

**Universidad del Quindío**

**Facultad de Ingeniería**

**Programa de Ingeniería Electrónica**

**Daniel Alejandro Quintero  
Carlos Andrés Rendón Soto  
Sebastián Pardo Ocampo  
Valentina Garibello Lizarazo**

**Avance 2 - Seminario de grado: Despliegue y monitorización de servicios en  
redes y Kubernetes**

**Armenia, octubre del 2024**

## Introducción

El mundo de las tecnologías de la información y las telecomunicaciones avanza rápidamente, y con ello, surgen nuevas herramientas para optimizar procesos y mejorar la eficiencia en el manejo de servicios. En este contexto, Kubernetes se posiciona como una de las plataformas más revolucionarias para gestionar aplicaciones y servicios en la nube, facilitando tanto su despliegue como su monitorización.

Este informe aborda la implementación de un sistema basado en Kubernetes, centrado en su capacidad para gestionar servicios de forma automatizada y eficiente. El proyecto incluye la configuración de una red con un router Juniper, la creación de un clúster de Kubernetes mediante Kubeadm y la integración de herramientas como Prometheus y Grafana para monitorizar los recursos y servicios desplegados. Además, se implementó un servidor de video que permite transmitir contenido de forma fluida, demostrando la versatilidad de Kubernetes en aplicaciones prácticas.

A través de este trabajo, se busca no solo implementar una solución técnica, sino también demostrar cómo estas tecnologías pueden ser útiles en contextos reales, como en el área de las telecomunicaciones y otras industrias que requieren infraestructura robusta y escalable.

## Objetivos

El objetivo principal de este proyecto es extender la red planteada en la primera entrega mediante la adición de un router y realizar la monitorización en un entorno Kubernetes configurado con kubeadm. Para ello, se busca construir una red que permita conectar tres o más dispositivos a través de un router Juniper, simulando una nube donde se desplegarán servicios. En este caso, se implementará un servidor de video Nginx, cuya monitorización se llevará a cabo mediante herramientas como lo son prometheus y Grafana.

## Descripción general del proyecto

Este proyecto consta de la descarga, creación, configuración e implementación de un clúster de Kubeadm, con la finalidad de obtener una plataforma que pueda ser gestionado y supervisado para la administración de aplicaciones y servicios. El trabajo realizado durante el seminario de grado comprende diversas fases técnicas y organizativas que abarcan desde la instalación y configuración del orquestador de contenedores hasta la puesta en marcha de instrumentos de monitorización de los servicios implementados.

Se realizó las siguientes tareas para el desarrollo del proyecto:

### 1. Configuración del Entorno de Trabajo:

- Preparación de los nodos maestro y trabajadores con los recursos necesarios (Docker, kubeadm, kubelet, kubectl, containerd).
- Configuración de un router Juniper para proporcionar conectividad entre los nodos y acceso a internet utilizando la red de la universidad.

### 2. Despliegue del Clúster con kubeadm:

- Inicialización del nodo maestro y unión de nodos trabajadores para formar el clúster.
- Creación de contenedores personalizados (servidor de video RTMP y cliente de video streaming) y despliegue de Pods basados en estos contenedores.

### **3. Monitorización del Clúster:**

- Implementación de herramientas como Prometheus, Grafana, cAdvisor y Node Exporter para la recopilación, visualización de métricas del clúster y sus componentes.

## **Herramientas Utilizadas**

Durante el desarrollo del proyecto, se emplean diversas herramientas que desempeñan roles esenciales en cada etapa, desde la configuración inicial hasta la monitorización del clúster Kubernetes. A continuación, las herramientas usadas:

### **1. Kubernetes (kubeadm, kubelet y kubectl)**

- kubeadm: Utilizado para la configuración inicial y el despliegue del clúster Kubernetes, incluyendo la inicialización del nodo maestro y la unión de los nodos trabajadores.
- kubelet: Gestor de nodos encargado de iniciar y mantener los contenedores en ejecución dentro del clúster.
- kubectl: Herramienta de línea de comandos para interactuar con el clúster, gestionar Pods, servicios y realizar operaciones administrativas.

### **2. Docker**

- Plataforma para la creación, gestión y ejecución de contenedores. Fue utilizada tanto para crear imágenes personalizadas como para alojar las aplicaciones y servicios desplegados dentro del clúster. Sin embargo, el container runtime utilizado en el clúster fue containerd, no Docker.

### **3. Router Juniper**

- Configurado para proporcionar conectividad entre los nodos del clúster y acceso a internet utilizando la red de la universidad. Su rol fue crucial para garantizar la comunicación y descarga de recursos esenciales para Kubernetes.

### **4. Prometheus**

- Sistema de monitorización y alerta utilizado para recopilar métricas de rendimiento del clúster y los contenedores en ejecución. Fue configurado para extraer datos de cAdvisor y Node Exporter.

### **5. Grafana**

- Herramienta de visualización que permitió crear dashboards personalizados para analizar métricas recolectadas por Prometheus, facilitando el monitoreo en tiempo real del clúster.

### **6. cAdvisor (Container Advisor)**

- Herramienta para la monitorización de recursos utilizados por los contenedores, como CPU, memoria y disco. Fue integrada con Prometheus para ampliar la visibilidad del clúster.

### **7. Node Exporter**

- Componente utilizado para recolectar métricas de los nodos del clúster, como el uso de CPU, memoria y red, proporcionando datos adicionales para el monitoreo de infraestructura.

### **8. Sistema Operativo Ubuntu 20.04.6 LTS (focal)**

- Base para la instalación de Kubernetes y los contenedores. Su flexibilidad y compatibilidad lo hicieron ideal para el entorno del proyecto.

## 9. YAML

- Lenguaje utilizado para definir configuraciones de Pods, servicios, y recursos del clúster en Kubernetes, permitiendo automatizar su despliegue.

## 10. Redes Físicas y Virtuales

- Cables UTP: Garantizaron la conectividad física entre los nodos y el router Juniper.
- Configuración de red IP: Para habilitar la comunicación entre los dispositivos y asegurar la correcta operación del clúster.

## Entorno de trabajo utilizado

El entorno de trabajo consistió en una infraestructura compuesta por hardware y software diseñado para soportar la creación y gestión del clúster Kubernetes.

### Hardware:

- **Router juniper:**  
Encargado de interconectar los computadores y la conectividad con internet.
- **Computadores (requisitos mínimos):**
  - CPU: 2 Núcleos o más por máquina (nodo).
  - RAM: 4 GB o más por máquina (nodo).
  - Almacenamiento: 40 GB por máquina (nodo).
  - Red física: Conexión entre los dispositivos mediante cable UTP y el router.

### Software

- Sistema operativo: Ubuntu 20.04.6 LTS (Focal) en todas las máquinas (nodos)
- Herramientas de Kubernetes: Kubectl, kubelet, kubeadm.
- Docker: Creador y gestor de contenedores.

## Configuración del router

Para realizar la estructura de red propuesta, es necesario realizar las configuraciones pertinentes de acuerdo a la marca del router que se vaya a utilizar. En este caso, se emplea un router Juniper SRX300, que cuenta con 6 puertos Ethernet para enrutamiento, 1 puerto para acceso a internet, 1 acceso USB para la configuración del router mediante consola y 2 puertos SFP.



Figura 1: Router Juniper SRX300

En la implementación de este proyecto, se propone la siguiente estructura:

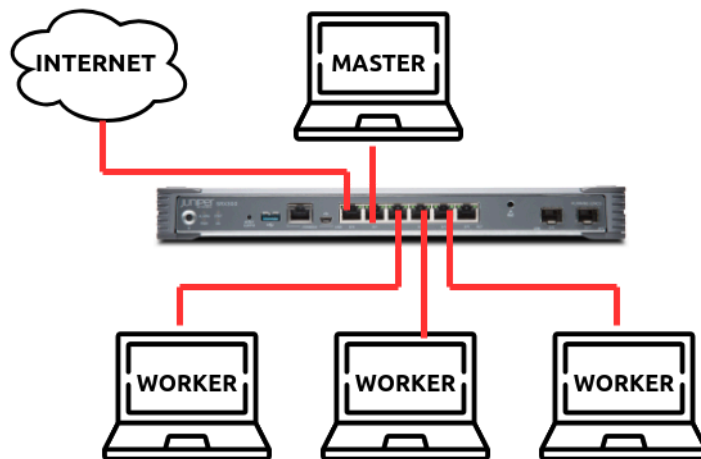


Figura 2: Estructura de red propuesta para la implementación del proyecto

Se debe ingresar al sistema de comandos del router Juniper, para ello se instala la herramienta **minicom** en un dispositivo con SO Linux, con el siguiente comando:

```
sudo apt install minicom
```

Luego de instalar la herramienta, se debe conectar el router al computador por medio de USB, y en modo de superusuario se ejecuta en el terminal:

```
minicom
```

Luego de asegurar de que la comunicación serial sí se esté realizando al dispositivo USB `dev/ttyUSB0`, se inicializa la herramienta en la que hay acceso automático a la línea de comandos del router Juniper.

Ahora, se debe configurar cada una de las interfaces que serán utilizadas para la implementación de la red, como se indica en la Figura 2.

Para cada interfaz se debe aplicar la misma configuración, según los cambios de direcciones IP que correspondan. Se debe habilitar la línea de comandos con `cli` y entrar a la sección de configuración con `configure`.

Luego, se ejecutan los comandos para cada una de las interfaces según las subredes correspondientes, para lo que se realizó una tabla de enrutamiento:

Subnet ID	Subnet Address	Host Address Range	Broadcast Address
1	172.30.0.0	172.30.0.1 - 172.30.0.62	172.30.0.63
2	172.30.0.64	172.30.0.65 - 172.30.0.126	172.30.0.127
3	172.30.0.128	172.30.0.129 - 172.30.0.190	172.30.0.191
4	172.30.0.192	172.30.0.193 - 172.30.0.254	172.30.0.255

Figura 3: Tabla de enrutamiento para la implementación de la red.

Este comando debe ejecutarse para cada una de las interfaces que están en uso, para este caso corresponden a cada dispositivo de la red.

```
set interfaces <interfaz ethernet> unit 0 family inet address <IP/MASCARA>
commit
```

Una vez configurada cada interfaz, es necesario establecer una NAT, pues para implementar una infraestructura de Kubernetes es necesario acceder a internet con el fin de descargar recursos y descargar las imágenes de los pods que se van a ejecutar en el clúster.

Para permitir que los dispositivos de la red privada accedan a internet por medio de una red pública es necesario configurar la NAT en el router. Para ello se destina una interfaz física en la que se conectará el ISP y se configura para que su dirección sea asignada mediante un servidor DHCP (proveniente de la red externa).

```
set interfaces ge-0/0/0 unit 0 family inet dhcp
commit
```

**Configurar las zonas de seguridad:**

```
set security policies from-zone trust to-zone untrust policy allow-internet
match source-address any
```

```
set security policies from-zone trust to-zone untrust policy allow-internet
match destination-address any
```

```
set security policies from-zone trust to-zone untrust policy allow-internet
match application any
```

```
set security policies from-zone trust to-zone untrust policy allow-internet
then permit
```

```
commit
```

Una zona de seguridad es un concepto lógico que agrupa interfaces según el nivel de confiabilidad. Estos comandos configuran políticas de seguridad en el router que permiten el tráfico entre dos zonas de seguridad: trust (zona confiable) y untrust (zona no confiable, como Internet).

Configurar NAT para traducir redes privadas:

```
set security nat source rule-set trust-to-untrust from zone trust

set security nat source rule-set trust-to-untrust to zone untrust

set security nat source rule-set trust-to-untrust rule internet-outbound
match source-address <Dirección de red>

set security nat source rule-set trust-to-untrust rule internet-outbound
then source-nat interface

commit
```

Estos comandos establecen una regla NAT que habilita el tráfico entre la zona trust y untrust con el fin de acceder a la red pública desde la red privada previamente establecida.

Ahora, se debe verificar que los cambios se aplicaron correctamente:

```
show configuration interfaces

show configuration routing-options

show configuration security nat

show route
```

Y verificar que se haya establecido la ruta a la red pública:

```
show route 0.0.0.0/0
```

Por último se verifica conectividad haciendo ping al servidor de Google:

```
ping 8.8.8.8
```

Así se permite acceso a internet por medio del router Juniper para que el resto de los dispositivos puedan acceder a la red para descargar recursos necesarios en la ejecución del clúster de Kubernetes. Ahora la red está lista para implementar la estructura de Kubernetes.

## Preparación del nodo maestro y nodos trabajadores

### Configuración Inicial con máquinas virtuales

En primer lugar, para el caso de emplear máquinas virtuales se requiere la configuración de los puertos de internet de la misma, debido a que Kubeadm para ser creado requiere internet para descargar archivos importantes y también para la conexión con los nodos trabajadores.

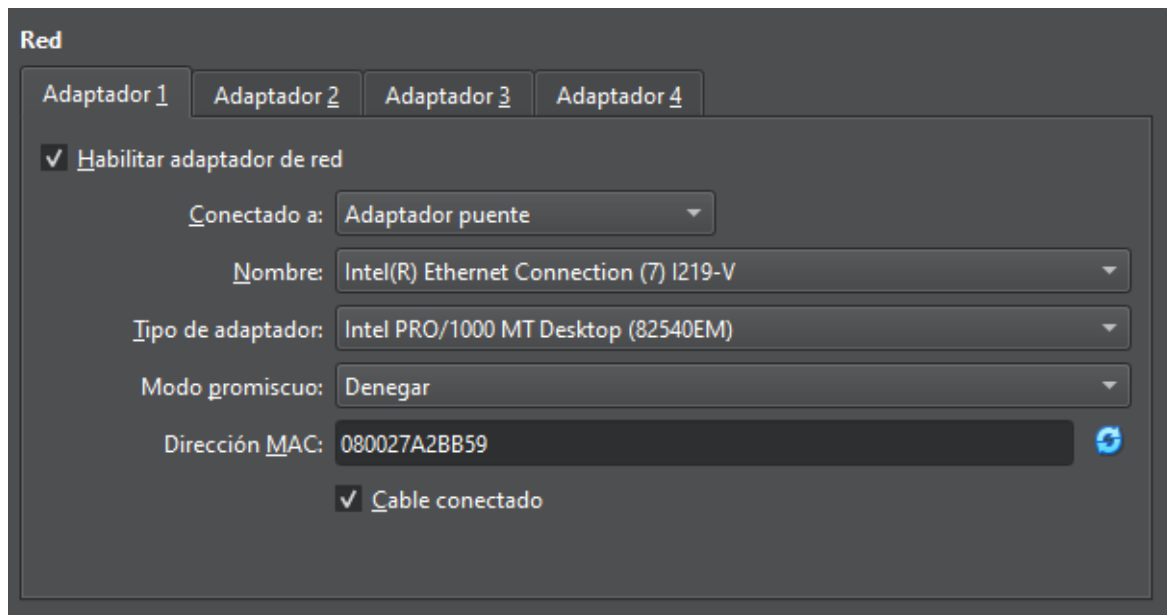


Figura 4: Adaptador de red 1

En la Figura 4 está la configuración recomendada, se debe configurar como adaptador puente y este estar asignado al adaptador de red que tiene internet.

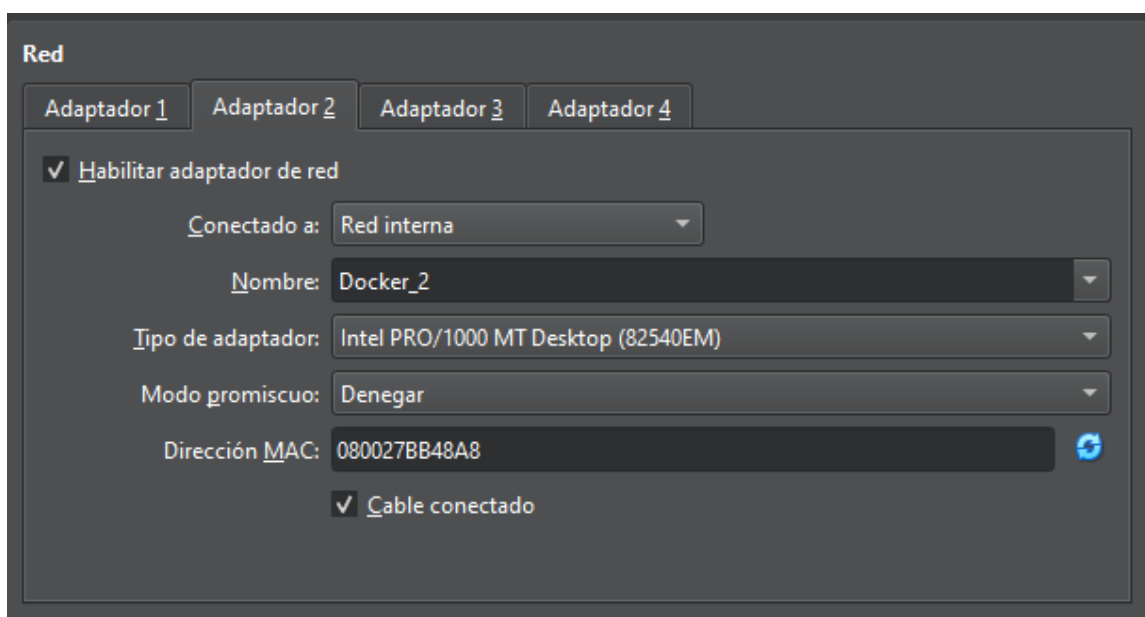


Figura 5: Adaptador de red 2



En la Figura 5 se configura el adaptador de red 2 como red interna y se crea con un nombre cualquiera, para este caso se escogió el nombre de “Docker\_2”, esta configuración de adaptadores debe ser igual para el nodo maestro y nodo trabajador.

## **Configuración Inicial computador nativo**

### **Instalación de Docker**

El siguiente comando actualiza la base de datos de los repositorios de software que hay en el sistema.

```
sudo apt-get update
```

Para la instalación del software curl encargado de la transferencia de los archivos a través de una URL requerida para la descarga de archivos de Docker.

```
sudo apt-get install ca-certificates curl
```

Con este comando se crea un directorio en la ubicación específica /etc/apt/keyrings con permisos de escritura lectura y ejecución.

```
sudo install -m 0755 -d /etc/apt/keyrings
```

El siguiente comando se encarga de transferir los archivos de la URL a la dirección específica /etc/apt/keyrings/docker.asc

```
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o  
/etc/apt/keyrings/docker.asc
```

Luego, se concede a todos los usuarios del sistema (propietario, grupo y otros) el permiso de lectura sobre el archivo docker.asc a través del siguiente comando:

```
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

El siguiente comando modifica un archivo de texto con las características necesarias para instalar Docker.

```
echo \  
"deb [arch=$(dpkg --print-architecture)  
signed-by=/etc/apt/keyrings/docker.asc]  
https://download.docker.com/linux/ubuntu \  
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \  
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Se actualiza la base de datos de los repositorios.

```
sudo apt-get update
```

El siguiente comando se encarga de instalar Docker Engine

```
sudo apt-get install docker-ce docker-ce-cli containerd.io  
docker-buildx-plugin docker-compose-plugin
```

### **Instalación de Kubeadm Master y Worker**

Para la instalación de kubeadm en nodos maestros y trabajadores se realiza la misma instalación la diferencia está en la configuración de los dos. Como primer paso se realiza la actualización del repositorio y aplicaciones.

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

Después de realizar la actualización se instalan los repositorios requeridos.

```
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.31/deb/Release.key | sudo  
gpg --dearmor -o /etc/apt/keyrings/kubernetes-apt-keyring.gpg
```

```
echo 'deb [signed-by=/etc/apt/keyrings/kubernetes-apt-keyring.gpg]  
https://pkgs.k8s.io/core:/stable:/v1.31/deb/ /' | sudo tee  
/etc/apt/sources.list.d/kubernetes.list
```

Ahora se instalan las aplicaciones que componen un cluster de Kubeadm.

```
sudo apt-get update
```

```
sudo apt-get install -y kubelet kubeadm kubectl
```

```
sudo apt-mark hold kubelet kubeadm kubectl
```

Después para verificar una correcta instalación.

```
kubectl version --client && kubeadm version
```

Para que kubeadm funcione se requiere deshabilitar la memoria SWAP. Esto debido a que es una partición del almacenamiento que funciona como memoria virtual, dando respaldo cuando la memoria RAM está llena, el problema es que Kubernetes tiene que gestionar los recursos del sistema para poder asignar esos recursos a los pods que se están ejecutando y el SWAP genera conflicto en la gestión de recursos, por ende se desactiva el SWAP con el siguiente comando.

```
sudo sed -i '/ swap / s/^\(.*\)$/#\1/g' /etc/fstab
```

Se requiere habilitar módulos del kernel para que kubeadm funcione.

```
sudo modprobe overlay
```

```
sudo modprobe br_netfilter
```

Se agregan configuraciones a sysctl.

```
sudo tee /etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
net.ipv4.ip_forward = 1
EOF
```

Después se reinicia sysctl.

```
sudo sysctl --system
```

Se realiza la configuración para que persistan los módulos.

```
sudo tee /etc/modules-load.d/containerd.conf <<EOF
overlay
br_netfilter
EOF
```

Kubeadm requiere un gestor de contenedores para este caso se usó containerd.

```
sudo apt update
sudo apt install -y containerd.io
```

Se configura containerd y se inician los servicios.

```
sudo mkdir -p /etc/containerd
sudo containerd config default|sudo tee /etc/containerd/config.toml
```

y finalmente se reinicia containerd.

```
sudo systemctl restart containerd
sudo systemctl enable containerd
systemctl status containerd
```

Ahora se instalan las imágenes que necesita kubeadm para funcionar.

```
sudo kubeadm config images pull
```

## Inicialización del Master

Para el caso del master se ejecuta el apiserver con la dirección IP de la interfaz de red que esté conectada al nodo Worker, como recomendación, es mejor escoger una IP diferente a la que proporciona el servicio de internet para evitar errores. Para la IP de pod-network-cidr se pueden poner cualquier IP siempre y cuando no sea la misma del api server.

```
docker@master:~$ sudo kubeadm init --apiserver-advertise-address=172.30.0.254 --pod-network-cidr=10.244.0.0/16
```

Figura 6: Comando para inicializar clúster.

```
sudo kubeadm init --apiserver-advertise-address=ip-interfaz-red  
--pod-network-cidr=10.244.0.0/16
```

```
Your Kubernetes control-plane has initialized successfully!  
  
To start using your cluster, you need to run the following as a regular user:  
  
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config  
  
Alternatively, if you are the root user, you can run:  
  
export KUBECONFIG=/etc/kubernetes/admin.conf  
  
You should now deploy a pod network to the cluster.  
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:  
https://kubernetes.io/docs/concepts/cluster-administration/addons/  
  
Then you can join any number of worker nodes by running the following on each as root:  
  
kubeadm join 172.30.0.254:6443 --token rarej1.zlgjglb1paxljgfn \  
--discovery-token-ca-cert-hash sha256:470c1ef70929f9083ae865536eeb94866c34b50fde9730743ab2e9100a92c6f7
```

Figura 7: Resultado de la inicialización del clúster.

Después de desplegar el nodo maestro se requiere crear el directorio `.kube` en el directorio del usuario actual `$HOME` si aún no existe. Esto es para almacenar el archivo de configuración `config` necesario para interactuar con el clúster de Kubernetes utilizando herramientas como `kubectl`.

```
mkdir -p $HOME/.kube
```

Posteriormente, se copia el archivo de configuración del clúster `admin.conf` a la ubicación estándar donde `kubectl` espera encontrarlo `$HOME/.kube/config`, este archivo `admin.conf` contiene las credenciales y la configuración necesarias para que el usuario pueda gestionar el clúster.

```
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
```

Se requiere cambiar la propiedad del archivo copiado al usuario actual. `$(id -u)`. Para ello, se obtiene el ID de usuario actual. `$(id -g)` y el ID del grupo actual, esto asegura que el usuario actual tenga permisos adecuados para leer y modificar el archivo de configuración.

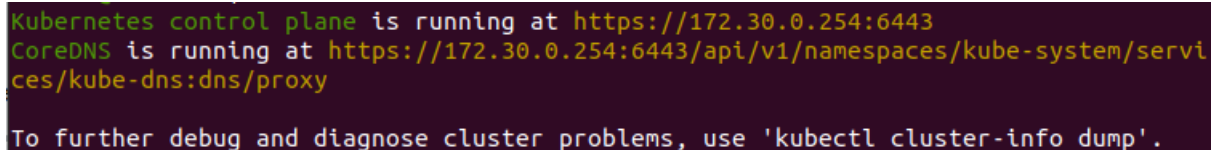
```
sudo chown $(id -u):$(id -g) $HOME/.kube/config
```

Ahora, se establece la variable de entorno `KUBECONFIG` para que apunte al archivo de configuración, esto permite a `kubectl` localizar automáticamente el archivo de configuración y usarlo para interactuar con el clúster.

```
export KUBECONFIG=$HOME/.kube/config
```

Para comprobar que el clúster funciona correctamente

```
kubectl cluster-info
```

A terminal window with a dark background and light-colored text. The output of the 'kubectl cluster-info' command is displayed. It shows the Kubernetes control plane running at https://172.30.0.254:6443 and CoreDNS running at https://172.30.0.254:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy. A note at the bottom suggests using 'kubectl cluster-info dump' for further debugging.

```
Kubernetes control plane is running at https://172.30.0.254:6443
CoreDNS is running at https://172.30.0.254:6443/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.
```

Figura 8: Información del api server en el clúster.

Ahora, se realiza la configuración de Calico. Como primera medida se descarga el archivo `.yaml` que ejecutara `tigera operator`, este puede desempeñar un papel crucial para administrar Calico de manera eficiente y automatizada

```
curl -O
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/tigera-operator.yaml
```

Se ejecuta el archivo `.yaml` de `tigera-operator`. Este comando se utiliza para crear cualquier servicio que esté alojado en un archivo `.yaml` en un clúster de `kubeadm`.

```
kubectl create -f tigera-operator.yaml
```

Después se descarga el archivo que contiene dos Custom Resources (CRs) para configurar una instalación de Calico en un clúster de Kubernetes. Ambos recursos son gestionados por el Tigera Operator y están relacionados con la configuración básica de redes y el servidor API de Calico.

```
curl -O
https://raw.githubusercontent.com/projectcalico/calico/v3.26.1/manifests/custom-resources.yaml
```

Ahora, se requiere modificar el archivo de (custom-resources.yaml) el cual si se ejecuta el comando curl por defecto en el terminal este se debe encontrar en la carpeta personal del sistema como se aprecia en la Figura 9 .

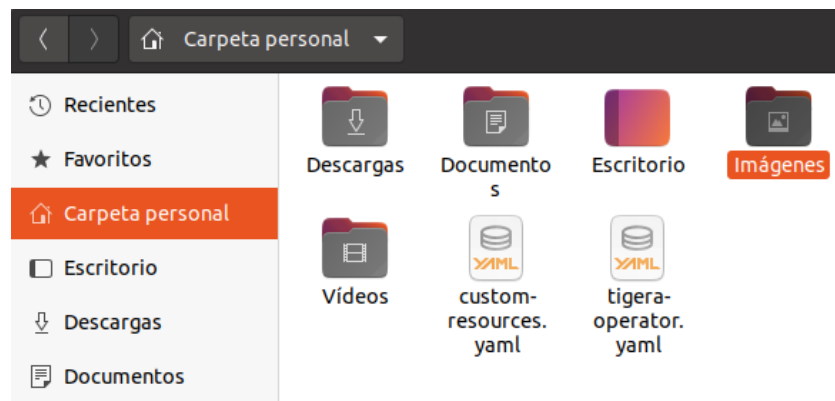


Figura 9: Ubicación de los archivos yaml de custom resource y tigera operator.

Ahora se modifica la línea 13 donde cidr: 10.244.0.0/16 debe tener la misma IP que se empleó cuando se ejecuta el comando --pod-network-cidr=10.244.0.0/16. Esto define el rango de IP que van a tener los pods en la red interna del clúster.

```
1 # This section includes base Calico installation configuration.
2 # For more information, see: https://projectcalico.docs.tigera.io/master/reference/installation/-
3 apiVersion: operator.tigera.io/v1
4 kind: Installation
5 metadata:
6   name: default
7 spec:
8   # Configures Calico networking.
9   calicoNetwork:
10     # Note: The ipPools section cannot be modified post-install.
11     ipPools:
12     - blockSize: 26
13       cidr: 10.244.0.0/16
14       encapsulation: VXLANCrossSubnet
15       natOutgoing: Enabled
16       nodeSelector: all()
17 ---
18
19
20 # This section configures the Calico API server.
21 # For more information, see: https://projectcalico.docs.tigera.io/master/reference/installation/-
22 apiVersion: operator.tigera.io/v1
23 kind: APIServer
24 metadata:
25   name: default
26 spec: {}
27
```

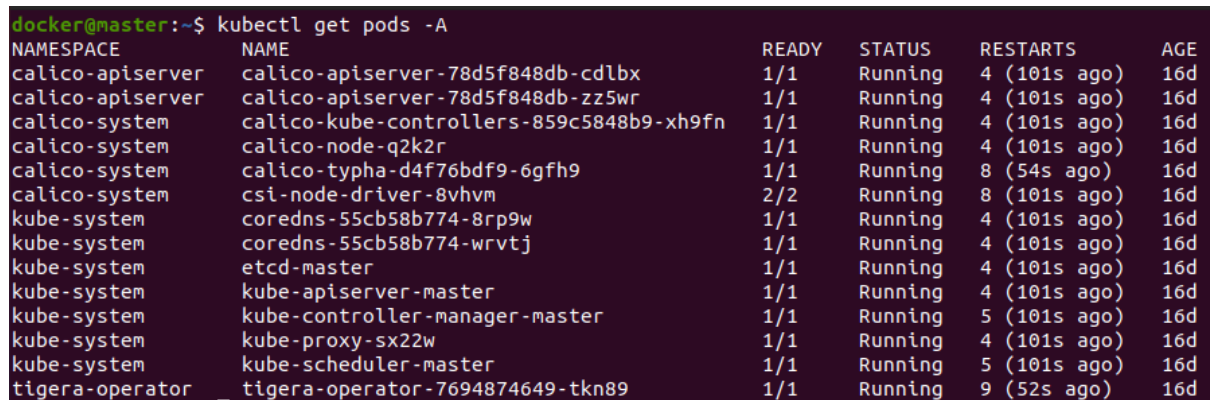
Figura 10: Configuración de IP archivo yaml custom resources.

Después de esta configuración se guarda el archivo y se ejecuta el documento o archivo .yaml

```
kubectl create -f custom-resources.yaml
```

Después de ejecutar los documentos se pueden ver los pods que están ejecutando y que requiere kubeadm para realizar conectividad y gestión en el clúster.

```
kubectl get pods -A
```



NAMESPACE	NAME	READY	STATUS	RESTARTS	AGE
calico-apiserver	calico-apiserver-78d5f848db-cdlbx	1/1	Running	4 (101s ago)	16d
calico-apiserver	calico-apiserver-78d5f848db-zz5wr	1/1	Running	4 (101s ago)	16d
calico-system	calico-kube-controllers-859c5848b9-xh9fn	1/1	Running	4 (101s ago)	16d
calico-system	calico-node-q2k2r	1/1	Running	4 (101s ago)	16d
calico-system	calico-typha-d4f76bdf9-6gfh9	1/1	Running	8 (54s ago)	16d
calico-system	csi-node-driver-8vhvm	2/2	Running	8 (101s ago)	16d
kube-system	coredns-55cb58b774-8rp9w	1/1	Running	4 (101s ago)	16d
kube-system	coredns-55cb58b774-wrvtj	1/1	Running	4 (101s ago)	16d
kube-system	etcd-master	1/1	Running	4 (101s ago)	16d
kube-system	kube-apiserver-master	1/1	Running	4 (101s ago)	16d
kube-system	kube-controller-manager-master	1/1	Running	5 (101s ago)	16d
kube-system	kube-proxy-sx22w	1/1	Running	4 (101s ago)	16d
kube-system	kube-scheduler-master	1/1	Running	5 (101s ago)	16d
tigera-operator	tigera-operator-7694874649-tnk89	1/1	Running	9 (52s ago)	16d

Figura 11: Pods por defecto nodo maestro kubeadm.

Cada pod que se observa en la figura 11 tiene la siguiente funcionalidad:

**calico-apiserver:** provee la REST API para calico y permite el manejo de apis usando kubectl en vez de calicoctl

**calico-kube-controllers:** se implementan en un clúster de Kubernetes. Los diferentes controladores monitorean la API de Kubernetes y realizan acciones basadas en el estado del clúster.

**calico-node:** es un componente de Kubernetes que se encarga de proporcionar redes y políticas de red para los pods que se encuentran dentro de un clúster.

**calico-typha:** El propósito principal es aumentar la escala reduciendo el impacto de cada nodo en el almacén de datos.

**csi-node:** Los controladores CSI generan información específica del nodo. En lugar de almacenarla en el objeto de la API de nodo de Kubernetes, se creó un nuevo objeto Kubernetes CSINode específico de CSI.

**coredns:** CoreDNS es un servidor de DNS flexible y extensible que puede servir como el DNS del clúster de Kubernetes.

**etcd:** es un almacén de claves-valores distribuido y fuertemente consistente que proporciona una forma confiable de almacenar datos que necesitan ser accedidos por un sistema distribuido o un clúster de máquinas.

Otros conceptos claves a tener en cuenta para el desarrollo de la práctica:

- **cadvisor:** Proporciona a los usuarios de contenedores una comprensión del uso de recursos y las características de rendimiento de sus contenedores en ejecución. Es un daemon en ejecución que recopila, agrega, procesa y exporta información sobre los contenedores en ejecución.
- **node-exporter:** Obtención de estadísticas de las aplicaciones.
- **configmap deployment:** Es un objeto de API que se utiliza para almacenar datos no confidenciales en pares clave-valor. Permite desacoplar la configuración específica del entorno de las imágenes del contenedor

- **pod service:** Es el objeto de la API de Kubernetes que describe cómo se accede a las aplicaciones.
- **daemonset:** Garantiza que todos los nodos ejecuten una copia de un Pod.
- **nodeport:** Expone el Service en cada IP del nodo en un puerto estático.
- **serviceaccount:** Proporciona una identidad distinta en un clúster de Kubernetes. Los Pods de aplicaciones, los componentes del sistema y las entidades dentro y fuera del clúster pueden usar las credenciales de una ServiceAccount específica para identificarse como esa ServiceAccount.

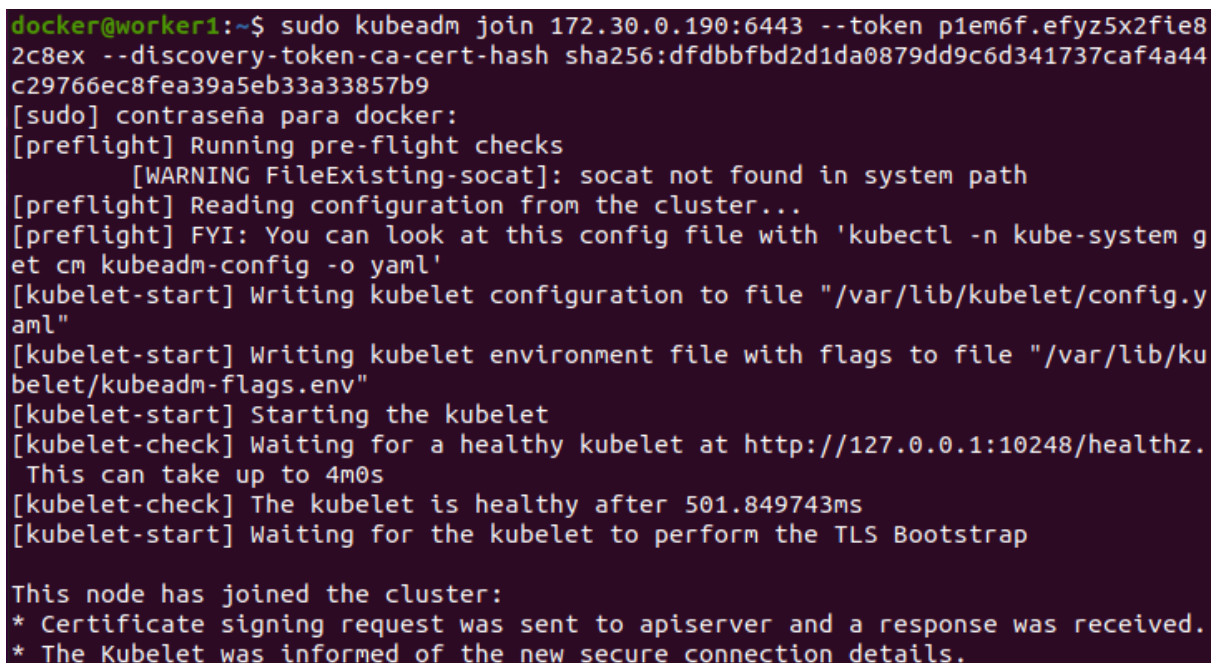
### Inicialización y enlace del nodo trabajador.

Ahora se ejecuta el nodo trabajador como se mencionó anteriormente. Este se instala del mismo método que en el caso del worker, así que, como primera medida se genera el token para conectar al nodo maestro y al nodo trabajador.

```
kubeadm token create --print-join-command
```

Este token genera un comando o serial el cual se debe ejecutar en el nodo trabajador para que este se conecte al nodo maestro.

```
sudo kubeadm join 172.30.0.190:6443 --token p1em6f.efyz5x2fie82c8ex
--discovery-token-ca-cert-hash
sha256:dfdbbfbd2d1da0879dd9c6d341737caf4a44c29766ec8fea39a5eb33a33857b9
```



```
docker@worker1:~$ sudo kubeadm join 172.30.0.190:6443 --token p1em6f.efyz5x2fie8
2c8ex --discovery-token-ca-cert-hash sha256:dfdbbfbd2d1da0879dd9c6d341737caf4a44
c29766ec8fea39a5eb33a33857b9
[sudo] contraseña para docker:
[preflight] Running pre-flight checks
[WARNING FileExisting-socat]: socat not found in system path
[preflight] Reading configuration from the cluster...
[preflight] FYI: You can look at this config file with 'kubectl -n kube-system g
et cm kubeadm-config -o yaml'
[kubelet-start] Writing kubelet configuration to file "/var/lib/kubelet/config.y
aml"
[kubelet-start] Writing kubelet environment file with flags to file "/var/lib/ku
belet/kubeadm-flags.env"
[kubelet-start] Starting the kubelet
[kubelet-check] Waiting for a healthy kubelet at http://127.0.0.1:10248/healthz.
This can take up to 4m0s
[kubelet-check] The kubelet is healthy after 501.849743ms
[kubelet-start] Waiting for the kubelet to perform the TLS Bootstrap

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.
```

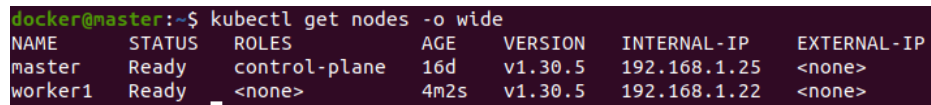
Figura 12: Vinculación del nodo trabajador al nodo maestro.

Ahora desde el nodo maestro se pueden ver los nodos trabajadores que tienen conectados, en especial, `-o wide` muestra los nodos y el número de IP interna del nodo maestro y nodo



worker la ip del nodo worker debe estar presente para poder acceder a los servicios de grafana, prometheus y el servidor de videostreaming nginx.

```
kubectl get pods -o wide
```



```
docker@master:~$ kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP
master	Ready	control-plane	16d	v1.30.5	192.168.1.25	<none>
worker1	Ready	<none>	4m2s	v1.30.5	192.168.1.22	<none>

Figura 13: Nodos totales del cluster

## Despliegue de servicios en el clúster.

Antes de desplegar los servicios se debe crear el namespace. Para el caso de los archivos yaml proporcionados a continuación es el namespace monitoring, pero se puede modificar siempre y cuando esté presente el nombre en los archivos yaml.

```
kubectl create namespace monitoring
```

Inicialmente, se obtiene el archivo .yaml del servicio de prometheus. Este compone de un configmap, un deployment y un service. El archivo se puede descargar desde el repositorio presente en la sección de anexos

Ahora, se ejecuta el archivo .yaml de prometheus creando el pod del mismo para este comando el terminal debe estar en la misma carpeta donde se encuentre el archivo.

```
kubectl apply -f prometheus-server.yaml
```

Para comprobar que la aplicación del archivo se ejecutan los siguientes comandos que muestran la información de que pods, deployment, configmap y servicios se encuentran en el namespace monitoring. El service define en qué puerto está escuchando el servicio, para este se emplea nodeport que permite redirigir el tráfico interno de ese puerto en kubeadm a la máquina anfitriona: Para acceder a este se debe usar la IP del nodo worker y el puerto, por ejemplo 172.30.0.189:30909

Del comando (kubectl get pods -n monitoring -o wide) la parte de (-o wide) permite en los pods ver a qué nodo trabajador fue asignado.

```
kubectl get pods -n monitoring -o wide
```

```
kubectl get deployment -n monitoring
```

```
kubectl get configmap -n monitoring
```

```
kubectl get service -n monitoring
```

```

docker@master:~$ kubectl get pods -n monitoring -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE   NOMINATED NODE   READINESS GATES
prometheus-deployment-7dbdd68697-5j94j  1/1     Running   0           4m43s  10.244.235.131  worker1  <none>           <none>
docker@master:~$ kubectl get deployment -n monitoring
NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
prometheus-deployment              1/1     1             1           4m57s
docker@master:~$ kubectl get configmap -n monitoring
NAME                                DATA   AGE
kube-root-ca.crt                   1       6m58s
prometheus-config                  1       5m12s
docker@master:~$ kubectl get service -n monitoring
NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
prometheus-service                 NodePort    10.109.50.132 <none>        9090:30909/TCP   5m35s
docker@master:~$

```

Figura 13. Pods, deployment, configmap y service de prometheus

Ahora se obtiene el archivo .yaml para Grafana, este servicio consta de solo el deployment y el servicio. En los anexos se encuentra el repositorio con cada uno de los códigos empleados para este proyecto.

Para ejecutar el servicio de Grafana se debe tener presente que el terminal esté en la misma carpeta que el archivo.

```
kubectl apply -f grafana-deployment.yaml
```

Para comprobar que el pod y el servicio están iniciados se ejecutan los siguientes comandos. En este servicio también se usa nodeport y se asigna en el puerto 30300, para acceder externamente se usa la IP del nodo trabajador y el puerto, por ejemplo 172.30.0.189:30300

```
kubectl get pods -n monitoring -o wide
```

```
kubectl get service -n monitoring
```

```

docker@master:~$ kubectl get pods -n monitoring -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP             NODE   NOMINATED NODE   READINESS GATES
grafana-deployment-79c84d99b5-45w49  1/1     Running   0           14s   10.244.235.132  worker1  <none>           <none>
prometheus-deployment-7dbdd68697-5j94j  1/1     Running   0           16m   10.244.235.131  worker1  <none>           <none>
docker@master:~$ kubectl get service -n monitoring
NAME                                TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
grafana-service                     NodePort    10.104.237.71 <none>        3000:30300/TCP   23s
prometheus-service                 NodePort    10.109.50.132 <none>        9090:30909/TCP   16m
docker@master:~$

```

Figura 14. Pods y service de Grafana.

Para el servicio de cadvisor se obtiene el archivo .yaml, por defecto este proporciona un daemonset y un service, este se encarga de proporcionar métricas de contenedor a prometheus, ya que este no tiene acceso a los contenedores del clúster. El archivo se puede descargar desde el repositorio presente en la sección de anexos

Ahora, se ejecuta el servicio. Se debe tener presente que el terminal esté en la misma carpeta que el archivo.

```
kubectl apply -f cadvisor.yaml
```

Para comprobar que el daemonset, pod y el servicio están iniciados se ejecutan los siguientes comandos. En esta configuración, no hay acceso a este servicio, porque no se usó un nodeport, ya que prometheus si puede acceder a él a través de su red interna y este se encarga de organizar las métricas que proporciona.

```
kubectl get daemonset -n monitoring
```

```
kubectl get pods -n monitoring -o wide
```

```
kubectl get service -n monitoring
```

```
docker@master:~$ kubectl get daemonset -n monitoring
NAME      DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
cadvisor  1          1          1          1             1           <none>          32s

docker@master:~$ kubectl get pods -n monitoring -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
cadvisor-vpzj8                      1/1     Running   0           45s   10.244.235.133   worker1   <none>            <none>
grafana-deployment-79c84d99b5-45w49 1/1     Running   0           27m   10.244.235.132   worker1   <none>            <none>
prometheus-deployment-7dbdd68697-5j94j 1/1     Running   0           43m   10.244.235.131   worker1   <none>            <none>

docker@master:~$ kubectl get service -n monitoring
NAME            TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
cadvisor-service  ClusterIP   10.96.99.191 <none>        8080/TCP         59s
grafana-service  NodePort    10.104.237.71 <none>        3000:30300/TCP   28m
prometheus-service NodePort    10.109.50.132 <none>        9090:30909/TCP   43m
```

Figura 15. Daemonset, pods y service de cadvisor.

Para el último servicio de monitoreo node-exporter se obtiene el archivo .yaml, este es similar a cadvisor con la diferencia que este contiene un serviceaccount, daemonset y service.

El archivo se puede descargar desde el repositorio presente en la sección de anexos

Ahora se ejecuta el servicio se debe tener presente que el terminal esté en la misma carpeta que el archivo.

```
kubectl apply -f node-exporter.yaml
```

Para comprobar que el daemonset, pod y el servicio están iniciados se ejecutan los siguientes comandos en esta configuración a este servicio no se tiene acceso porque no se usó un nodeport, ya que prometheus si puede acceder a él a través de su red interna y este se encarga de organizar las métricas que proporciona.

```
kubectl get daemonset -n monitoring
```

```
kubectl get pods -n monitoring -o wide
```

```
kubectl get service -n monitoring
```

```
kubectl get serviceaccount -n monitoring
```

```

docker@master:~$ kubectl get daemonset -n monitoring
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
cadvisor      1         1         1       1             1           <none>          11m
node-exporter 1         1         0       1             0           <none>          6s
docker@master:~$ kubectl get daemonset -n monitoring
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
cadvisor      1         1         1       1             1           <none>          11m
node-exporter 1         1         1       1             1           <none>          10s
docker@master:~$ kubectl get pods -n monitoring -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE     NOMINATED NODE   READINESS GATES
cadvisor-vpzj8                      1/1    Running   0           11m   10.244.235.133   worker1   <none>            <none>
grafana-deployment-79c84d99b5-45w49 1/1    Running   0           38m   10.244.235.132   worker1   <none>            <none>
node-exporter-rtlr7                 1/1    Running   0           20s   192.168.1.22     worker1   <none>            <none>
prometheus-deployment-7dbdd68697-5j94j 1/1    Running   0           54m   10.244.235.131   worker1   <none>            <none>
docker@master:~$ kubectl get service -n monitoring
NAME          TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
cadvisor-service ClusterIP   10.96.99.191 <none>        8080/TCP         11m
grafana-service NodePort    10.104.237.71 <none>        3000:30300/TCP   38m
node-exporter-service ClusterIP   10.110.150.170 <none>        9100/TCP         30s
prometheus-service NodePort    10.109.50.132 <none>        9090:30909/TCP   54m
docker@master:~$ kubectl get serviceaccount -n monitoring
NAME          SECRETS   AGE
default       0         56m
node-exporter 0         47s
docker@master:~$

```

Figura 16. Daemonset, pods, service y serviceaccount de node exporter.

Y para finalizar los servicios requeridos se ejecuta el archivo .yaml que tiene el servicio de video streaming, la imagen usada en el archivo tiene ya instalado un video si se quiere usar otro video se debe crear la imagen con un dockerfile o utilizar la imagen nativa de tiangolo/nginx-rtmp y modificar está en el archivo yaml en la línea 19. El archivo se puede descargar desde el repositorio presente en la sección de anexos.

```

16 spec:
17   containers:
18   - name: nginx
19     image: docker.io/cars213/servidor_hub:latest
20     imagePullPolicy: IfNotPresent
21     resources:
22       limits:
23         cpu: "1.2"
24         memory: "3Gi"
25       requests:
26         cpu: "0.9"
27         memory: "2Gi"
28     ports:
29     - containerPort: 1935
30   ---

```

Figura 17. Modificación de imagen en el archivo yaml de nginx.

Después se ejecuta el servicio de videostreaming.

```
kubectl apply -f nginx-deployment.yaml
```

Para comprobar que el pod y el servicio están iniciados se ejecutan los siguientes comandos. En este servicio también se usa nodeport y se asigna en el puerto 31935 para acceder externamente se usa la IP del nodo trabajador y el puerto por ejemplo (172.30.0.189:31935).

```
kubectl get deployment -n monitoring
```

```
kubectl get pods -n monitoring -o wide
```

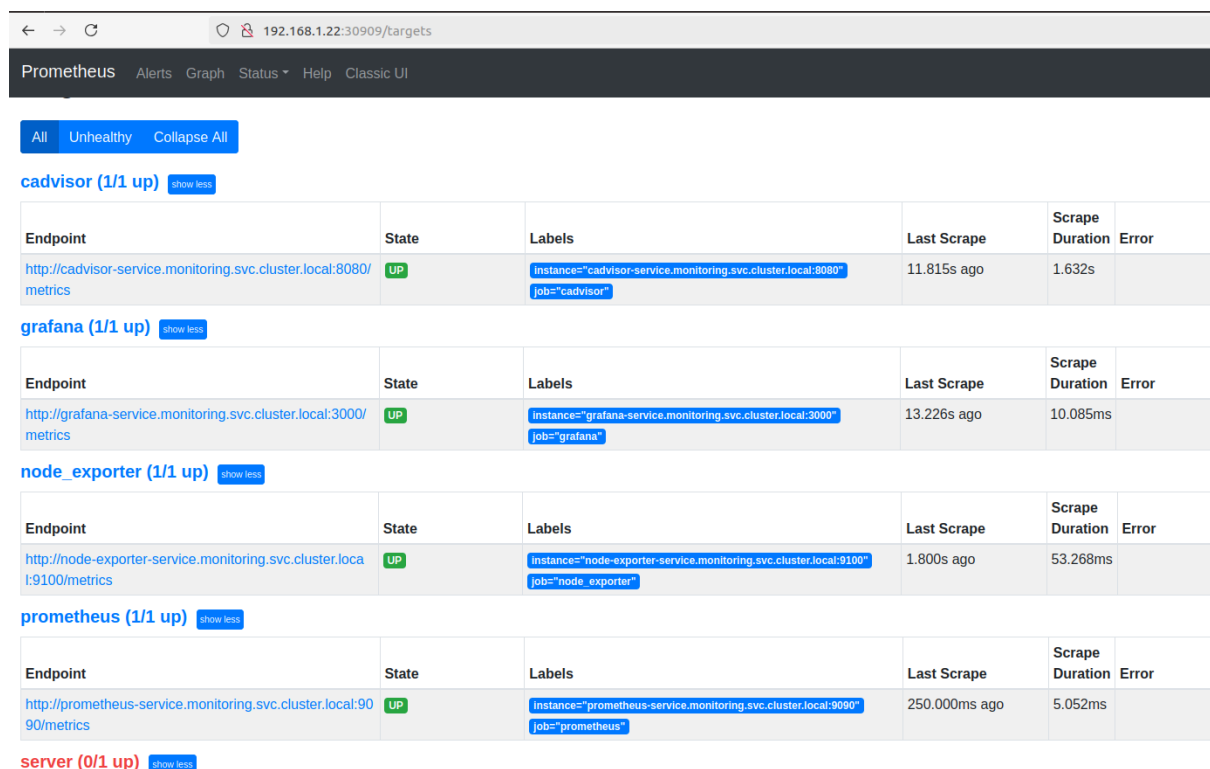
```
kubectl get service -n monitoring
```

```
docker@master:~$ kubectl get deployment -n monitoring
NAME                READY   UP-TO-DATE   AVAILABLE   AGE
grafana-deployment  1/1     1             1           49m
nginx-deployment    1/1     1             1           2m28s
prometheus-deployment 1/1     1             1           65m
docker@master:~$ kubectl get pods -n monitoring -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP            NODE     NOMINATED NODE   READINESS GATES
cadvisor-vpzj8      1/1     Running   0           22m   10.244.235.133 worker1   <none>            <none>
grafana-deployment-79c84d99b5-45w49 1/1     Running   0           49m   10.244.235.132 worker1   <none>            <none>
nginx-deployment-76654f99b7-qsgxl 1/1     Running   0           2m37s 10.244.235.134 worker1   <none>            <none>
node-exporter-rtlr7 1/1     Running   0           11m   192.168.1.22  worker1   <none>            <none>
prometheus-deployment-7dbdd68697-5j94j 1/1     Running   0           65m   10.244.235.131 worker1   <none>            <none>
docker@master:~$ kubectl get service -n monitoring
NAME                TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
cadvisor-service    ClusterIP   10.96.99.191    <none>        8080/TCP         22m
grafana-service      NodePort    10.104.237.71   <none>        3000:30300/TCP   49m
nginx-service        NodePort    10.99.192.77    <none>        1935:31935/TCP   2m44s
node-exporter-service ClusterIP   10.110.150.170 <none>        9100/TCP         11m
prometheus-service   NodePort    10.109.50.132   <none>        9090:30909/TCP   65m
docker@master:~$
```

Figura 18. Deployment, pods y service de nginx.

Todos los servicios fueron asignados a un solo nodo trabajador, cuando se usan más nodos el sistema asigna a cualquiera nodo trabajador.

Como ultima medida se accede a la página de prometheus con la IP interna del nodo trabajador por ejemplo <http://192.168.1.22:30909/targets>



The screenshot shows the Prometheus web interface at the URL [192.168.1.22:30909/targets](http://192.168.1.22:30909/targets). The interface includes a navigation bar with 'Prometheus', 'Alerts', 'Graph', 'Status', 'Help', and 'Classic UI'. Below the navigation bar, there are buttons for 'All', 'Unhealthy', and 'Collapse All'. The main content area displays a list of targets, each with a status indicator (UP or DOWN) and a 'show less' button. The targets are grouped by service: cadvisor (1/1 up), grafana (1/1 up), node\_exporter (1/1 up), and prometheus (1/1 up). Each target entry includes a table with columns for Endpoint, State, Labels, Last Scrape, Scrape Duration, and Error.

Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
<a href="http://cadvisor-service.monitoring.svc.cluster.local:8080/metrics">http://cadvisor-service.monitoring.svc.cluster.local:8080/metrics</a>	UP	instance="cadvisor-service.monitoring.svc.cluster.local:8080" job="cadvisor"	11.815s ago	1.632s	
<a href="http://grafana-service.monitoring.svc.cluster.local:3000/metrics">http://grafana-service.monitoring.svc.cluster.local:3000/metrics</a>	UP	instance="grafana-service.monitoring.svc.cluster.local:3000" job="grafana"	13.226s ago	10.085ms	
<a href="http://node-exporter-service.monitoring.svc.cluster.local:9100/metrics">http://node-exporter-service.monitoring.svc.cluster.local:9100/metrics</a>	UP	instance="node-exporter-service.monitoring.svc.cluster.local:9100" job="node_exporter"	1.800s ago	53.268ms	
<a href="http://prometheus-service.monitoring.svc.cluster.local:9090/metrics">http://prometheus-service.monitoring.svc.cluster.local:9090/metrics</a>	UP	instance="prometheus-service.monitoring.svc.cluster.local:9090" job="prometheus"	250.000ms ago	5.052ms	

Figura 19. Targets de prometheus.

Para que prometheus pueda tomar las métricas se requiere que todas estén en **up** si están en **down** o de color rojo quiere decir que hay un problema de conexión interno de los pods, en

el caso del servidor, que aparezca en rojo no significa que esté mal porque este no proporciona métricas http entonces prometheus no puede conectarse a él.

## Transmisión de video streaming nginx

Como primera medida se debe acceder al contenedor que fue creado anteriormente, para esto se ejecuta el siguiente comando.

```
kubectl exec -it nombre-del-pod -n monitoring -- /bin/bash
```

Después de acceder al contenedor, ejecutar el comando para acceder a la carpeta raíz y revisar si el video se encuentra, estos pasos solo se hacen si se utiliza la misma imagen que está por defecto en el archivo yaml, si se utiliza otra deben tener presente la configuración dada a la imagen creada.

```
cd
```

```
ls
```

```
docker@master:~$ kubectl exec -it nginx-deployment-76654f99b7-qsxgl -n monitoring -- /bin/bash
root@nginx-deployment-76654f99b7-qsxgl:/# cd
root@nginx-deployment-76654f99b7-qsxgl:~# ls
Video.mp4
root@nginx-deployment-76654f99b7-qsxgl:~#
```

Figura 20. Acceso al contenedor de nginx.

Por último, se ejecuta el comando para repetir el video.mp4 alojado en el contenedor.

```
ffmpeg -re -stream_loop -1 -i "Video.mp4" -c:v copy -c:a aac -ar 44100 -ac 2 -f flv rtmp://localhost/live/nginx
```

En el comando, en la sección de rtmp://localhost/live/nginx se puede cambiar la dirección, pero esta debe ser igual cuando se quiera recibir la transmisión en el VLC, si se quiere acceder desde una máquina externa, en la parte de localhost se reemplaza por la IP interna del nodo trabajador asignado al pod, por ejemplo rtmp://192.168.1.22:31935/live/nginx

```
ISO Media file produced by Google Inc.
Stream mapping:
  Stream #0:0 -> #0:0 (copy)
  Stream #0:1 -> #0:1 (aac (native) -> aac (native))
Press [q] to stop, [?] for help
Output #0, flv, to 'rtmp://localhost/live/nginx':
  Metadata:
    major_brand      : isom
    minor_version    : 512
    compatible_brands: isomiso6iso2avc1mp41
    encoder          : Lavf58.45.100
  Stream #0:0(und): Video: h264 (Main) ([7][0][0][0] / 0x00007), yuv420p(tv, bt709), 854x480 [SAR 1:1 DAR 427:240], q=2-31, 170 kb/s, 23.98 fps, 23.98 tbr, 1k tbn, 24k tbc (default)
  Metadata:
    handler_name     : ISO Media file produced by Google Inc.
  Stream #0:1(eng): Audio: aac (LC) ([10][0][0][0] / 0x0000A), 44100 Hz, stereo, fltp, 128 kb/s (default)
  Metadata:
    handler_name     : ISO Media file produced by Google Inc.
    encoder          : Lavc58.91.100 aac
frame= 341 fps= 24 q=1.0 size=      520kB time=00:00:14.18 bitrate= 300.2kbits/s speed= 1x
```

Figura 21. Transmisión de nginx.

Ahora para recibir la transmisión se dirige a la parte de Medio, en Abrir ubicación de red.

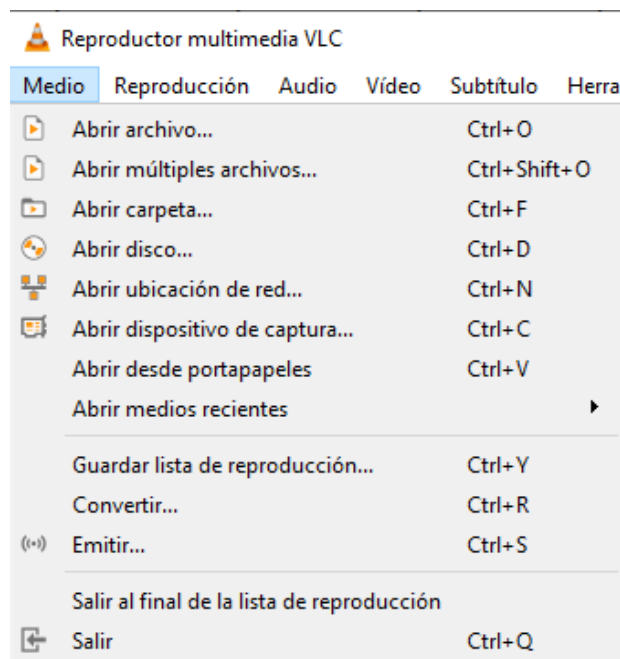


Figura 22. Abrir ubicación de red.

Después, se escribe la dirección de la transmisión, en este caso es `rtmp://192.168.1.22:31935/live/nginx`, cualquier computador con acceso a la red del worker puede recibir la transmisión.

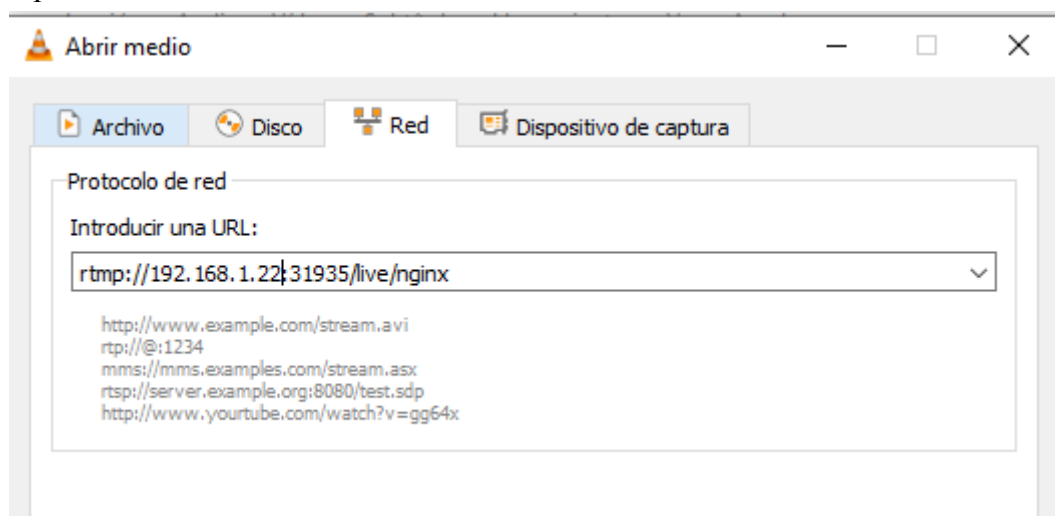


Figura 23. Ejemplo de URL para acceder al video streaming.

## Monitorización de Grafana y Prometheus

Para poder observar las métricas de Grafana se busca desde cualquier navegador la URL de la máquina anfitriona dirigida al puerto en el que opera Grafana `http://localhost:3000`. Por defecto, Grafana exige un usuario y una contraseña, que normalmente son “admin” para usuario y contraseña.

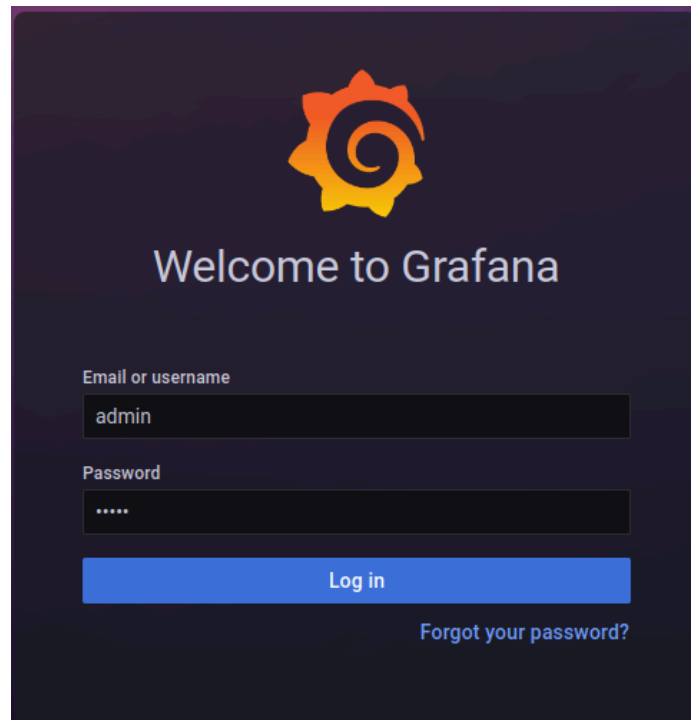


Figura 24. Inicio de sesión Grafana.

Para observar las métricas se necesita configurar a Grafana la URL de Prometheus, para la configuración se presiona la el botón remarcado en rojo y se selecciona la opción de datasource.

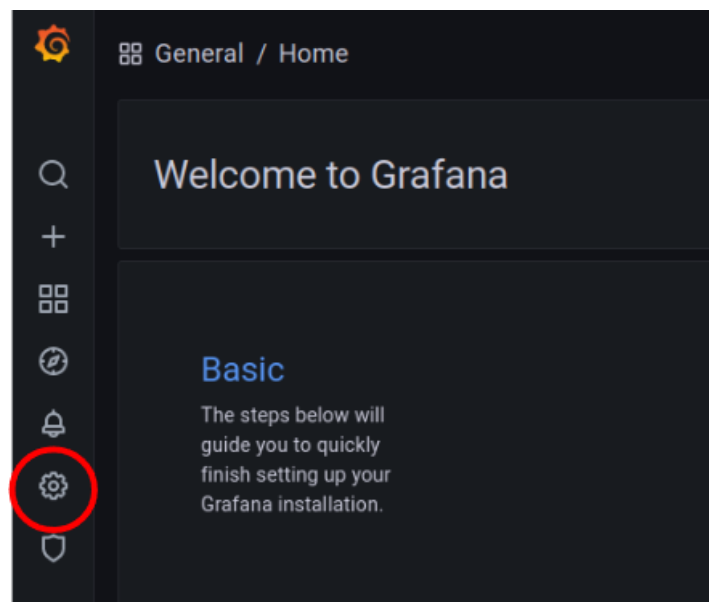


Figura 25: Panel lateral de Grafana.

Posteriormente, se presiona el botón `add data source`.



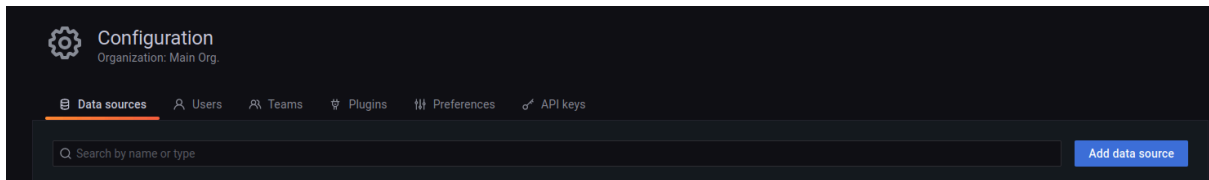


Figura 26: Configuración data source Grafana.

A continuación, en la Figura 27, se escribe la URL interna de Prometheus, en este caso es `prometheus-service.monitoring.svc.cluster.local:9090`.

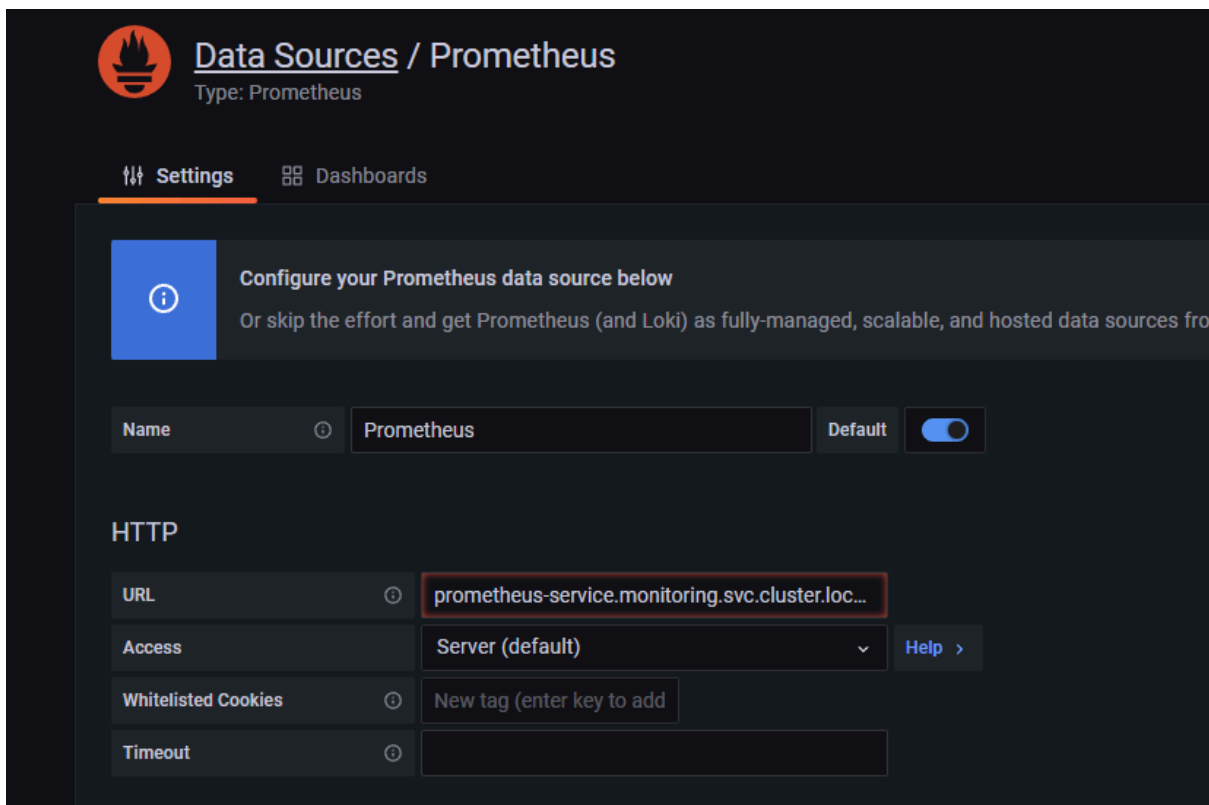


Figura 27: Parámetros para datasource Grafana

Para finalizar con la configuración de Prometheus se presiona en el botón `save & test`. Para verificar que Grafana accede a prometheus debe aparecer el recuadro en verde y con `data source is working`.

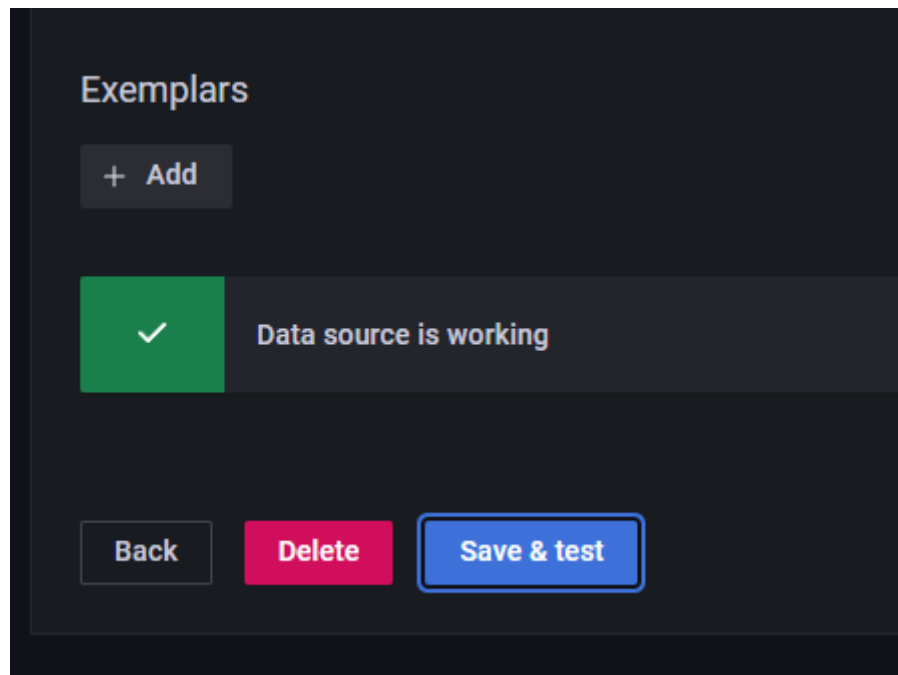


Figura 28: Guardar los cambios para el datasource de Grafana.

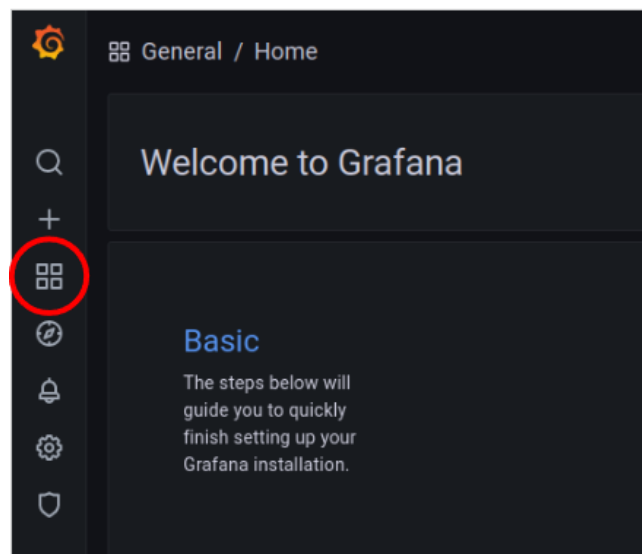


Figura 29: Panel lateral de Grafana.

Posteriormente en la Figura 30 se presiona el botón `import` ubicado en la esquina superior derecha [11].

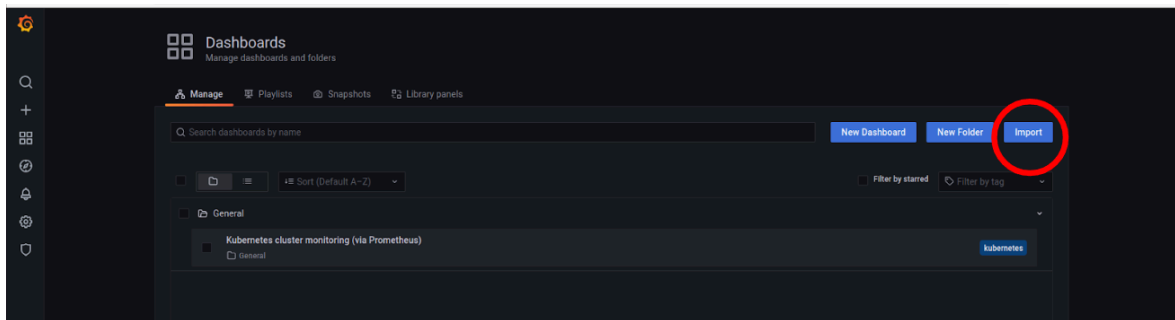


Figura 30: Importar Dashboard para Grafana

Seguidamente en la Figura 31 se escribe en el recuadro import via grafana.com el código 3119, que es una plantilla con configuración de métricas predeterminada por Grafana.

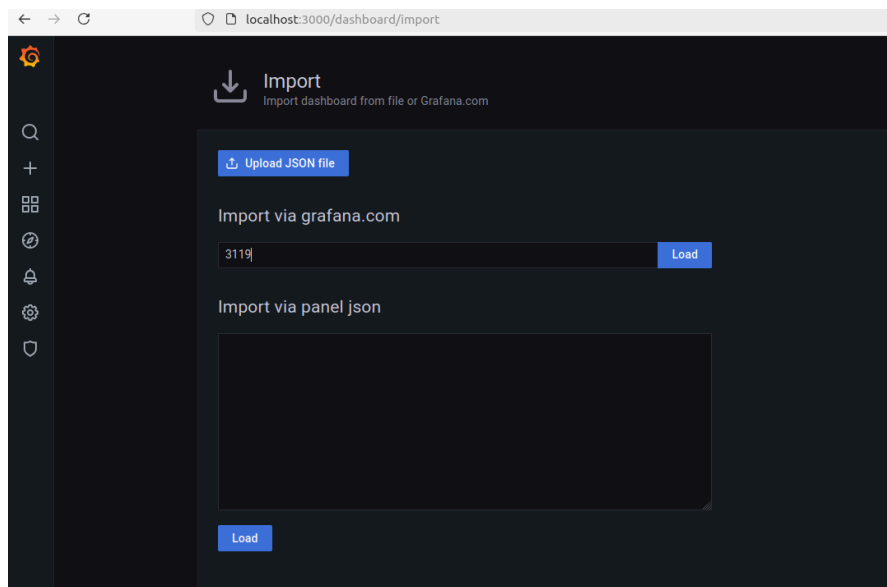


Figura 31: Importar la plantilla 3119 para el dashboard de Grafana.

Y finalmente en la Figura 32 se presiona el botón de importar.

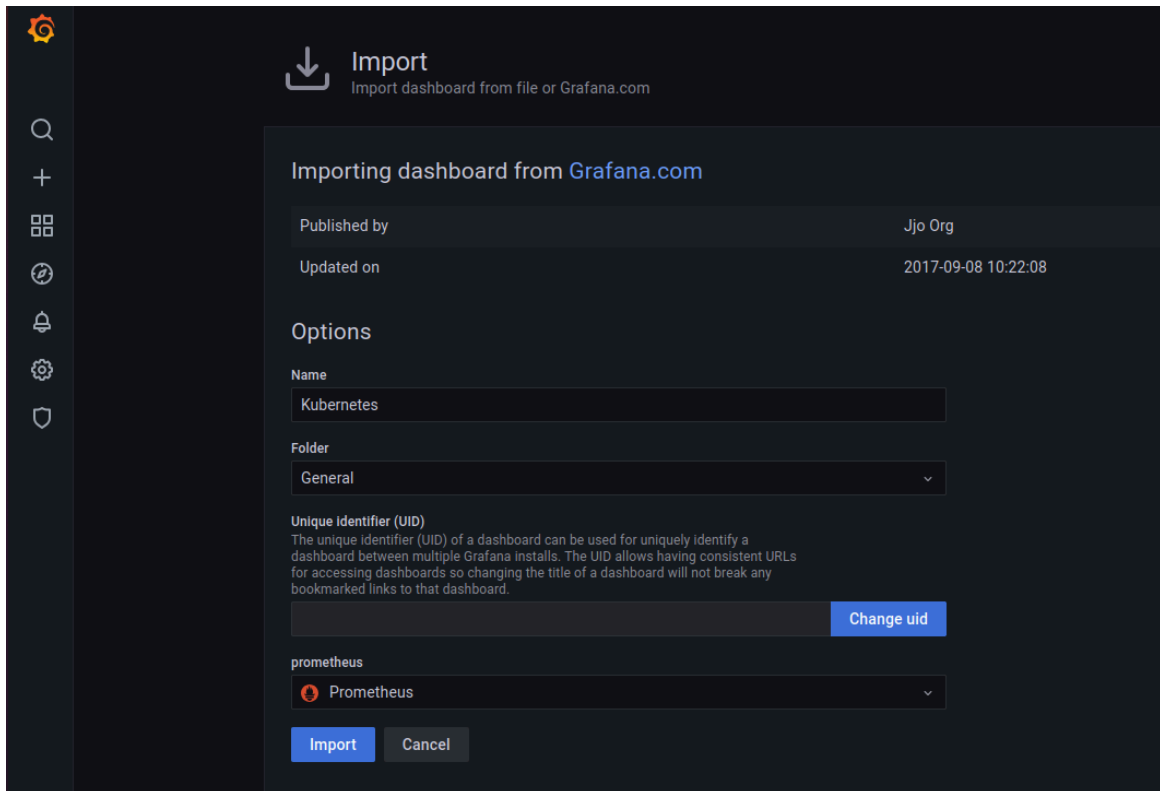


Figura 32. Importar el datasource de prometheus.

Al crear el dashboard, como se muestra en la Figura 33, Grafana arroja métricas de uso de CPU, memoria usada y uso de internet.

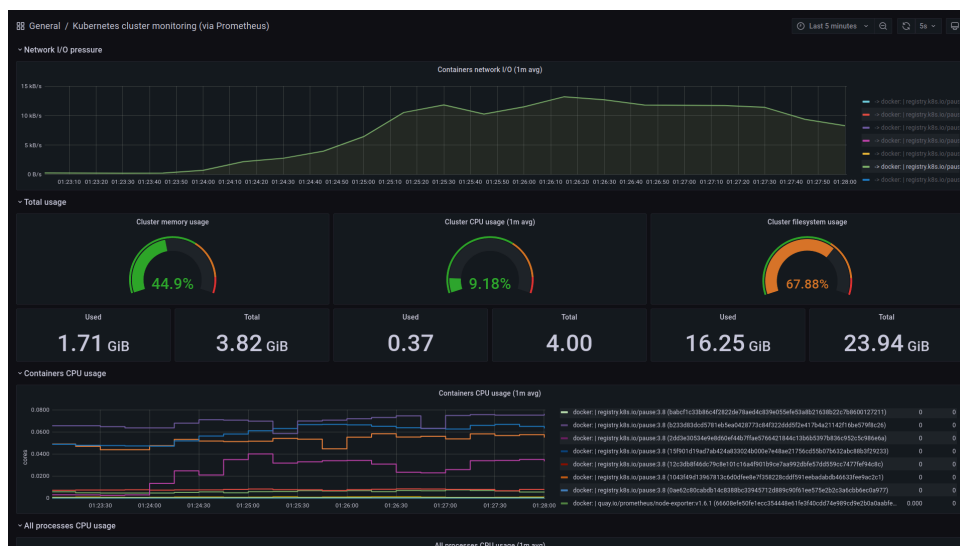


Figura 33: dashboard de Grafana

Para observar métricas de red se dirige al siguiente apartado en el dashboard normalmente las métricas aparecen con un código para saber qué servicio está asignado ese código se puede revisar en prometheus.

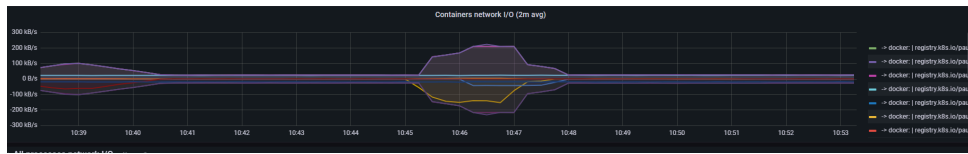


Figura 34 Métricas de red de contenedores

En la Figura 35 se muestra la arquitectura final del proyecto luego de haber creado la arquitectura de kubernetes y la implementación de los servicios:

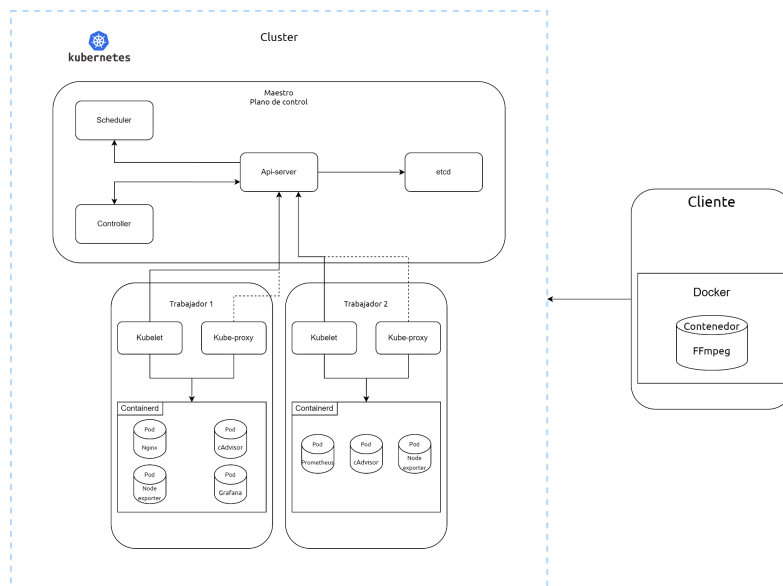


Figura 35: Componentes del clúster y cliente contenerizado.

## Conclusiones

- La integración de Kubernetes junto con herramientas como Prometheus y Grafana demostró ser una solución eficaz para la gestión y monitorización de servicios en una red simulada. Esta experiencia refuerza que estas tecnologías no solo son adecuadas para proyectos educativos y de pequeña escala, sino que también pueden escalarse para implementaciones en redes empresariales y entornos de producción más complejos.
- La configuración del clúster mediante Kubeadm, junto con el despliegue de servicios contenerizados, evidenció la flexibilidad de Kubernetes para soportar diferentes tipos de aplicaciones, incluyendo servidores de video streaming. Esto permite optimizar recursos y adaptar la infraestructura a las necesidades específicas de diferentes industrias, como telecomunicaciones, educación y comercio.
- El proyecto sentó las bases para implementar soluciones más robustas en el futuro. Aunque no se integró una red VPN en esta fase, los resultados obtenidos muestran

que Kubernetes y las herramientas complementarias tienen el potencial de ser escaladas e integradas con tecnologías avanzadas, lo que abre la puerta a explorar configuraciones más complejas y distribuidas en futuras iteraciones del proyecto

## **Anexos**

<https://github.com/seminario20242/CI-ster-basado-en-Kubeadm-y-monitorizaci-n-de-servicios/settings>