# *Diplomatura en DevOps*
# *Mundos E*

Edición 2403


# “Informe Desarrollo Práctico Integrador (PIN FINAL)”

Tutor:

- Guazzardo, Marcelo

Grupo 1:

- Cabral, Damian Esteban
- Huataquispe Poma, Arnold
- Gonzalez, Claudio
- Rico, Cristian
- Ferreira, Alexander

2024

# Proyecto de automatización de Infraestructura en AWS, creación y monitoreo de un Clúster EKS.

## ***1. Introducción***

Este proyecto, denominado PIN FINAL, ha sido diseñado para desplegar y gestionar un clúster de Kubernetes en AWS utilizando Elastic Kubernetes Service (EKS). Su propósito es integrar diferentes herramientas vistas durante la diplomatura de DevOps en MundosE, Utilizamos AWS para crear recursos como EC2, EKS, Load Balancer, etc… y herramientas de monitoreo para crear una infraestructura optimizada con enfoque DevOps.

El proyecto se compone de los siguientes aspectos principales:

- Aprovisionamiento de infraestructura: Se crea una instancia EC2 en AWS que funciona como máquina de administración, con herramientas esenciales instaladas en el User Data, como AWS CLI, kubectl, eksctl, Docker, necesarias para interactuar con los recursos de AWS, es como un Bastion.
- Creación y configuración de un clúster Kubernetes (EKS): Se utiliza eksctl para desplegar un clúster de Kubernetes administrado con tres nodos.
- Gestión de accesos y permisos: Se configura IAM y el configmap/aws-auth para administrar usuarios y accesos al clúster.
- Monitoreo y visualización de métricas: Se despliega Prometheus para recolectar métricas del clúster o sus pods, y Grafana para visualizarlas a través de dashboards personalizables.

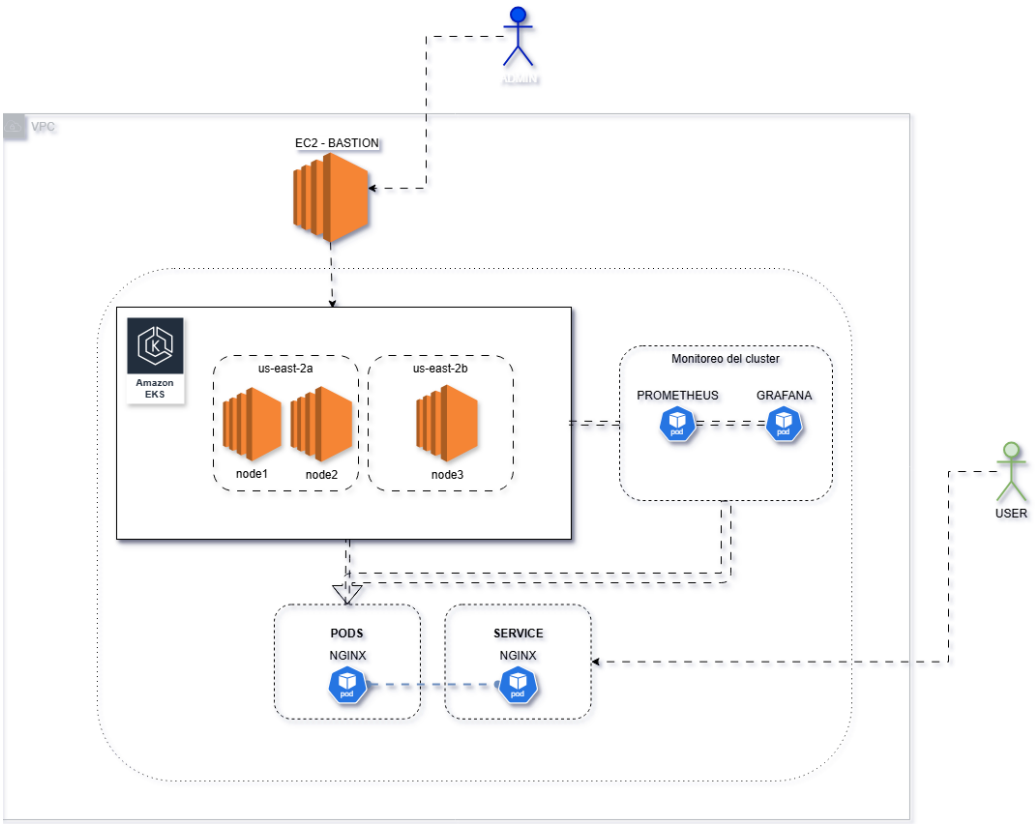
En términos generales, el proyecto automatiza la creación y gestión de un entorno Kubernetes en AWS, integrando monitoreo y logging centralizado con herramientas modernas de DevOps.

## 2. Objetivos

El proyecto tiene como objetivo general del proyecto es aprender y poner en práctica diversas tecnologías y herramientas a través de un laboratorio. Para ello, se implementa una infraestructura en AWS, de manera automatizada y eficiente. Para ello, se desplegará un clúster EKS con eksctl, configurando los nodos de trabajo y los permisos de acceso adecuados. Además, se utilizarán scripts para instalar herramientas clave como AWS CLI, Docker, Helm y Terraform en una instancia EC2, facilitando la gestión de la infraestructura.

Para recolectar las métricas de los POds, se habilitará un sistema centralizado con Prometheus y Grafana para la recopilación y visualización de métricas. Se realizarán pruebas para verificar la correcta operación del clúster. Finalmente, se establecerá un proceso de limpieza automatizado para eliminar de forma segura los recursos cuando ya no sean necesarios, asegurando una administración eficiente y escalable del entorno.

## 3. Topología general



Topología general del proyecto DevOps. Representa cómo los diferentes componentes interactúan entre sí:

- **EC2 Instance:** Es la máquina base donde se instalan herramientas necesarias como AWS CLI, kubectl, Docker, Helm y Terraform.
- **AWS CLI:** Se usa para interactuar con AWS y configurar el clúster EKS.
- **EKS Cluster:** Contiene los nodos del clúster donde se ejecutan los workloads.
- **Worker Nodes:** Son los nodos donde se despliegan los contenedores de Kubernetes.
- **Prometheus:** Monitorea el rendimiento del clúster y sus aplicaciones (Nginx).
- **Grafana:** Visualiza los datos de Prometheus mediante dashboards.

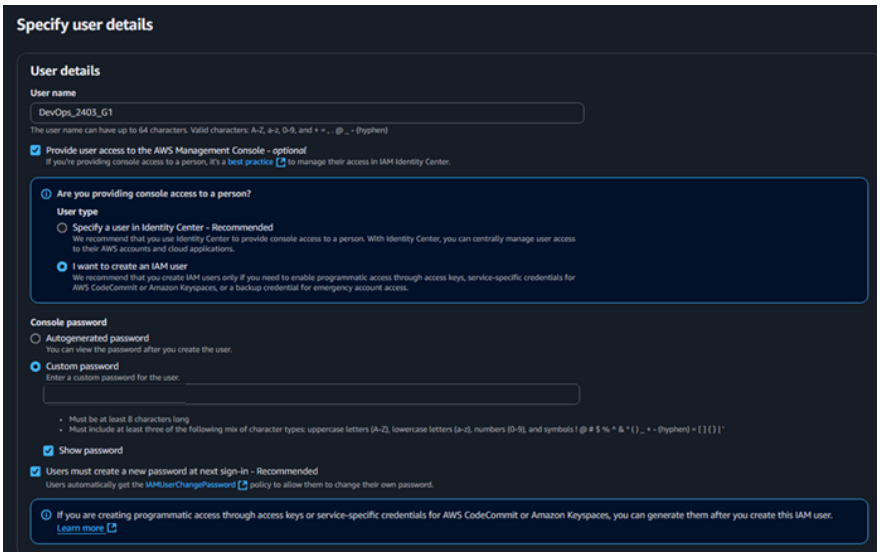
## ***4. Descripción del Proyecto***

Este proyecto tiene como objetivo la creación y configuración de un clúster Kubernetes en AWS EKS, proporcionando una solución integral para la gestión de contenedores con una infraestructura robusta de monitoreo y logging. Mediante la combinación de Prometheus y Grafana, se garantiza la observabilidad del sistema, facilitando la detección y solución de problemas.
La implementación se desarrolla en seis fases estructuradas, asegurando un despliegue organizado y eficiente, desde la preparación de la infraestructura hasta la posible eliminación de recursos.

### 4.1 FASE 0: prerrequisitos

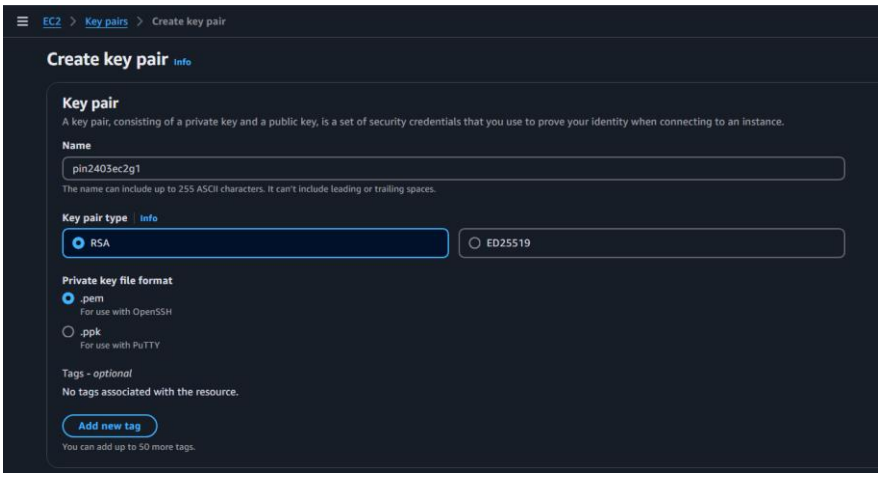
(Paso 1) Se creara un usuario en AWS

ID : xlexferr02aws
Ur : DevOps_2403_G1
Pass: **********

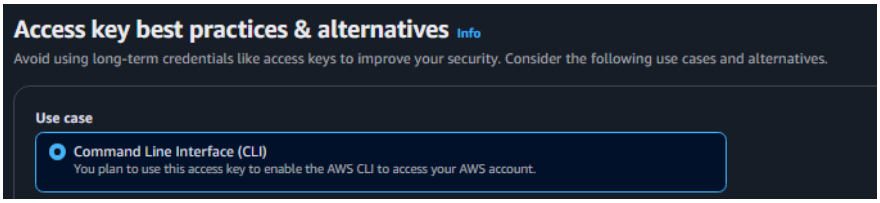


(Paso 2) Se creara un par de claves SSH en la consola de AWS, para acceder a la EC2

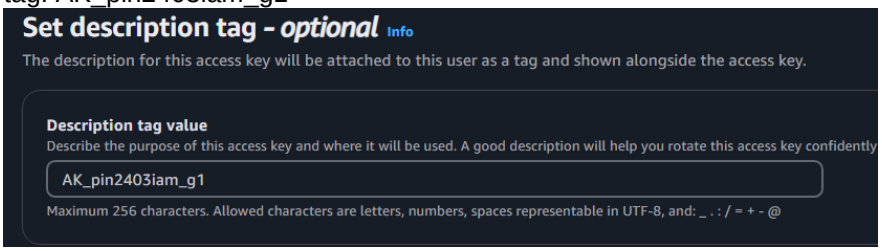
Name: pin2403ec2g1




Se crea Access Key – IAM – ur: DevOps_2403_G1




tag: AK_pin2403iam_g1



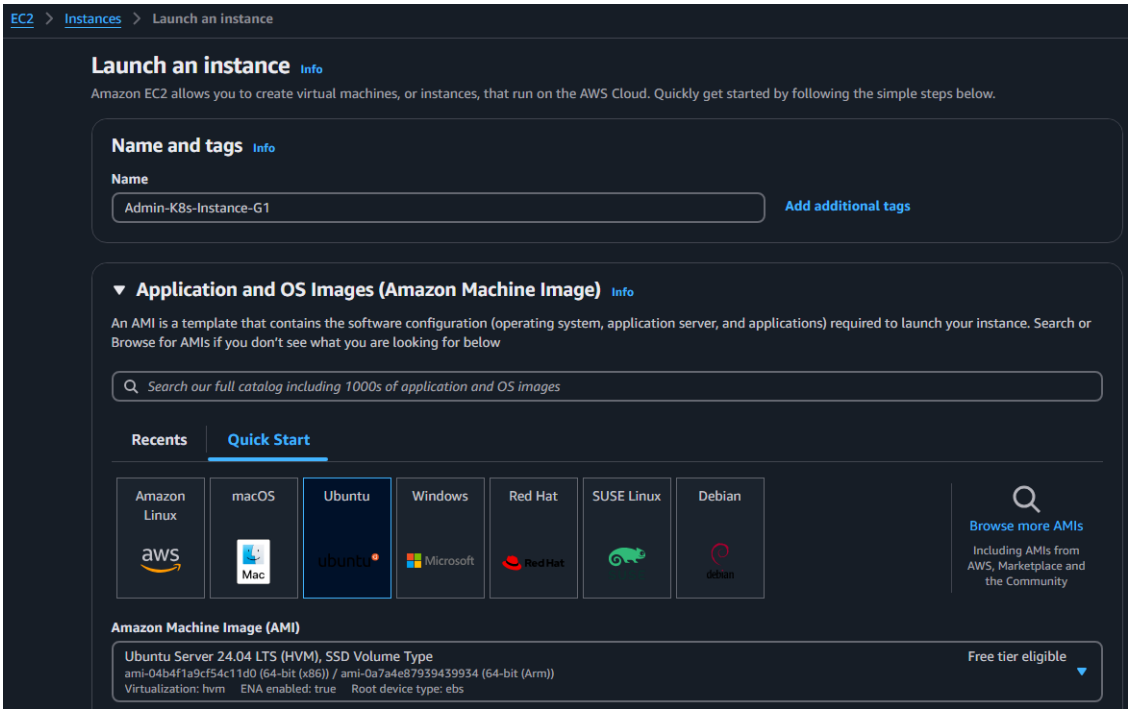

*Access key*: ********

*Secret access key*: ********

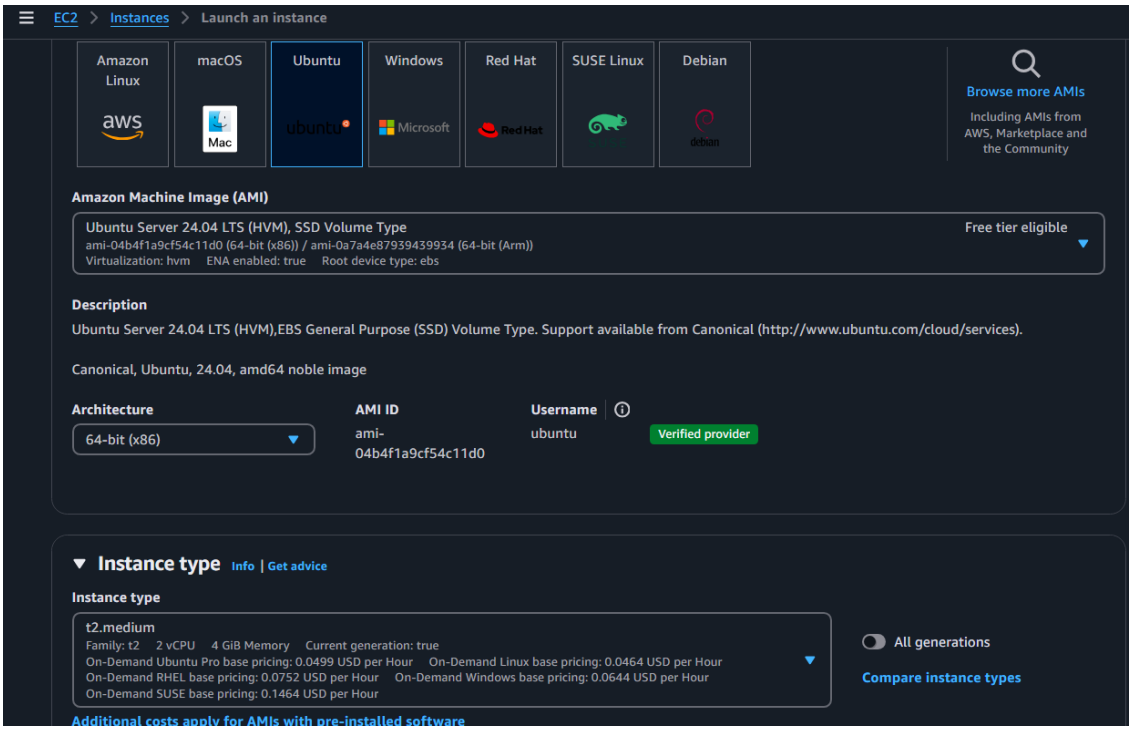
## 4.2 FASE 1: Preparación de la infraestructura

(Paso 1) Se crea una instancia EC2 en AWS con Ubuntu 22.04.

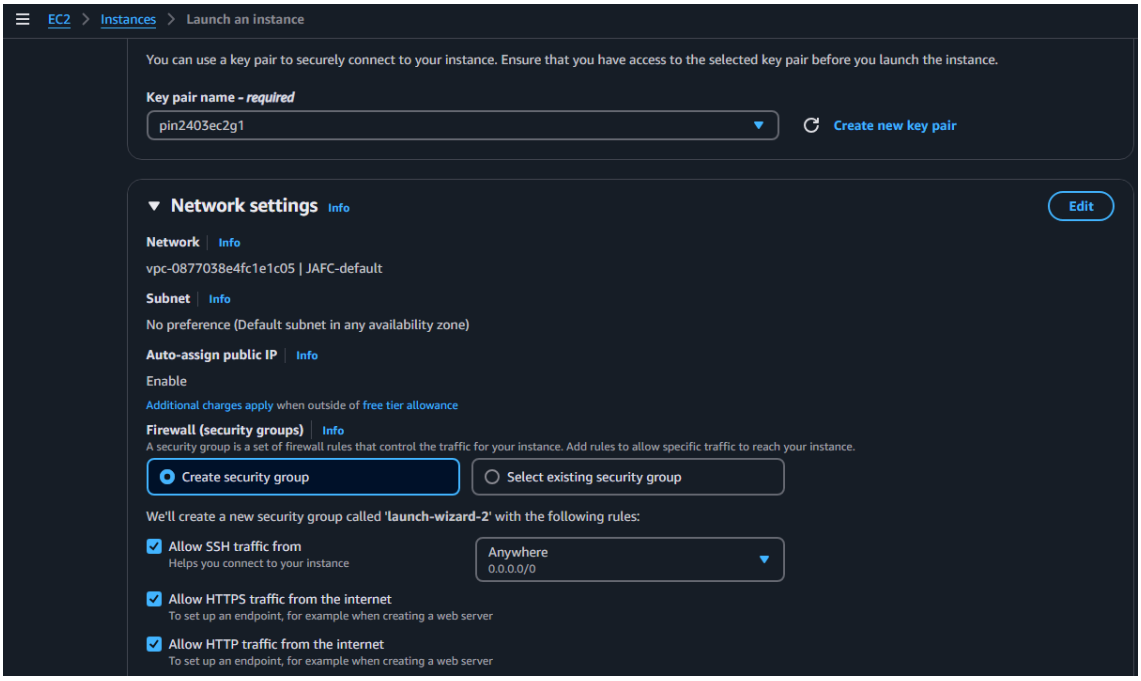
Instancia name: Admin-K8s-Instance-G1



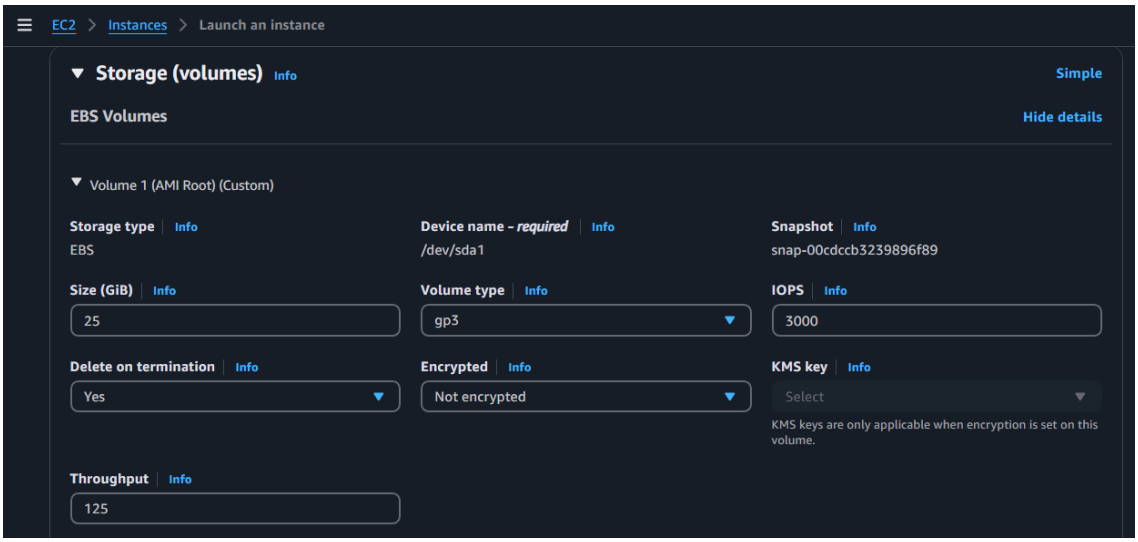

Tipo: t2.medium (2 vCPU, 4GB RAM)



Key pair




Storage




User data

**User data – *optional*** | Info

Upload a file with your user data or enter it in the field.

Choose file

```
#!/bin/bash
echo "Installing AWS CLI"
sudo apt update -y && sudo apt upgrade -y
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
sudo apt install -y unzip
unzip awscliv2.zip
sudo ./aws/install

# Instalar kubectl
echo "Instalando kubectl..."
curl -fsSL -o /usr/local/bin/kubectl https://dl.k8s.io/release/v1.29.0/bin/linux/amd64/kubectl
chmod +x /usr/local/bin/kubectl

# Instalar eksctl
echo "Instalando eksctl..."
curl --silent --location
"https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_Linux_amd64.tar.gz" | tar xz
-C /tmp
sudo mv /tmp/eksctl /usr/local/bin


# Instalar Docker
echo "Instalando Docker..."
sudo apt install -y docker.io
sudo usermod -aG docker $USER
sudo systemctl enable docker
sudo systemctl start docker

# Instalar Docker Compose
echo "Instalando Docker Compose..."
curl -fsSL -o /usr/local/bin/docker-compose
https://github.com/docker/compose/releases/latest/download/docker-compose-linux-x86_64
chmod +x /usr/local/bin/docker-compose

echo "Instalando Helm..."
curl -fsSL https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3 | bash

# Instalar Terraform
echo "Instalando Terraform..."
sudo apt install -y gnupg software-properties-common
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install -y terraform
```

## Summary

**Number of instances** | Info

1

**Software Image (AMI)**
Canonical, Ubuntu, 24.04, amd64 noble image
ami-04b4f1a9cf54c11d0

**Virtual server type (instance type)**
t2.medium

**Firewall (security group)**
New security group

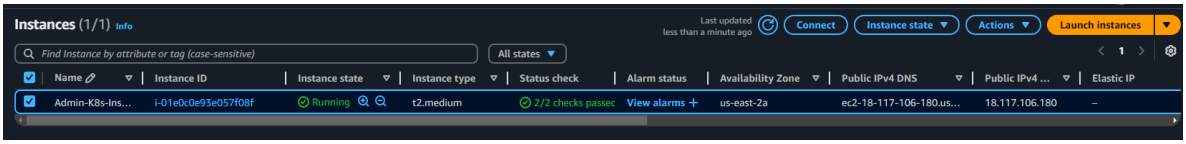
**Storage (volumes)**
1 volume(s) - 25 GiB

**Free tier:** In your first year of opening an AWS account, you get 750 hours per month of t2.micro instance usage (or t3.micro where t2.micro isn't available) when used with free tier AMIs, 750 hours per month of public IPv4 address usage, 30 GiB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.

Cancel | Launch instance | Preview code

**Success**
Successfully initiated launch of instance (i-09f34eba96632dbed)

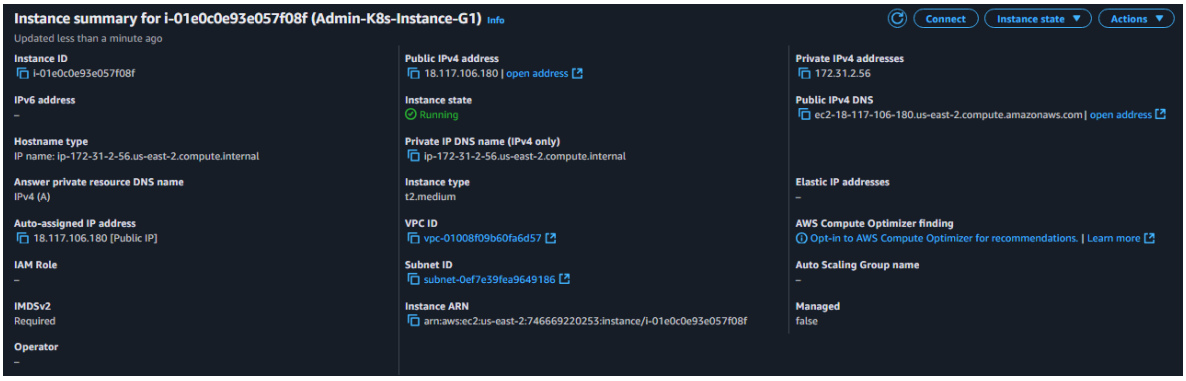
(Paso 2) Se instalan herramientas necesarias en la instancia EC2, como AWS CLI, kubectl, eksctl, Docker, Helm y Terraform (ec2_user_data.sh).



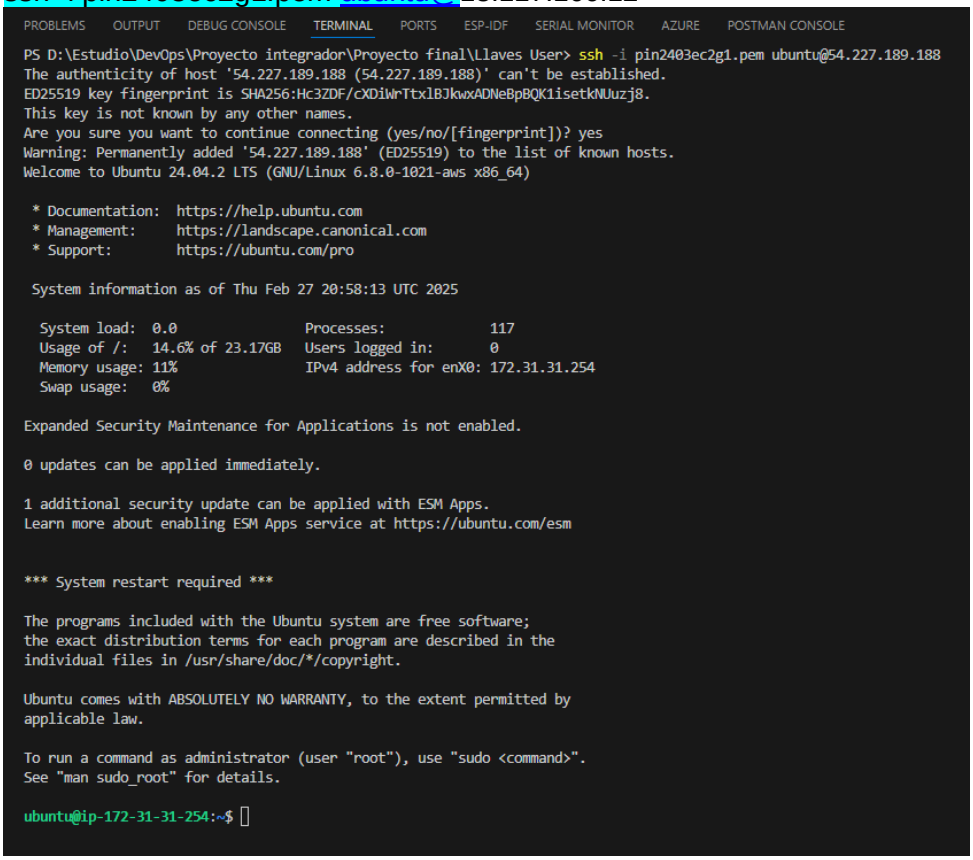

(Paso 3) Se prueban credenciales AWS en la instancia EC2 para poder operar con AWS EKS.

Esta IP, cambia cada vez que se apaga la máquina, para que quede fija la misma IP, deberíamos utilizar una ELASTIC IP

IP Public: 18.117.106.180

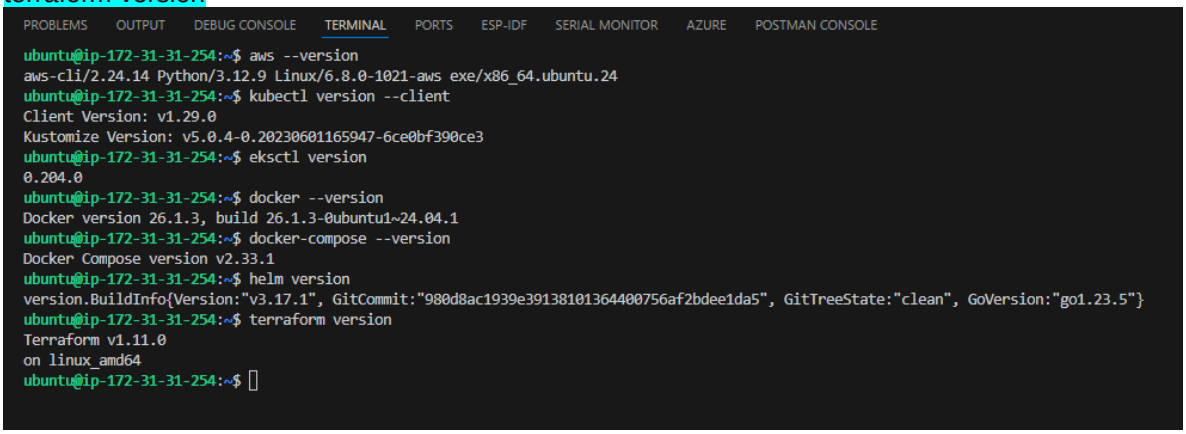



ssh -i pin2403ec2g1.pem ubuntu@18.117.166.12



Validación de los servicios instalados

aws --version
kubectl version --client
eksctl version
docker --version
docker-compose --version
helm version
terraform version




(Paso 4) Desde la EC2, se configuran las credenciales para AWS

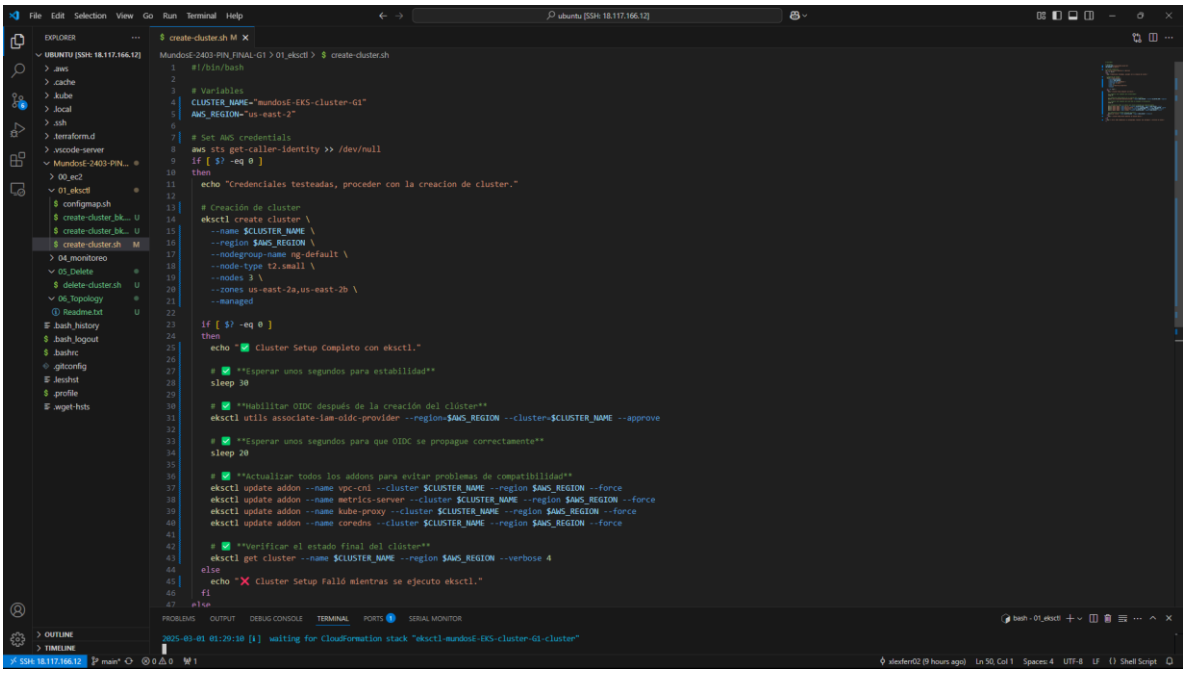
aws configure
aws sts get-caller-identity


## 4.3 FASE 2: Creación del clúster Kubernetes en AWS EKS

(Paso 4) Se crea un clúster Kubernetes en AWS EKS con 3 nodos (create-cluster.sh).

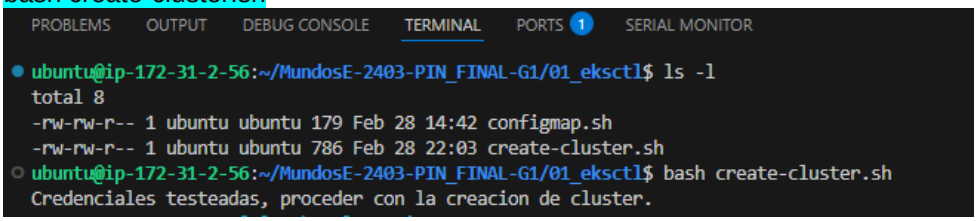
Se han cambiado parámetros importantes en create-cluster.sh como

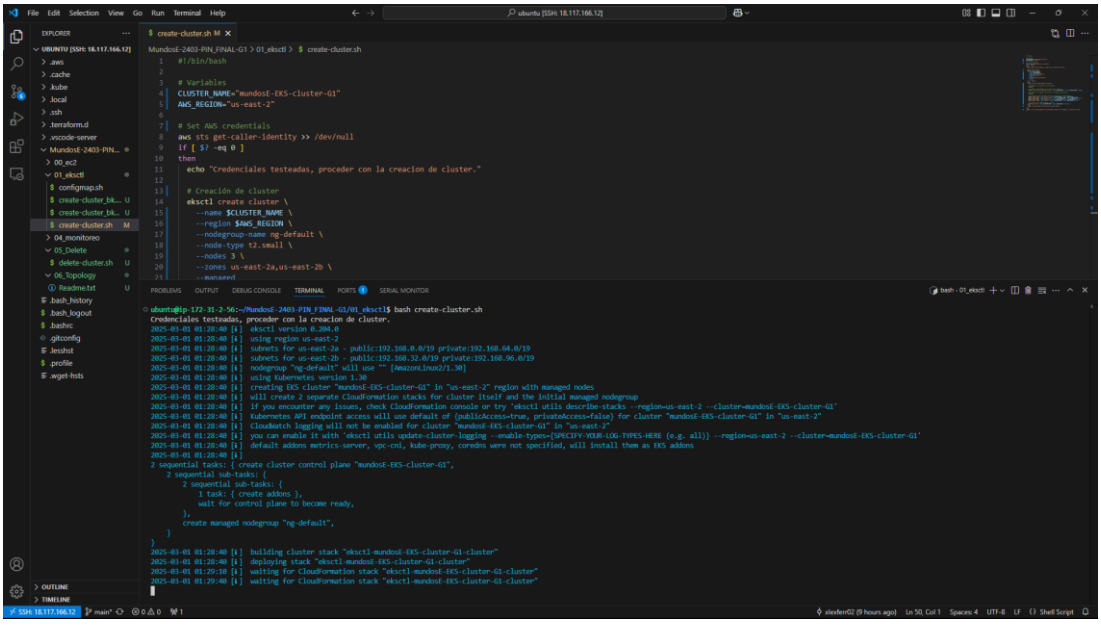
- CLUSTER_NAME=mundosE-EKS-cluster-G1 (Cambio el nombre del clúster)
- --name mundosE-EKS-cluster-G1 \ (Reemplazamos el nombre del Cluster)
- --region us-east-2 \ (Reemplazamos la región)
- #--ssh-access \
- #--ssh-public-key jenkins \ (No se requiere acceso por SSH a los nodos, porque estos los administra AWS)
- --nodegroup-name ng-default \ (grupo de nodos ng-default, facilita la administración y escalabilidad.)
- --zones us-east-2ª (Trabajamos con 2 solo zona de disponibilidad)
- eksctl utils associate-iam-oidc-provider --region=$AWS_REGION --cluster=$CLUSTER_NAME --approve (Habilitación de OIDC Después de la Creación del Clúster)
  Actualización de Add-ons




eksctl → Es la herramienta específica CLI, que se utiliza para crear y administrar clústeres **EKS** en AWS.
Este comando iniciará la creación del clúster en AWS con los parámetros actualizados.

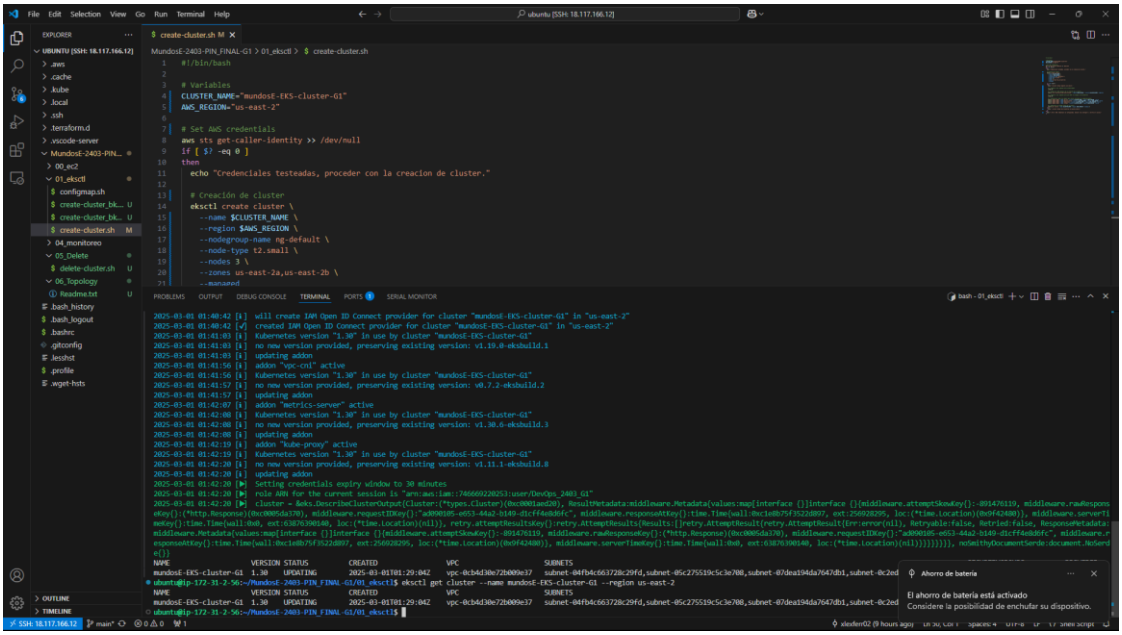
bash create-cluster.sh



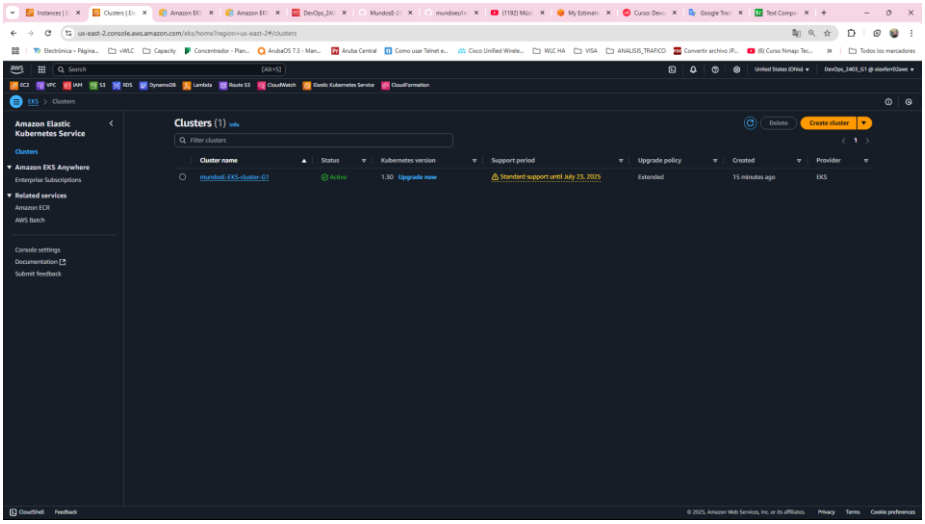



Verificación del clúster desde CLI

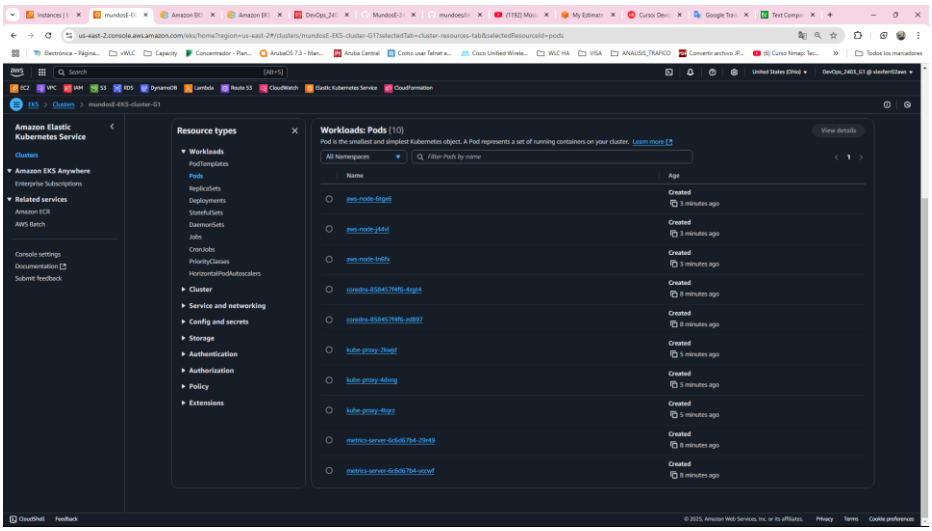
eksctl get cluster --name mundosE-EKS-cluster-G1 --region us-east-2



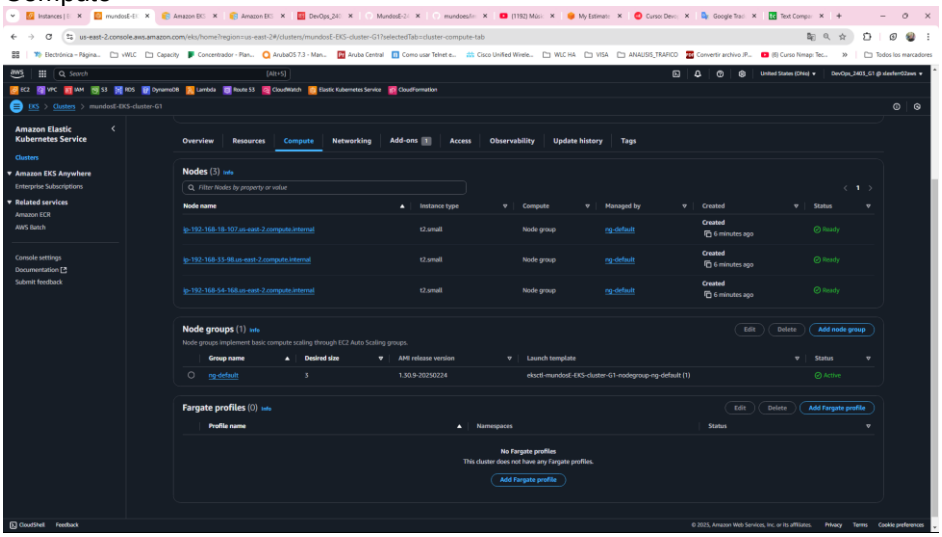

Verificación del clúster desde GUI



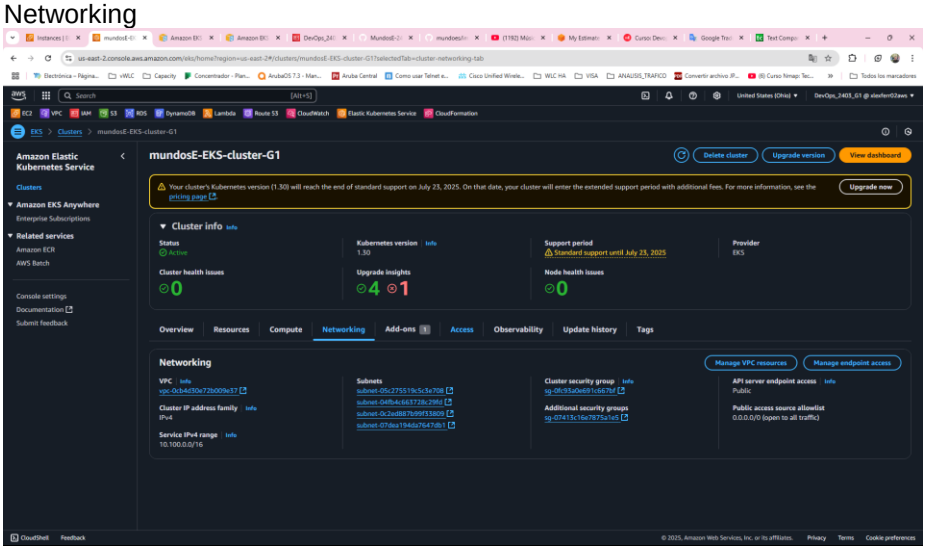

PODS



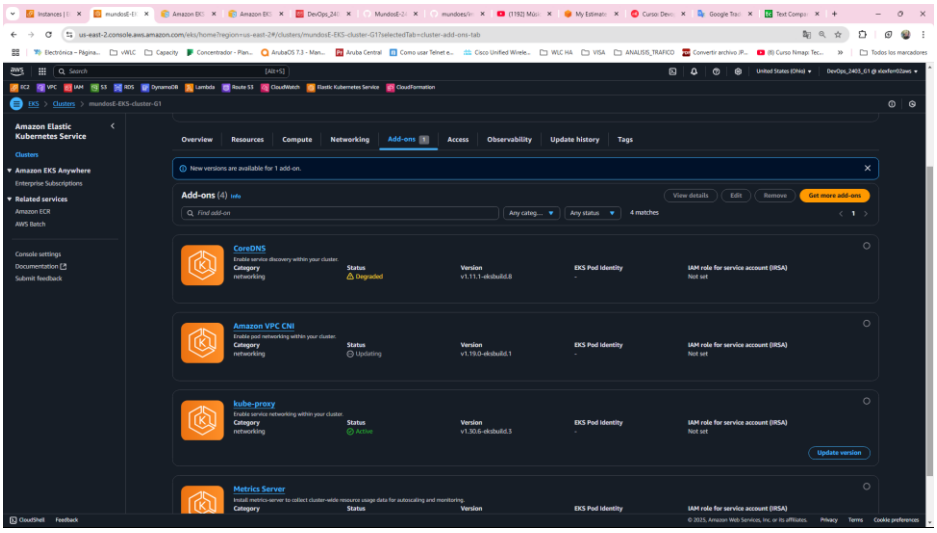
Computo



Networking



Add-ons

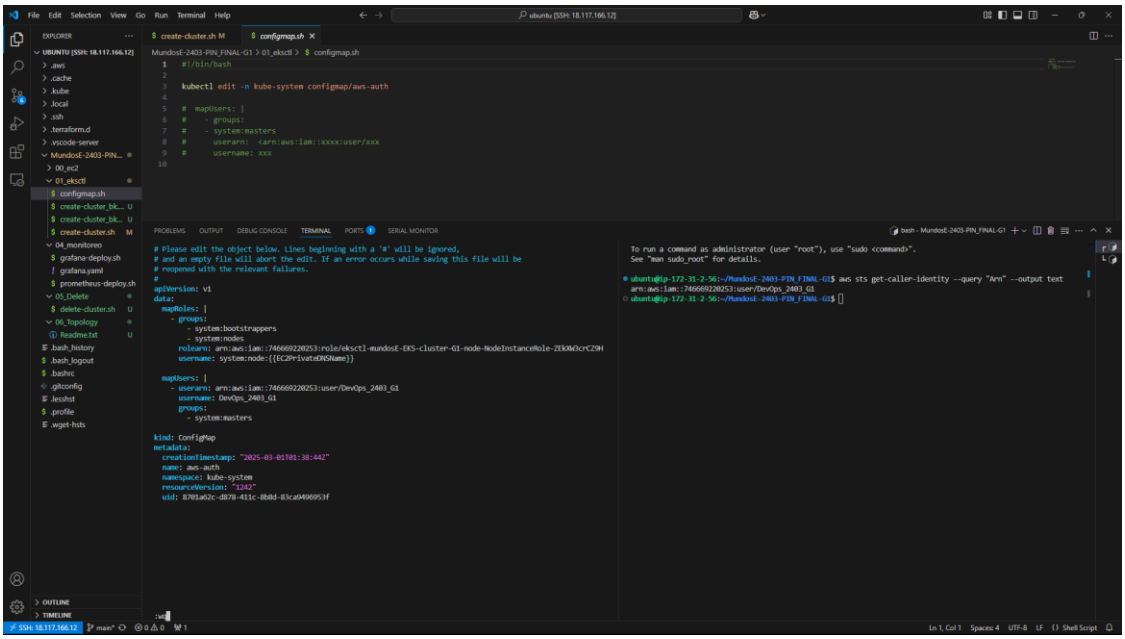



(Paso 5) Se edita el ConfigMap de Kubernetes para gestionar permisos de acceso (configmap.sh).

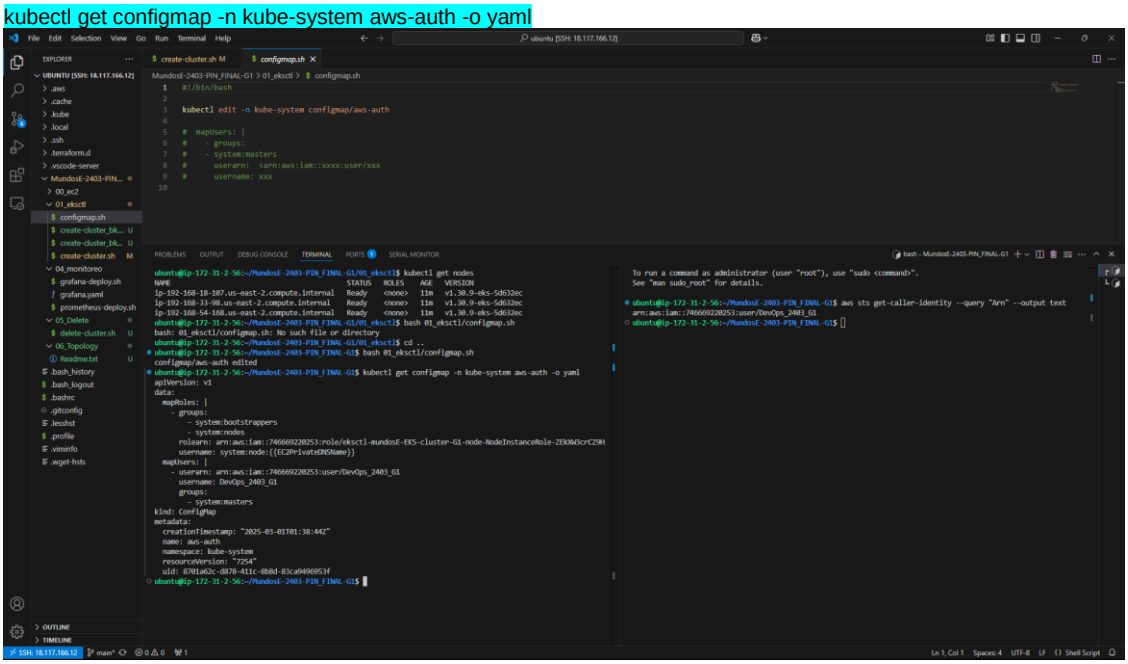
Este paso permite que usuarios o roles específicos tengan permisos administrativos dentro del clúster EKS.

- Validación que el clúster está activo y que `kubectl` está configurado correctamente.
  kubectl get nodes
- Editar el ConfigMap manualmente
  bash 01_eksctl/configmap.sh
- Se sustituye `arn:aws:iam::746669220253:user/DevOps_2403_G1` con el ARN correcto de tu usuario IAM.
- Para obtener el ARN del usuario IAM, se ejecuta:
  aws sts get-caller-identity --query "Arn" --output text
- Después de guardar ConfigMap, se aplica configuración con:
  kubectl get configmap -n kube-system aws-auth -o yaml

Se edita con vi ->  ESC :wq



kubectl get configmap -n kube-system aws-auth -o yaml



Cambios realizados y explicación

- Se mantiene mapRoles: (permite que los nodos EC2 se unan al clúster).
- Se agregó mapUsers: (permite que el usuario IAM DevOps_2403_G1 tenga acceso administrativo al clúster).

## 4.4 FASE 4: Implementación del monitoreo con Prometheus y Grafana

Prometheus almacena métricas históricas en un volumen persistente para evitar pérdida de datos si el pod se reinicia.
Entonces los volúmenes gp2 es una clase de almacenamiento en AWS EBS (Elastic Block Store) utilizada comúnmente para volúmenes persistentes en Kubernetes.

Antes que nada debemos verificar si en mi región tenemos disponible almacenamiento gp2

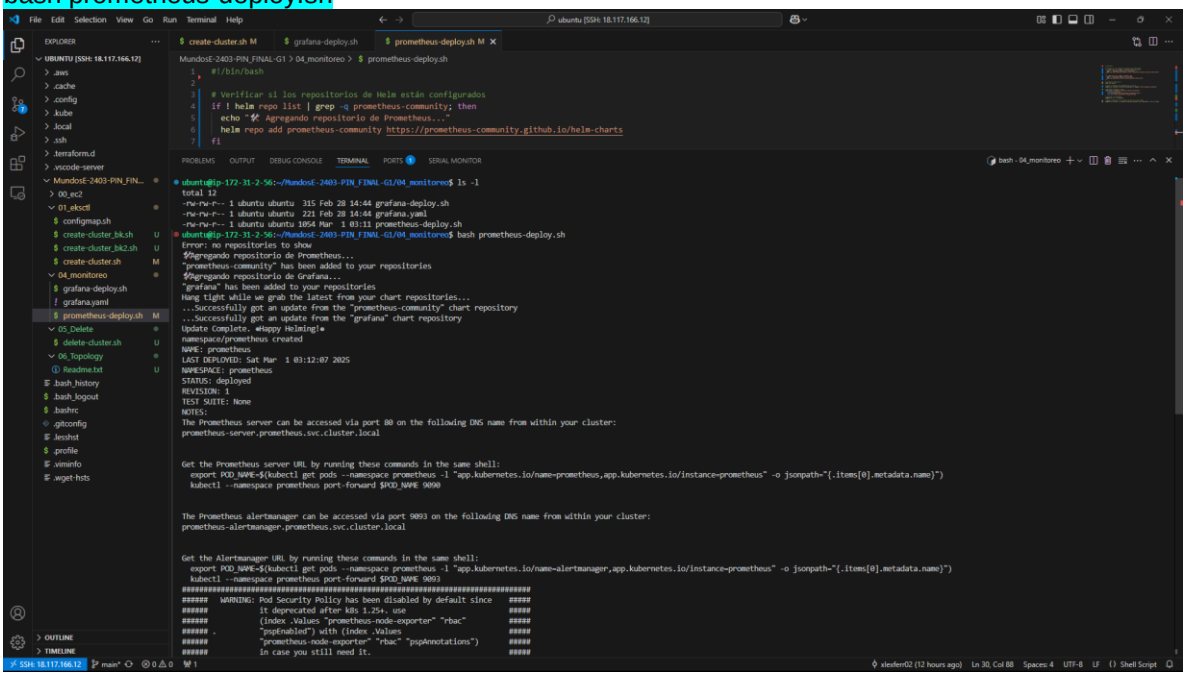
kubectl get storageclass

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS 1  SERIAL MONITOR

ubuntu@ip-172-31-2-56:~/MundosE-2403-PIN_FINAL-G1$ kubectl get storageclass
NAME   PROVISIONER             RECLAIMPOLICY   VOLUMEBINDINGMODE      ALLOWVOLUMEEXPANSION   AGE
gp2    kubernetes.io/aws-ebs   Delete          WaitForFirstConsumer   false                  72m
ubuntu@ip-172-31-2-56:~/MundosE-2403-PIN_FINAL-G1$
```

(Paso 12) Se despliega Prometheus en Kubernetes para recolectar métricas (prometheus-deploy.sh).

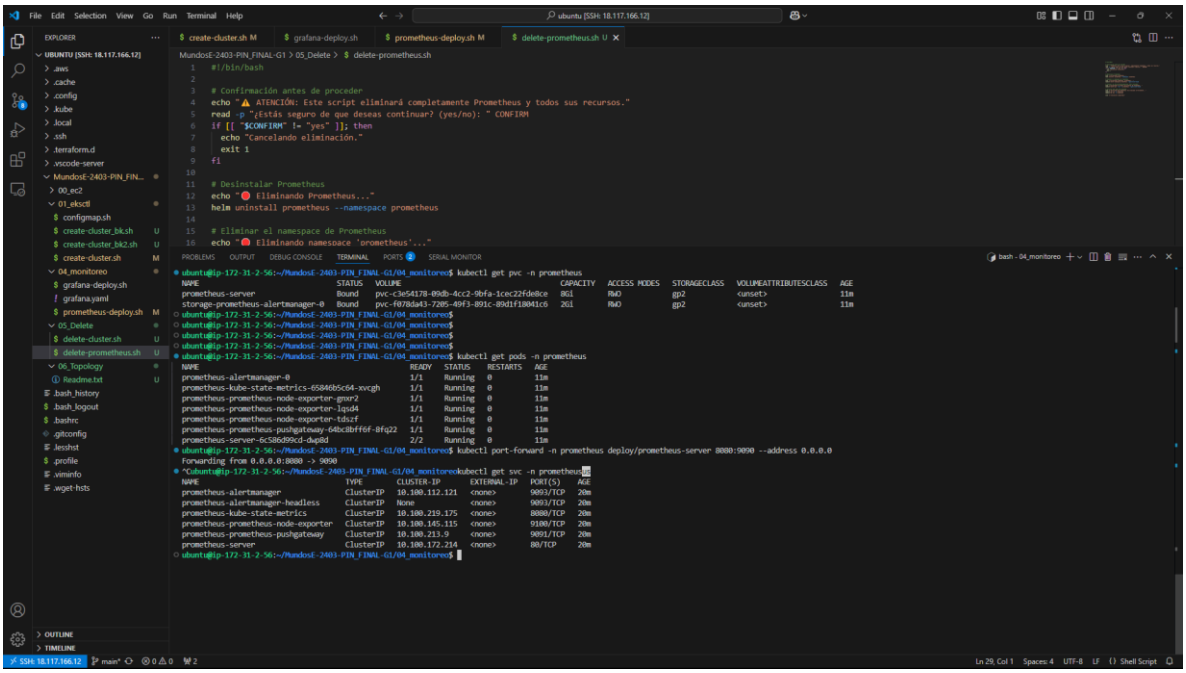
bash prometheus-deploy.sh



comando para verificar los volúmenes persistentes
kubectl get pvc -n prometheus

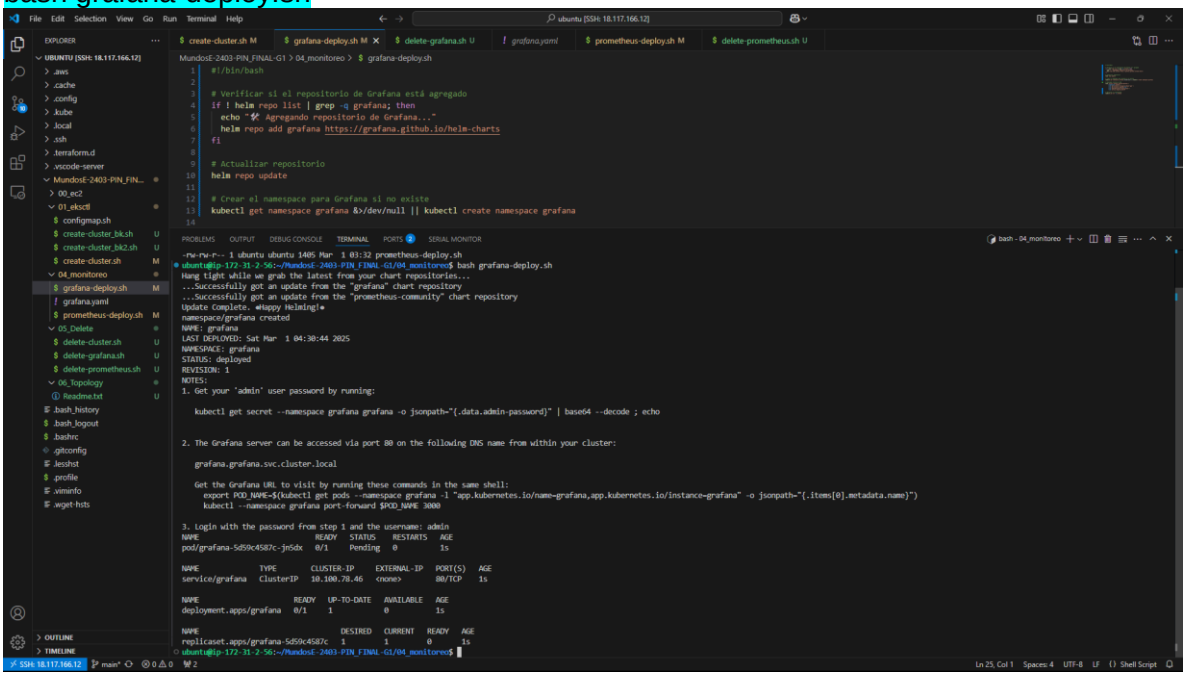
comando para ver el estado de los pods:
kubectl get pods -n prometheus

comando para verificar el tipo de servicio de Prometheus:
El servicio prometheus-server tiene TYPE: ClusterIP, lo cual significa que solo puede ser accedido dentro del clúster.
kubectl get svc -n prometheus



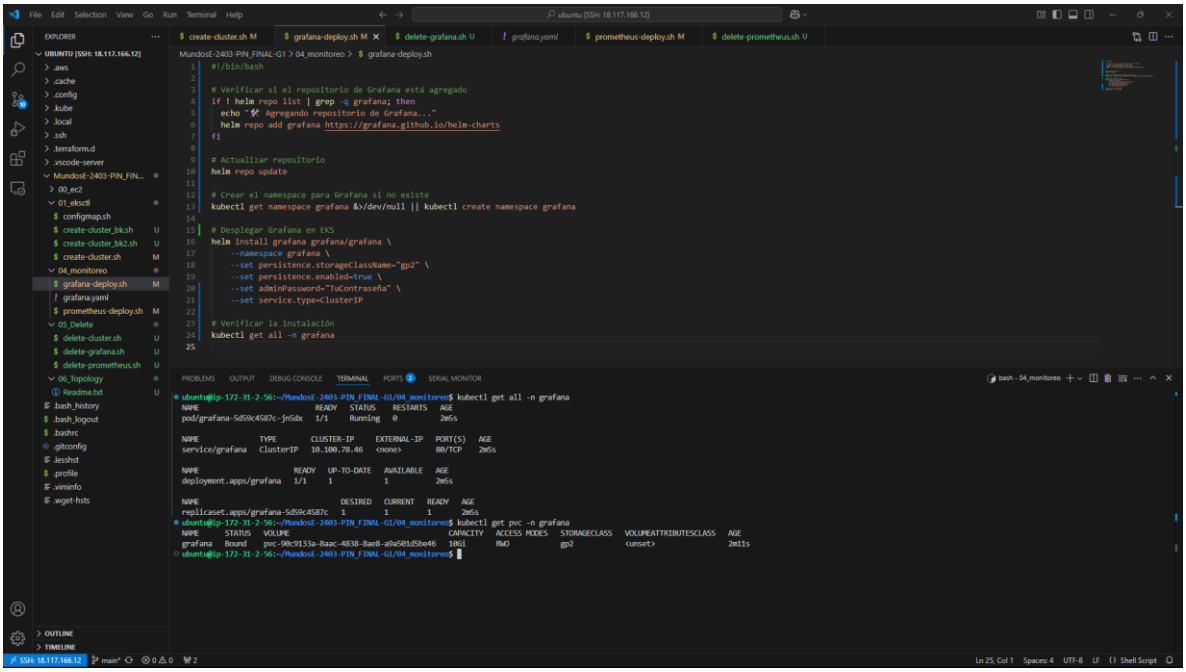

(Paso 13) Se despliega Grafana para visualizar métricas (grafana-deploy.sh).

bash grafana-deploy.sh




Muestra todos los recursos en el namespace grafana
kubectl get all -n grafana

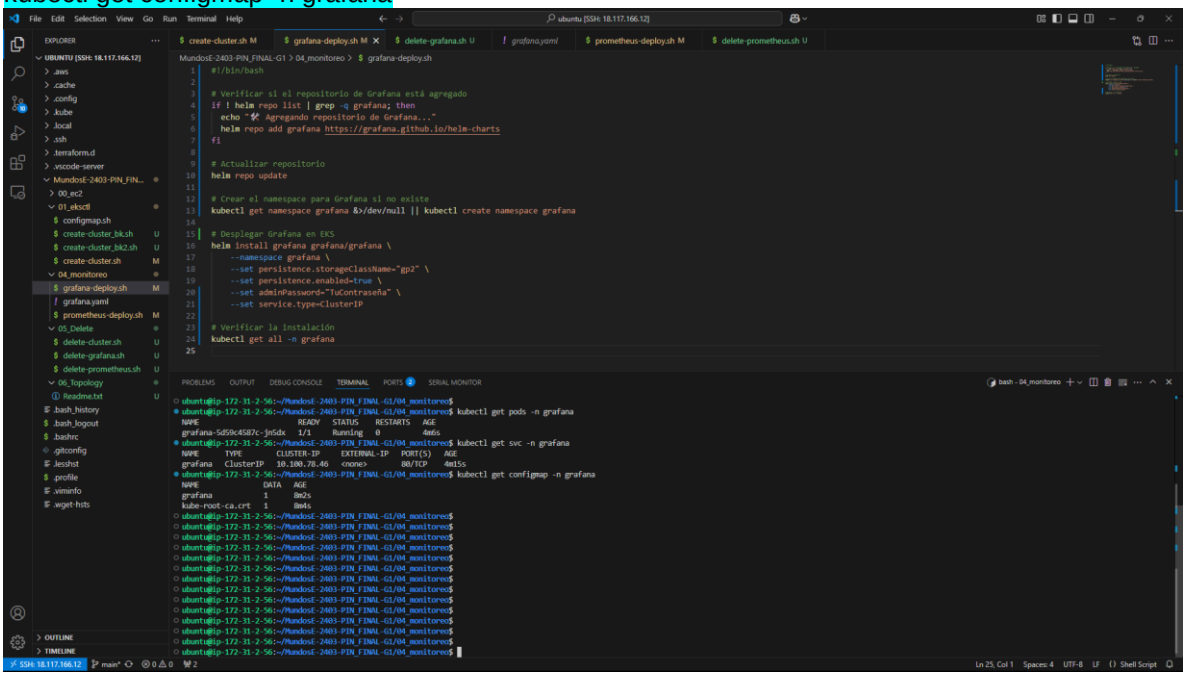
Lista los PersistentVolumeClaims (PVC) en el namespace grafana
kubectl get pvc -n grafana




Verificación de los pods en Grafana:
kubectl get pods -n grafana

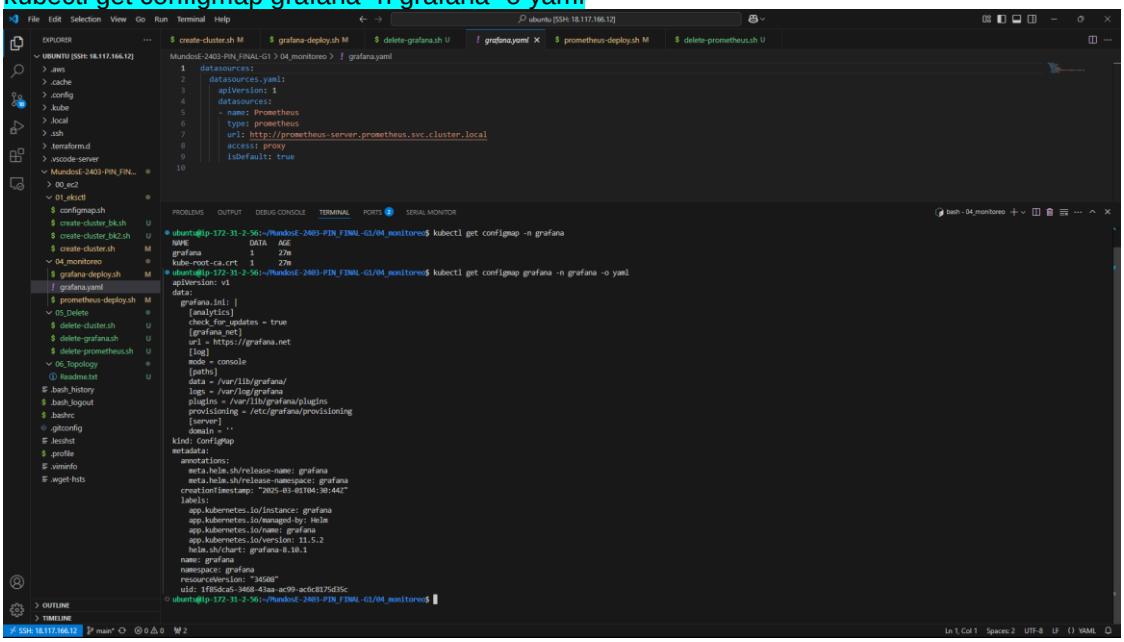
Verificación del servicio (svc) de Grafana:
kubectl get svc -n grafana

Grafana tiene su configuración almacenada en un ConfigMap. Grafana configurado correctamente dentro del clúster.
kubectl get configmap -n grafana



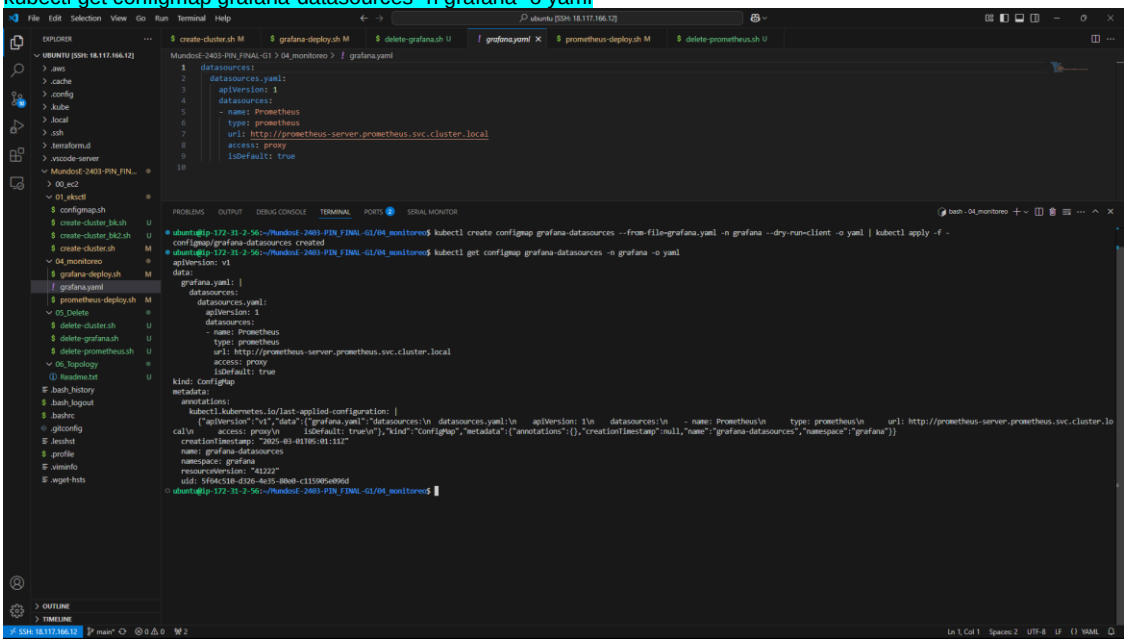
(Paso 14) Se configura Grafana para obtener datos desde Prometheus (grafana.yaml).

kubectl get configmap -n grafana
kubectl get configmap grafana -n grafana -o yaml




Aplicar la configuración
Ejecuta el siguiente comando para cargar la configuración en Grafana:
kubectl create configmap grafana-datasources --from-file=grafana.yaml -n grafana --dry-run=client -o yaml | kubectl apply -f -

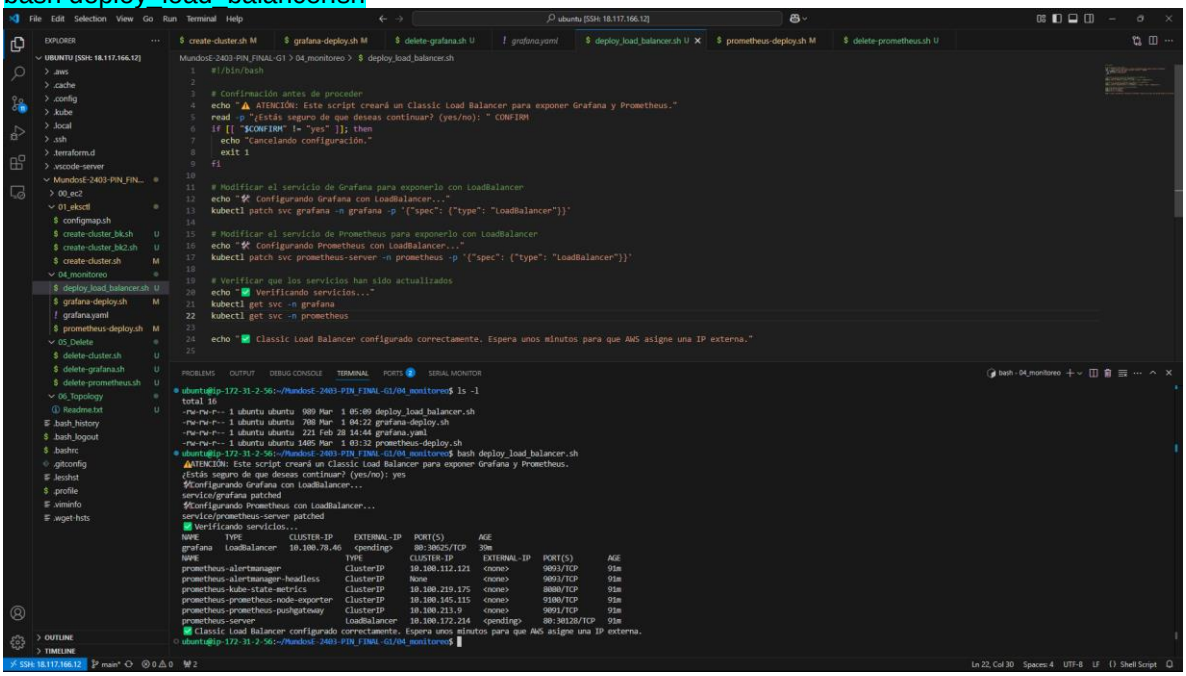
Verifica que la configuración se aplicó correctamente:
kubectl get configmap grafana-datasources -n grafana -o yaml



## 4.5 FASE 5: Instalación de Classic Load Balancer

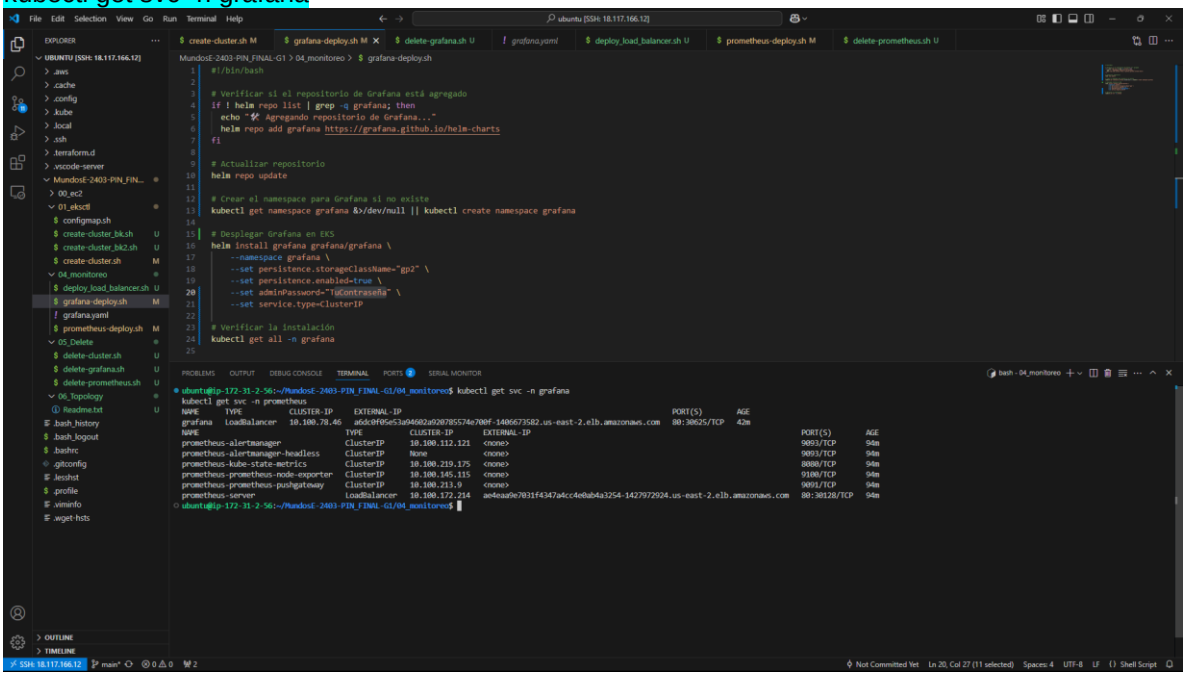
(Paso 15) Se despliega el servicio de Load Balancer para acceder a los recursos de manera externa.

bash deploy_load_balancer.sh



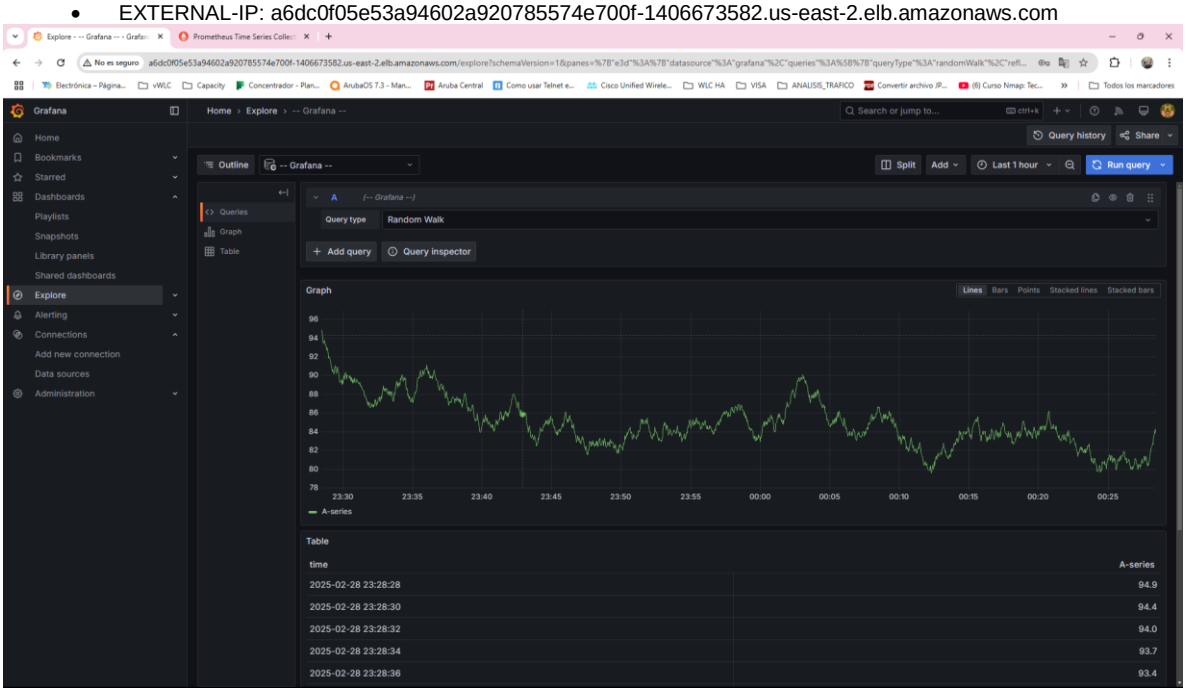
(Paso 16) validar acceso a los recursos.

kubectl get svc -n grafana



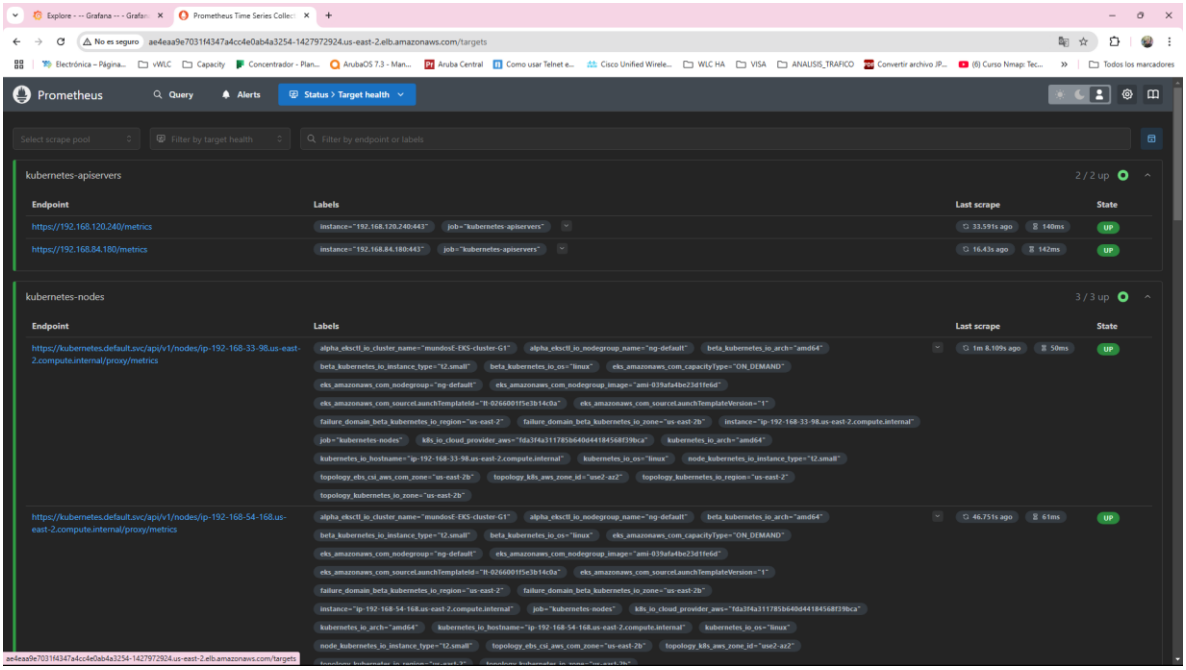
## Grafana

- TYPE: LoadBalancer → Grafana ahora está expuesto públicamente.
- EXTERNAL-IP: a6dc0f05e53a94602a920785574e700f-1406673582.us-east-2.elb.amazonaws.com

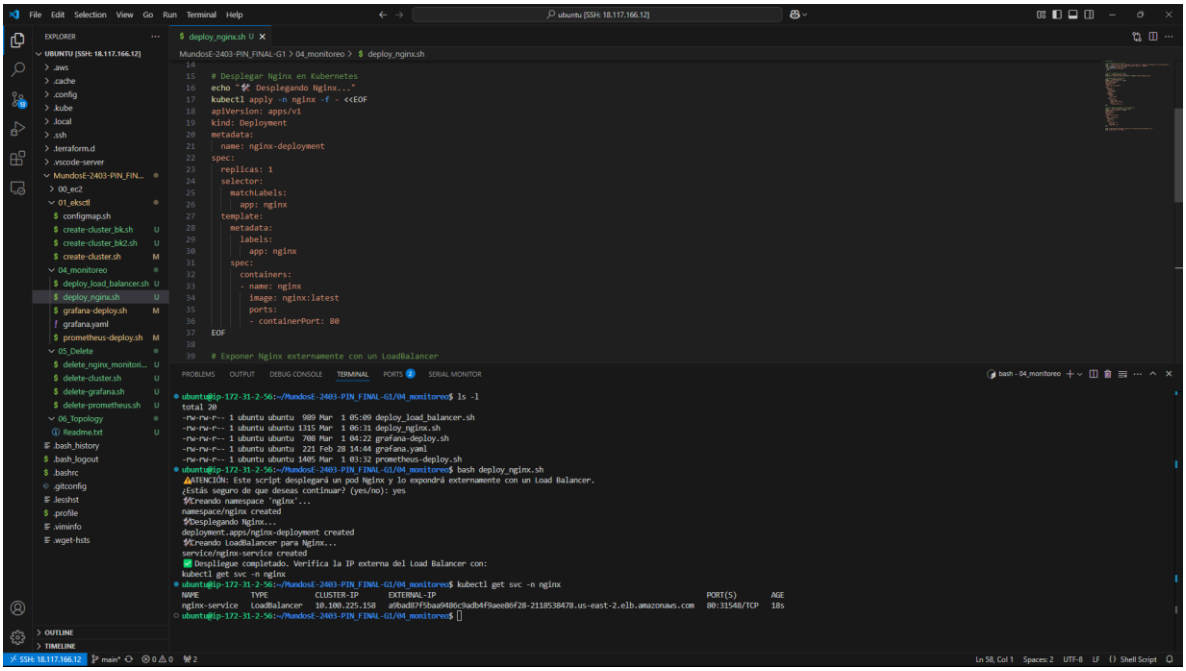



## Prometheus

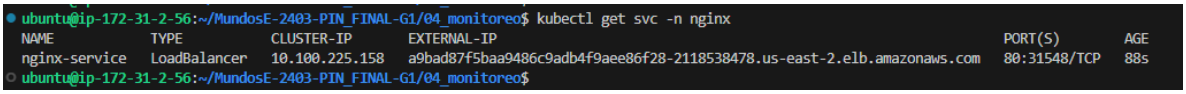
- TYPE: LoadBalancer → Prometheus también está expuesto públicamente.
- EXTERNAL-IP: ae4eaa9e7031f4347a4cc4e0ab4a3254-1427972924.us-east-2.elb.amazonaws.com

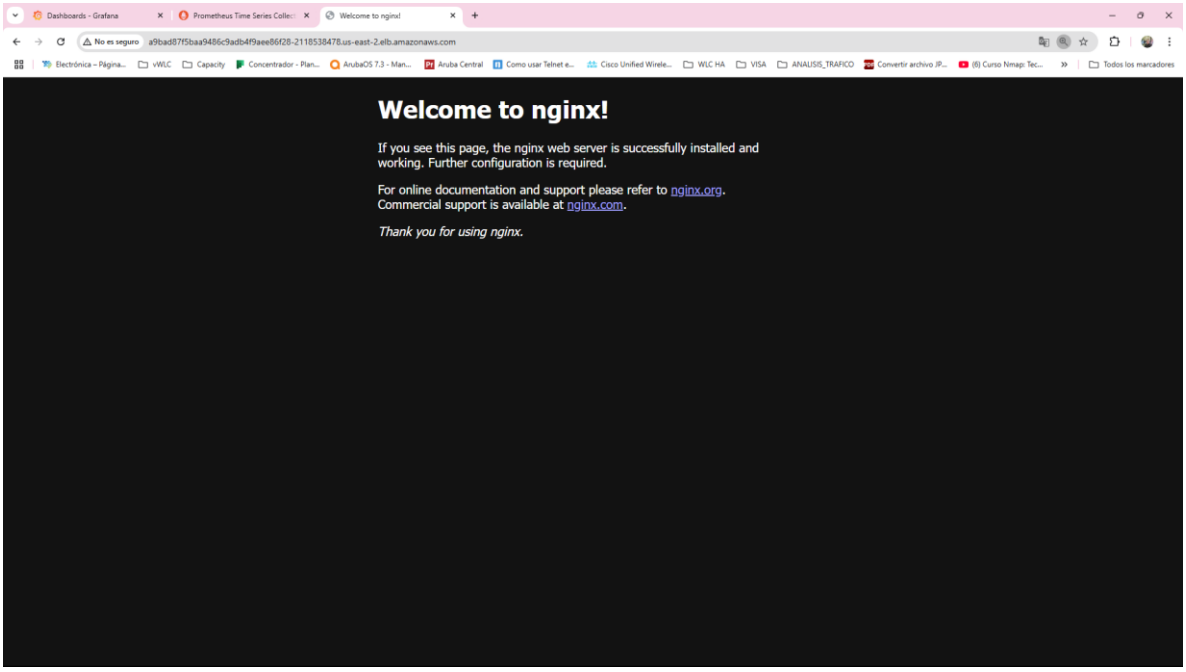



(Paso 17) Se despliega un Nginx en Kubernetes, para que el prometheus, pueda recolectar las métricas y con grafana podamos observarlas a través de un dashboard, también se accede externamente.

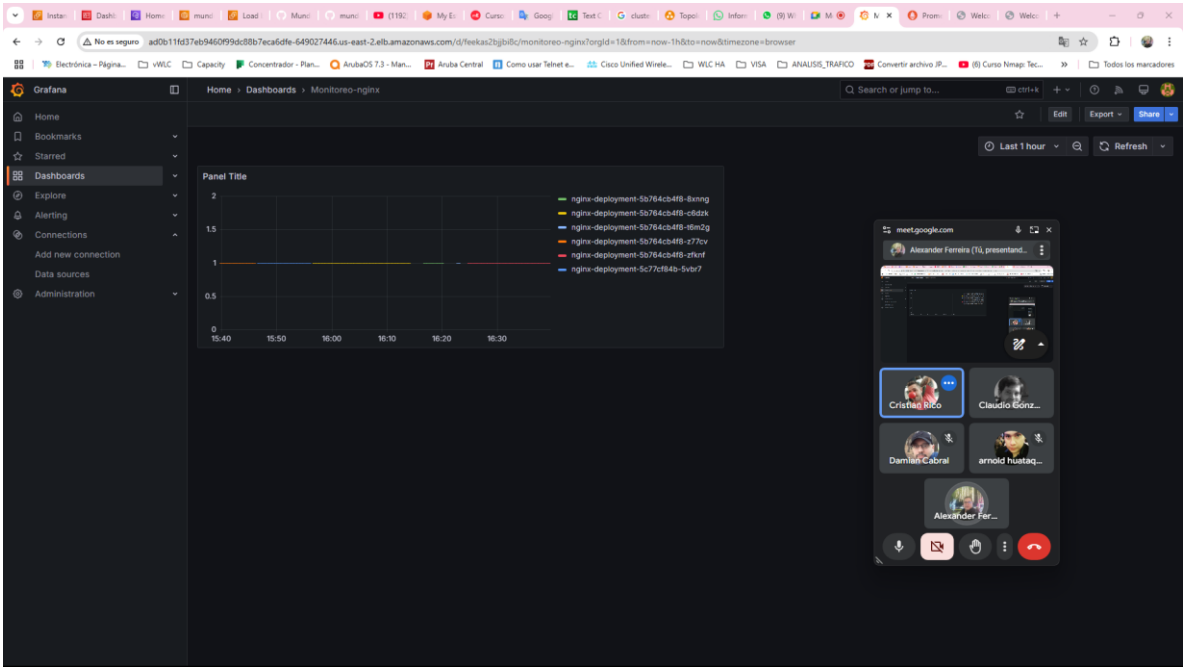


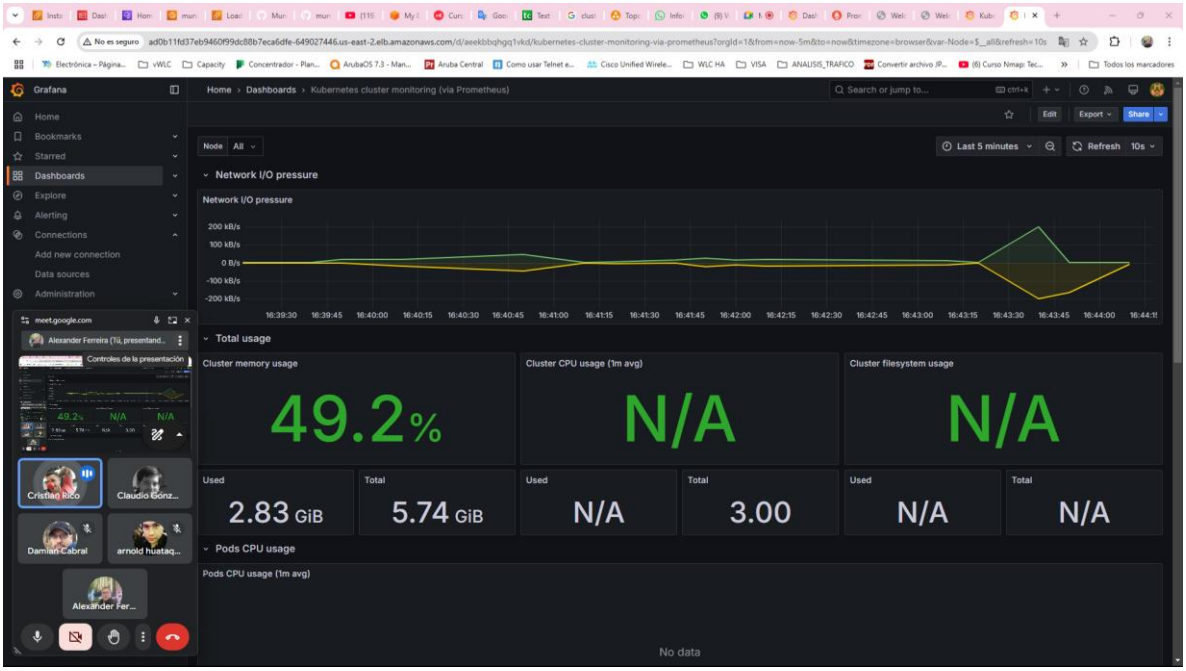

kubectl get svc -n nginx

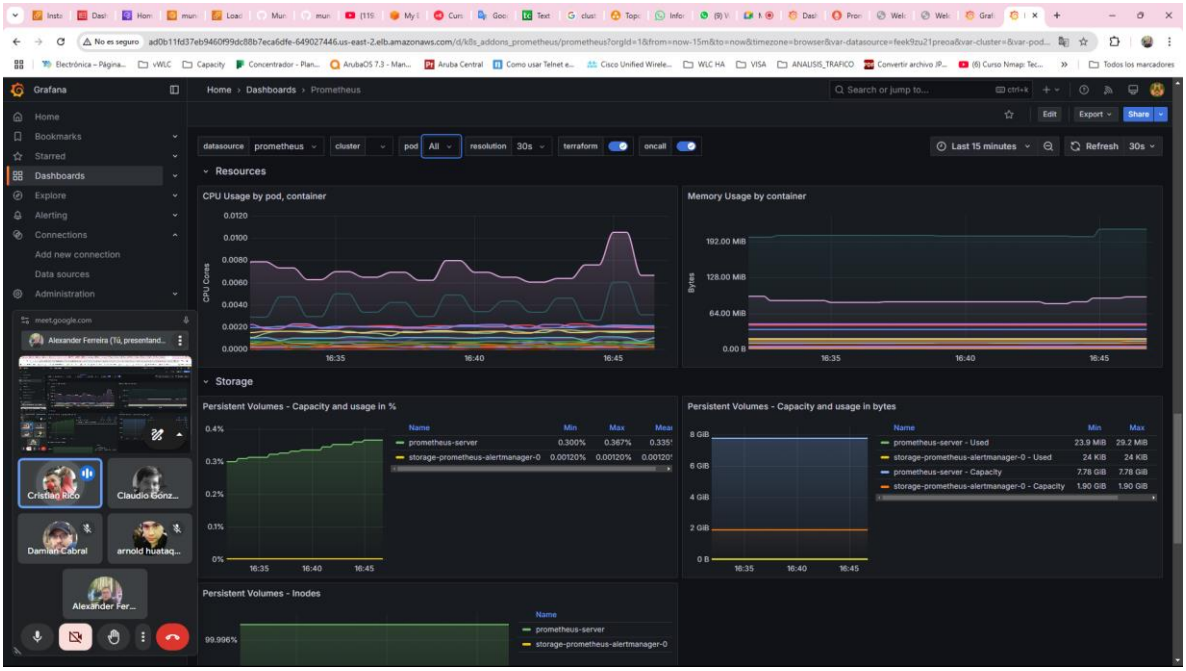


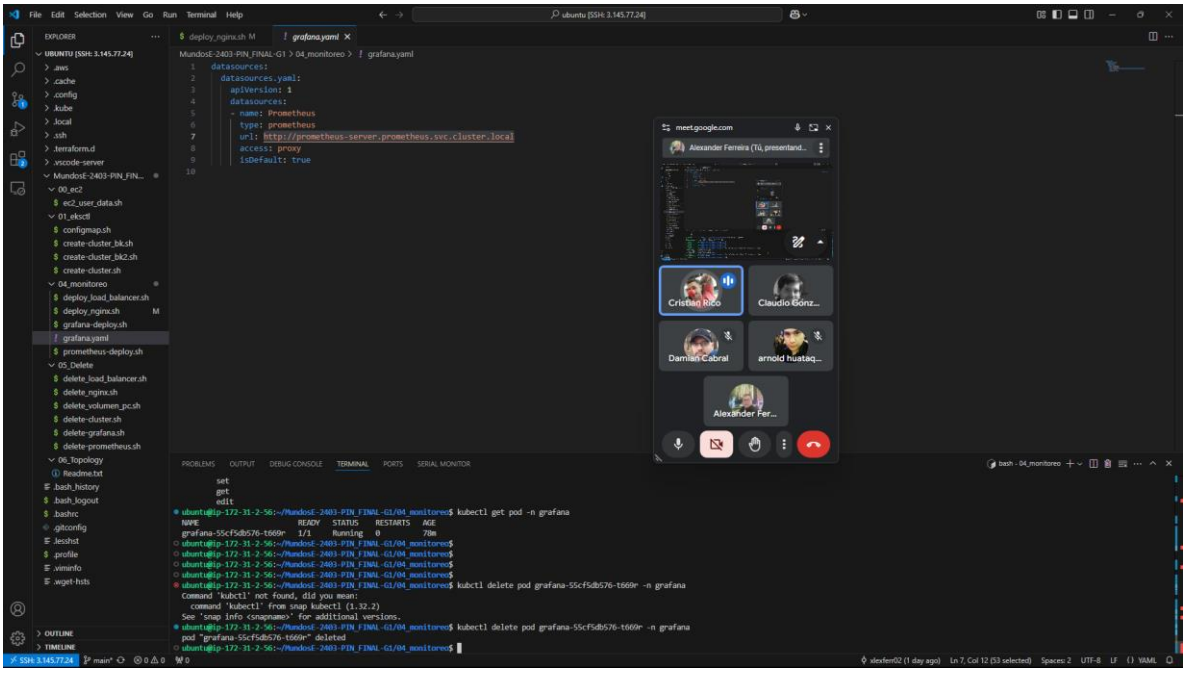

Çonfiguración grafana


Grafana
Home › Dashboards › Monitoreo-nginx
Panel Title
nginx-deployment-5b764cb4f8-8xnng
nginx-deployment-5b764cb4f8-c6dzk
nginx-deployment-5b764cb4f8-t6lm2g
nginx-deployment-5b764cb4f8-z77cv
nginx-deployment-5b764cb4f8-zfknf
nginx-deployment-5c77cf84b-5vbr7
meet.google.com
Cristian Rico
Claudio Gonz...
Damian Cabral
arnold huataq...
Alexander Fer...

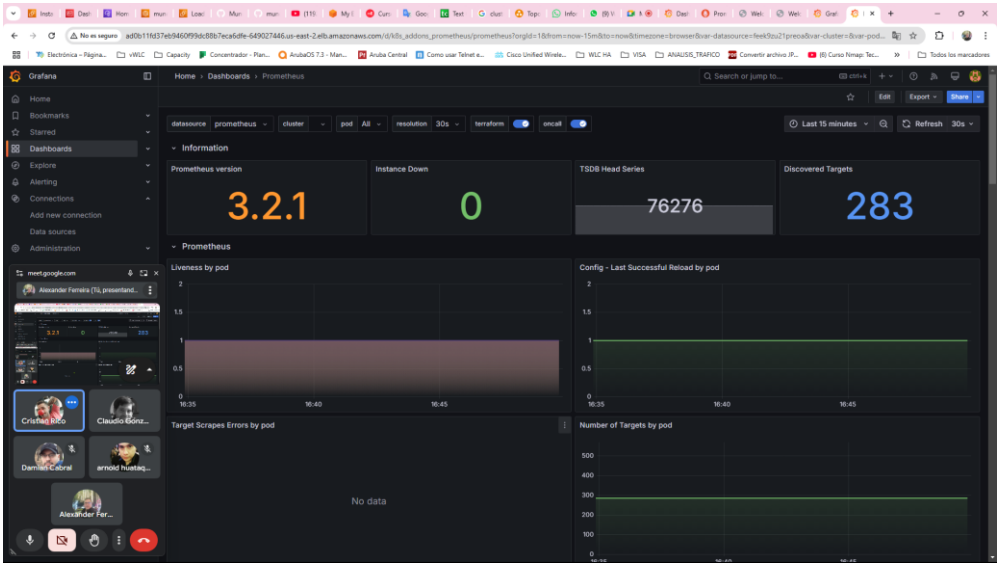

Grafana
Home › Dashboards › Kubernetes cluster monitoring (via Prometheus)
Network I/O pressure
Total usage
Cluster memory usage
49.2%
Cluster CPU usage (1m avg)
N/A
Cluster filesystem usage
N/A
Used
2.83 GiB
Total
5.74 GiB
Used
N/A
Total
3.00
Used
N/A
Total
N/A
Pods CPU usage
Pods CPU usage (1m avg)
No data




Bajamos POD de grafana




La configuración de guarda en el volumen persistente




## 4.6 FASE 6: Eliminación de recursos (Opcional)

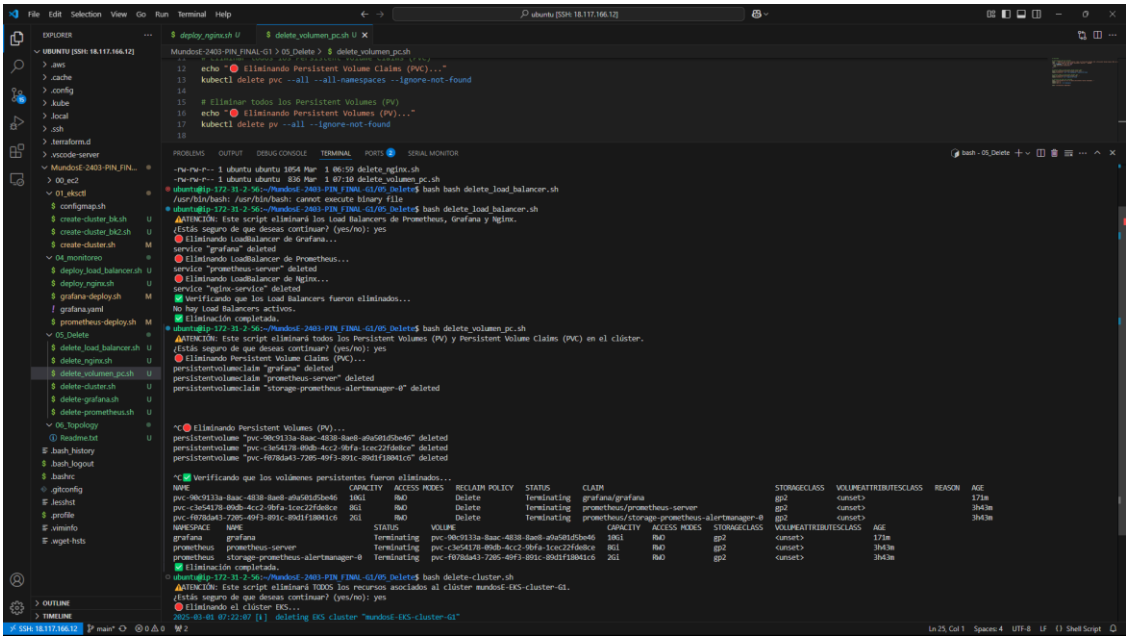
(Paso 17) Se eliminan los recursos creados si es necesario

Vamos a borrar los recursos en este orden:

Eliminar Load Balancers
Bash delete_load_balancer.sh

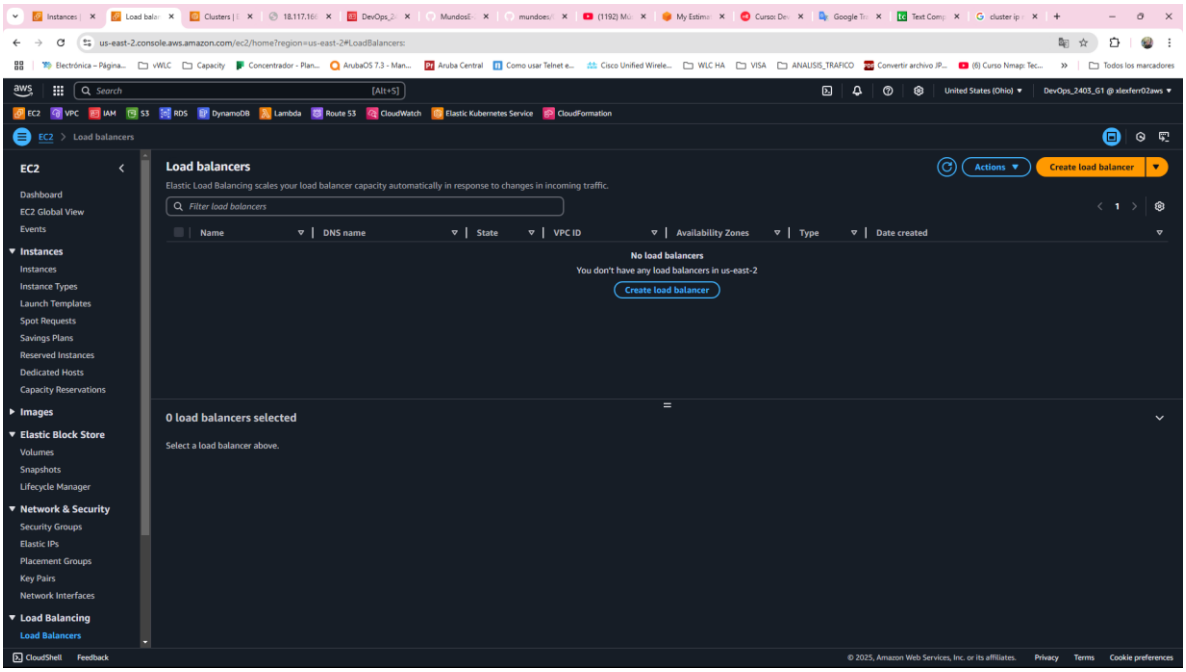
Eliminar volúmenes persistentes
Bash delete_volumen_pc.sh

Eliminar el clúster de EKS
bash delete-cluster.sh

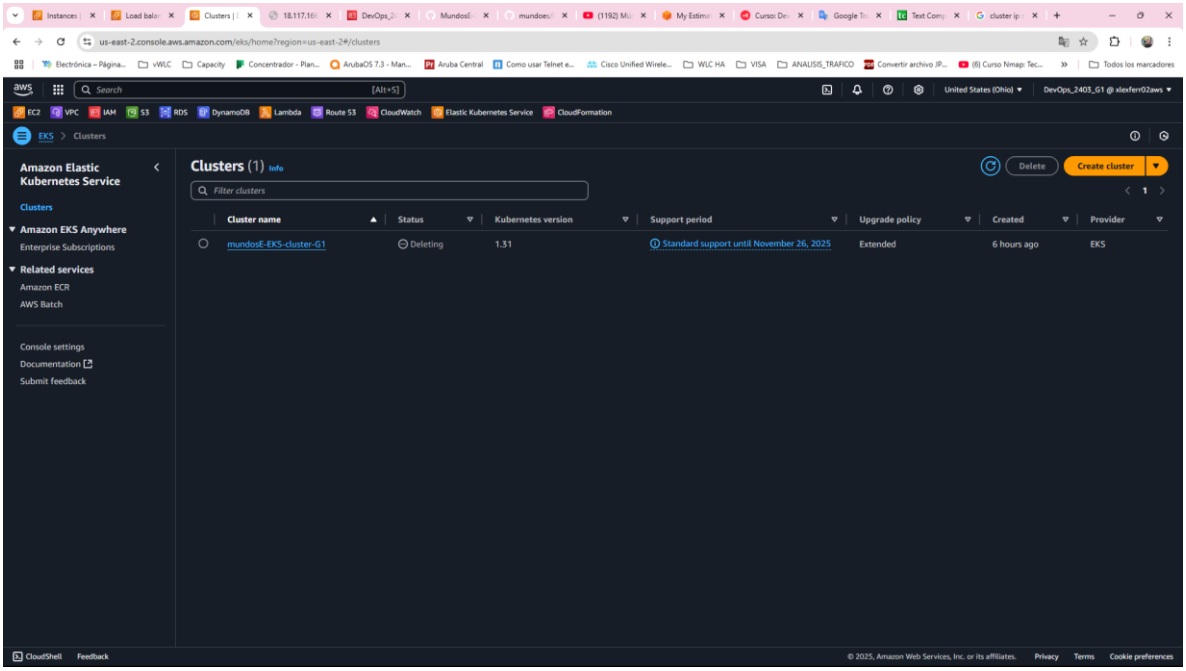


Verificar si AWS dejó recursos huérfanos
aws elb describe-load-balancers --query "LoadBalancerDescriptions[*].LoadBalancerName"
aws ec2 describe-volumes --query "Volumes[*].VolumeId"

Eliminación del Laod balancer




Eliminación del Cluster



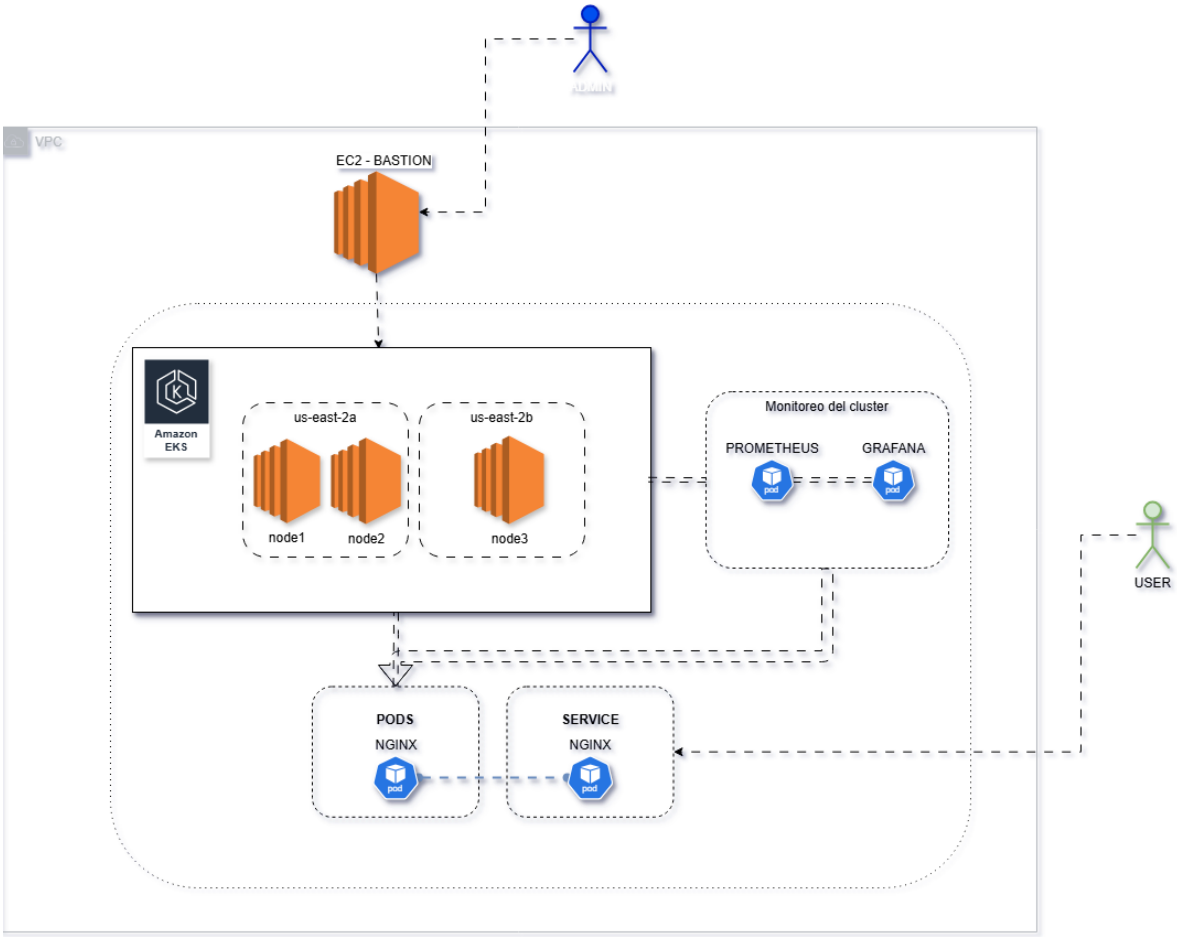
## ***5. Conclusiones***

Se logró automatizar completamente la infraestructura en AWS con eksctl, kubectl y helm, optimizando despliegues y reduciendo errores manuales. La configuración correcta de permisos, volúmenes y puertos fue clave para garantizar el funcionamiento estable de los servicios.

El monitoreo en tiempo real con Prometheus y Grafana permitió la observabilidad del clúster, implementando Exporters y ServiceMonitors para recolectar métricas críticas. Se utilizaron Load Balancers para exponer servicios como Nginx, asegurando accesibilidad y escalabilidad.

Se resolvieron desafíos en almacenamiento, compatibilidad de versiones y permisos mediante validaciones de storageClass y actualización de addons.

Se implementó una estrategia ordenada para la eliminación de recursos, evitando bloqueos en AWS. La modularidad del proyecto permite escalar y agregar nuevos servicios sin afectar la estabilidad. En conclusión, esta infraestructura automatizada y monitoreada con Kubernetes y AWS demuestra buenas prácticas de DevOps, garantizando eficiencia, seguridad y escalabilidad.




https://github.com/XLEXFERR02/MundosE-2403-PIN_FINAL-G1.git