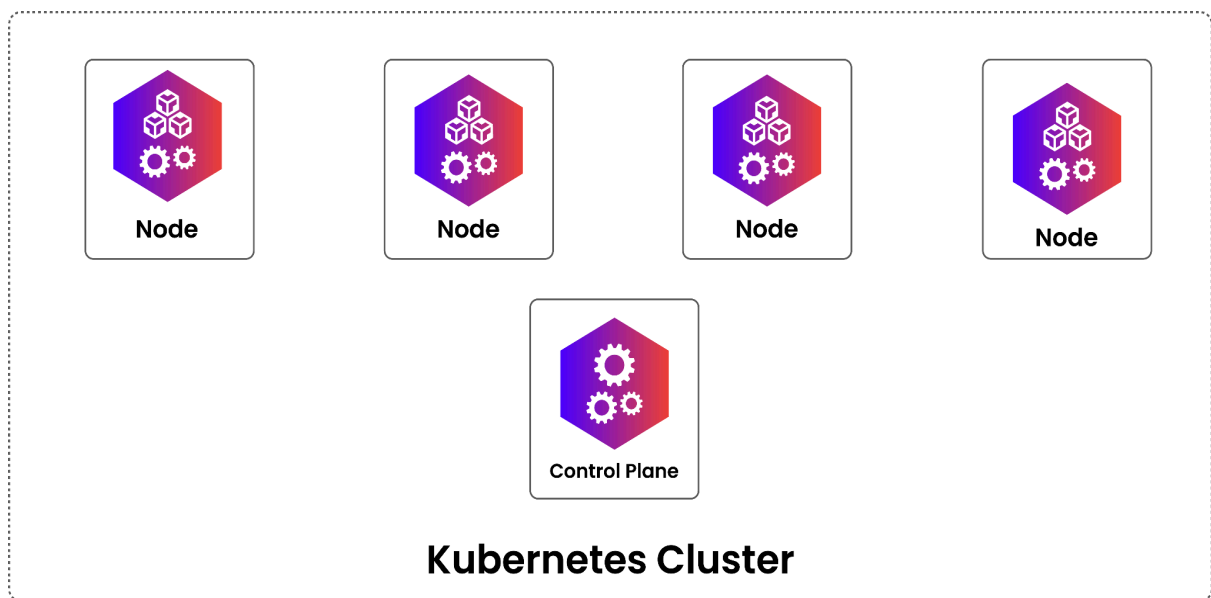


MONITORIZACIÓN DEL CLUSTER CON KUBERNETES



Steven Nata Zumbana

2º ASIR 2024-2025

9 de Junio 2025



ÍNDICE

INTRODUCCIÓN.....	3
OBJETIVOS.....	3
ESTADO DEL ARTE.....	4
PLANTEAMIENTO / ARQUITECTURA DEL PROYECTO.....	4
DESARROLLO DEL PROYECTO (PARTE EXPERIMENTAL).....	7
CONCLUSIONES.....	24
BIBLIOGRAFÍA Y REFERENCIAS.....	25
ANEXOS.....	26



INTRODUCCIÓN

Ya que durante este año ha habido bastante despliegue de aplicaciones, servicios que gestionar en red y ya tenía en mente la idea de hacer un cluster, al hacer las primeras búsquedas sobre esto encontré que existe Kubernetes para hacerlo, que alguna vez se mencionó en el curso, por lo que el trabajo tratará sobre la monitorización de un cluster con Kubernetes que se ha convertido en una de las principales plataformas para la orquestación de contenedores debido a su escalabilidad y capacidad de gestión automatizada.

El proyecto consiste en crear una infraestructura de un clúster con tres máquinas virtuales mediante Kubernetes, dentro desplegamos varios servicios, entre ellos los de monitorización, Prometheus, Grafana y Loki y una base de datos PostgreSQL con una interfaz gráfica de Adminer para administrar y con PostgreSQL Exporter para recoger las métricas. Para ello usaremos máquinas virtuales en VMware para conocer el programa como Virtualbox y simular el ecosistema y que sea lo más realista posible.

La documentación del proyecto contendrá la explicación de como se ha instalado, configurado y gestionado el cluster, volúmenes y redes, junto con la información de los servicios que se han utilizado para conseguir los objetivos

OBJETIVOS

El objetivo principal de este proyecto es desplegar y monitorizar una infraestructura de Kubernetes en un entorno de laboratorio, simulando un entorno real en producción, con herramientas que permitan observar el estado del cluster y de los servicios desplegados. Uno de los objetivos también es comprender el funcionamiento de un cluster y la infraestructura que tiene para que se use en el mundo de la informática. Al usar Kubernetes quiero aprender sobre esta plataforma y el tema de la orquestación con los contenedores que tiene el clúster. También reforzar el conocimiento en la implantación de servicios, ya que estamos en un entorno diferente a lo que he visto anteriormente.





Y por último con todo el conocimiento que consiga poder utilizarlo tanto a nivel personal como a nivel profesional.

ESTADO DEL ARTE

En el mercado existen este tipo de infraestructuras, para organizar servidores y controlarlo desde múltiples servidores físicos, grandes plataformas como Google Cloud, AWS o Azure ofrecen servicios gestionados de Kubernetes, aunque el uso de kubernetes es más utilizada para mejorar el rendimiento de las aplicaciones que usan y tener la mayor certeza de que si falla algo no suponga un gran inconveniente ya que kubernetes facilita la gestión del ecosistema.

Prometheus es uno de los sistemas más utilizados para recoger las métricas , junto con loki que utiliza un sistema de logs ligero y añadiendo a grafana que ofrece una interfaz con la visualización de la recolección de métricas mediante dashboards, esto se convierte en un gran conjunto de servicios para la monitorización.

En lo que destaca PostgreSQL es en el rendimiento y la robustez utilizada en sistemas críticos, por ello se usa Postgres Exporter para obtener las métricas internas y conocer el estado y comportamiento que tiene la base de datos. Y por último Adminer que es una alternativa ligera y rápida a otras, como PgAdmin o phpMyAdmin, para administrar el servicio desde un navegador.

PLANTEAMIENTO / ARQUITECTURA DEL PROYECTO

El proyecto se desarrolla en un entorno virtualizado mediante VMware Workstation, utilizando tres máquinas virtuales: una con Ubuntu 24.04 que actúa como nodo maestro, y dos con Ubuntu Server 24.04 configuradas como nodos worker. Todas las máquinas están conectadas en una red NAT proporcionada por VMware, que permite su comunicación tanto entre ellas como con el exterior.

En cada máquina se instalan los repositorios necesarios de Docker (para usar containerd como runtime) y Kubernetes (kubeadm, kubelet y kubectl) para su integración dentro del clúster. Tras inicializar y unir todos los nodos al clúster, se utiliza Helm como gestor de paquetes para desplegar distintos servicios esenciales.





Uno de los primeros componentes desplegados es Cilium, una CNI (Container Network Interface) que crea una red interna superpuesta y asigna una dirección IP a cada pod. Esta red permite la comunicación eficiente entre pods y nodos, y posibilita también exponer servicios al exterior cuando sea necesario.

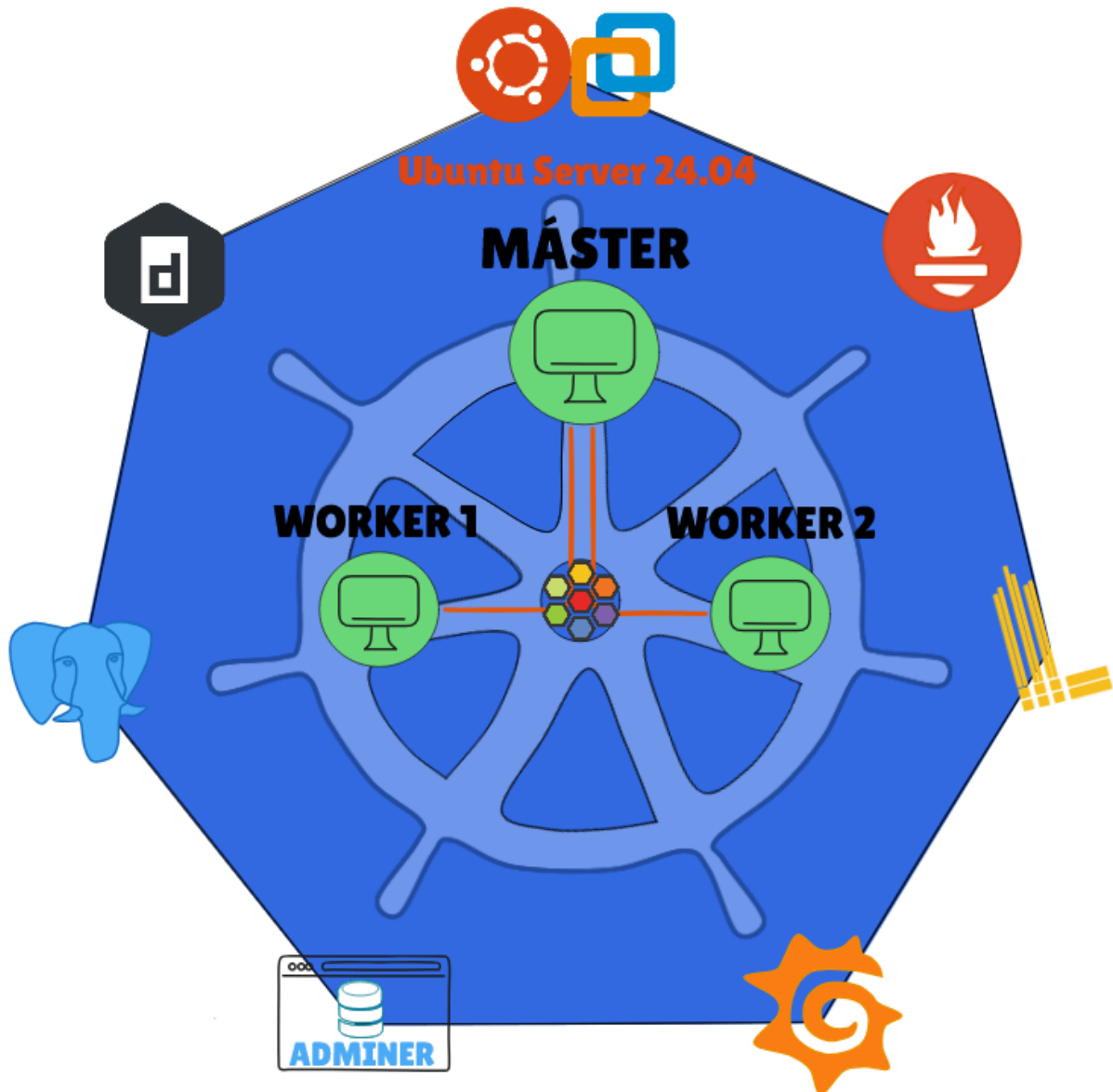
Una vez establecida la red, se instalan las herramientas de monitorización:

- Prometheus, encargado de recolectar métricas del clúster y los servicios desplegados.
- Grafana, que se integra con Prometheus para mostrar de forma visual y configurable las métricas mediante dashboards.
- Loki, que complementa la monitorización mediante la recolección y búsqueda de logs, también integrándose directamente con Grafana.

Como parte de la capa de persistencia del clúster, se despliega PostgreSQL como servicio de base de datos, que se integra con:

- PostgreSQL Exporter, el cual permite que Prometheus recoja métricas específicas del servidor de base de datos.
- Adminer, una interfaz web sencilla para acceder, visualizar y gestionar la base de datos manualmente.

Todas las herramientas y servicios se encuentran alojados dentro del clúster, y su instalación ha sido automatizada mediante scripts con archivos YAML y comandos Helm..



ARQUITECTURA DE LAS REDES

En cuanto a la arquitectura de red, el clúster utiliza dos redes diferenciadas:

- Una red NAT proporcionada por VMware, que asigna direcciones IP a las máquinas virtuales (nodos) y permite su comunicación externa e interna.
- Una red interna gestionada por el CNI (Container Network Interface) Cilium, que asigna direcciones IP a cada pod dentro del clúster. Además, crea una dirección IP virtual para cada nodo dentro de su red superpuesta, que se





utiliza para enrutar el tráfico entre los pods y facilitar la comunicación dentro del clúster.

CLUSTER KUBERNETES	LUBUNTU	UBUNTU SERVER 1	UBUNTU SERVER 2
RED NAT (192.168.145.0)	192.168.145.140	192.168.145.141	192.168.145.142
CILIUM (10.0.0.0)	10.0.0.48	10.0.1.83	10.0.3.51

ARQUITECTURA DE LA BASE DE DATOS

La base de datos se compone de cuatro tablas: producto, clientes, compra y compra_detalle, se relacionan a la hora de comprar cliente, compra y compra_detalle.

En cuanto a la administración de la base de datos:

POSTGRE SQL	PRODUCTO	CLIENTES	COMPRA	COMPRA_DETALLES
ADMIN (steven)	TODOS LOS PERMISOS	TODOS LOS PERMISOS	TODOS LOS PERMISOS	TODOS LOS PERMISOS
USU1	PERMISO WRITE	PERMISO WRITE	PERMISO WRITE	PERMISO WRITE
USU2	PERMISO READ	PERMISO READ	PERMISO READ	PERMISO READ

DESARROLLO DEL PROYECTO (PARTE EXPERIMENTAL)

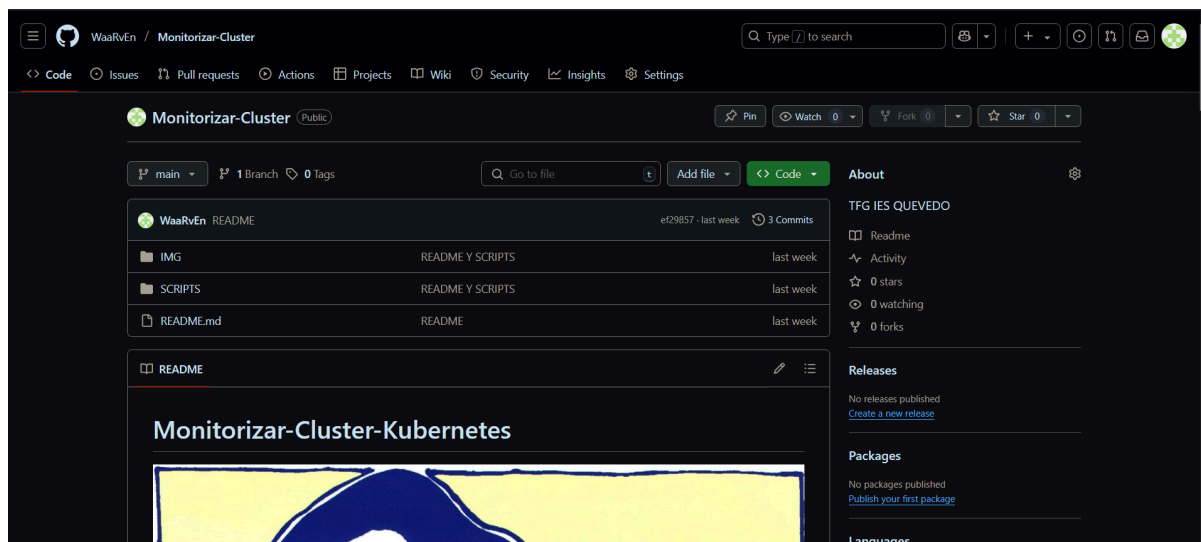
Lo primero que haremos al iniciar las máquinas virtuales de VMware será configurar los Lubuntu 24.04 y los Ubuntu Server 24.04, tanto la ip fija para que no haya





problemas con la red como cambiar el nombre del equipo en el archivo
/etc/hostname.

Luego clonamos el repositorio de github para empezar con la instalación a través de los scripts, pero primero moveremos los archivos a otra carpeta para guardar los archivos relacionados con la instalación



```
steven@MASTER:~/Monitorizar-Cluster/SCRIPTS$ ls
Conf_kube.sh  Crear_nodos.sh  servicios.sh
steven@MASTER:~/Monitorizar-Cluster/SCRIPTS$ mv Conf_kube.sh Crear_nodos.sh servicios.sh ../../Desktop/KUBE/
steven@MASTER:~/Monitorizar-Cluster/SCRIPTS$ cd !$
cd ../../Desktop/KUBE/
steven@MASTER:~/Desktop/KUBE$ ls
Conf_kube.sh  Crear_nodos.sh  servicios.sh
steven@MASTER:~/Desktop/KUBE$ sudo chmod +x Conf_kube.sh Crear_nodos.sh servicios.sh
steven@MASTER:~/Desktop/KUBE$ ls
Conf_kube.sh  Crear_nodos.sh  servicios.sh
steven@MASTER:~/Desktop/KUBE$
```

En el script de **Crear_nodos.sh** lo primero que hará será detectar si se ejecutan como root o no ya que es necesario ejecutar los comandos como root, luego los pasos previos para empezar a instalar las herramientas esenciales. Se desactiva el swap ya que Kubernetes asume completamente el uso de la memoria RAM, carga módulos necesarios para la comunicación entre pods y Crea un archivo de configuración para que el tráfico en los bridges sea visible para iptables y se permita el reenvío de paquetes (IP forwarding).





```
# Comprobar si se ejecuta como root

if [[ "$EUID" -ne 0 ]]; then
    echo -e "${rojo}Este script debe ejecutarse como root.${NC}"
    exit 1
fi

echo -e "${azul}-----REQUISITOS PREVIOS-----${NC}"
apt update && apt upgrade -y
apt install -y curl apt-transport-https git wget software-properties-common lsb-release ca-certificates socat

echo -e "${azul}-----DESACTIVAR SWAP-----${NC}"
swapoff -a
sed -i '/swap/s/^\(.*\)$/#\1/g' /etc/fstab

echo -e "${azul}-----CARGA MODULOS Y CONFIGURA SYSCTL-----${NC}"
modprobe overlay
modprobe br_netfilter

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

sysctl --system
```

Crea la carpeta para guardar las claves GPG y luego instalar containerd mediante los repositorios de docker, que gestiona el ciclo de vida de los contenedores, desde la descarga de la imagen hasta la ejecución y supervisión. A continuación se instala Kubernetes, primero se descarga y se guarda la clave del repositorio de Kubernetes, luego se añade el repositorio y al final instala y bloquea la versión específica de kubeadm, kubelet y kubectl.

kubeadm:

Es una herramienta de línea de comandos que simplifica el proceso de creación e inicialización de un clúster de Kubernetes.

kubelet:

Es un agente que se ejecuta como servicio en cada nodo del clúster de Kubernetes. Se encarga de gestionar los pods y los contenedores que se ejecutan en ese nodo, asegurando que estén funcionando según lo especificado en las configuraciones de los pods.





kubecttl:

Es la herramienta de línea de comandos que permite interactuar con el clúster de Kubernetes. Permite realizar tareas como crear, listar, eliminar y modificar objetos (pods, servicios, despliegues, etc.) del clúster.

Y después añadir al /etc/hosts la ip del nodo maestro con el nombre que elijamos para identificarlo.

```
echo -e "${azul}-----INSTALAR CONTAINERD-----${NC}"
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

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

apt-get update && apt-get install -y containerd.io

containerd config default | tee /etc/containerd/config.toml
sed -i 's/SystemdCgroup = false/SystemdCgroup = true/' /etc/containerd/config.toml
systemctl restart containerd

echo -e "${azul}-----INSTALAR KUBERNETES-----${NC}"
curl -fsSL https://pkgs.k8s.io/core:/stable:/v1.30/deb/Release.key | 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:/v1.30/deb/ /" \
| tee /etc/apt/sources.list.d/kubernetes.list

apt-get update
apt-get install -y kubeadm=1.30.1-1.1 kubectl=1.30.1-1.1 kubecttl=1.30.1-1.1
apt-mark hold kubelet kubeadm kubecttl

echo -e "${azul}-----AÑADIR HOST MAESTRO-----${NC}"
read -p "¿Que ip tiene el nodo maestro? " ip_maestro
read -p "¿nombre para identificarlo?" nombre_kube
echo "$ip_maestro $nombre_kube" >> /etc/hosts
```

Finalizando el script dependerá si lo ejecutamos en el nodo master o en un nodo worker, ya que si es en el nodo master ejecutará el comando que creará el cluster y mostrará el comando que se deberá ejecutar en el nodo worker con su token y hash correspondiente. Y si es el nodo worker simplemente colocar el token y hash para unir el nodo al cluster





```
read -p "¿QUÉ NODO ERES, master O worker? " nodo

#NODO MASTER

if [[ $nodo == "master" ]]
then

    echo -e "${azul}-----INICIAR EL CLUSTER CON KUBEADM-----${NC}"
    read -p "¿rango ip? ej:192.168.0.0/16" ip
    kubeadm init --pod-network-cidr=$ip --control-plane-endpoint=k8scp:6443

    echo "Token y hash del cluster"
    kubeadm token create --print-join-command

fi

#NODO WORKER

if [[ $nodo == "worker" ]]
then

    echo -e "${azul}-----UNIR EL WORKER AL CLUSTER-----${NC}"

    echo -e "${negrita}PUEDES CREAR EL COMANDO EJECUTANDO EN EL MASTER 'kubeadm token create --print-join-command'${NC}"

    read -p "token del cluster: " token

    read -p "hash del cluster: " hash

    kubeadm join k8scp:6443 --token $token --discovery-token-ca-cert-hash sha256:$hash

fi
```

El script **Conf_kube.sh** es exclusivo para el nodo master que configurará las herramientas de Kubernetes (kubeadm, kubelet y kubectl) para utilizarlas en el usuario donde se ejecute y autocompletando los comandos.

```
#!/bin/bash

negrita='\e[1m'
verde='\e[32m'
azul='\e[34m'
rojo='\e[31m'
NC='\e[0m'
amarillo='\e[33'

#Comprobar si se ejecuta como usuario
if [[ "$EUID" -eq 0 ]]; then
    echo -e "${rojo}Este script debe ejecutarse como un usuario.${NC}"
    exit 1
fi

echo -e "${azul}-----CONFIGURAR KUBECTL E INSTALAR AUTOCOMPLETADO-----${NC}"

mkdir -p $HOME/.kube

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

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

sudo apt-get install bash-completion -y

source <(kubectl completion bash)

echo 'source <(kubectl completion bash)' >> ~/.bashrc # persistir autocompletado
```





Y luego instalará helm descargando las claves GPG y añadiendo los repositorios HELM, al final instalará el CNI Cilium generando el archivo .yaml y luego aplicándolo al cluster.

```
echo -e "${azul}-----INSTALAR HELM-----${NC}"

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

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

sudo apt-get update

sudo apt-get install helm -y

echo -e "${azul}-----INSTALAR CILIUM, CNI QUE CONECTARA LOS PODS ENTRE SI-----${NC}"

helm repo add cilium https://helm.cilium.io/

helm repo update

helm template cilium cilium/cilium --version 1.16.1 \
--namespace kube-system > cilium.yaml

kubectl apply -f cilium.yaml
```

En el último script de Servicios.sh primero creará un local-path permitiendo que los volúmenes persistentes usen el disco local y lo marca como el almacenamiento por defecto.

```
#!/bin/bash
You, last week • README Y SCRIPTS

negrita='\e[1m'

verde='\e[32m'

rojo='\e[31m'

NC='\e[0m'

azul='\e[34m'

amarillo='\e[33'

echo -e "${azul}-----CREAR LOCAL-PATH-----${NC}"

kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml

kubectl patch storageclass local-path -p '{"metadata": {"annotations":{"storageclass.kubernetes.io/is-default-class":"true"}}}'
```

Luego empieza a instalar PostgreSQL donde primero crea el namespace donde estarán los demás servicios que usaremos, luego añade repositorios de bitnami necesario para instalarlo, después esperamos a que el servicio esté listo a través de un bucle “until”. A continuación se inicia la creación de la base de datos y usuarios, primero guarda la contraseña del usuario predeterminado que se crea y poder





ejecutar comandos dentro del contenedor de postgres, segundo pedirá los datos para establecer los nombres de la base de datos, usuario y contraseña

```
echo -e "${azul}-----INSTALAR POSTGRESQL-----${NC}"

kubectl create namespace monitoreo

helm repo add bitnami https://charts.bitnami.com/bitnami
helm repo update

helm install my-postgresql bitnami/postgresql --namespace monitoreo

echo -e "${azul} Esperando a que PostgreSQL esté listo...${NC}"
until kubectl get pods -n monitoreo | grep my-postgresql | grep -q 'Running'; do
  sleep 5
done
echo -e "${verde} PostgreSQL está listo.${NC}"

echo -e "${azul}-----CREACION DE BASE DE DATOS Y USUARIO-----${NC}"

#-----CONTRASEÑA DE POSTGRE-----

pswd_pg=$(kubectl get secret my-postgresql -n monitoreo -o jsonpath="{.data.postgres-password}" | base64 --decode)
#-----

read -p "nombre de la base de datos: " db_name
read -p "nombre de usuario: " usuario
read -s -p "contraseña usuario: " pswd_usu_pg

echo

kubectl exec my-postgresql-0 -n monitoreo -- bash -c "PGPASSWORD='${pswd_pg}' psql -U postgres -c \"
CREATE DATABASE ${db_name};\""

kubectl exec my-postgresql-0 -n monitoreo -- bash -c "PGPASSWORD='${pswd_pg}' psql -U postgres -c \"
CREATE USER ${usuario} WITH PASSWORD '${pswd_usu_pg}';
GRANT ALL PRIVILEGES ON DATABASE ${db_name} TO ${usuario};\""

kubectl exec my-postgresql-0 -n monitoreo -- bash -c \"
PGPASSWORD='${pswd_pg}' psql -U postgres -d ${db_name} -c \"
GRANT ALL PRIVILEGES ON SCHEMA public TO \\\"${usuario}\\\";
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO \\\"${usuario}\\\";
\\\""
```

Luego se instala POSTGRESQL EXPORTER donde se utilizará la contraseña y el nombre de la base de datos aplicando la estructura .yaml (Yet Another Markup Language) configurando los recursos que necesitamos del servicio. En la primera parte tenemos el tipo de Deployment para iniciar un contenedor con el nombre postgres-exporter con su imagen latest que usa DATA_SOURCE_NAME con credenciales para conectarse a PostgreSQL recién creado y lo expone métricas en el puerto 9187.

También hay un tipo Service el cuál expone a la red la aplicación del postgres exporter





```
echo -e "${azul}-----INSTALAR POSTGRESQL EXPORTER-----${NC}"

kubectl apply -f - <<EOF
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres-exporter
  namespace: monitoreo
spec:
  selector:
    matchLabels:
      app: postgres-exporter
  replicas: 1
  template:
    metadata:
      labels:
        app: postgres-exporter
    spec:
      containers:
        - name: postgres-exporter
          image: prometheuscommunity/postgres-exporter:latest
          env:
            - name: DATA_SOURCE_NAME
              value: "postgresql://postgres:${pswd_pg}@my-postgresql:5432/${db_name}?sslmode=disable"
          ports:
            - containerPort: 9187
              name: metrics
---
apiVersion: v1
kind: Service
metadata:
  name: postgres-exporter
  namespace: monitoreo
spec:
  selector:
    app: postgres-exporter
  ports:
    - protocol: TCP
      port: 9187
      targetPort: 9187
EOF
```

Después tenemos la instalación del Prometheus para las métricas de la base de datos, primero el el PersistentVolumeClaim para guardar en el almacenamiento de forma persistente las métricas de prometheus en local-path creado anteriormente. Luego tenemos ConfigMap donde se configura el prometheus y se definen los jobs de Prometheus para extraer los datos del postgre-exporter.



```
echo -e "${azul}-----INSTALAR PROMETHEUS Y CONFIGURACION DE MONITORIZACIÓN DB-----${NC}"

kubectyl apply -f - <<EOF
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: prometheus-data
  namespace: monitoreo
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: local-path
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
  namespace: monitoreo
data:
  prometheus.yml: |
    global:
      scrape_interval: 15s
    scrape_configs:
      - job_name: 'prometheus'
        scrape_interval: 5s
        static_configs:
          - targets: ['localhost:9090']
      - job_name: 'postgres-exporter-namespaceone'
        scrape_interval: 15s
        static_configs:
          - targets: [postgres-exporter.monitoreo.svc.cluster.local:9187]
      - job_name: 'postgres-exporter-namespacetwo'
        scrape_interval: 15s
        static_configs:
          - targets: [postgres-exporter.monitoreo.svc.cluster.local:9187]
```

Lo siguiente monta el volumen en el Deployment donde se inicia el contenedor con su imagen y las configuraciones de los volúmenes y ConfigMap hechos.





```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: prometheus
  namespace: monitoreo
spec:
  selector:
    matchLabels:
      app: prometheus
  replicas: 1
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
        - name: prometheus
          image: prom/prometheus:latest
          args:
            - "--auto-discover-databases"
          ports:
            - containerPort: 9090
          volumeMounts:
            - name: prometheus-data
              mountPath: /etc/prometheus
            - name: prometheus-config
              mountPath: /etc/prometheus/prometheus.yml
              subPath: prometheus.yml
          args:
            - "--config.file=/etc/prometheus/prometheus.yml"
      volumes:
        - name: prometheus-data
          persistentVolumeClaim:
            claimName: prometheus-data
        - name: prometheus-config
          configMap:
            name: prometheus-config
```

Y por último el Service que expone la aplicación en la red y con el bucle de espera para que la aplicación esté listo





```
---
apiVersion: v1
kind: Service
metadata:
  name: prometheus
  namespace: monitoreo
spec:
  selector:
    app: prometheus
  ports:
    - protocol: TCP
      port: 9090
      targetPort: 9090
  type: NodePort
EOF

echo -e "${azul} Esperando a que Prometheus-DB esté listo...${NC}"
until kubectl get pods -n monitoreo | grep prometheus | grep -q 'Running'; do
  sleep 5
done

echo -e "${verde} Prometheus-DB está listo.${NC}"
```

Aquí la otra forma de desplegar los servicios, a través del repositorio HELM, a través de los charts que es un paquete que tiene los archivos YAML para el despliegue de la aplicación y con el bucle hasta que esté activo

```
echo -e "${azul}-----INSTALAR PROMETHEUS CLUSTER-----${NC}"

helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo update
helm install prometheus prometheus-community/prometheus -n prometheus --create-namespace

echo -e "${azul} Esperando a que Prometheus-cluster esté listo...${NC}"
until kubectl get pods -n prometheus | grep prometheus | grep -q 'Running'; do
  sleep 5
done
echo -e "${verde} Prometheus-cluster está listo.${NC}"
```

Con la instalación de grafana es un poco diferente ya que le aplicamos unos valores con el archivo values.yaml para modificar la instalación, activa loki con un volumen para que los datos no se pierdan si se reiniciara el pod, activa promtail que recoge los logs de los pods y los envía a loki y las características de grafana que habilita que habilita la detección de los datasources configurados como loki. También habrá una espera hasta que se inicie el pod.





```
echo -e "${azul}-----INSTALAR GRAFANA-----${NC}"

helm repo add grafana https://grafana.github.io/helm-charts
helm repo update

cat <<EOF > values.yaml
loki:
  enabled: true
  persistence:
    enabled: true
    size: 1Gi

promtail:
  enabled: true

grafana:
  enabled: true
  sidecar:
    datasources:
      enabled: true
  image:
    tag: 10.2.5
EOF

helm install loki-stack grafana/loki-stack --values values.yaml -n monitoreo

echo -e "${negrita} Esperando a que Grafana esté listo...${NC}"
until kubectl get pods -n monitoreo | grep grafana | grep -q 'Running'; do
  sleep 5
done
echo -e "${verde} Grafana está listo.${NC}"
```

El último despliegue será Adminer donde primero describimos el tipo Service que expone a la red la aplicación Adminer con el puerto indicado y por último el Deployment para iniciar el contenedor con su imagen y el puerto que se le asigna al contenedor con el que podemos acceder desde fuera con IP del nodo y el puerto.



```
echo -e "${azul}-----AÑADIR ADMINER-----${NC}"
cat <<EOF > adminer.yaml
apiVersion: v1
kind: Service
metadata:
  name: adminer
  namespace: monitoreo
spec:
  type: NodePort
  selector:
    app: adminer
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
      nodePort: 32080
```

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: adminer
  namespace: monitoreo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: adminer
  template:
    metadata:
      labels:
        app: adminer
    spec:
      containers:
        - name: adminer
          image: adminer:latest
          ports:
            - containerPort: 8080
EOF

kubectl apply -f adminer.yaml
```

Finalmente en el script tendremos algunas aclaraciones de lo que se ha usado como los comandos de la creación de la base de datos, su archivo .sql para ejecutar los comandos y crear tanto nuevos usuarios como las tablas y en otro archivos tendremos contraseñas que deberíamos saber y los identificadores de dashboards que usaremos más adelante.





```
echo -e "${amarillo}=== COMANDOS QUE SE EJECUTARON EN POSTGRESQL ===${NC}"
echo -e "CREATE DATABASE ${db_name};"
echo -e "CREATE USER ${usuario} WITH PASSWORD '[oculto]';"
echo -e "GRANT ALL PRIVILEGES ON DATABASE ${db_name} TO ${usuario};"
echo -e "GRANT ALL ON SCHEMA public TO ${usuario};"
echo -e "GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO ${usuario};"

export GRAFANA_PASSWORD=`kubectl -n monitoreo get secret loki-stack-grafana -o jsonpath="{.data.admin-password}" | base64 -d`

echo -e "${azul} Usuarios para la base de datos${NC}"

read -p "nombre usuario1: " usuario1

read -s -p "contraseña usuario1: " pswd_usu1_pg

echo

read -p "nombre usuario2: " usuario2

read -s -p "contraseña usuario2: " pswd_usu2_pg

echo

cat > tablas.sql <<EOF
-- Crear base de datos (si se hace con postgres, se hace desde fuera con CREATE DATABASE ...)

-- Crear tablas
CREATE TABLE producto (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100),
  precio NUMERIC
);

CREATE TABLE clientes (
  id SERIAL PRIMARY KEY,
  nombre VARCHAR(100),
  email VARCHAR(100)
);

CREATE TABLE compra (
  id SERIAL PRIMARY KEY,
  cliente_id INT REFERENCES clientes(id),
  fecha DATE
);

CREATE TABLE compra_detalles (
  id SERIAL PRIMARY KEY,
  compra_id INT REFERENCES compra(id),
  producto_id INT REFERENCES producto(id),
  cantidad INT
);

-- Crear usuarios
CREATE USER ${usuario1} WITH PASSWORD '${pswd_usu1_pg}';
CREATE USER ${usuario2} WITH PASSWORD '${pswd_usu2_pg}';

-- Dar permisos WRITE a usu1
GRANT INSERT, UPDATE ON producto, clientes, compra, compra_detalles TO ${usuario1};

-- Dar permisos READ a usu2
GRANT SELECT ON producto, clientes, compra, compra_detalles TO ${usuario2};
EOF

echo -e """"contraseña grafana = $GRAFANA_PASSWORD

contraseña postgre = $pswd_pg

Dashboards Grafana:| You, 3 days ago • Mejora scripts ...

Prometheus: 18283 - 1860

Loki: 16966

PostgreSQL: 455 - 9628"""" > otros

echo -e "${amarillo} revisar archivo --> otros${NC}"
```



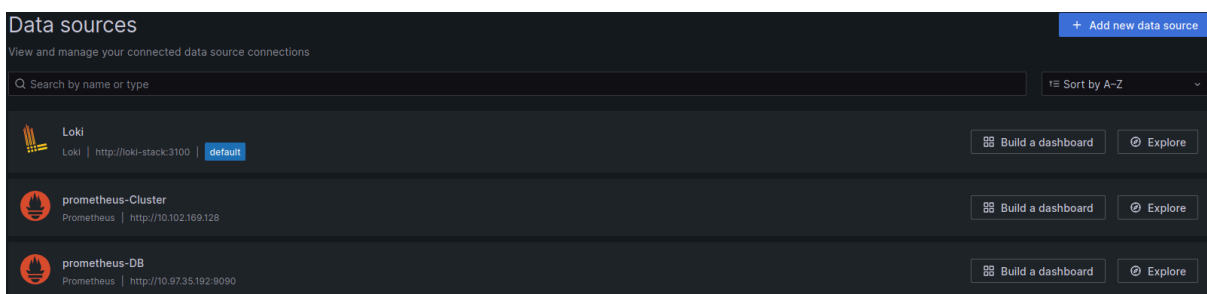


Una vez levantado los servicios iremos a comprobar que se han hecho bien los despliegues haciendo uso de los comandos necesarios con las herramientas de kubernetes. También podemos observar como Kubernetes ha distribuido los servicios en los nodos que hay en el cluster, tanto en el worker 1 como en el worker 2.

```
steven@MASTER:~/Desktop/KUBE$ kubectl get pods -n monitoreo -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
adminer-6ddb66fbd6-jc644            1/1     Running   0           8m41s 10.0.0.122      worker-2          <none>            <none>
loki-stack-0                         1/1     Running   0           10m    10.0.0.229      worker-2          <none>            <none>
loki-stack-grafana-76cbf4789f-m4pw4 2/2     Running   0           10m    10.0.2.87       worker-1          <none>            <none>
loki-stack-promtail-bxxfm           1/1     Running   0           10m    10.0.2.209      worker-1          <none>            <none>
loki-stack-promtail-l9m5k           1/1     Running   0           10m    10.0.0.83       worker-2          <none>            <none>
loki-stack-promtail-sm9b6           1/1     Running   0           10m    10.0.1.198      master            <none>            <none>
my-postgresql-0                     1/1     Running   0           12m    10.0.2.133      worker-1          <none>            <none>
postgres-exporter-8476c6b798-6pdzp 1/1     Running   0           11m    10.0.2.104      worker-1          <none>            <none>
prometheus-85ffbd59b8-vsdbb         1/1     Running   0           11m    10.0.0.91       worker-2          <none>            <none>
steven@MASTER:~/Desktop/KUBE$
```

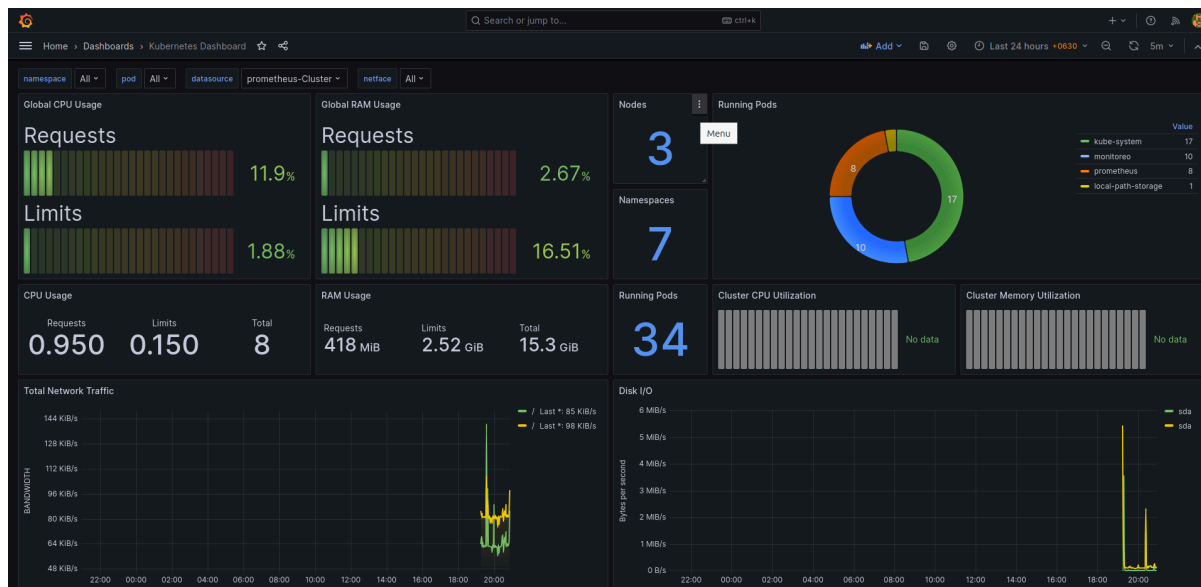
```
steven@MASTER:~/Desktop/KUBE$ kubectl get pods -n prometheus -o wide
NAME                                READY   STATUS    RESTARTS   AGE   IP              NODE             NOMINATED NODE   READINESS GATES
prometheus-alertmanager-0           1/1     Running   0           10m    10.0.0.86       worker-2          <none>            <none>
prometheus-kube-state-metrics-786488967b-4qzm5 1/1     Running   0           10m    10.0.0.71       worker-2          <none>            <none>
prometheus-prometheus-node-exporter-dqrl9 1/1     Running   0           10m    192.168.145.141 worker-1          <none>            <none>
prometheus-prometheus-node-exporter-lkcqh 1/1     Running   0           10m    192.168.145.142 worker-2          <none>            <none>
prometheus-prometheus-node-exporter-qmmh9 1/1     Running   0           10m    192.168.145.140 master            <none>            <none>
prometheus-prometheus-pushgateway-849bb86467-dm6bx 1/1     Running   0           10m    10.0.2.129      worker-1          <none>            <none>
prometheus-server-6648f86775-87jd4      2/2     Running   0           10m    10.0.2.122      worker-1          <none>            <none>
steven@MASTER:~/Desktop/KUBE$
```

Luego vamos a integrar a grafana los servicios de Prometheus y obtener sus métricas, tanto las del cluster como las de PostgreSQL. Loki ya está integrado porque se desplegó junto con grafana. Esto se hace en el apartado de Connections > Data sources.

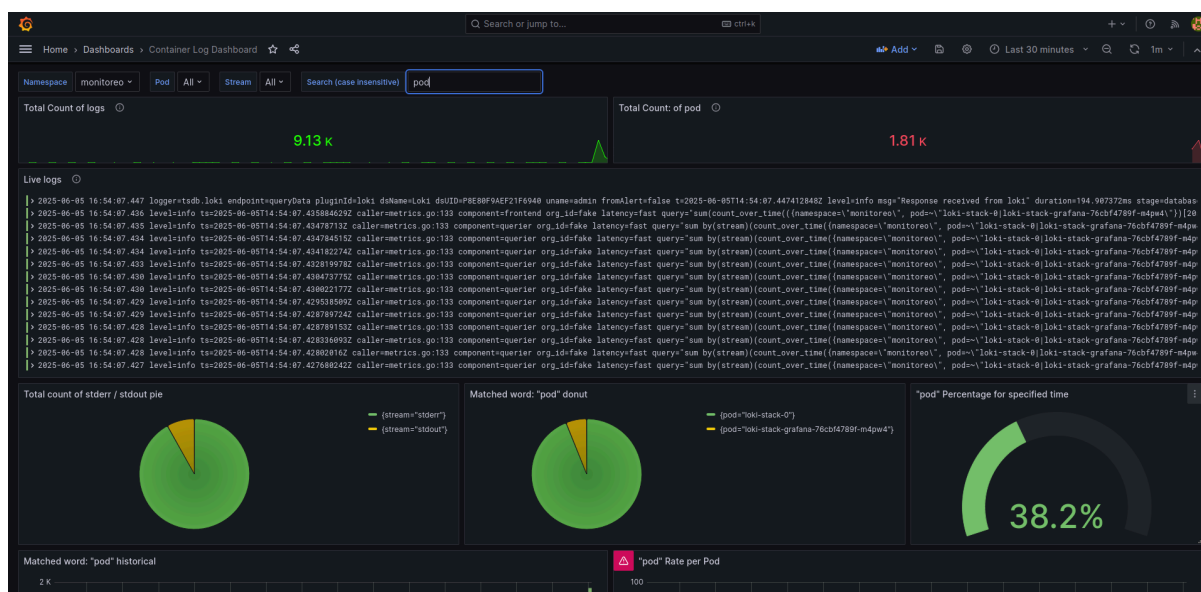


Una vez integrados importamos los dashboards a grafana para visualizar las métricas mejor. Tenemos para el Cluster, con ello podremos observar el rendimiento del cluster para saber si estamos al límite de recursos o no.



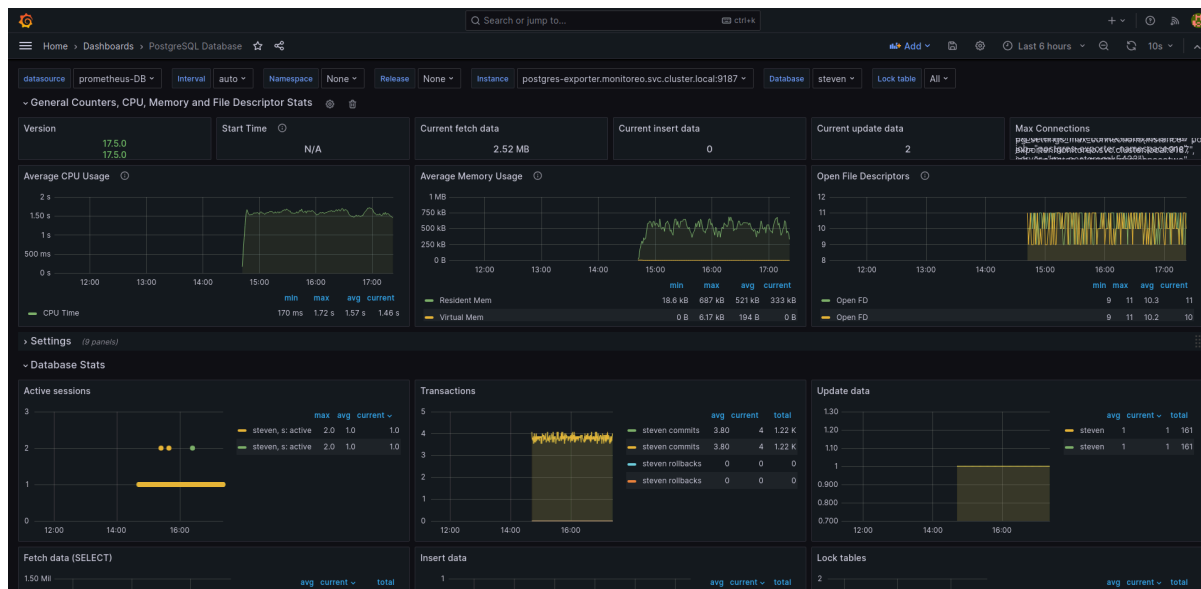


Para los logs de Loki, con él podríamos hacer un búsqueda más a fondo de algun problema o información que necesitemos en ese momento



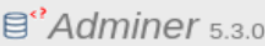
Para la base de datos, ver el estado del servicio y lo que está ocurriendo dentro de él, tanto las querys, los usuarios que hay o tráfico que está manejando





Con Adminer gestionamos la base de datos de una forma más sencilla ya que podemos acceder a ella con una interfaz gráfica por lo que nos logueamos con el usuario postgres y creamos las tablas con el archivo .sql creado

Idioma: Español



Login

(PostgreSQL) postgres@10.109.149.132
(PostgreSQL) steven1@10.109.149.132
(PostgreSQL) steven2@10.109.149.132

Motor de base de datos PostgreSQL

Servidor 10.109.149.132

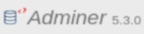
Usuario postgres

Contraseña *****

Base de datos steven

☐ Guardar contraseña

Idioma: Español



BD: steven
Esquema: public

[Comando SQL](#) [Importar](#)
[Exportar](#) [Crear tabla](#)

[registros clientes](#)
[registros compra](#)
[registros compra_detalle](#)
[registros producto](#)

PostgreSQL » 10.109.149.132 » steven » public » Importar

Importar

8 sentencias SQL ejecutadas correctamente. (0.013 s)

Importar archivo
SQL[.gz] (< 128MB): No se han seleccionado archivos.

☒ Parar en caso de error ☒ Mostrar solamente errores

Desde servidor
Archivo de servidor web adminer.sql[.gz]





Vemos como cada usuario tiene distintos privilegios en las tablas, en el usuario 1 no vemos el contenido mientras que el usuario 2 puede leer las tablas.

CONCLUSIONES

Con el desarrollo del proyecto he adquirido nuevos conocimientos sobre este tipo de entornos y una primera toma de contacto con la orquestación de servicios en un cluster hecho por Kubernetes, también a enfrentar los distintos problemas que ha habido durante el proceso del proyecto que me ha ayudado a comprender cómo funciona un clúster distribuido, cómo se despliegan y supervisan servicios en él, y cómo se comportan ante distintas cargas y configuraciones.

El trabajo me ha servido para ampliar mis conocimientos en el despliegue de servicios, en este caso con Kubernetes en relación con la monitorización mediante herramientas como Prometheus, Grafana y Loki, así como la integración de servicios como PostgreSQL y Adminer. Esta experiencia me ha aportado una visión más completa del ciclo de vida de los contenedores y del papel que juega la orquestación en la infraestructura moderna.

Además, he desarrollado destrezas que difícilmente se habrían conseguido en el contexto habitual del ciclo formativo. Por ejemplo, he aprendido a investigar por mi cuenta para comprender conceptos avanzados como redes entre pods, volúmenes persistentes o el uso de Helm charts. También he fortalecido mi capacidad de resolución de problemas al enfrentarme a errores reales de configuración,





incompatibilidades entre versiones y comportamientos inesperados de los servicios, utilizando logs, eventos y documentación oficial para encontrar soluciones.

Uno de los principales retos ha sido la escasa documentación clara y unificada sobre la instalación manual de Kubernetes, ya que muchos ejemplos utilizan herramientas que automatizan gran parte del proceso. Esto me obligó a profundizar más en la arquitectura subyacente y entender qué hace cada componente. También fue complicado combinar distintas fuentes de información (tutoriales, documentación y foros) sin generar conflictos entre configuraciones.

A nivel de mejora, el proyecto podría evolucionar incorporando una única instancia de Prometheus que recoja tanto métricas del clúster como de PostgreSQL, en lugar de separar en dos instalaciones. Pese a haberlo intentado en varias ocasiones, no logré obtener todos los datos deseados de una única fuente. Una mejora en cuanto a los volúmenes sería tener un disco para guardar todo lo que contenga los servicios y guardarlos de tal manera que no se pierdan si alguna vez los pods fallan. Además, los scripts de automatización podrían perfeccionarse para manejar mejor los errores de entrada del usuario, repitiendo las preguntas en caso de respuestas inválidas y haciendo el proceso aún más robusto.

En definitiva, este proyecto ha sido un reto enriquecedor que me ha permitido consolidar habilidades técnicas, mejorar mi autonomía en el aprendizaje y preparar mejor mi perfil profesional para entornos reales de administración y despliegue de servicios.

BIBLIOGRAFÍA Y REFERENCIAS

[MONITORIZACIÓN \(PROMETHEUS, GRAFANA, LOKI\)](#)

[MONITORIZACIÓN \(PROMETHEUS, GRAFANA, POSTGRESQL\)](#)

[DOCUMENTACIÓN KUBERNETES](#)





[CURSO KUBERNETES \(PABPEREZA\)](#)

[SHELL SCRIPTING](#)

ANEXOS

[GITHUB DEL PROYECTO](#)

[VIDEO DEL PROYECTO](#)

[PRESENTACIÓN DEL PROYECTO](#)

