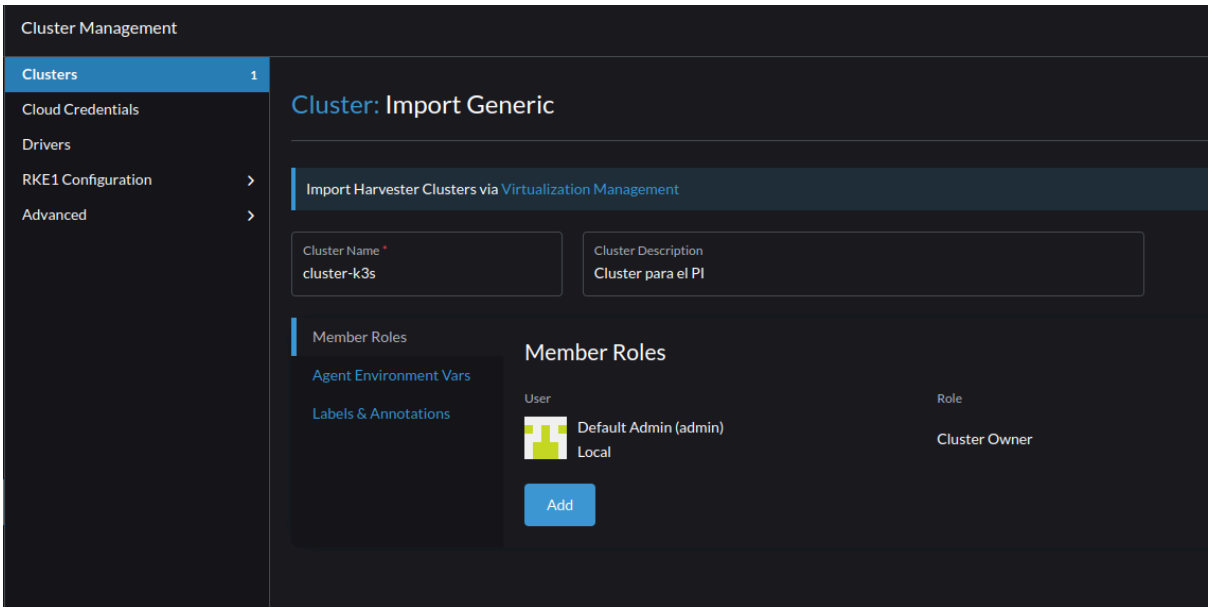
Ya instalado, pasamos a importar el clúster desde la interfaz gráfica, indicando **Import Existing.**




Nos pedirá el tipo de cluster a importar y como podemos observar, están los clústers **EKS, AKS y GKE** que comentamos en el inicio de la memoria.

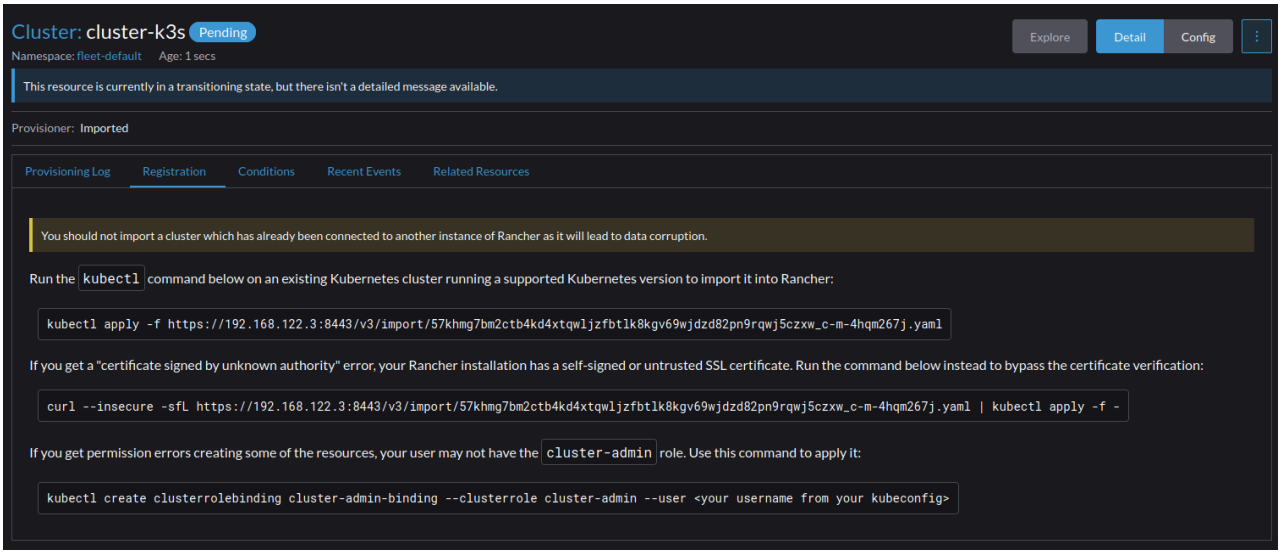
A nosotros nos interesa presionar **Generic** para importar un clúster de K3s y indicamos el nombre y descripción de éste:



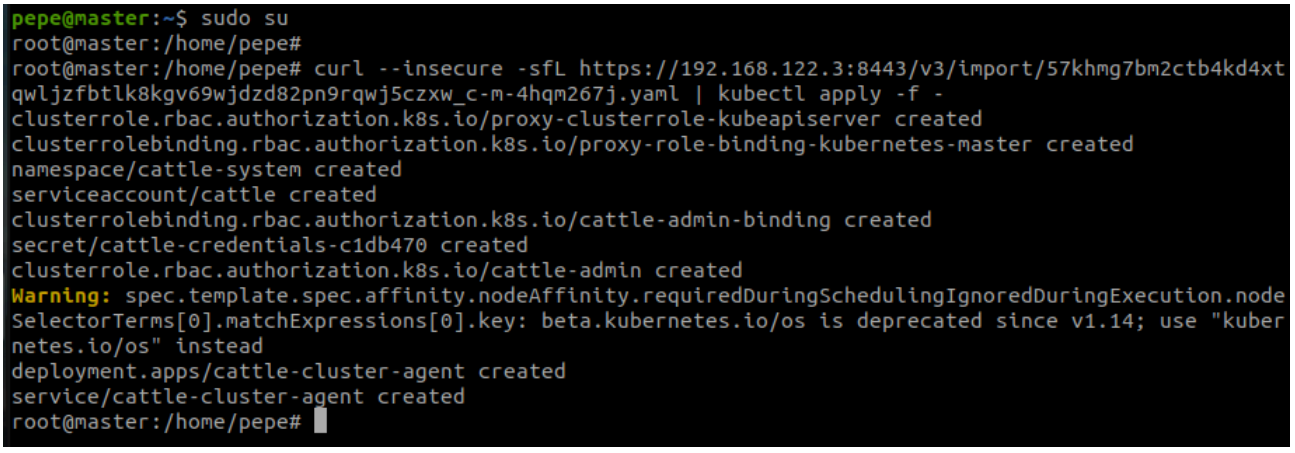

Cuando ya tengamos los nombres, hacemos clic en **Crear** para establecer la base que nos permitirá importar nuestro clúster.

Luego, Rancher nos pedirá que indiquemos el tipo de certificado que tiene. En nuestro caso, debemos elegir la **segunda opción**, ya que nuestro certificado no es confiable porque está en la máquina manager.

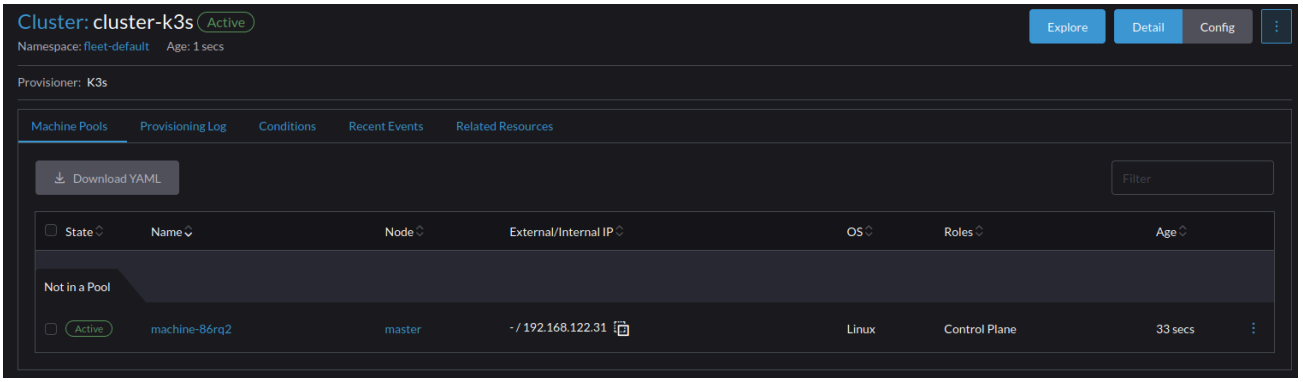



Copiamos este comando y lo ejecutamos en nuestra máquina máster como root. Veremos que se aplican todos los **service, deployments, secret, namespaces...**

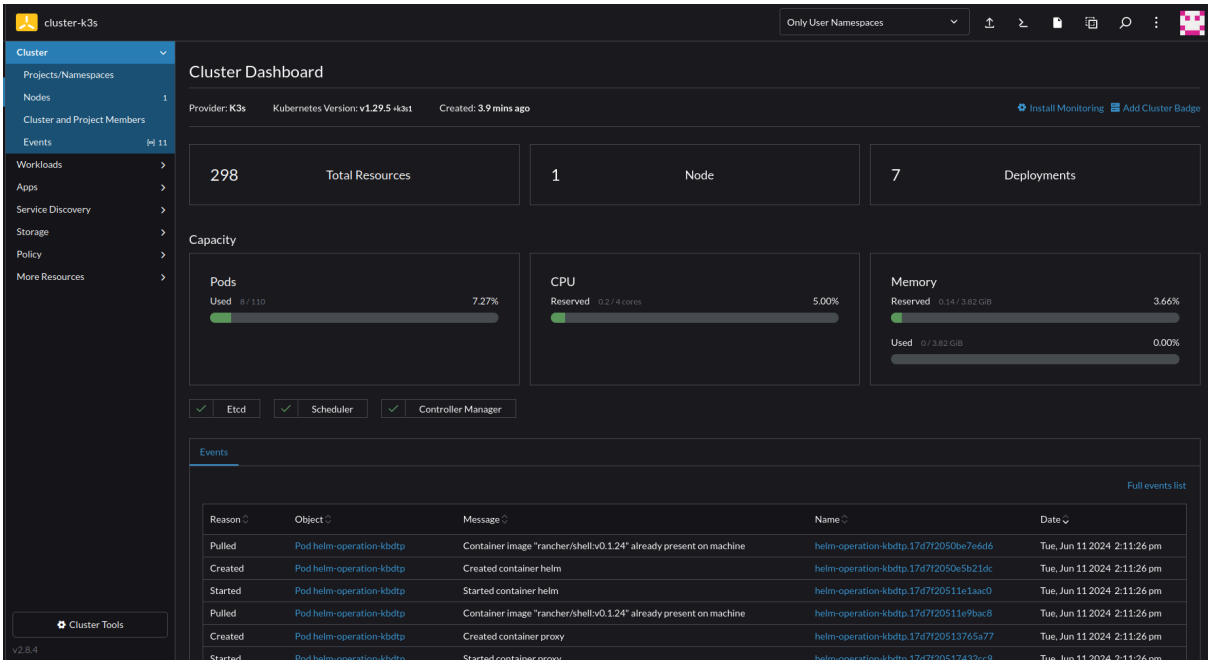
```
$ sudo su
$ curl --insecure -sfL https://[ip_ext_máquina_manager]:8443/[url].yaml |
kubectl apply -f -
```

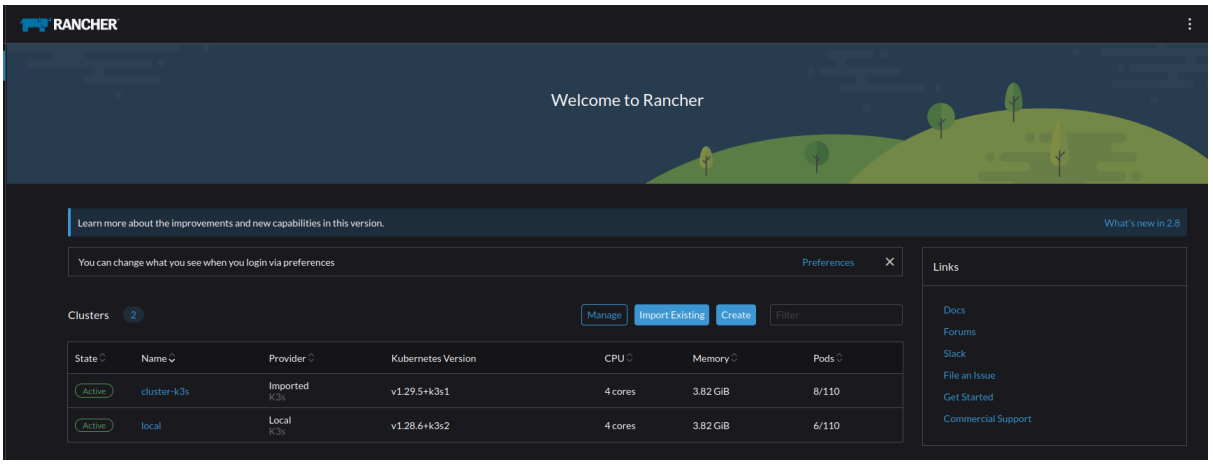



Por último, deberemos esperar unos segundos y en la misma interfaz que lo dejamos, refrescamos en el navegador y vemos que se ha conectado correctamente.




Con ésto, ya tendríamos creado e importado el **cluster de alta disponibilidad con K3s en Rancher**.



### 3.3.3. Grafana + Prometheus

Para la correcta instalación de Grafana y Prometheus, tendremos que instalar el chart de ambas herramientas mediante **Helm**. Para este proceso, tendremos que utilizar la terminal con la herramienta **Helm**. La instalación se hará en la máquina **Master**, por ello, instalaremos dicha herramienta ejecutando el script que descargamos.

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
$ sudo chmod 700 get_helm.sh
$ sudo ./get_helm.sh
```

Seguido de esto, nos descargamos el repositorio donde vamos a sacar los **charts** de Grafana y Prometheus para poder instalarlo. Para ello, utilizaremos la herramienta Helm (como superusuario):

```
$ helm repo add grafana https://grafana.github.io/helm-charts
$ helm repo add prometheus-community
https://prometheus-community.github.io/helm-charts
$ helm repo update
```

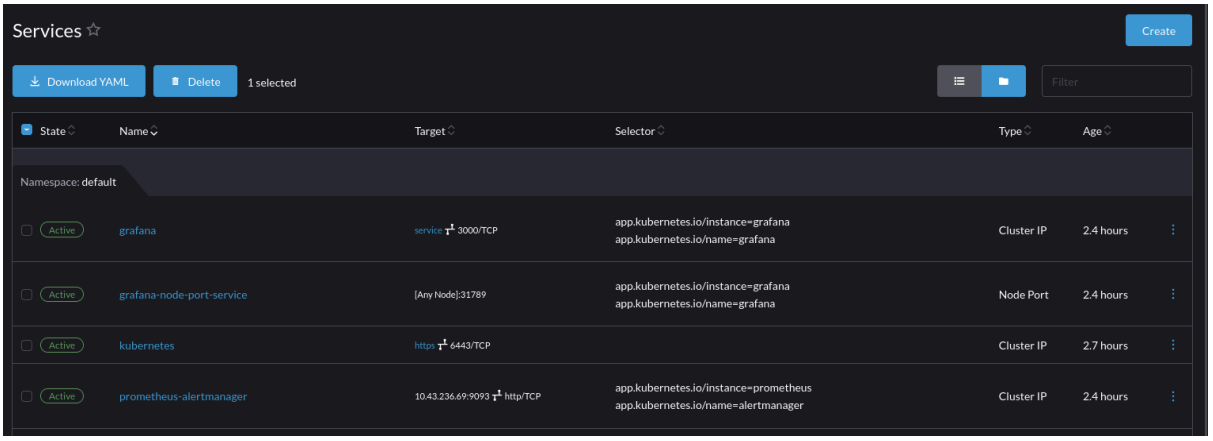
La instalación de las aplicaciones es bastante sencilla ya que disponemos de un solo comando para cada servicio para instalarlo que haciendo referencia a nuestro **kubeconfig** y al mismo repositorio, conseguimos instalarlo correctamente.

```
$ helm install prometheus prometheus-community/prometheus --kubeconfig
/etc/rancher/k3s/k3s.yaml
$ helm install grafana grafana/grafana --kubeconfig
/etc/rancher/k3s/k3s.yaml
```

Tras ésto, para las pruebas referentes a nuestras máquinas, crearemos 2 **services** para ver ambas aplicaciones mediante **NodePort**.

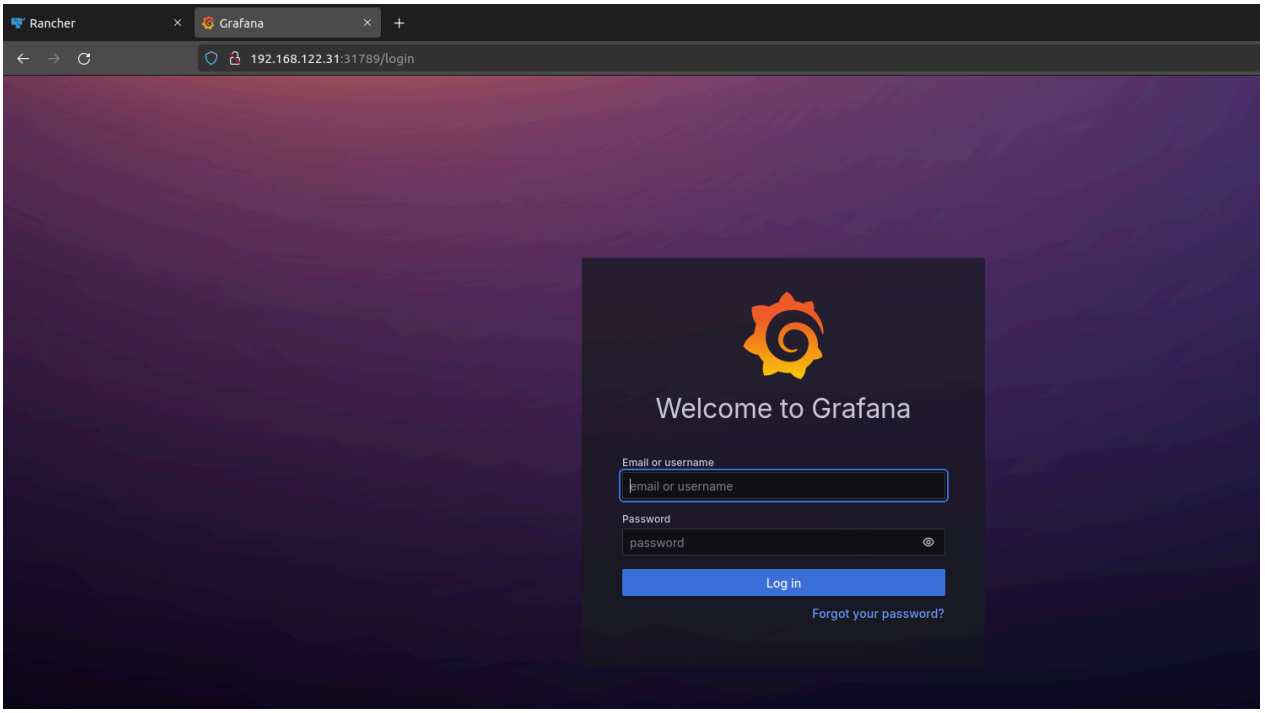
```
$ kubectl expose service prometheus-server --type=NodePort
--target-port=9090 --name=prometheus-server-ext
$ kubectl expose service grafana --type=NodePort --target-port=3000
--name=grafana-ext --name=grafana-node-port-service
```

Para poder observar las aplicaciones de manera correcta, tendremos que acceder a nuestra aplicación en Rancher en el apartado de **Services** y copiar el puerto de las aplicaciones. Para poder ver las aplicaciones de manera correcta, ponemos la ip del nodo y el puerto copiado anteriormente:




Y vemos que las aplicaciones están perfectamente instaladas 🙂.

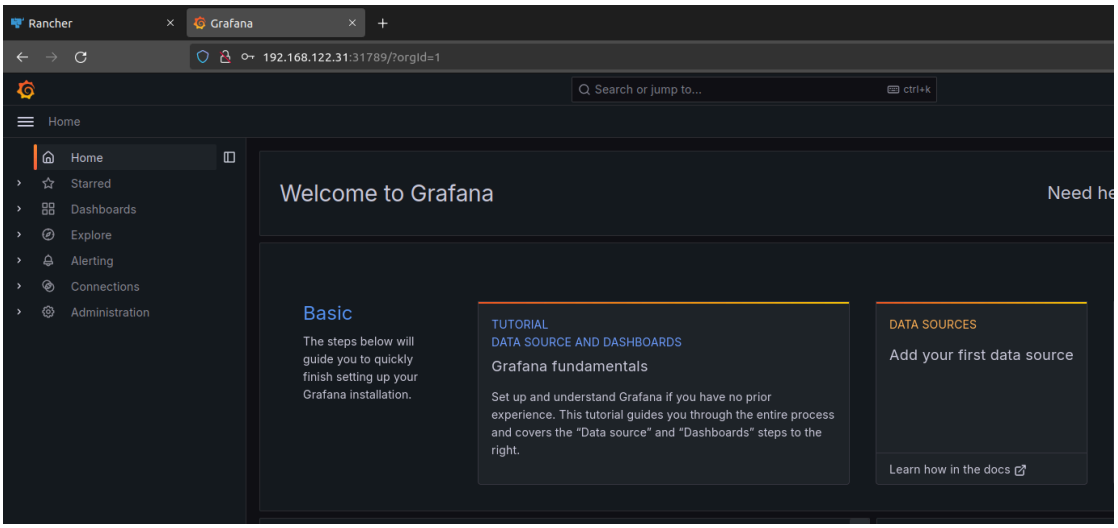
- Grafana:



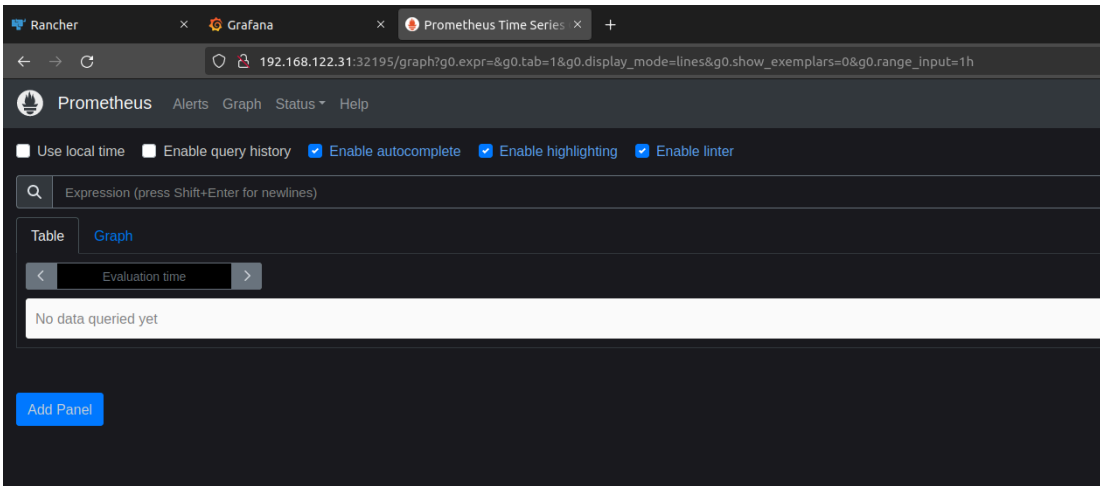
Para poder acceder a la aplicación, tenemos que obtener la **contraseña de administrador** y lo conseguiremos mediante este comando:

```
$ kubectl get secret --namespace default  grafana -o
jsonpath="{.data.admin-password}" | base64 --decode ; echo
```

Con éllo, copiamos la contraseña y con el usuario **admin** accederemos a la aplicación:




- Prometheus:




Para la **configuración y conexión de estas herramientas**, se hará en **apartado 3.4.2**.

### 3.3.4. Gitlab y Gitlab Runner

Para la correcta integración de Gitlab en nuestro clúster de K3s, tendremos que instalar la aplicación en nuestra máquina Gitlab dedicada. Para este proceso, primero instalamos la aplicación y posteriormente el Runner.

Para ello, **descargamos del repositorio de Gitlab oficial el script de instalación** para obtener este paquete **gitlab-ce** (aparte de instalar otros paquetes como curl, postfix para el correo... etc).

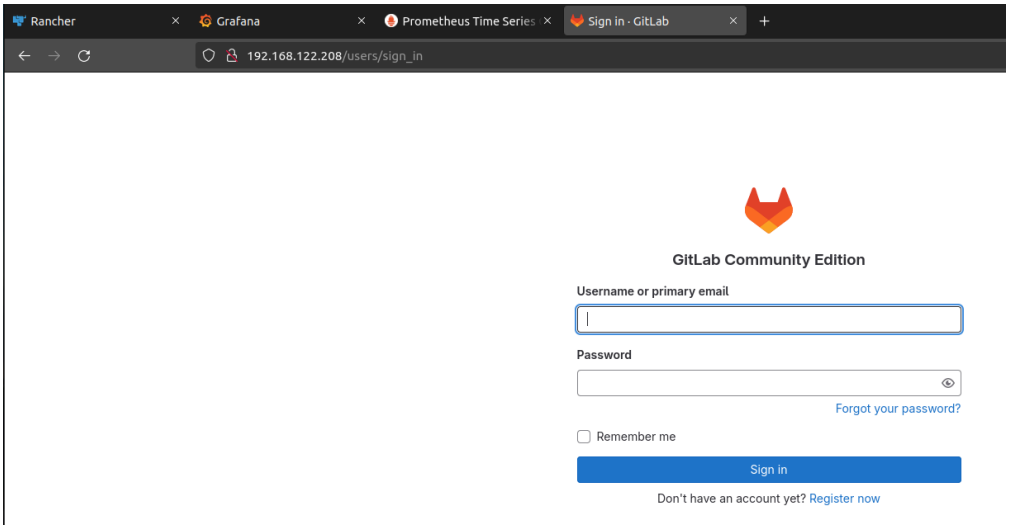
```
$ sudo apt-get update
$ sudo apt-get install -y curl openssh-server ca-certificates tzdata perl
$ sudo apt-get install -y postfix
$ curl
https://packages.gitlab.com/install/repositories/gitlab/gitlab-ce/script.deb
.sh | sudo bash
```

Cuando lo tengamos, pasamos a la instalación de la aplicación con este comando, especificando la IP externa de la máquina de gitlab ya que vamos a utilizar el protocolo **HTTP** sin nombre de dominio ninguno.

```
$ sudo EXTERNAL_URL="http://[ip_externa_máquina_gitlab]" apt-get install
gitlab-ce
```

Este proceso se puede demorar unos minutos ya que tiene que montar muchos servicios por detrás para poder desplegarlo.

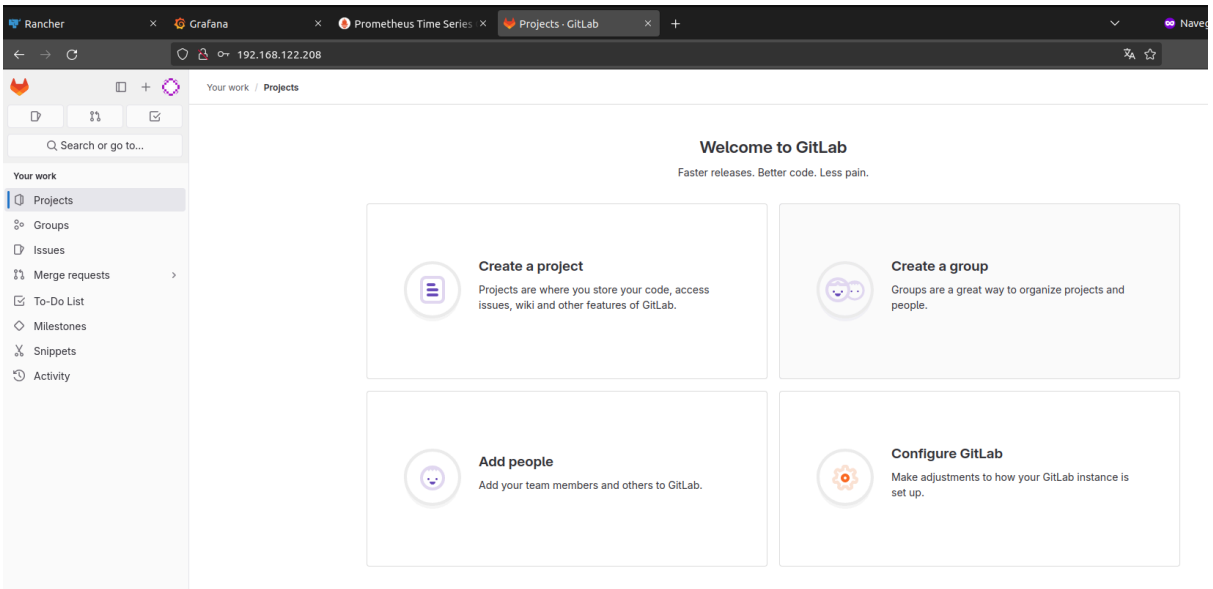
Cuando este finalice, accederemos a la URL indicada en la instalación y comprobaremos que se ha instalado correctamente.




Si queremos conocer la contraseña del usuario **root** con el acceso a la aplicación, tendremos que ver el contenido de este fichero:

```
$ sudo cat /etc/gitlab/initial_root_password
```

Cuando tengamos esta contraseña, con el usuario root accederemos a la aplicación de manera correcta y ordenada.

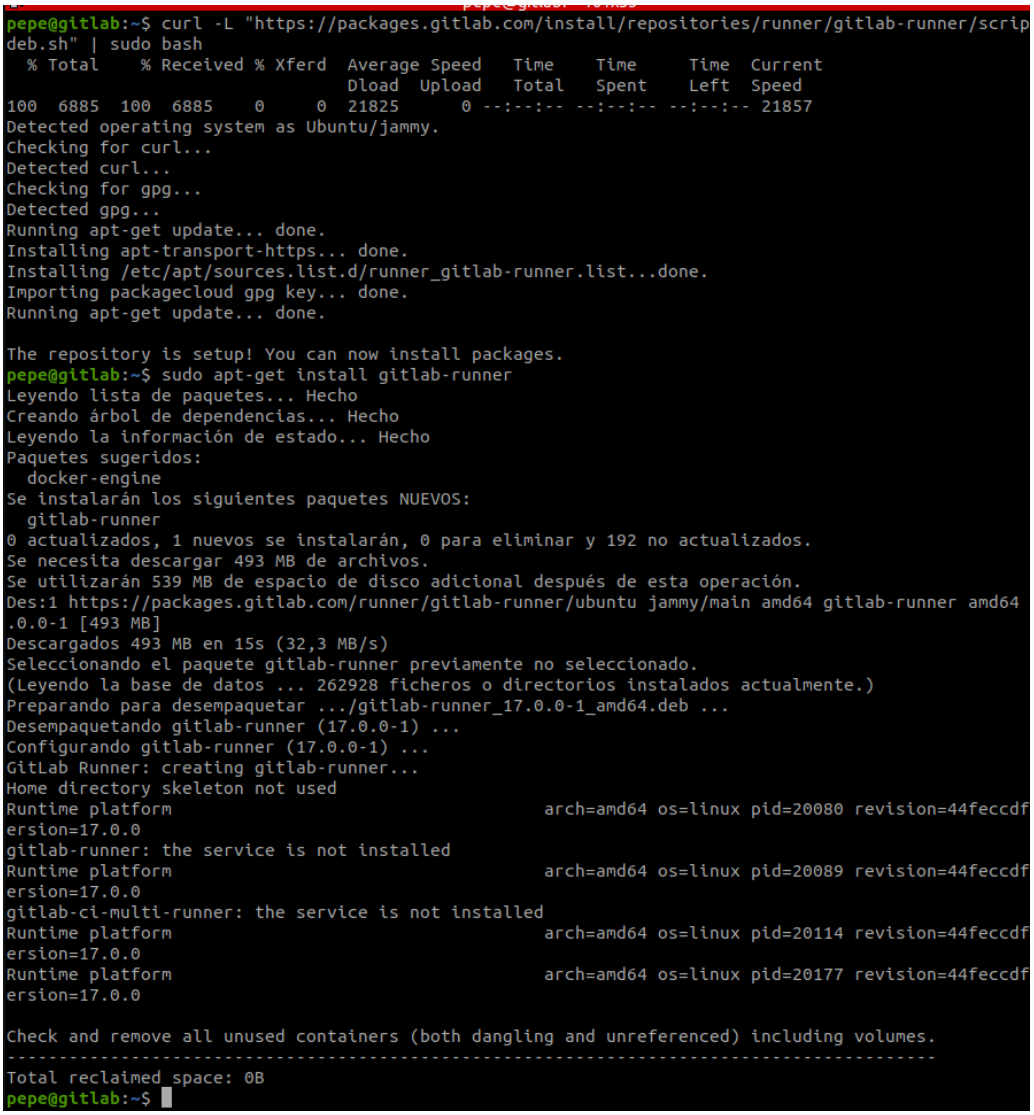



Con esto, ya tenemos desplegado **GitLab**.

Pasamos a la instalación de nuestro **Gitlab Runner**. Este runner estará instalado en la misma máquina de la aplicación para hacer la conexión a este mismo de manera rápida y sencilla.

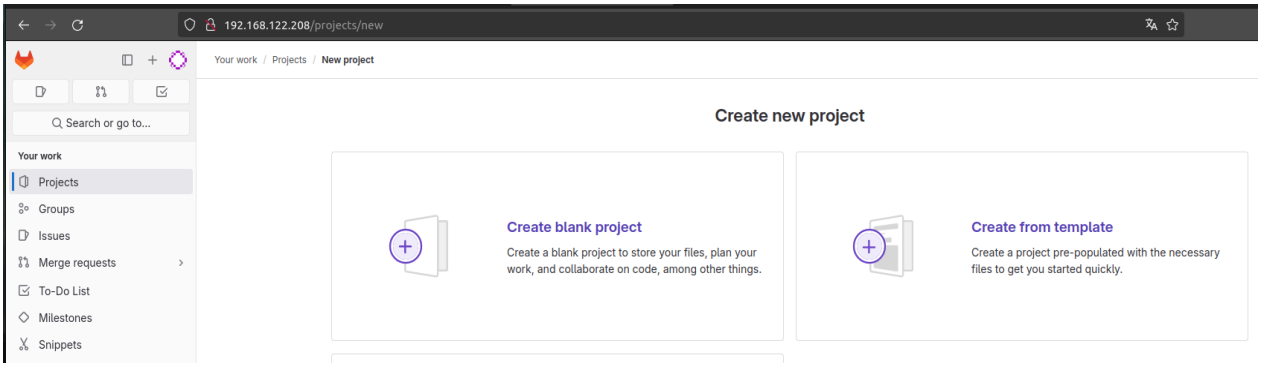
Por esto mismo, lo primero que deberemos realizar es descargar del repositorio de Gitlab oficial el script de instalación para el paquete **gitlab-runner.** Tras esto, instalamos el runner con nuestro gestor de paquetería correspondiente.

```
$ curl -L
"https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" | sudo bash
$ sudo apt-get install gitlab-runner
```

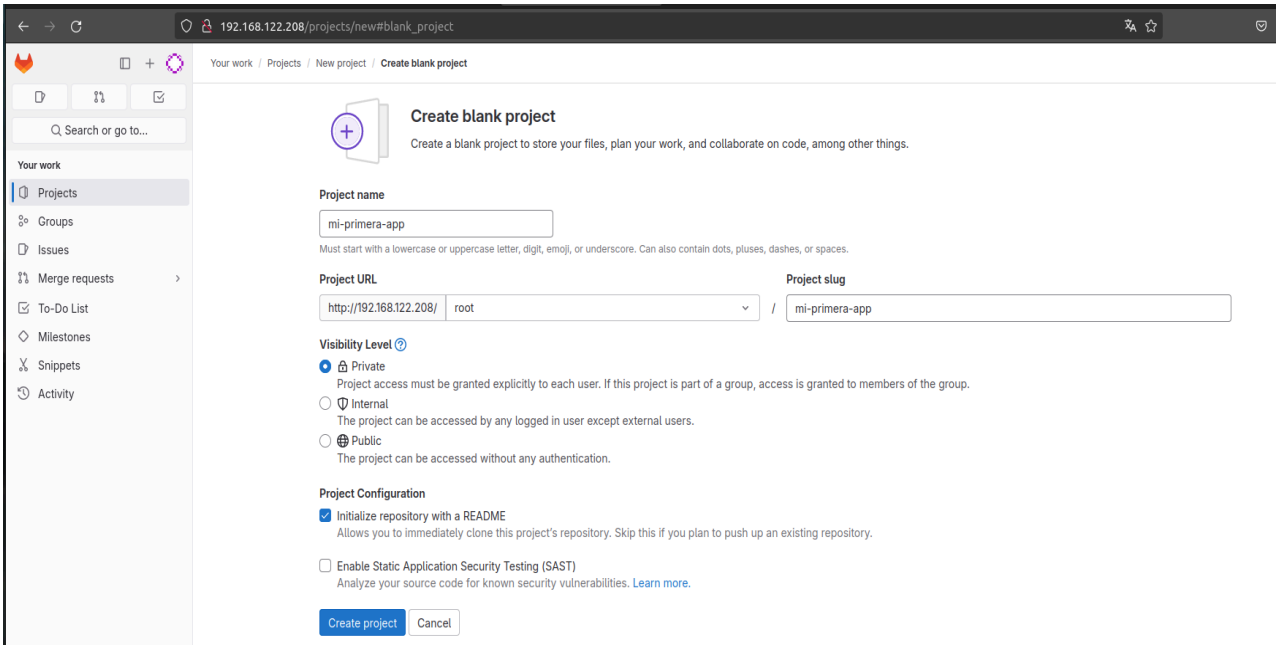


Este proceso de instalación puede tardar algunos minutos. Cuando se termine la instalación tendremos que crear el Runner mediante un **token de verificación**, pero primero, tendremos que crear un nuevo proyecto.

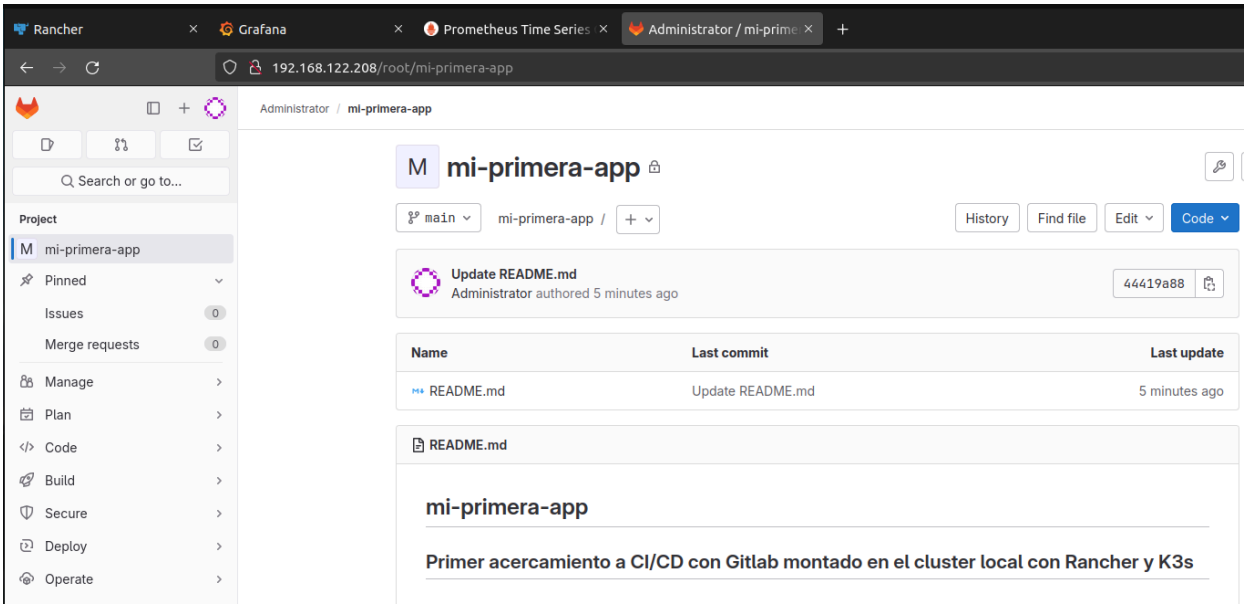
Primero, seleccionamos **Create a project** -> **Create blank project**



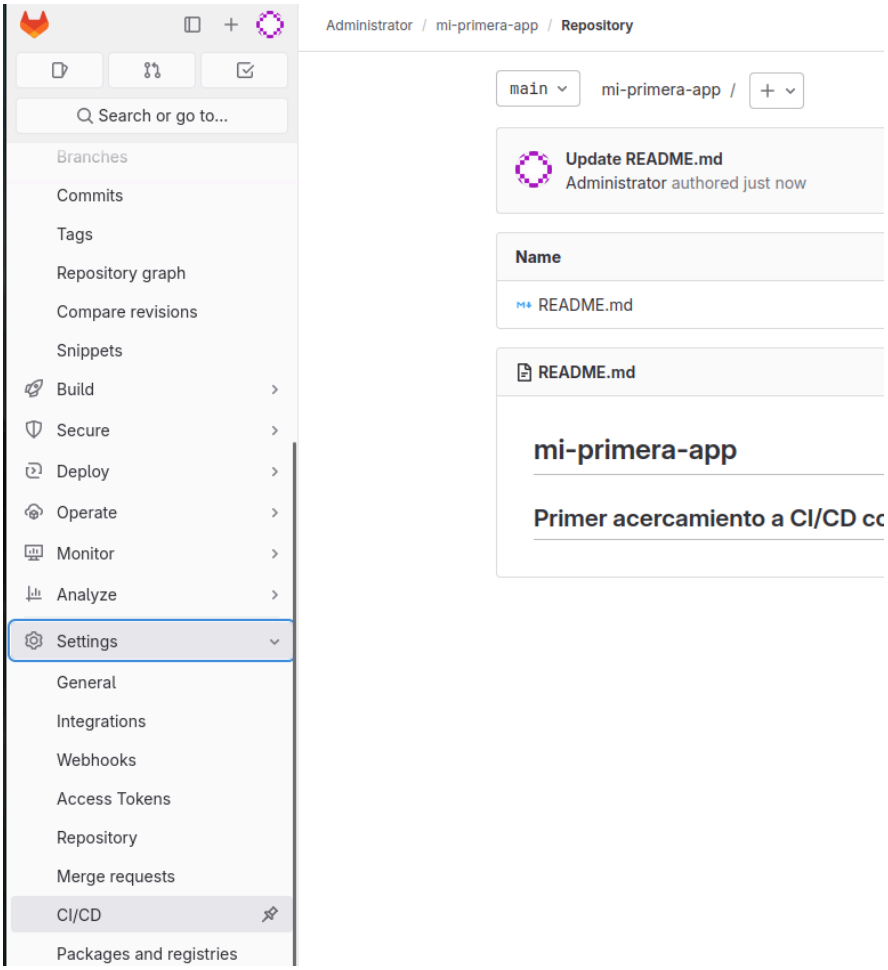

Seguido de esto, ponemos un nombre al proyecto y seleccionamos un **namespace** para el proyecto. Con esto comentado, creamos el proyecto.



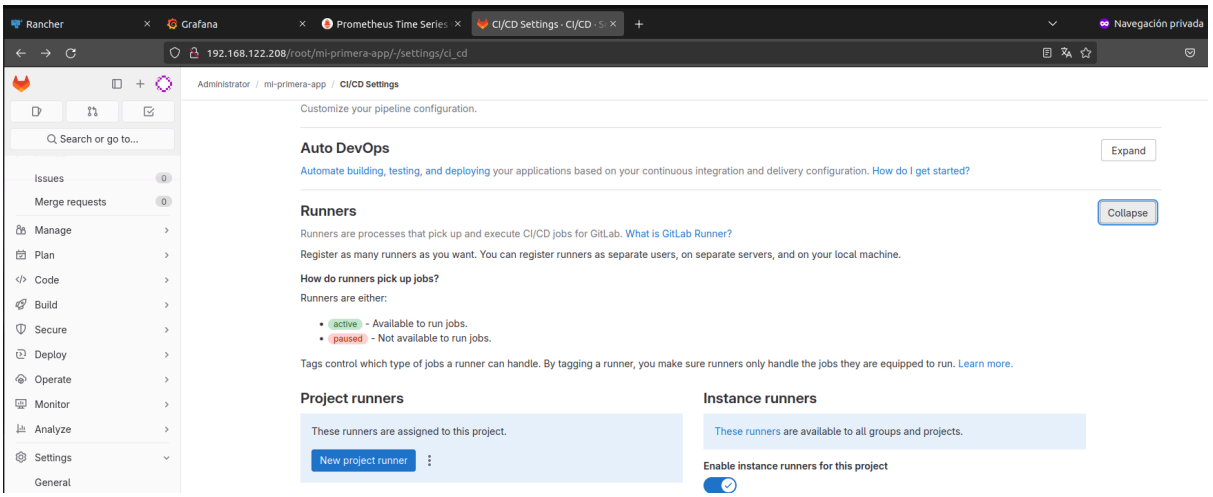

Cuando lo creemos, ya tendremos nuestro proyecto en blanco a la espera de poner nuestro contenido pero... ahora mismo esto no nos interesa.



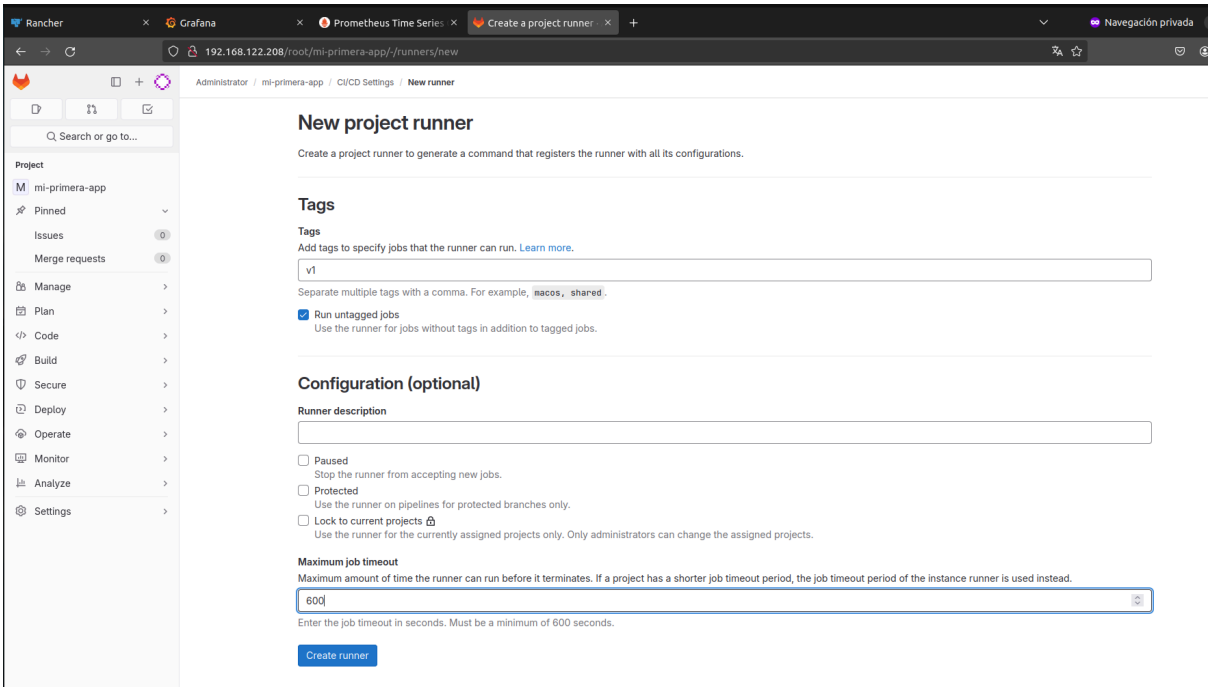

Pasamos a la importación del Runner; para ello, accederemos a **Settings -> CI/CD**.



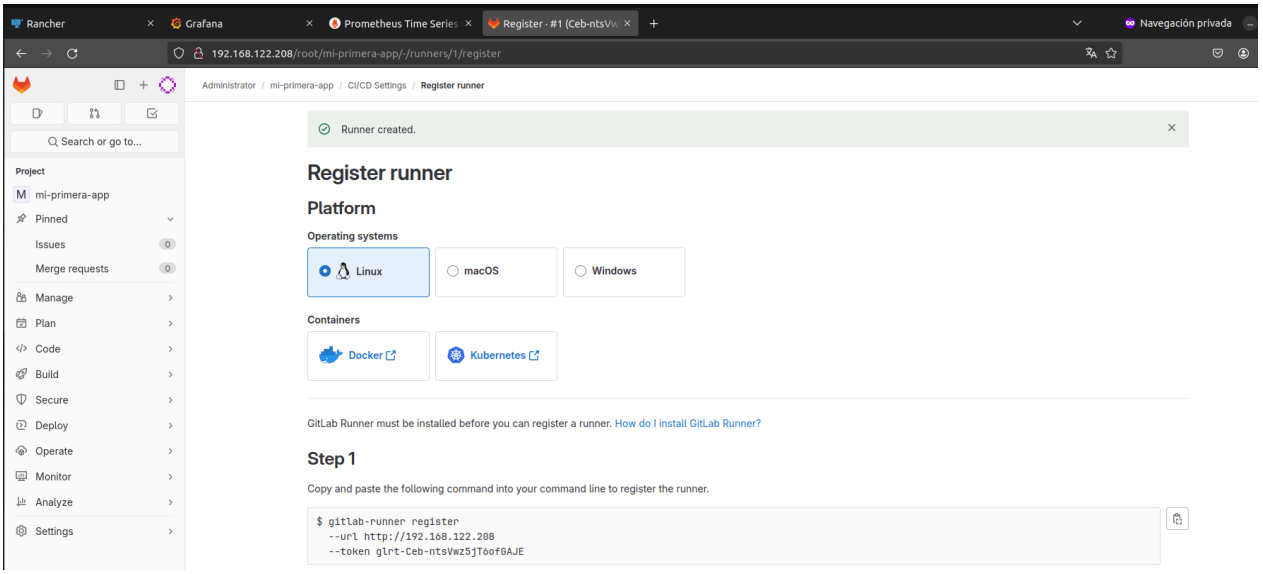
Clicamos en **Runners -> Expand** y se nos desplegará un contenido informativo sobre qué son los runners y cómo funcionan. Con esto comentado, creamos el Runner clicando en **New project runner.**




En la creación del Runner, tendremos que indicar el nombre, marcar que este Runner pueda lanzar el **pipeline** de los jobs que no están ***taggeados*** y ponemos 600 segundos como tiempo mínimo de **job**.



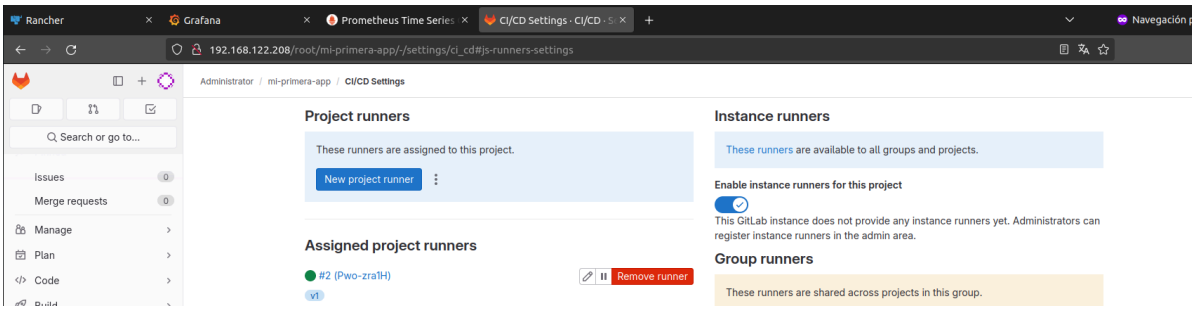

Cuando tengamos esta configuración lo creamos y nos mostrará la siguiente interfaz.




Como podemos ver, tendremos que correr el Runner mediante el token generado que tendremos que copiar. El comando en cuestión para correr dicho Runner es el siguiente:

```
$ sudo gitlab-runner register -n  --url http://[ip_externa_máquina_gitlab] --registration-token  [token_generado_gitlab] -executor docker --description "Deployment Runner" --docker-image "docker:stable" --tag-list deployment --docker-privileged
```

Le daremos a **View Runners** en nuestra aplicación (cuando ejecutemos el comando anterior), y veremos que sale el Runner en verde, es decir de manera activa.




Si queremos ver información sobre el Runner, miraremos el siguiente fichero -> **/etc/gitlab-runner/config.toml**
Con ésto, ya tendríamos instalado nuestro **Runner y conectado con Giitlab**.

## 3.4. Configuraciones

Para la correcta configuración del clúster, tendremos que tener en cuenta las siguiente configuraciones que son:

- **La propia configuración de este mismo**, en la cuál abordaremos el mantenimiento de las aplicaciones y el rendimiento de estas mismas.

- **La configuración y conexión entre de Grafana y Prometheus** para poder tener una monitorización de nuestro cluster en cuanto a rendimiento y disponibilidad.

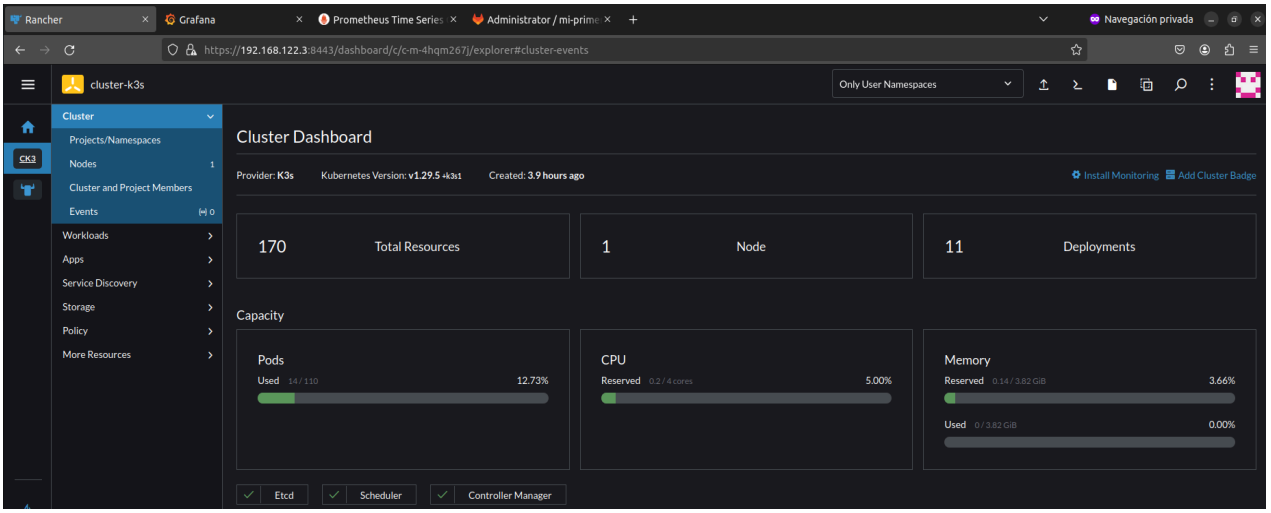
- **La configuración de Gitlab y su respectivo Runner** para realizar la integración continua de nuestra aplicación.

Dicho esto, comenzamos con las respectivas configuraciones.

### 3.4.1. Configuración del clúster

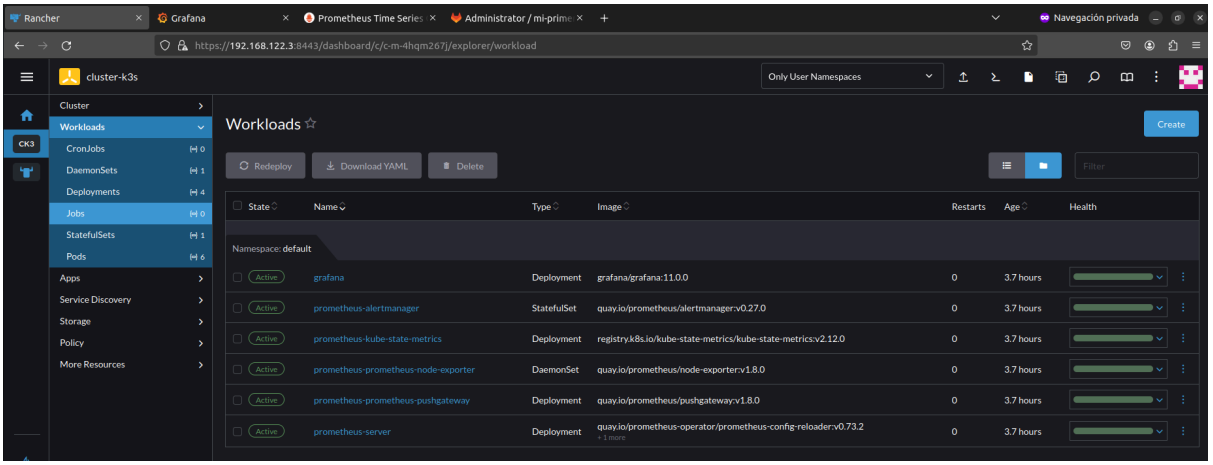
En este apartado, veremos los diferentes apartados que nos ofrece nuestro cluster, limitaremos el uso de nuestras aplicaciones y el uso del escalado de pods desde la interfaz gráfica.

Por ello, al acceder a nuestro cluster, podemos observar el uso de nuestra **CPU, RAM y la cantidad de pods** de nuestro nodo.

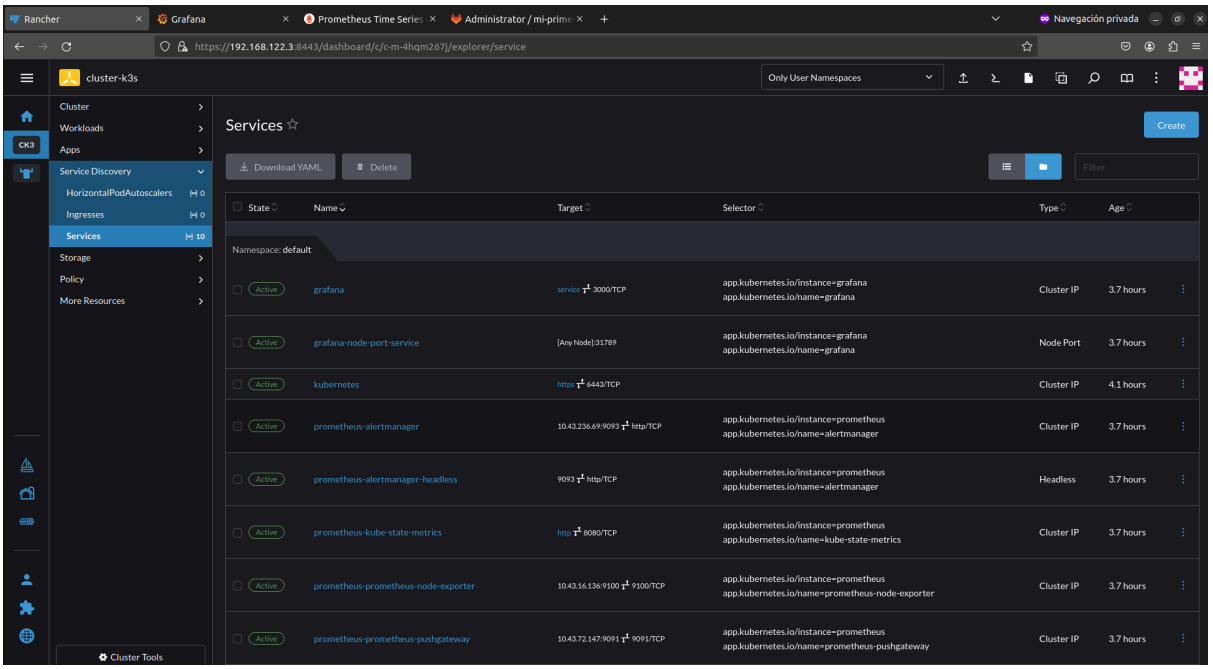


Como podemos observar, vemos que tenemos algunos apartados en los cuales se pueden ver proyectos o **namespaces**, estado de los nodos y los eventos sucedidos en este cluster.

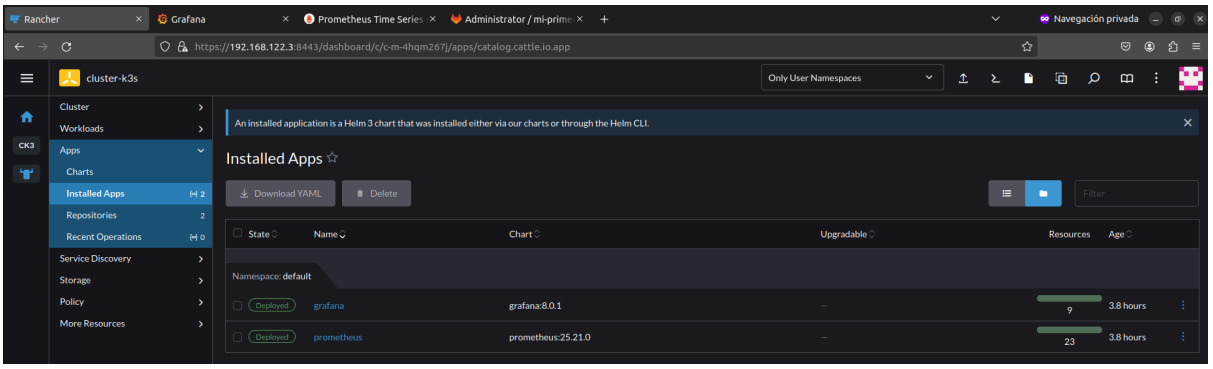
Posteriormente en el apartado de los **Workloads**, podemos ver que podemos configurar nuestros despliegues de las aplicaciones con **CronJobs**, **DaemonSets**, **Deployments**, **Jobs**, **StatefulSets** e incluso ver todos los **pods** correspondientes.



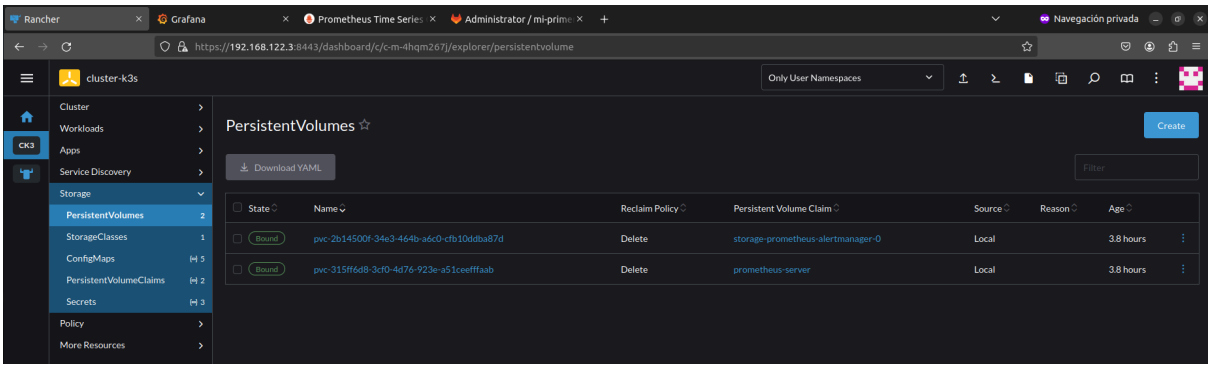

A continuación, podemos ver tanto las aplicaciones con sus respectivos **charts** y **repositorios**, como todo el tema de servicios **Ingress**, **Services** y los **HorizontalPodAutoscalers**.



Un dato importante es que la instalación de las aplicaciones la hemos realizado con Helm, por ello **los repositorios añadidos para nuestra instalación no se ven reflejados**. Estos mismos se encuentran en nuestro nodo master.



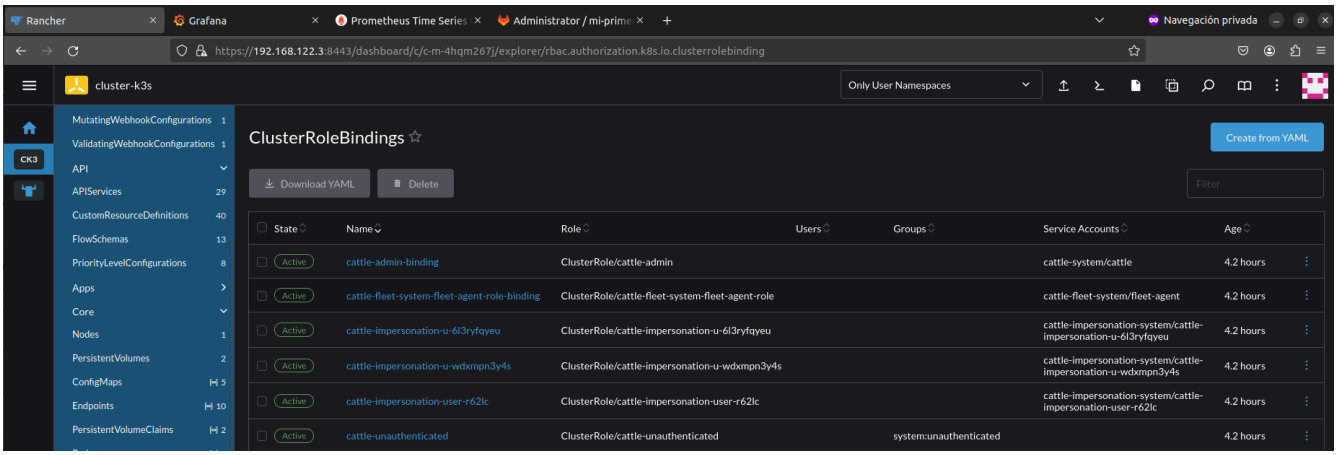

Un apartado también muy importante para el despliegue de nuestras aplicaciones es el almacenamiento o **Storage**. Este está compuesto lso **PersistentVolumes**, **StorageClasses**, **Configmaps** (donde se encuentran las variables de entorno), **PersistentVolumesClaims y Secrets** (donde se almacenan todas las contraseñas de manera encriptada) de nuestro clúster.




También una opción muy importante puede ser la **parte de políticas, seguridad y monitoreo**. En estos podemos ver los límites de rangos, políticas de seguridad de las redes, cuotas de almacenamiento y los pods de reserva para los deployments expuestos.

Además en el apartado de monitoreo, podemos controlar las alertas, los tipos de monitorización y la configuración avanzada de nuestras máquinas de Grafana y Prometheus.

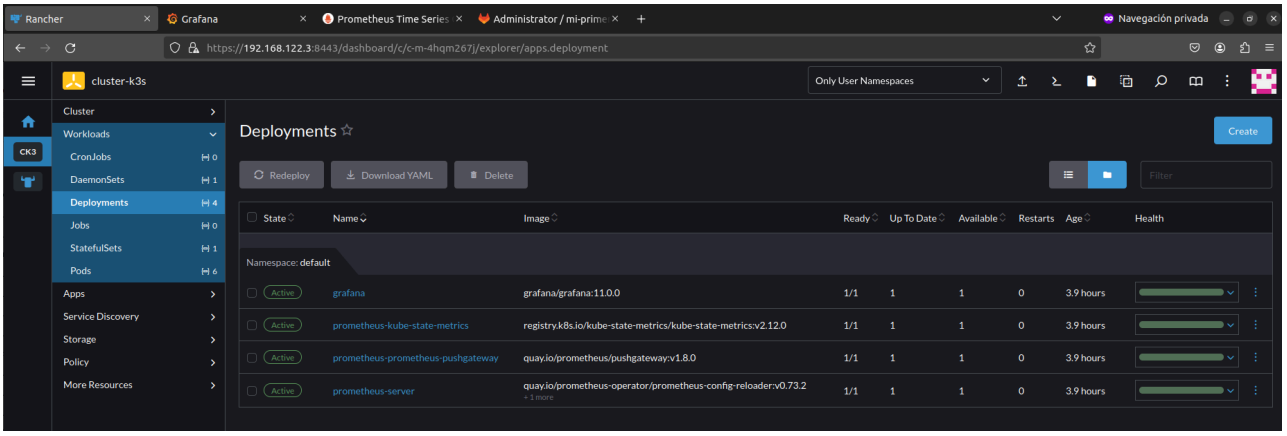
Por último, tenemos todos los demás recursos que nos ofrece Rancher que pueden ser **APIs**, **aplicaciones**, **autoescalado**, **redes**, **nodos**, **organizador de los eventos**, **RBAC...**



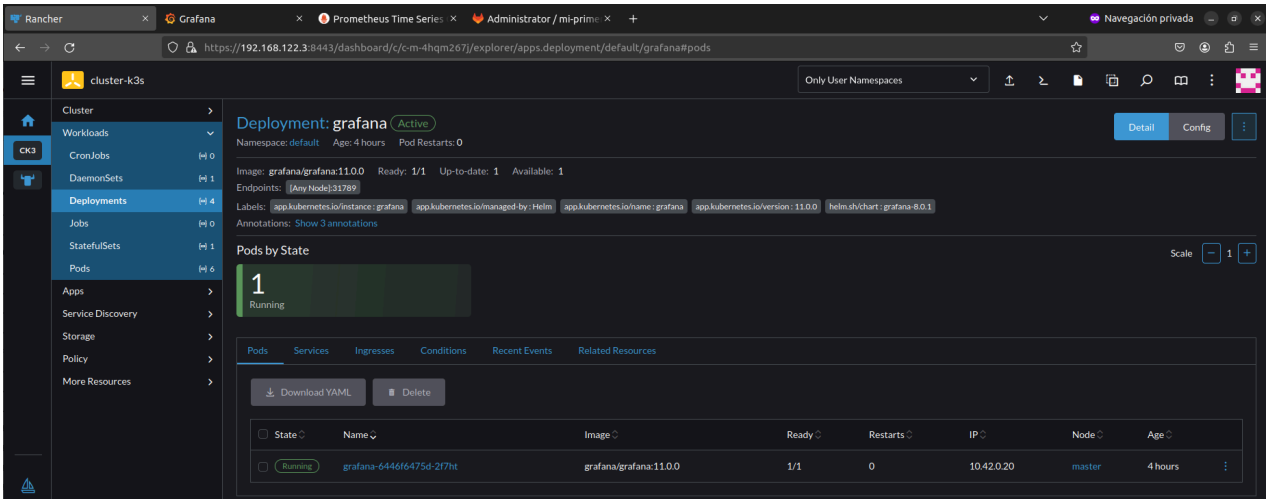

Nota: **RBAC** es un **mecanismo de control de acceso** que define los roles y los privilegios para determinar si a un usuario se le debe dar acceso a un recurso.

Tras esta breve introducción a la interfaz de Rancher según nuestro clúster, pasamos a ver cómo se pueden limitar nuestros recursos de nuestros **deployments** y como podemos **escalarlos** de manera gráfica.

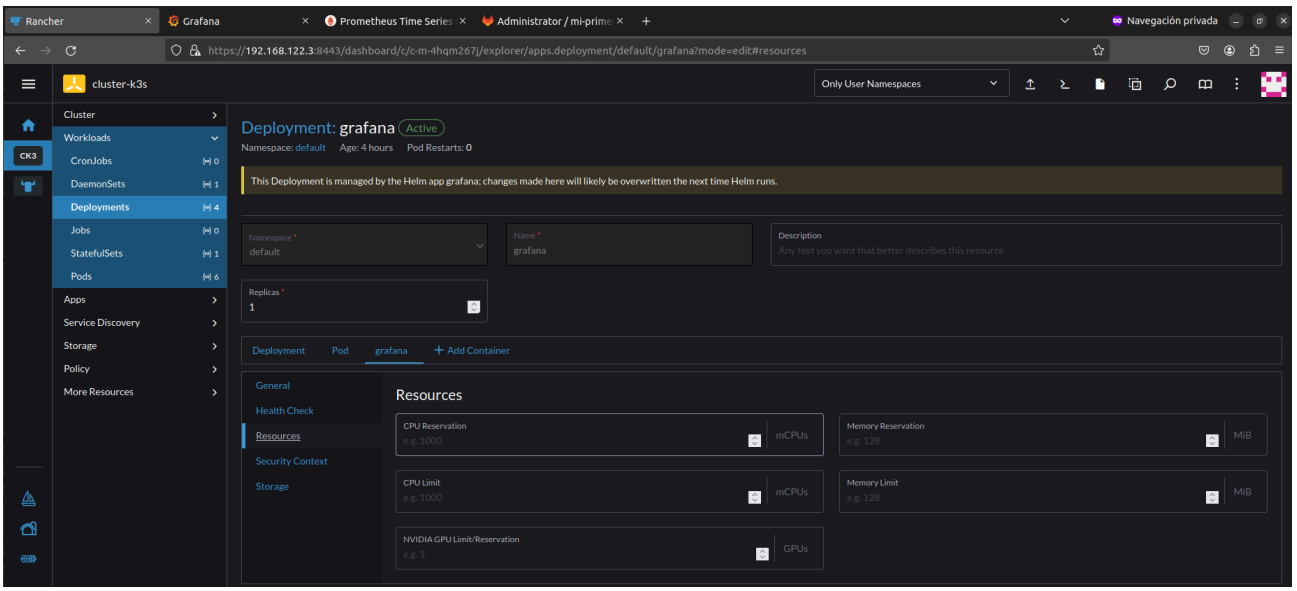
Para poder realizar estas 2 tareas, nos vamos al apartado de **Workloads -> Deployments** y dentro de este mismo podemos ver todos los despliegues de los distintos namespaces.



Elegimos un deploy que queremos, que en mi caso es el **grafana** y podemos ver que escalamos los pods mediante la interfaz gráfica:




Si queremos limitar el uso de recursos de un deployment, le damos a los 3 puntos y le damos a **Edit Config.** Tras esto, en el mismo servicio **grafana**, le damos a **Resources** y limitamos los recursos como queramos.



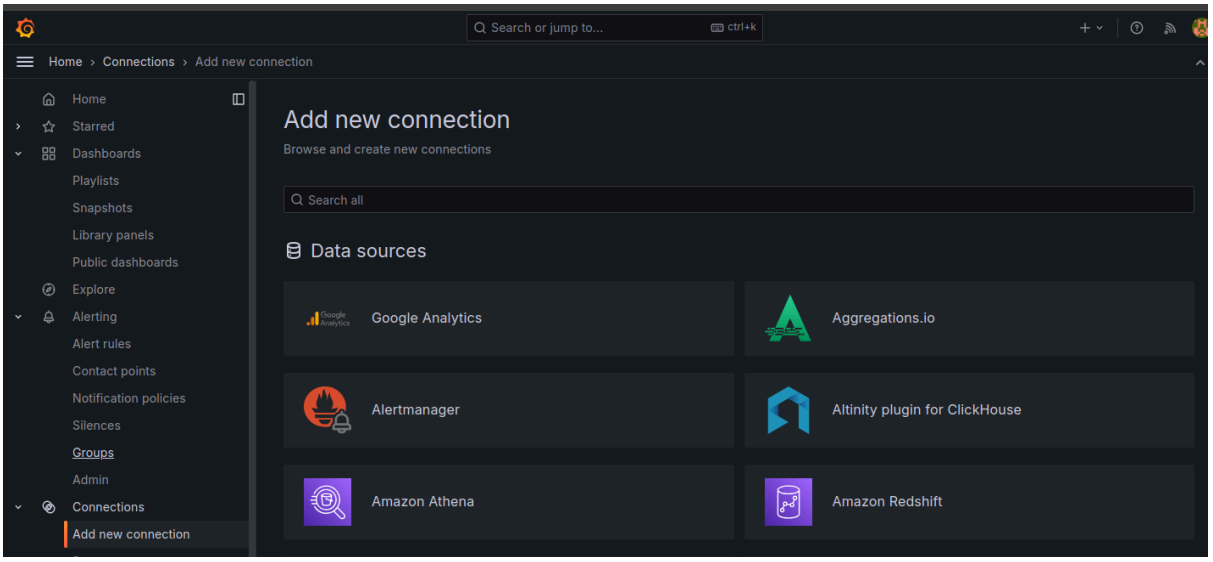

Tras hacer esto en algunos otros deployments con alta carga de trabajo y recursos. Tras un tiempo de espera para la reestructuración de los pods, **tendremos el cluster limitado sin la utilización de todos los recursos**.

## 3.4.2. Configuración de métricas con Grafana + Prometheus

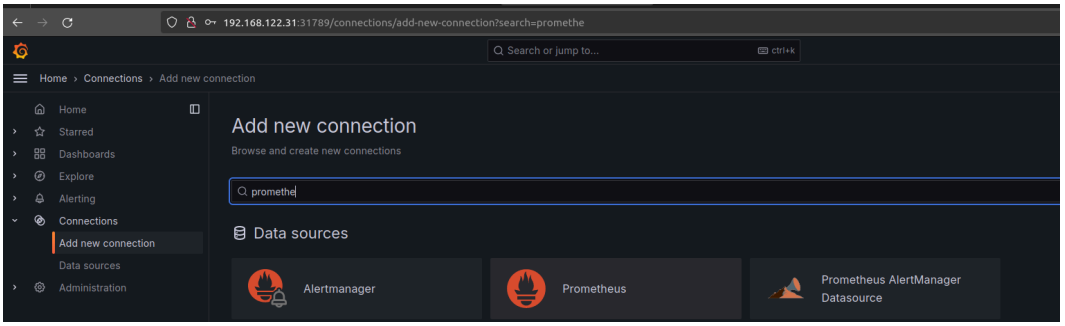
Pasamos a la configuración y conexión entre Grafana y Prometheus. Como tenemos expuesto los servicios mediante **NodePort** podemos ver los servicios creados.

Para la conexión entre las máquinas de **Grafana** y **Prometheus** , tendremos que añadir una nueva conexión mediante la interfaz gráfica.

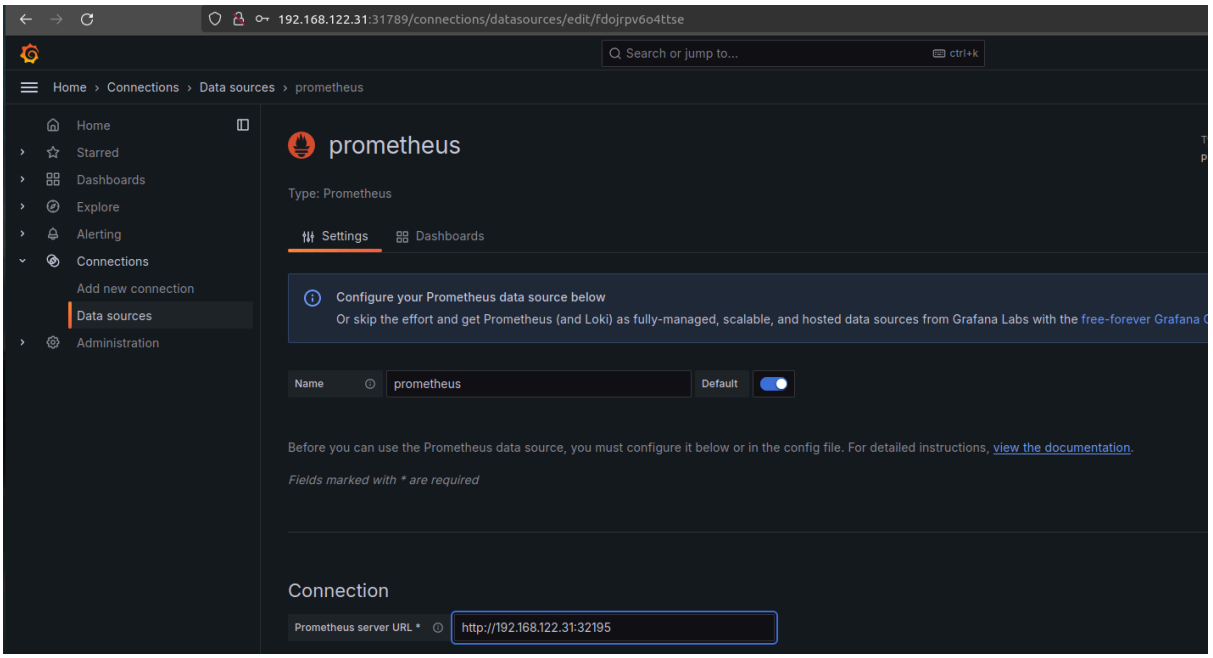
Por ello, accederemos a Grafana y podemos ver que tiene distintos apartados como los **Dashboards de las métricas**, las **alertas** de notificación mediante un límite de cuotas o reglas, las diferentes conexiones para añadir como **Azure**, **AWS**, **GitHub**, **GitLab**, **Oracle**, **PostreSQL**...




En el mismo apartado de conexiones, buscamos Prometheus y clicamos en él para añadir una nueva conexión.

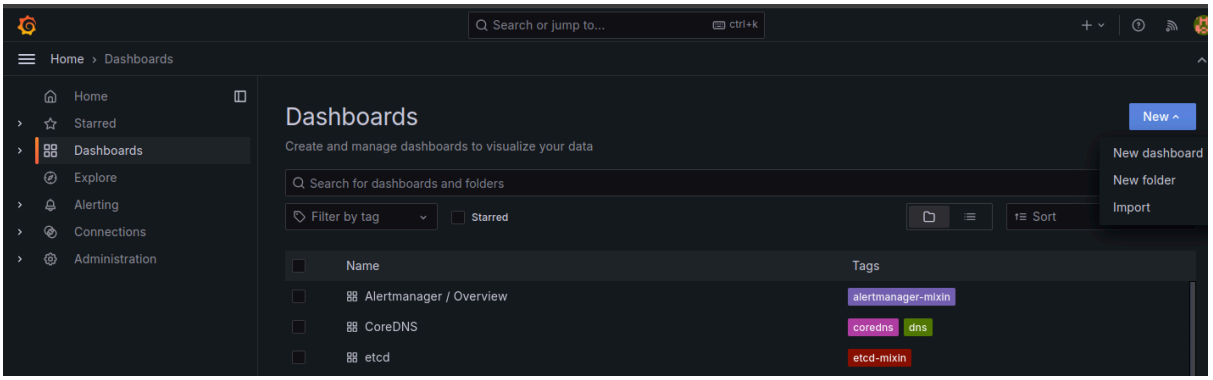


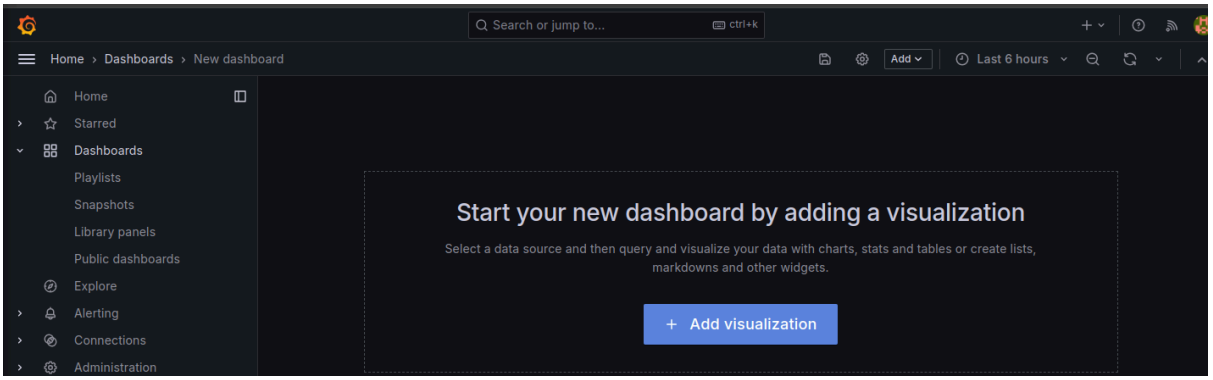
Cuando accedemos, tendremos que ingresar la URL de nuestro servicio de Prometheus con el respectivo **NodePort**, ya que se encuentran en la misma máquina:



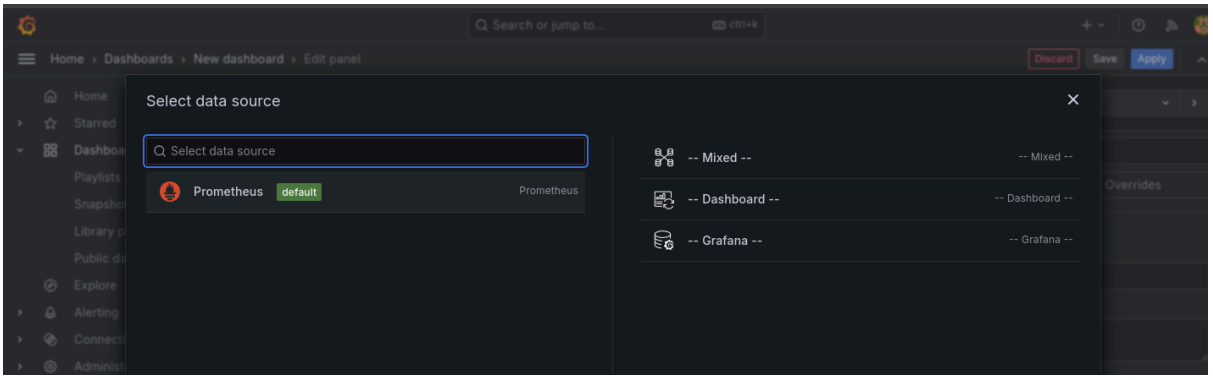

Tras esto, bajamos hasta donde pone **Save and Test** y guardamos la conexión con nuestra máquina Prometheus.

Con esto comentado, pasamos a crear el **Dashboard** con las métricas respectivas para hacer ver una interfaz bonita para el monitoreo de métricas. Por ello, nos vamos a este apartado y creamos un nuevo Dashboard y añadimos una nueva visualización:

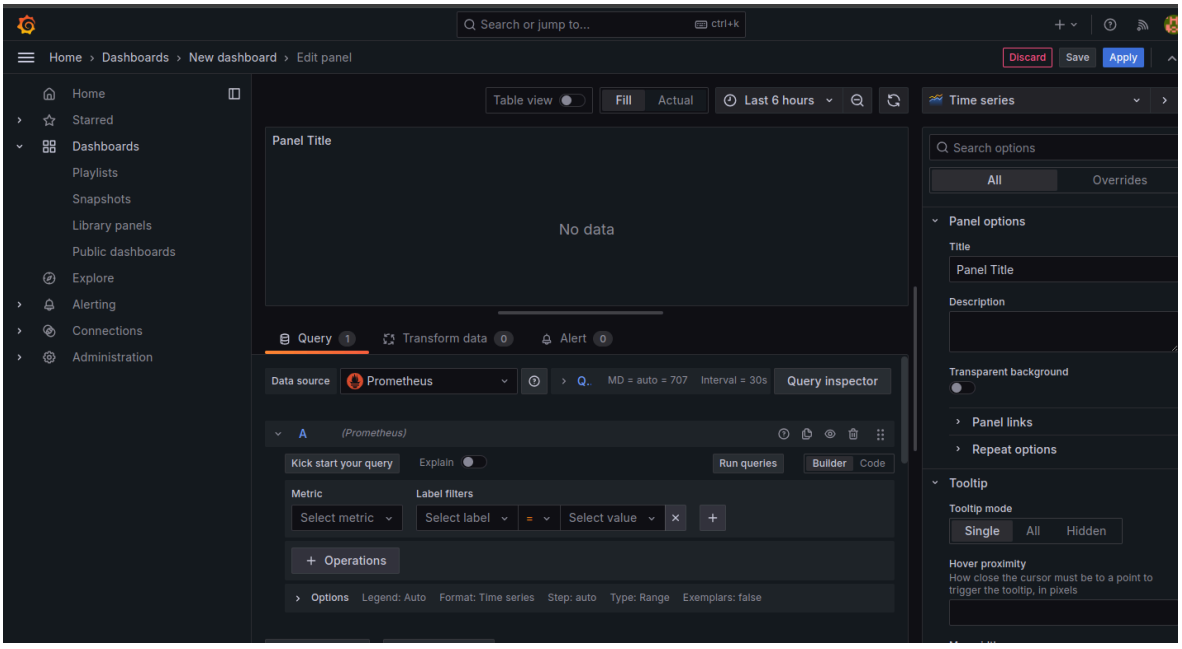





Seguido de esto, apuntamos a nuestro Prometheus por defecto ya viene configurado en nuestro stack de Helm.



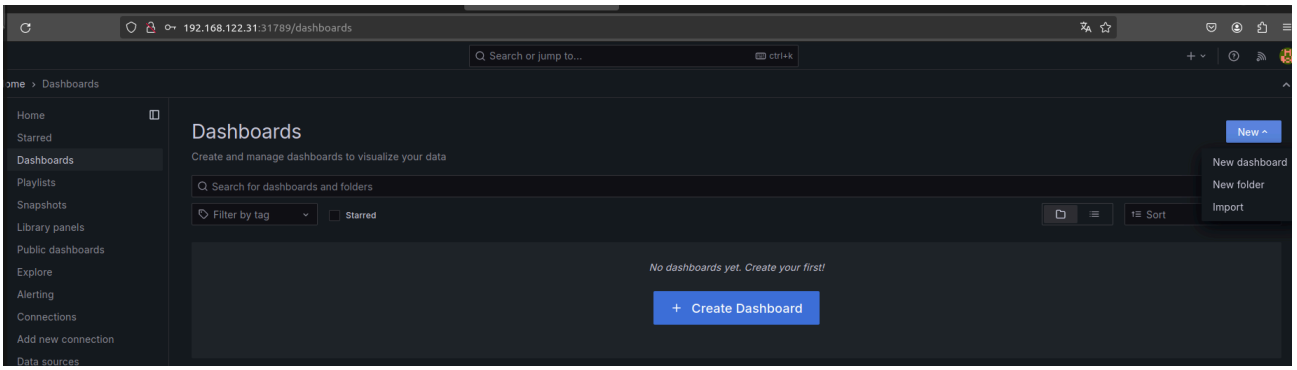

Cuando accedemos, tendremos la interfaz la cual podremos configurar las métricas de **Prometheus** mediante una interfaz gráfica.



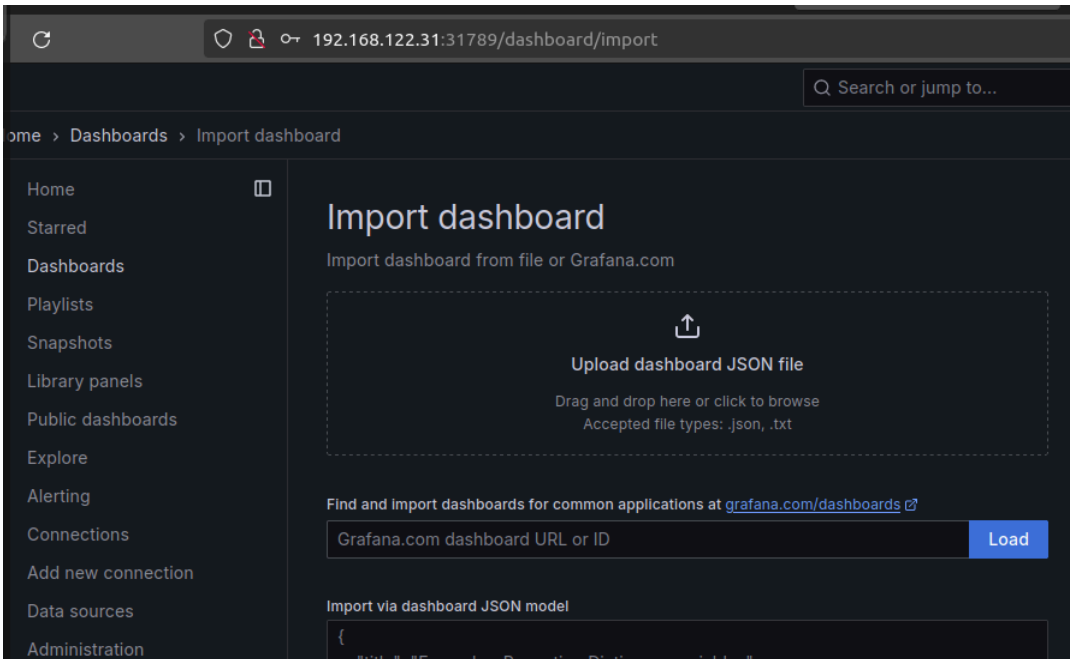
Algunas de las métricas están relacionadas con el **uso de CPU, el consumo de bytes para los procesos, el almacenamiento de nuestro cluster...**

También en el apartado Dashboards, podemos importar Dashboards con información referente a nuestro cluster de manera más rápida y sencilla.

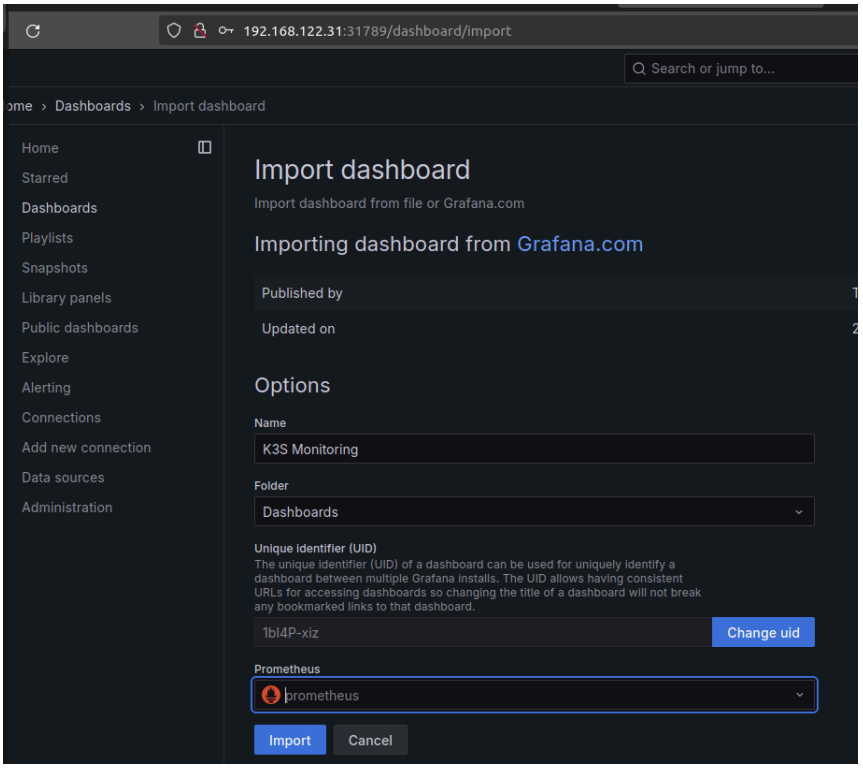
Para esto, accederemos a estos mismos y le damos a **New -> Import** e importamos la plantilla que queramos.



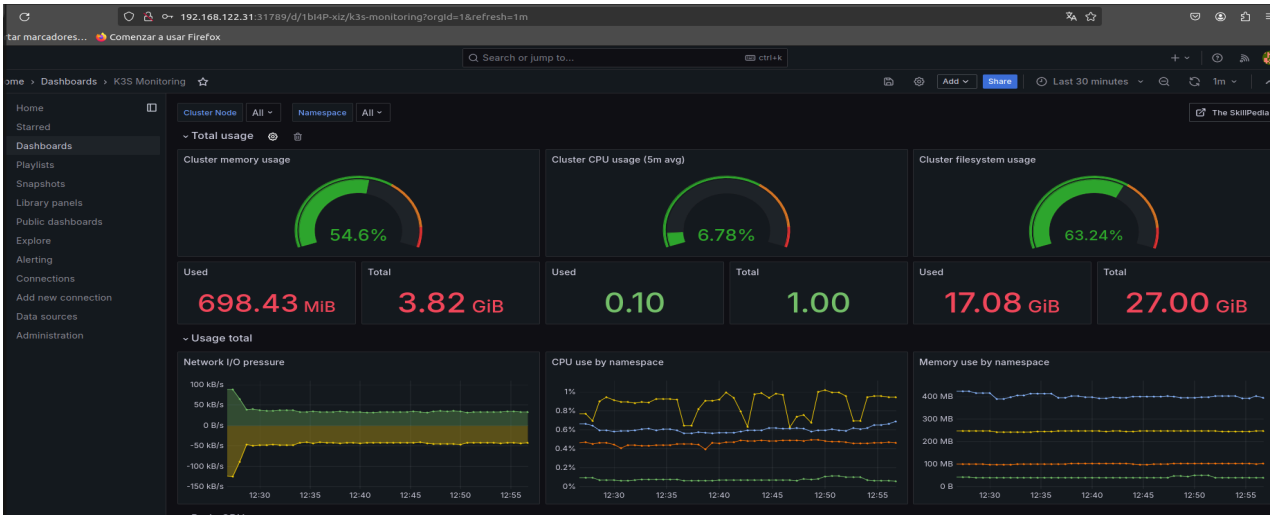

Para poder importar un Dashboard correctamente, indicamos un **ID** un import mediante un formato **JSON** de alguien de la comunidad que tenga métricas para nuestro cluster de K3S.

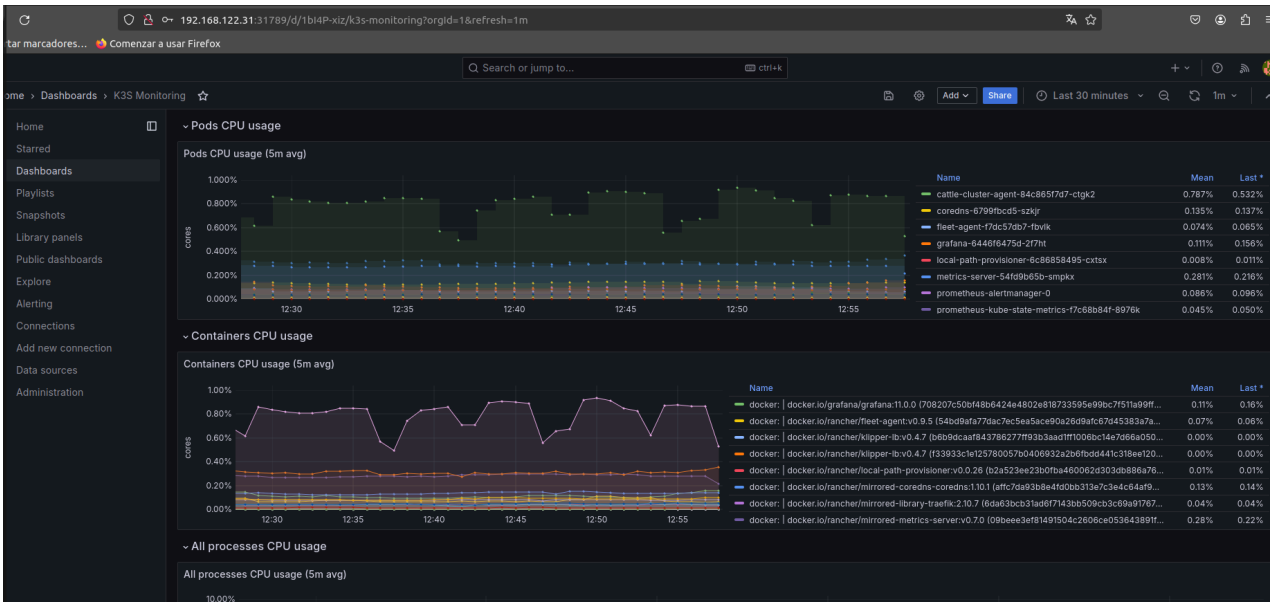


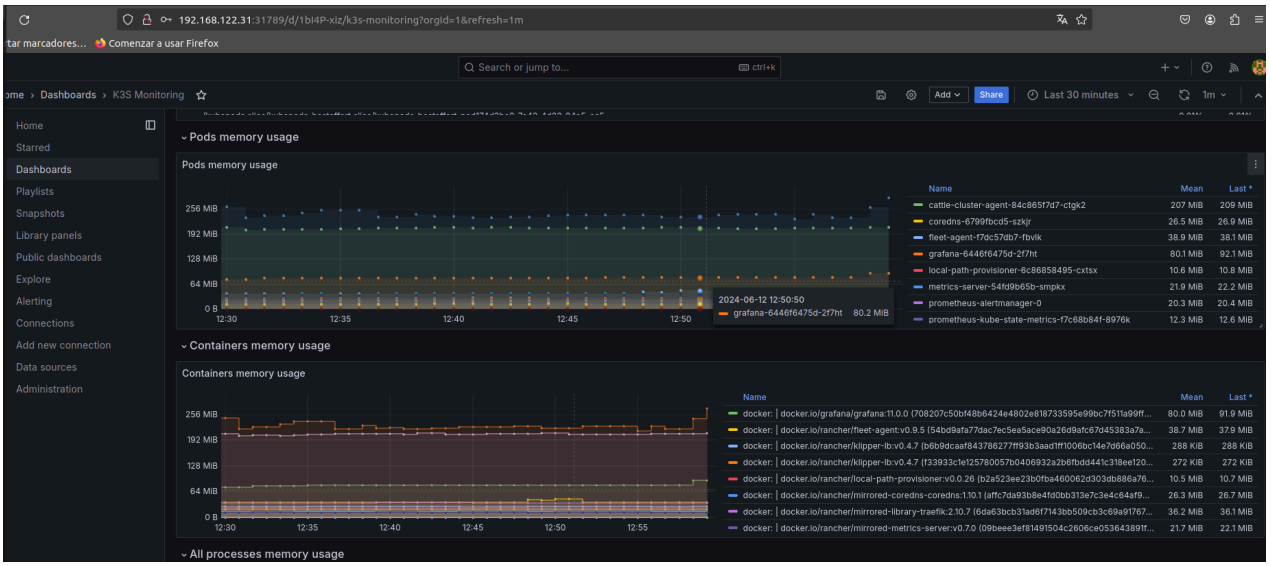
Un ejemplo puede ser **19972,** el cual hace referencia al cómputo de recursos que tiene nuestro cluster de K3s. Cuando le demos a **Load**, nos pedirá que indiquemos el **resource** donde obtendremos las métricas (el cual, por defecto, es la conexión Prometheus de nuestro cluster).




Para finalizar, lo importamos y accedemos a este mismo mediante Dashboards y veremos que tiene todo tipo de información sobre nuestro cluster.






Desde el **uso de nuestra CPU, RAM y el almacenamiento ocupado**, hasta el **uso de CPU y RAM de los pods, los procesos de CPU y RAM** que se están ejecutando...

Con esto, **podemos ver la integración de nuestro cluster de manera rápida con las métricas mediante Grafana y Prometheus de nuestro cluster desplegado mediante Rancher + K3s.**

### 3.4.3. Despliegue y configuración del Runner en Gitlab

Ya con **Gitlab** y **Runner** instalados, pasamos a la creación del **Registry privado** donde almacenaremos las imágenes generadas por nuestro Runner cada vez que se dispare. Para ello, crearemos dicho Registry privado en la máquina manager como hemos comentado al principio de la documentación.

Primero, crearemos el entorno de trabajo antes de ponernos con la creación del escenario mediante **docker compose**. Los siguientes pasos que realizaremos, los haremos como root o superusuario.

```
$ mkdir ~/private-registry
$ mkdir ~/private-registry/registry-data
```

Accederemos al directorio del mismo y crearemos el fichero **docker-compose.yaml** donde crearemos el respectivo contenedor.

```
$ cd ~/private-registry
$ nano docker-compose.yml

version: '3'

services:
  registry:
      image: registry:latest
      ports:
      - "5000:5000"
      environment:
      REGISTRY_AUTH: htpasswd
      REGISTRY_AUTH_HTPASSWD_REALM: Registry
      REGISTRY_AUTH_HTPASSWD_PATH: /auth/registry.password
      REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY: /registry-data
      volumes:
      - ./auth:/auth
      - ./registry-data:/registry-data
```

Este archivo **docker-compose.yaml** configura un servicio de registro privado de contenedores Docker usando la imagen **registry:latest**. Además, expone el puerto 5000 del contenedor al host, permitiendo el acceso al registro a través de http://localhost:5000 (en este caso sería **http://[ip_ext_máquina_manager]:5000**).

Implementamos la autenticación básica usando un archivo de contraseñas ubicado en **./auth/registry.password** y almacena las imágenes de contenedores en **./registry-data**, proporcionando persistencia de datos.

Tras esto, desplegamos el escenario con el siguiente comando:

```
$ docker compose up -d
```

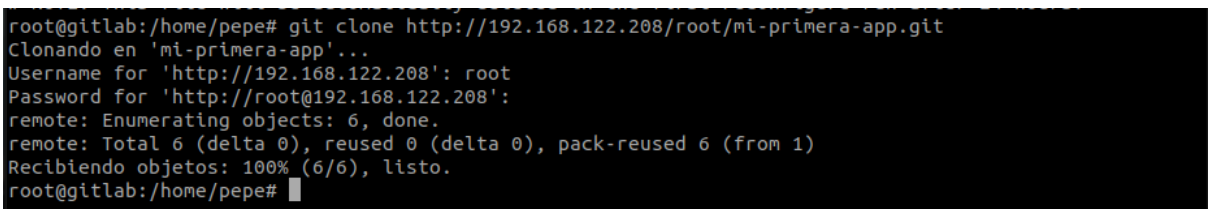
Con esto, ya tendríamos nuestro Registry privado pero antes, deberemos cambiar la contraseña del usuario de este mismo ya que nos lo genera por defecto cuando levantamos el contenedor de **Docker** mediante el comando **htpasswd**.

```
$ apt install apache2-utils (si no está instalado)
$ htpasswd -Bc auth/registry.password admin
```

Nos pedirá una contraseña que tendremos que acordarnos para el **registry** que crearemos con **Rancher** posteriormente. Seguido de esto, pasamos a la máquina de gitlab, la cual tenemos el proyecto, y clonamos el repo en la máquina en local.

```
$ git clone http://[ip_ext_máquina_manager]/root/mi-primera-app.git
```

Para poder clonarnos el proyecto, deberemos ingresar el usuario y la contraseña al acceder a gitlab. El usuario es **root** y la contraseña se puede obtener en el directorio **/etc/gitlab/initial_root_password.**




Con esto, ya tendríamos clonado el proyecto. Con el repo copiado, añadimos los archivos en cuestión que son los siguientes:

- **index.html:** una plantilla de ejemplo para ver el funcionamiento de la integración continua.

```
$ nano index.html

<!DOCTYPE html>
<html lang="es">
<head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <title>Página de Ejemplo</title>
      <style>
      body {
            display: flex;
            justify-content: center;
            align-items: center;
            flex-direction: column;
            height: 100vh;
            margin: 0;
            background-color: #f0f0f0;
            font-family: Arial, sans-serif;
      }
      h1, h2 {
            color: #8c52ff;
            text-align: center;
            border: 2px solid #8c52ff;
            padding: 20px;
            border-radius: 10px;
            background-color: white;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
            margin: 10px;
      }
      h2 {
            color: #21ef80;
            border: 2px solid #21ef80;
      }
      </style>
</head>
<body>
      <h1>Prueba de funcionamiento CI/CD mediante Gitlab con Rancher +
K3S</h1>
      <h2>Realizado por Jose Carlos Rodríguez Cañas</h2>
</body>
</html>
```

- **Dockerfile:** donde copiaremos el fichero **index.html** que hemos modificado en un imagen por defecto de servidor web. En mi caso es una imagen de **nginx**.

```
$ nano Dockerfile

FROM nginx
COPY index.html /usr/share/nginx/html
```

- **.gitlab-ci.yml**: fichero que lee el runner para poder ejecutar el pipeline.

```
$ nano .gitlab-ci.yml

stages:
  - build-containers
  - deploy

variables:
  DOCKER_IMAGE: "192.168.122.3:5000/mi-primera-app:v${CI_PIPELINE_ID}"
  K8S_NAMESPACE: "default"
  K8S_DEPLOYMENT: "mi-primera-app"

docker_build:
  stage: build-containers
  image:
      name: gcr.io/kaniko-project/executor:debug
      entrypoint: [""]
  before_script:
      - cp "$KANIKO_SECRET" /kaniko/.docker/config.json
  script: >
      /kaniko/executor --context "${CI_PROJECT_DIR}" --dockerfile
"${CI_PROJECT_DIR}/Dockerfile" --destination "${DOCKER_IMAGE}"

deploy:
  stage: deploy
  image:
      name: bitnami/kubectl:latest
      entrypoint: [""]
  before_script:
      - cp "$KUBECONFIG_SECRET" /.kube/config

  script:
      - kubectl -n $K8S_NAMESPACE set image deployment/$K8S_DEPLOYMENT
$K8S_DEPLOYMENT=$DOCKER_IMAGE
```

Como podemos ver, el fichero **.gitlab-ci.yml** es el fichero más importante de nuestro, es por ello, que explicaré su debido funcionamiento.

Este archivo **gitlab-ci.yml** configura una pipeline de CI/CD en GitLab con dos etapas principales: c**onstrucción de contenedores y despliegu**e. A continuación, explicaré cada sección del archivo:

- **Etapas**:

  - **build-containers**: Esta etapa es para la construcción de contenedores Docker.
  - **deploy**: Esta etapa es para desplegar la aplicación en Kubernetes.

- **Variables**:

  - **DOCKER_IMAGE**: Definimos la imagen Docker con la dirección del registro (192.168.122.3:5000) y la etiqueta basada en el ID del pipeline (v${CI_PIPELINE_ID}).
  **K8S_NAMESPACE**: Definimos el espacio de nombres en Kubernetes donde se desplegará la aplicación (default).
  **K8S_DEPLOYMENT**: Definimos el nombre del despliegue en Kubernetes (mi-primera-app).

**NOTA:** Hay que comentar que hay 2 variables que no están declaradas en este pipeline que son **KANIKO_SECRET** y **KUBECONFIG_SECRET.** Cuando terminemos de subir los cambios y conoceré explicaré como declaramos estas variables generales.

- **Job docker_build**:

  - Pertenece a la etapa build-containers.
  - Usa la imagen **gcr.io/kaniko-project/executor:debug** para construir el contenedor.
  - Antes de ejecutar el script, copia el secreto de Kaniko ($**KANIKO_SECRET**) para la autenticación con el Docker registry.
  - El script ejecuta Kaniko para construir la imagen Docker usando el Dockerfile en el directorio del proyecto y sube la imagen al registro definido.

- **Job deploy:**

  - Pertenece a la etapa deploy.
  - Usa la imagen **bitnami/kubectl**:latest para el despliegue.
  - Antes de ejecutar el script, copia el secreto de configuración de kubectl ($**KUBECONFIG_SECRET**) para la autenticación con Kubernetes.
  - El script usa **kubectl** para actualizar la imagen del despliegue en Kubernetes con la nueva imagen construida.

Habiendo explicado el uso de estos ficheros y qué contenido tienen, pasamos a subir los cambios al proyecto.

```
$ cd mi-primera-app
$ git add .
$ git commit -am “Contenido añadido”
$ git push origin main
```

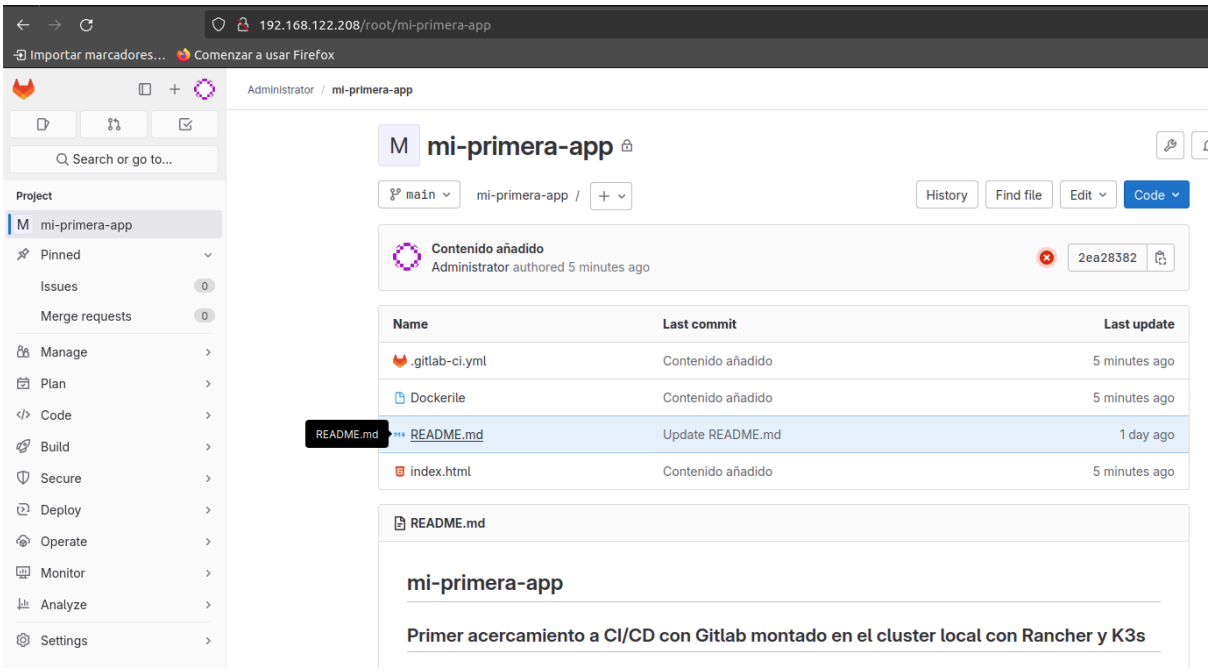
Como no estamos logueados en la terminal para poder hacer el comando **git push**, tendremos que acceder a nuestra aplicación de gitlab y editaremos el perfil del administrador.


Tras esto, bajamos y copiamos el email general por nuestra aplicación y lo introducimos en este comando:

```
$ git config --global user.email
"gitlab_admin_[cadenas_por_defecto]@example.com"
```

```
root@gitlab:/home/pepe/mi-primera-app#   git config --global user.email "gitlab_admin_b30248@example.
com"
root@gitlab:/home/pepe/mi-primera-app# git commit -am "Contenido añadido"
[main 2ea2838] Contenido añadido
 3 files changed, 69 insertions(+)
 create mode 100644 .gitlab-ci.yml
 create mode 100644 Dockerile
 create mode 100644 index.html
root@gitlab:/home/pepe/mi-primera-app# git push origin main
Username for 'http://192.168.122.208': root
Password for 'http://root@192.168.122.208':
Enumerando objetos: 6, listo.
Contando objetos: 100% (6/6), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (5/5), 1.32 KiB | 1.32 MiB/s, listo.
Total 5 (delta 0), reusados 0 (delta 0), pack-reusados 0
To http://192.168.122.208/root/mi-primera-app.git
   44419a8..2ea2838  main -> main
root@gitlab:/home/pepe/mi-primera-app#
```

Ya tendríamos añadidos los ficheros en nuestro proyecto.

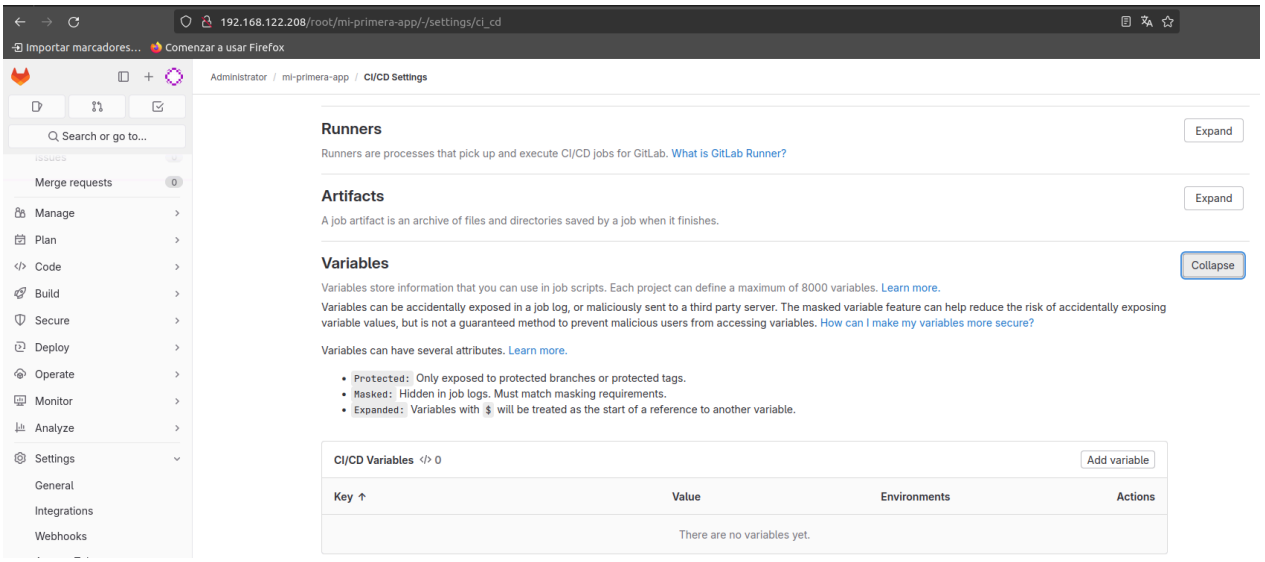



Antes de añadir las variables globales de Gitlab, tendremos que añadir a todas las máquinas que tenemos instalado Docker el siguiente **demonio** para que acceda el **registry** de manera no segura por el protocolo **HTTP.**

Para que los cambios surjan, tiene que reiniciarse el servicio y volver a levantar el **compose** del **registry**.

```
$ nano /etc/docker/daemon.json

{
  "insecure-registries": ["[ip_externa_manager]:5000"]
}

$ sudo systemctl restart docker
```

Pasamos a la configuración de las variables generales que nos faltan para que funcione nuestro pipeline. Para ello, accederemos **Settings -> CI/CD** y al apartado de **Variables -> Expand**.




Las variables que añadiremos son **KANIKO_SECRET** y **KUBECONFIG_SECRET** que ambas son de tipo **File** y se encuentran en los siguientes directorios.

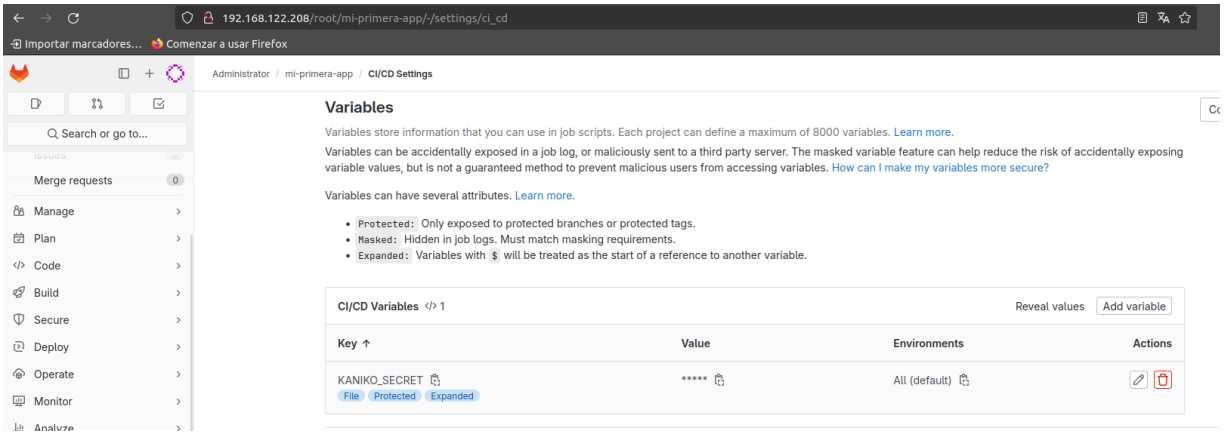
- **KANIKO_SECRET**: se encuentra en el fichero **/root/.docker/config.json en la máquina manager**, el cual sí no existe lo que tienes que hacer es loguearte a tu registry mediante ip con este comando:

```
$ docker login [ip_externa_manager]:5000
```

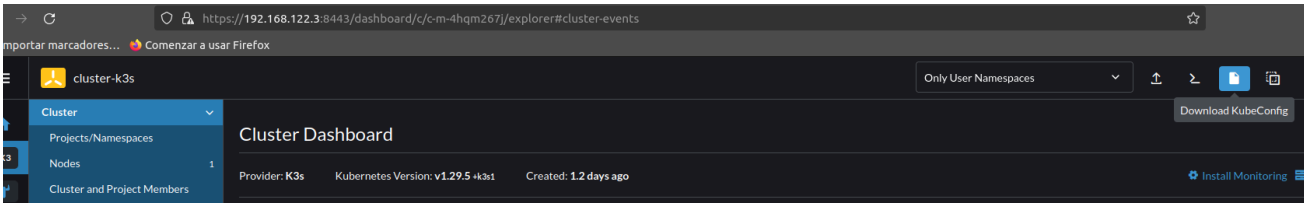
Cuando te logues, este fichero se creará automáticamente y es por ello que tendrá el siguiente contenido:

```
{
    "auths": {
        "[ip_externa_manager]:5000": {
                "auth": "Contraseña en cuestión"
        }
    }
}
```

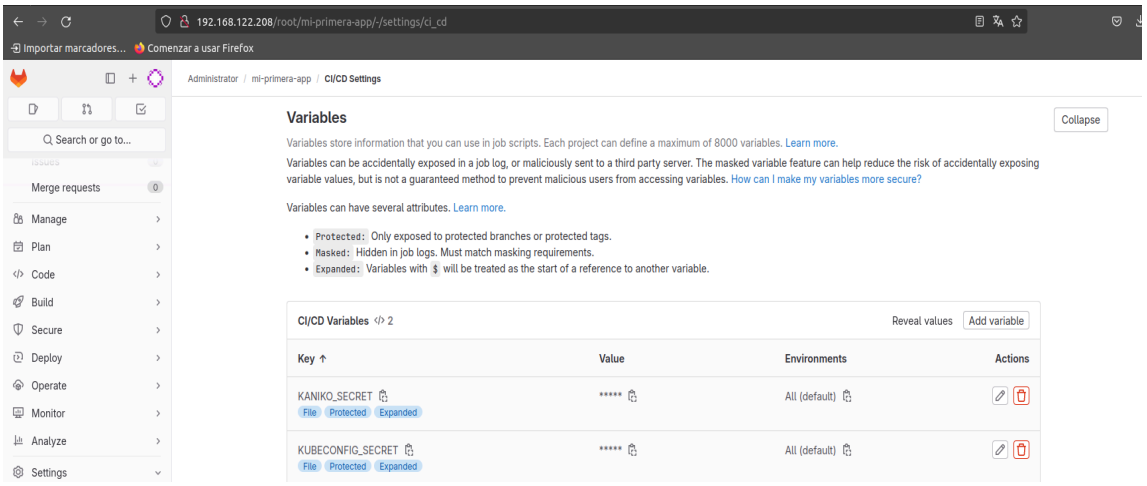
Este contenido lo copiamos y lo añadimos a la variable de gitlab.




- **KUBECONFIG_SECRET**: este fichero se ubica en la interfaz de Rancher, en el siguiente icono que pone **Download Kubeconfig.**




Tendremos que descargarlos en nuestra máquina local y añadir todo el contenido en crudo en la variable que crearemos en **Gitlab**.




Con esto ya tendríamos la configuración de **Gitlab y ru respectivo Runner** de manera correcta.

# 4. Demostración final

En esta sección, llevaremos a cabo una demostración práctica que abarca varios aspectos clave de la integración y monitorización de aplicaciones. Es decir, vamos a realizar una **prueba de la integración continua con Gitlab mediante una aplicación con un clúster en Rancher y monitorizado con Grafana y Prometheus**.

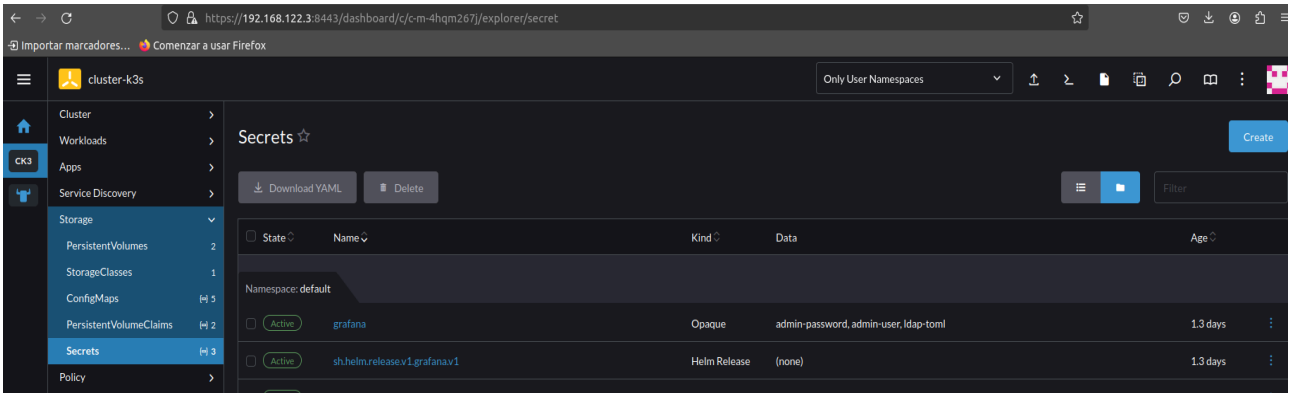
Pero antes de realizar la integración, vamos a configurar el deployment para poder correr la aplicación y secret para poder acceder al **registry** privado.

Para ello, primero, tendremos que acceder a la máquina **master** donde tenemos ubicado el cluster de **K3s**, modificamos y añadimos el siguiente archivo:

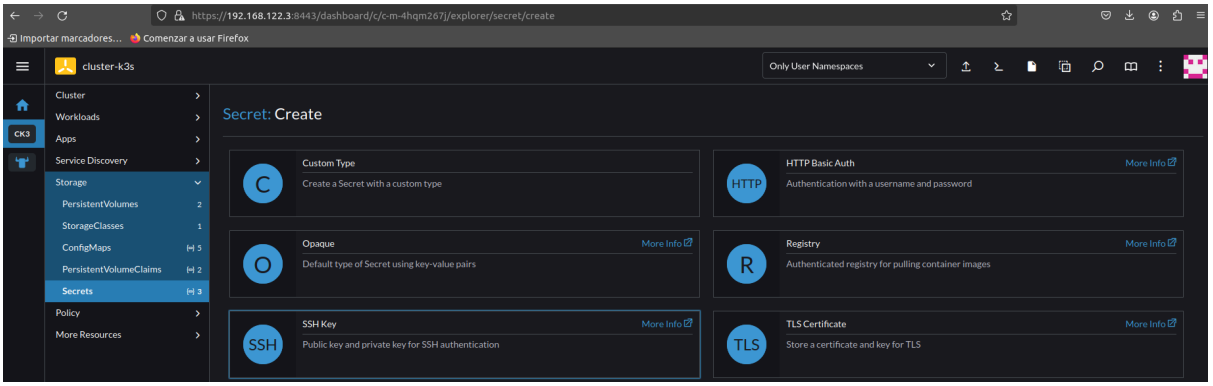
```
$ nano /etc/rancher/k3s/registries.yaml

mirrors:
  "http://[ip_externa_manager]:5000":
      endpoint:
      - "http://[ip_externa_manager]:5000"
```

Tras esto, reiniciamos la máquina master para que actualicen los cambios correctamente.

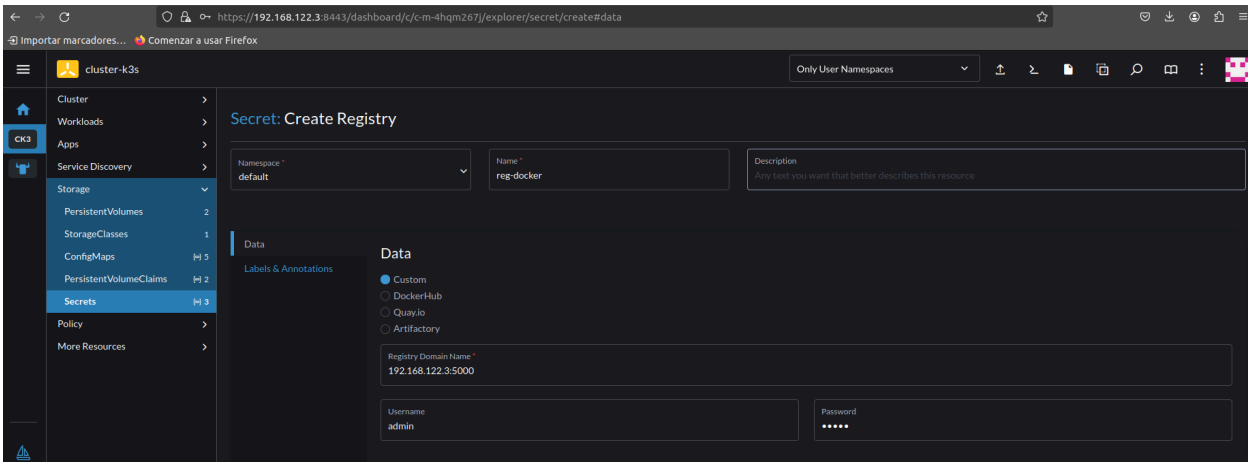
Una vez terminada, pasamos a crear el **secret** para el **registry** privado. Para ello accederemos al apartado de **Storage -> Secrets** y dándole a crear.



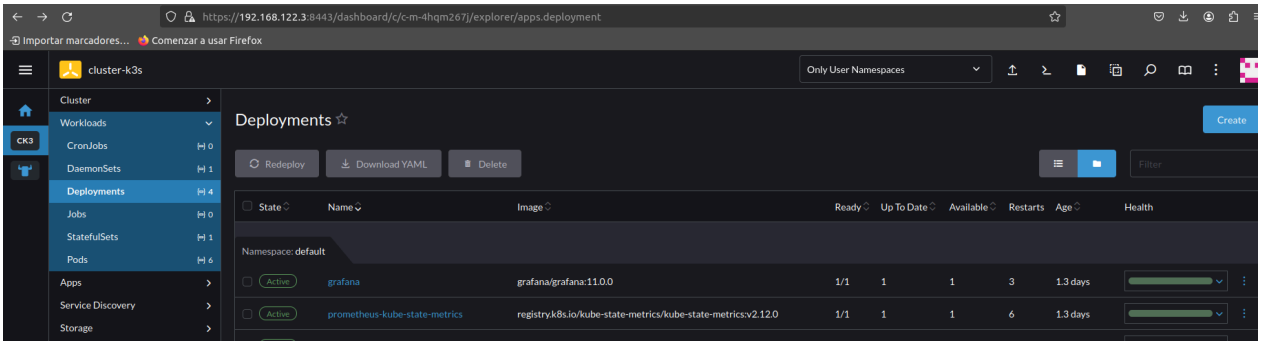
Tras esto, crearemos un Registry y lo seleccionamos.



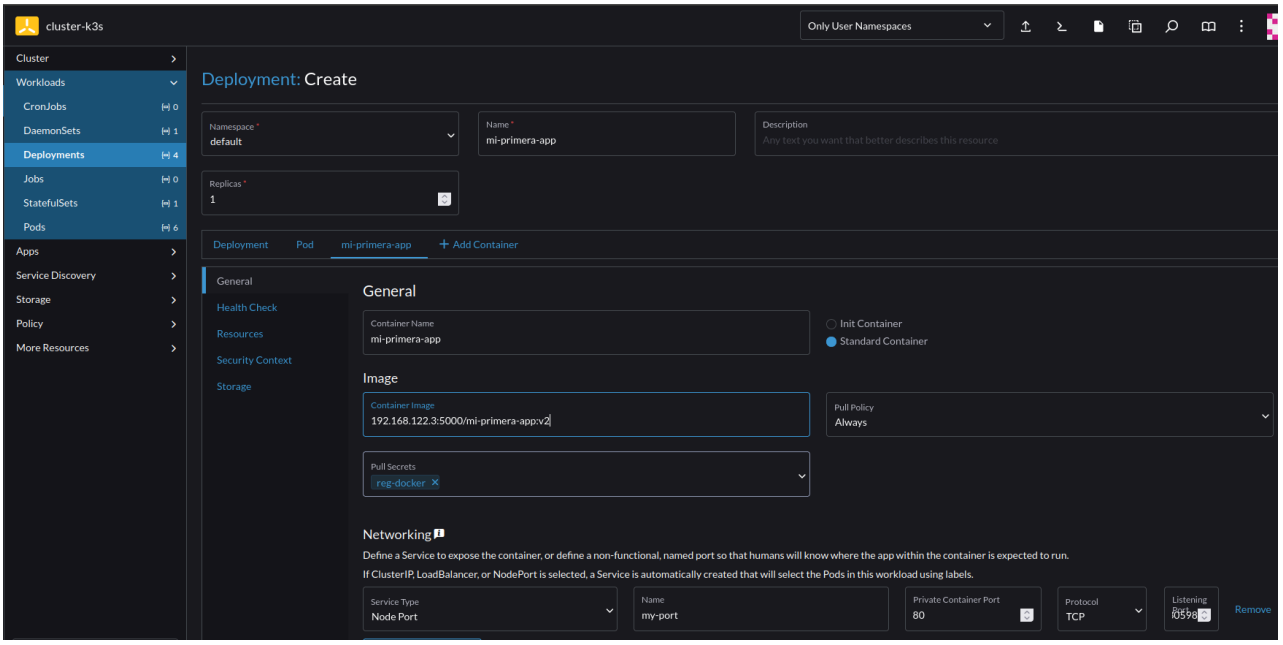

Por último, para crearlo tendremos que indicar la ip de la máquina manager con el puerto 5000 de nuestro **registry** más el usuario y la contraseña que cambiamos mediante el comando **htpasswd.**




Pasamos a crear el deployment desde la interfaz accediendo **Workloads -> Deployments** y dándole a crear.



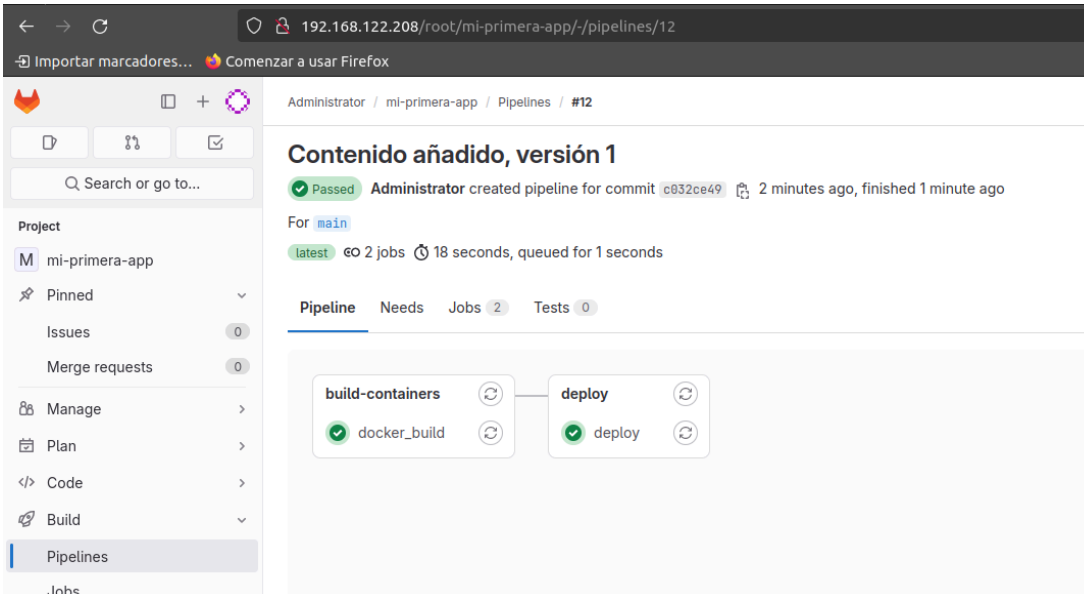
Por último, en la configuración del deployment **añadiendo nombre de la imagen** que queremos importar, el **registry** que hemos creado anteriormente y el puerto del contenedor el 80 (por defecto por nginx) al **Nodeport** 30598 (se puede autogenerar si no indicamos alguno).




Creamos el deployment y ya tendríamos todos los cambios de manera que nos faltaría hacer la integración continua. Para ello, realizaremos un **commit** añadiendo un punto al primer h1 de nuestro **index.html** y subiremos el cambio.

```
root@gitlab:/home/pepe/mi-primera-app# nano index.html
root@gitlab:/home/pepe/mi-primera-app# git add .
root@gitlab:/home/pepe/mi-primera-app# git commit -am "Contenido añadido, versión 1"
[main c032ce4] Contenido añadido, versión 1
 1 file changed, 1 insertion(+), 1 deletion(-)
root@gitlab:/home/pepe/mi-primera-app# git push origin main
Username for 'http://192.168.122.208': root
Password for 'http://root@192.168.122.208':
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 316 bytes | 316.00 KiB/s, listo.
Total 3 (delta 2), reusados 0 (delta 0), pack-reusados 0
To http://192.168.122.208/root/mi-primera-app.git
   6ec099a..c032ce4  main -> main
root@gitlab:/home/pepe/mi-primera-app#
```

Cuando realicemos el push, automáticamente nuestro Runner disparará los **jobs** correspondientes y los ejecutará de manera correcta.




Como vemos se ha disparado correctamente y si accedemos a la URL siguiente podremos ver la aplicación funcionando de manera correcta.

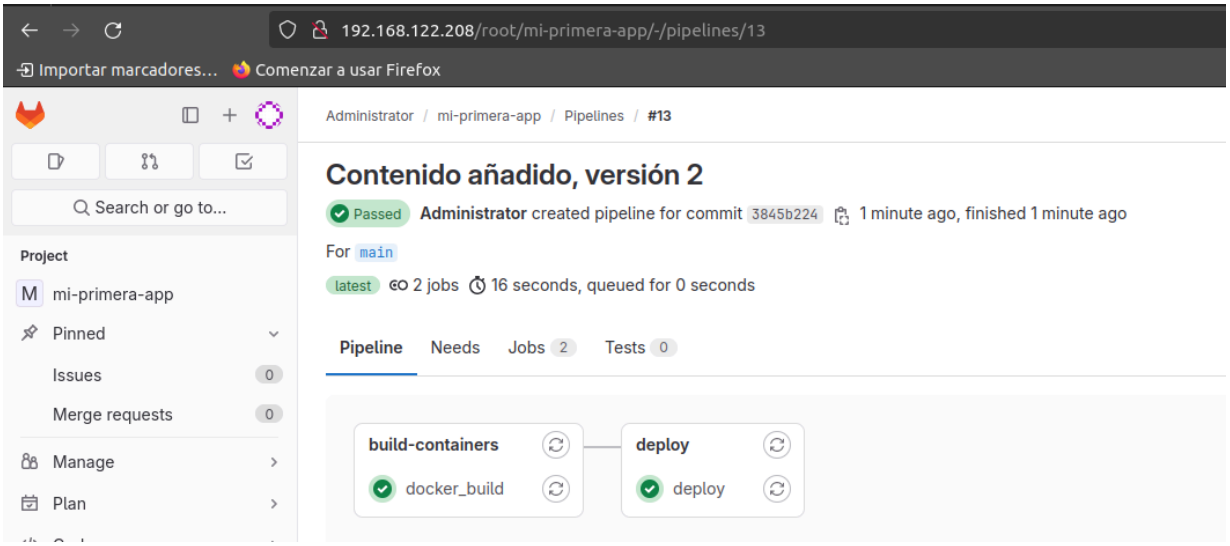
```
$ http://[ip_externa_master]:[puerto_node_port]/

En mi caso...

$ http://192.168.122.31:30598/
```


Como podemos ver, se ha desplegado de manera correcta pero para verlo de manera más explícita, vamos a ver un cambio en la aplicación mediante el index.html

Le vamos a añadir al h1 un v2 al final para ver que se despliega de manera automática con el Runner.

```
root@gitlab:/home/pepe/mi-primera-app# nano index.html
root@gitlab:/home/pepe/mi-primera-app# cat index.html
<!DOCTYPE html>
<html lang="es">
<head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>Página de Ejemplo</title>
        <style>
        body {
                display: flex;
                justify-content: center;
                align-items: center;
                flex-direction: column;
                height: 100vh;
                margin: 0;
                background-color: #f0f0f0;
                font-family: Arial, sans-serif;
        }
        h1, h2 {
                color: #8c52ff;
                text-align: center;
                border: 2px solid #8c52ff;
                padding: 20px;
                border-radius: 10px;
                background-color: white;
                box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
                margin: 10px;
        }
        h2 {
                color: #21ef80;
                border: 2px solid #21ef80;
        }
        </style>
</head>
<body>
        <h1>Prueba de funcionamiento CI/CD mediante Gitlab con Rancher + K3S v2</h1>
        <h2>Realizado por Jose Carlos Rodríguez Cañas</h2>
</body>
</html>
root@gitlab:/home/pepe/mi-primera-app# git add .
root@gitlab:/home/pepe/mi-primera-app# git commit -am "Contenido añadido, versión 2"
[main 3845b22] Contenido añadido, versión 2
 1 file changed, 1 insertion(+), 1 deletion(-)
root@gitlab:/home/pepe/mi-primera-app# git push origin main
Username for 'http://192.168.122.208': root
Password for 'http://root@192.168.122.208':
Enumerando objetos: 5, listo.
Contando objetos: 100% (5/5), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (3/3), 316 bytes | 316.00 KiB/s, listo.
Total 3 (delta 2), reusados 0 (delta 0), pack-reusados 0
To http://192.168.122.208/root/mi-primera-app.git
   c032ce4..3845b22  main -> main
root@gitlab:/home/pepe/mi-primera-app#
```

Tras éllo, saltará dicho Runner y se actualizará la aplicación de manera rápida y sencilla.




Volvemos a acceder a la aplicación y ...


Con ésto, ya tendríamos la integración continua mediante **Gitlab en nuestro cluster de alta disponibilidad de K3s orquestado por Rancher**.

Si queréis ver el video de como lo hago en tiempo real, accede a este video creado por mi para comprobar la veracidad del despliegue:

```
$ https://youtu.be/gz273b_rsNw
```

# 5. Dificultades encontradas

Con respecto a las dificultades encontradas, realmente han sido cuestión de tiempo y de plantear opciones a la hora de montar nuestro cluster de alta disponibilidad.

Algunos de los ejemplos de conflictos o dificultades encontradas pueden ser la **conexión del Registry privado con nuestro Gitlab**, la **integración continua con nuestro cluster de manera automática**...

Éstas y muchas más pueden ser las dificultades que me he podido encontrar en mi proyecto integrado, pero realmente los problemas que me han generado mayor incertidumbre son los siguientes.

## 5.1. Conexión del cluster con Rancher

Una de las dificultades que me han dado tantísimos problemas ha sido la instalación del cluster con nuestro Rancher.

Como ya comentamos en la instalación del mismo, lo haríamos con **Docker** y ese fue el problema al principio. Lo intenté desplegar de la manera más básica como es la siguiente.

```
$ docker run -d --restart=unless-stopped \
  -p 80:80 -p 443:443 \
  --privileged \
  rancher/rancher:stable
```

Lo que yo no contaba es que este contenedor Docker suele estar configurado para poder utilizar **HTTPS** aunque no disponga de certificado.

Por ello que cuando instalamos el curl de importación del mismo cluster de K3s, nos dará un problema **curl 60** por la URL **HTTPS** que utiliza.

- Info del error en cuestión

La máquina master no encuentra ningún certificado firmado por la **CA** ya que ese certificado es autofirmado en la máquina manager.

Una de las soluciones que apliqué para el correcto funcionamiento de Rancher fue **añadir mi certificado y mi clave de mi dominio comprado en IONOS** (pepepfoter15.es) ya que está firmado por una CA que puede reconocer todas las máquinas del cluster mediante la salida al exterior.

Ésto lo haría copiando los certificados de mi máquina local a la máquina manager y generar un **volumen bind mount** de los mismos certificados.

```
$ sudo docker run -d --name rancher --restart=unless-stopped \
  -p 8080:80 -p 8443:443 \
  -v
/etc/ssl/rancher/pepepfoter15.es_ssl_certificate.cer:/etc/rancher/ssl/pepepf
oter15.es_ssl_certificate.cer \
  -v
/etc/ssl/rancher/_.pepepfoter15.es_private_key.key:/etc/rancher/ssl/_.pepepf
oter15.es_private_key.key \
  -v /var/lib/docker/rancher:/var/lib/rancher \
  --privileged \
  rancher/rancher:stable
```

Tras éllo, el problema se vió resuelto en instantes.

## 5.2. Configuración de Gitlab y su respectivo Runner

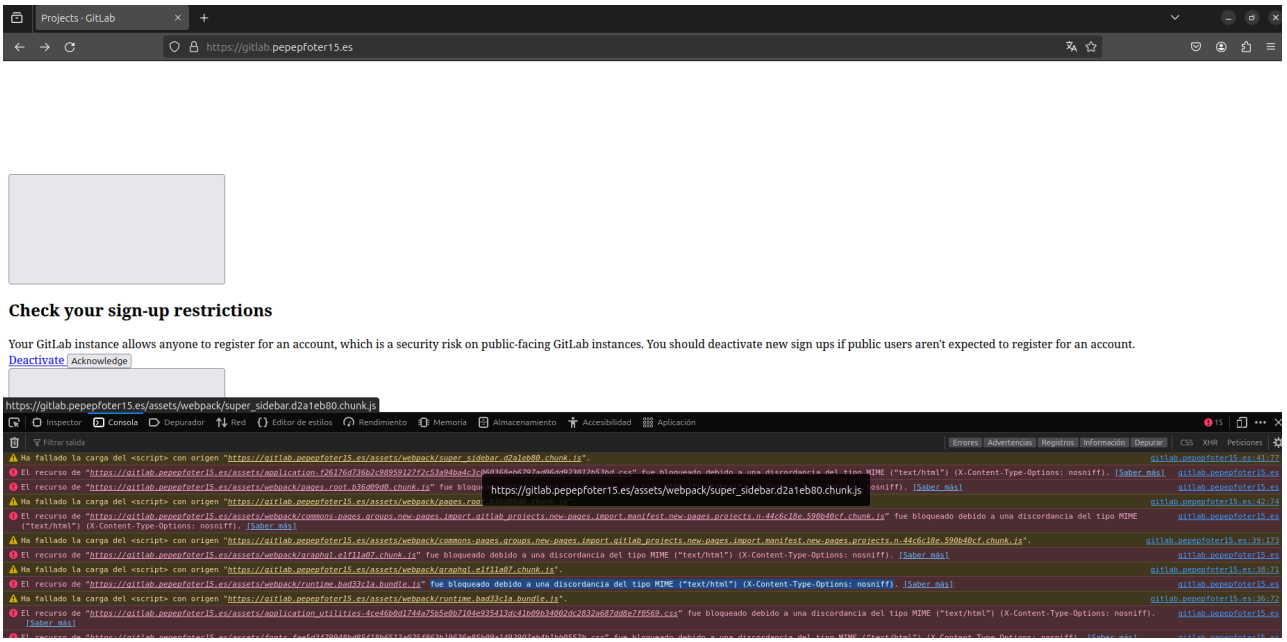
A la hora de la configuración de nuestro Gitlab, tenía una primera opción de instalación, realizada mediante el chart de Helm como realizamos con los contenedores de Grafana y Prometheus.

Uno de los inconvenientes de este mismo era el proceso de instalación ya que al instalarlo mediante el protocolo **HTTPS** y crear un **ingress** en nuestro **Rancher**, nos sucedía un problema con las URL a la hora de cargar la hora de estilos.

- Info del error en cuestión

Como podemos ver en la siguiente imagen el mismo **ingress**, mostraba el contenido de los contenedores pero sin ningún tipo de hoja de estilos, haciendo la instalación de este mismo inútil.

```
$ El recurso de
“https://gitlab.pepepfoter15.es/assets/webpack/runtime.bad33c1a.bundle.js”
fue bloqueado debido a una discordancia del tipo MIME (“text/html”)
(X-Content-Type-Options: nosniff).
```




El comando de instalación en cuestión era el siguiente:

```
$ helm upgrade --install gitlab gitlab/gitlab \
  --namespace gitlab \
  --kubeconfig /etc/rancher/k3s/k3s.yaml \
  --timeout 600s \
  --set global.hosts.domain=pepepfoter15.es \
  --set global.hosts.externalIP=[ip_externa_master] \
  --set certmanager-issuer.email=pepepfoter15@gmail.com \
  --set global.hosts.https=true \
  --set global.ingress.enabled=true \
  --set global.ingress.configureCertmanager=false \
  --set certmanager.install=false \
  --set global.gitlabVersion=17.0.1 \
  --set nginx-ingress.enabled=false \
  --set global.edition=ce \
```

Para poder solucionar dicho problema, simplemente **segmenté los servicios**. Instalé una máquina **Gitlab que tuviese dicha aplicación con su Runner y eliminé la instalación del chart de Helm del mismo**.

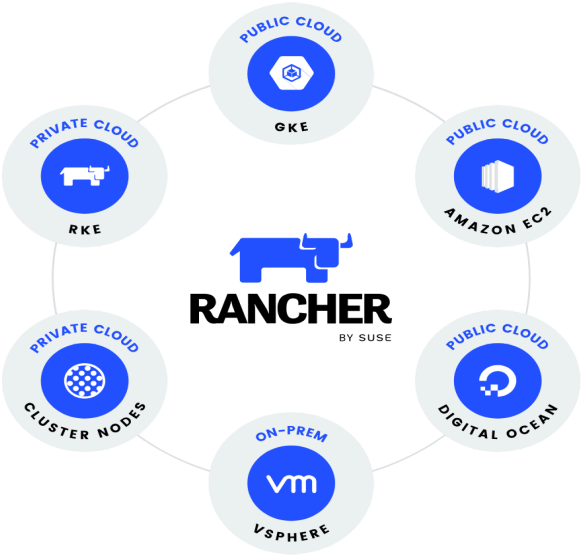
Dicho proceso de instalación se encuentra en el apartado [3.3.4](#).

# 6. Conclusión

En este proyecto, se intenta buscar la construcción de una plataforma integral para gestionar clústeres de contenedores utilizando **Rancher y K3s**.

La combinación de estas herramientas nos permite **implementar, monitorear y administrar aplicaciones de manera eficiente y segura, optimizar recursos y garantizar la máxima disponibilidad**.

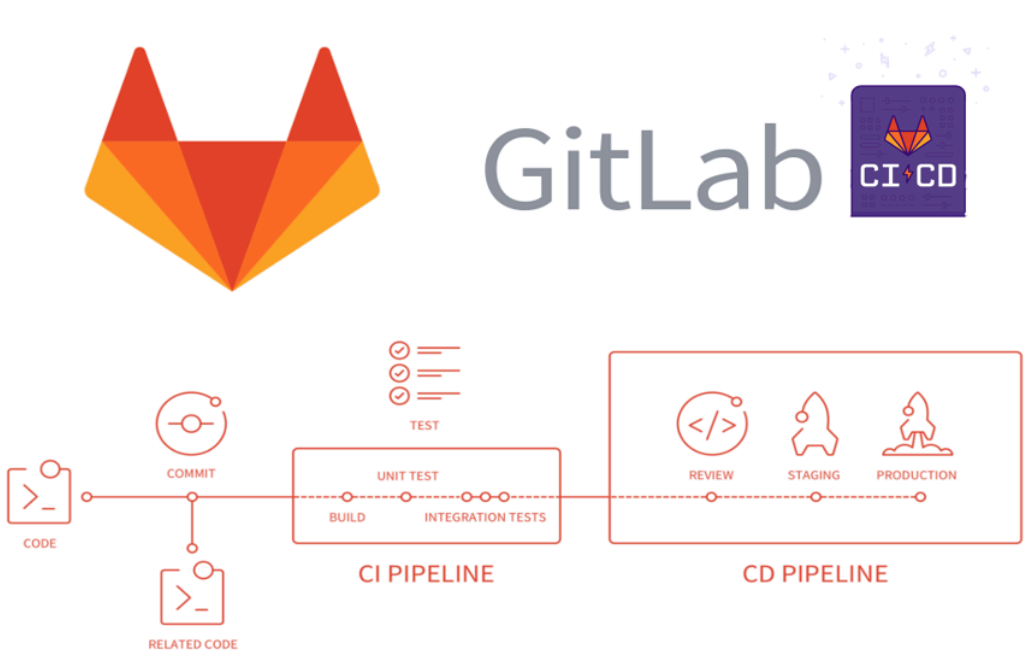
Elegir **Rancher** como orquestador y K3s como distribución de Kubernetes es clave para lograr una arquitectura robusta, escalable y fácil de administrar, especialmente en entornos con recursos limitados.




Durante el proyecto se han implementado soluciones de monitorización y visualización utilizando **Grafana y Prometheus**.

Además, la incorporación de **GitLab y GitLab Runner** para integración y entrega continuas (CI/CD), me ha simplificado enormemente el flujo de trabajo del desarrollador. **Esta automatización permite el desarrollo, las pruebas y la implementación de aplicaciones**.

La capacidad de **GitLab para gestionar todo el ciclo de vida del desarrollo de software proporciona una visión unificada y flexible del proceso**, desde la planificación hasta el lanzamiento.




Como resultado, el proyecto que he creado nos da una p**lataforma sólida y eficiente para la gestión de clusters de contenedores**, incorporando herramientas líderes en el mercado para garantizar disponibilidad, eficiencia y seguridad.

La implementación de **Rancher, K3s, Grafana, Prometheus y GitLab** ha creado un entorno unificado que respalda a los administradores y desarrolladores de sistemas y promueve prácticas de **DevOps** que optimizan el desarrollo y la implementación de aplicaciones.

**Esta base no solo maneja las operaciones diarias, sino que también proporciona una base sólida para la expansión y la automatización de tareas**.

# 7. Bibliografía

¿Qué es Kubernetes?
¿Qué es K3s? y Documentación de K3s

¿Qué es Rancher?
Página oficial - Rancher

Prometheus Exporters personalizados
Curso de OpenWebinars - Grafana y Prometheus

Saber todo sobre el repositorio Git para DevOps - GitLab
¿Qué es GitLab Runner?

Recomendaciones de arquitectura - Rancher
Instalación del clúster de K3s con Rancher
Instalación de Gitlab y Gitlab Runner
Instalación de Grafana y Prometheus con Helm