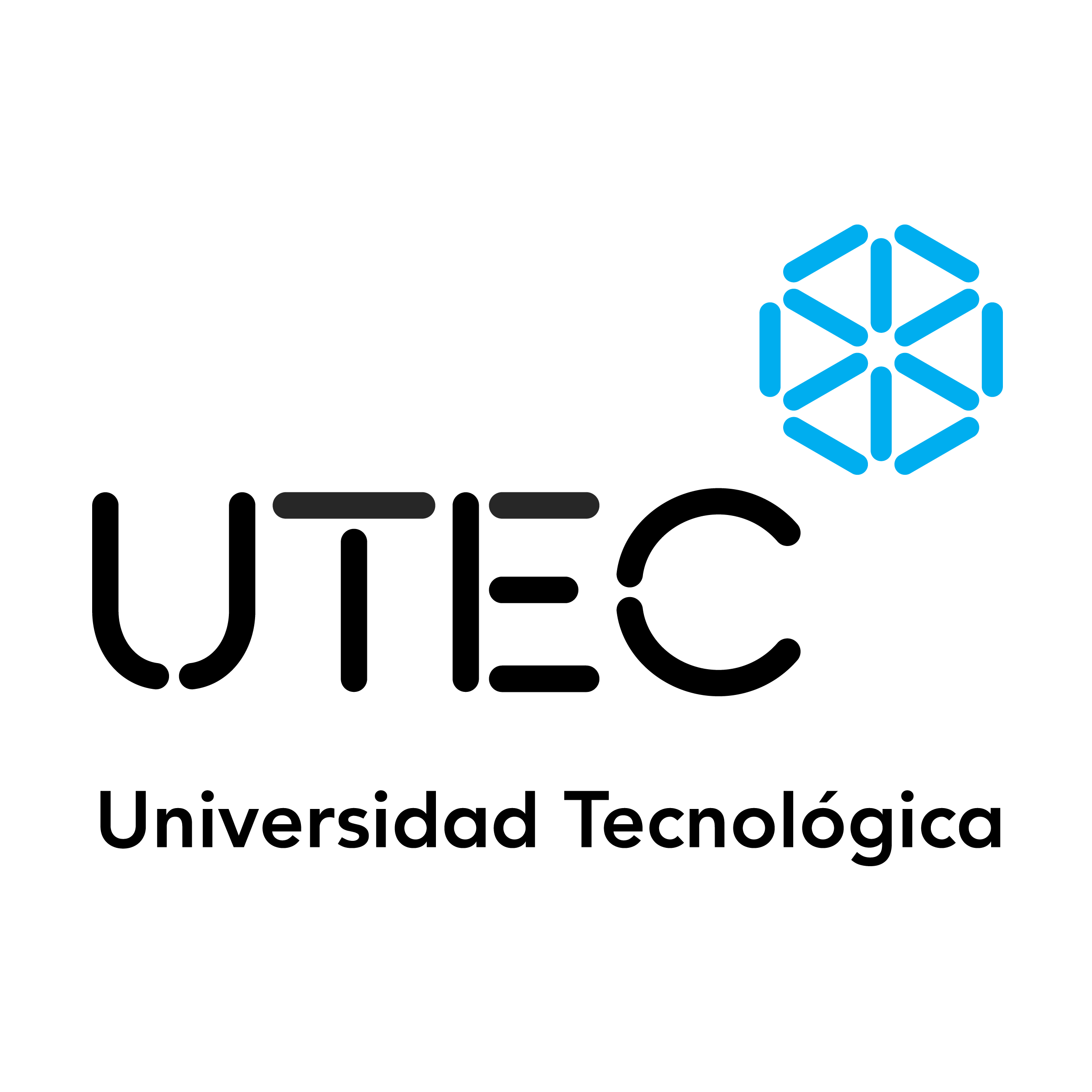
# Laboratorio AWS EKS



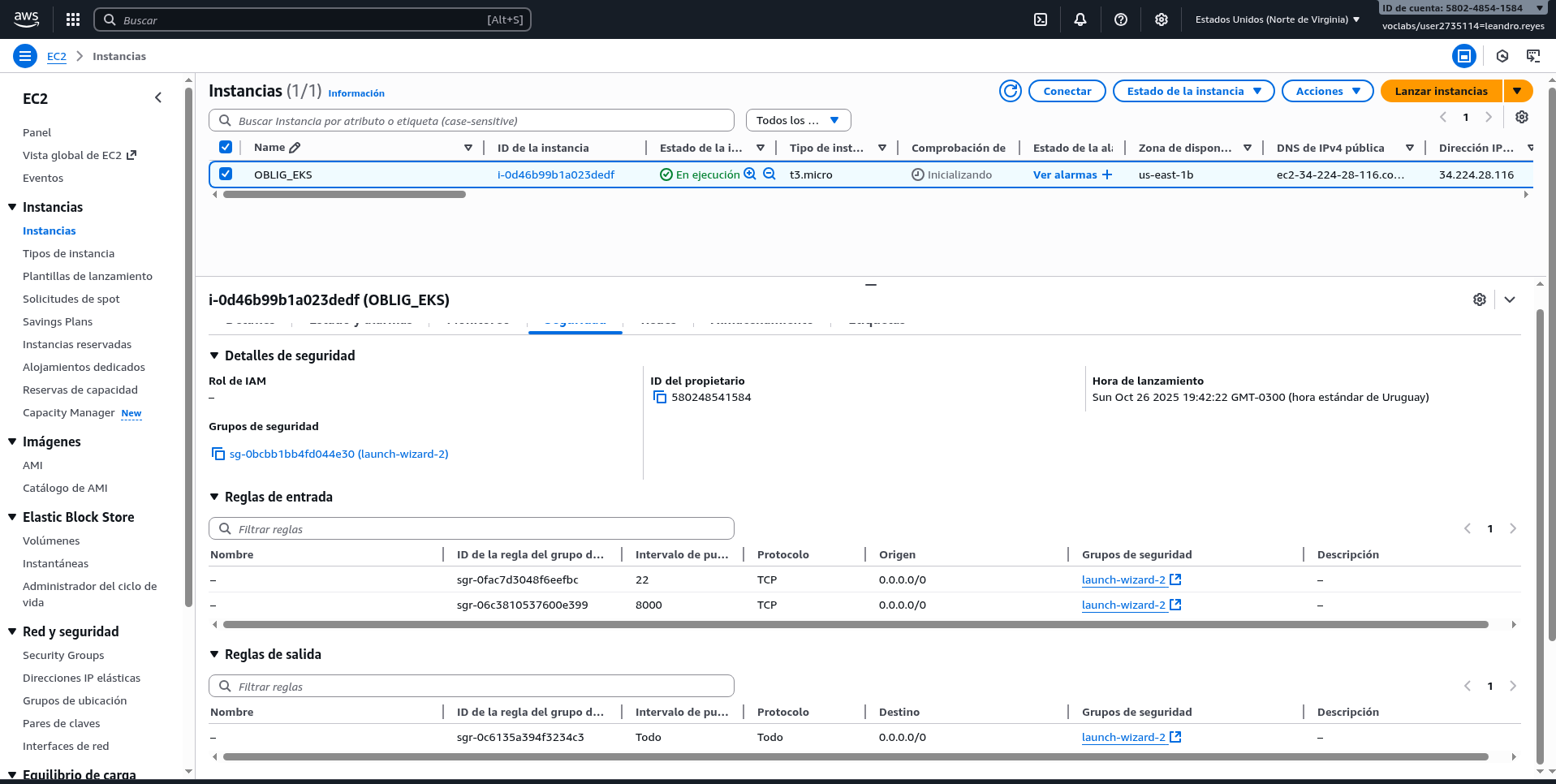
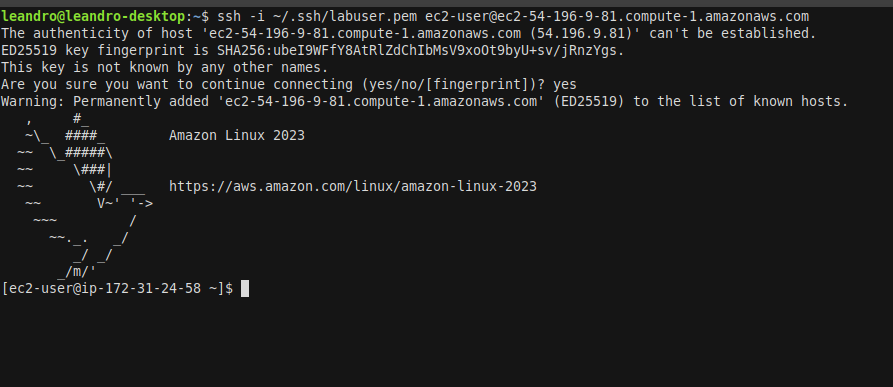
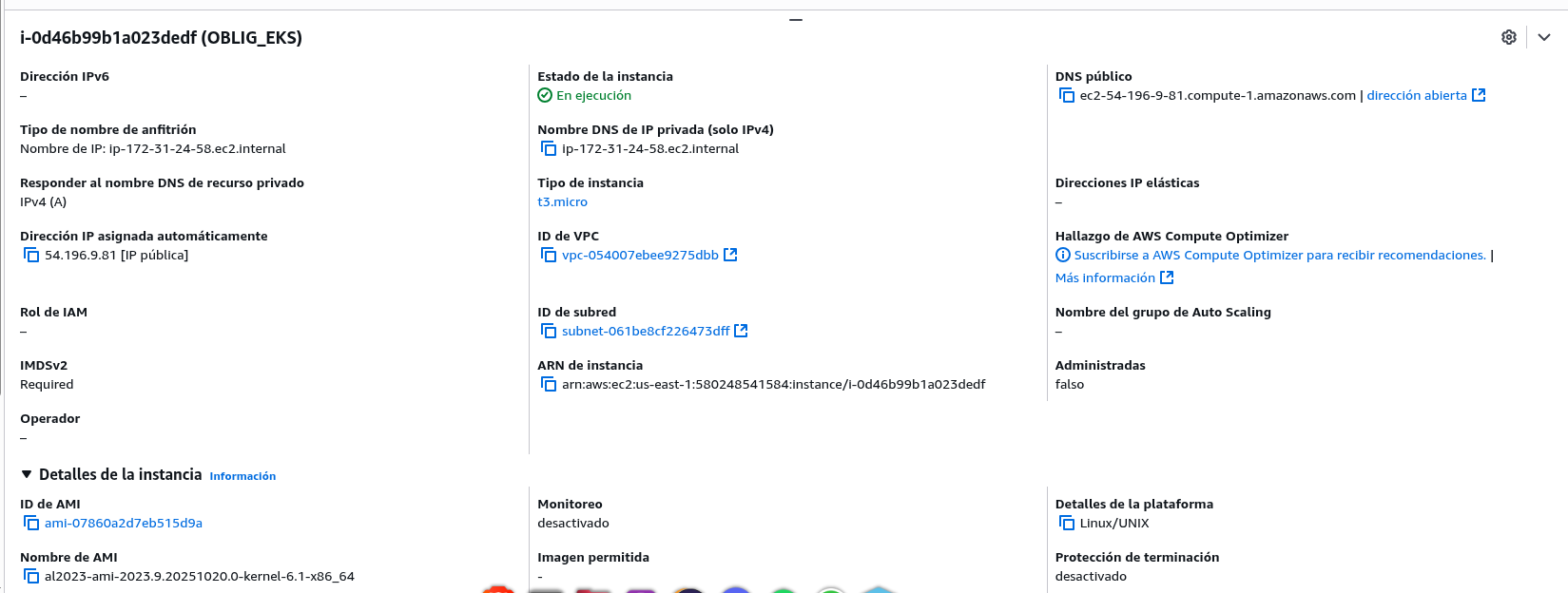
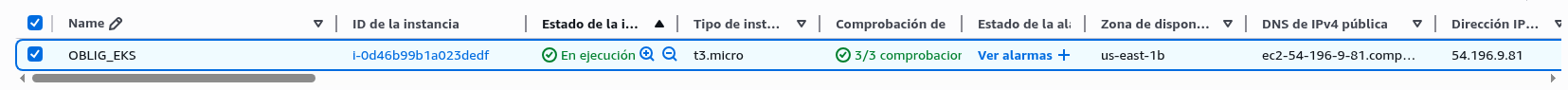
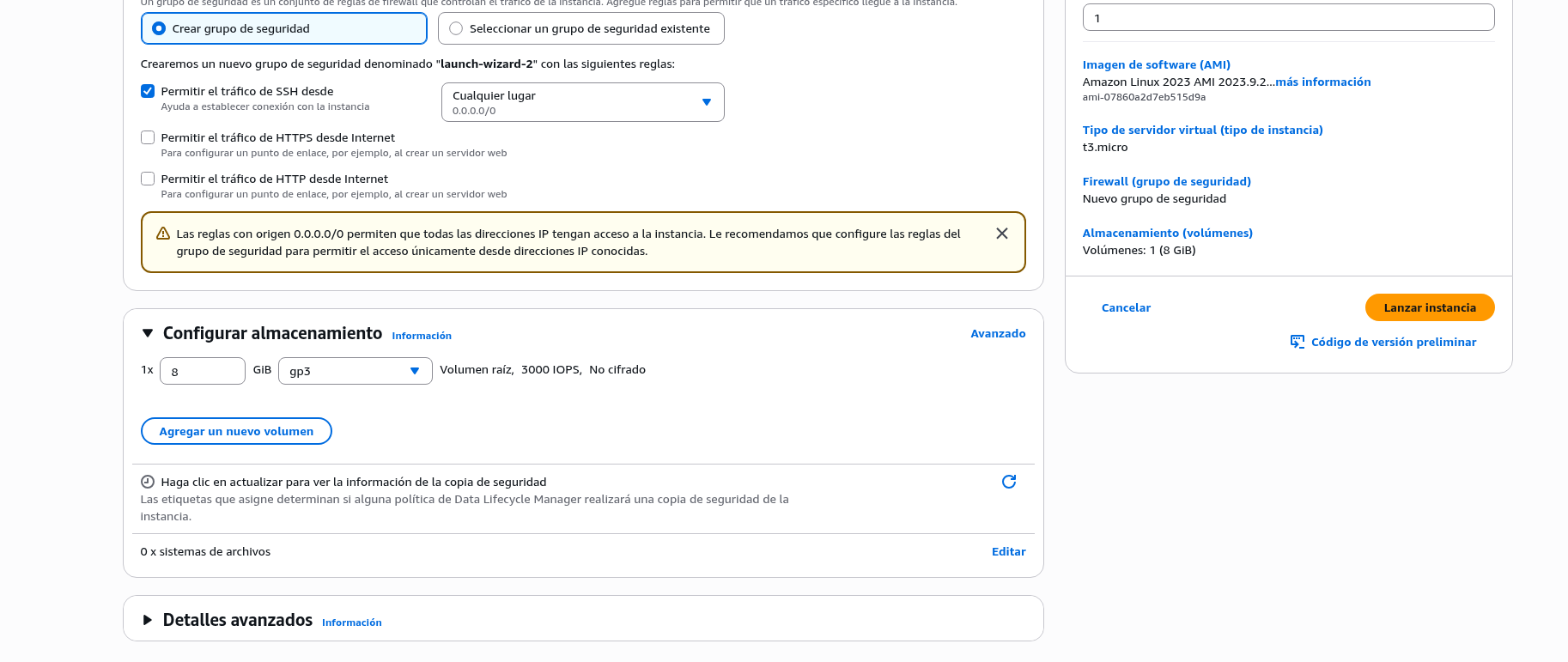
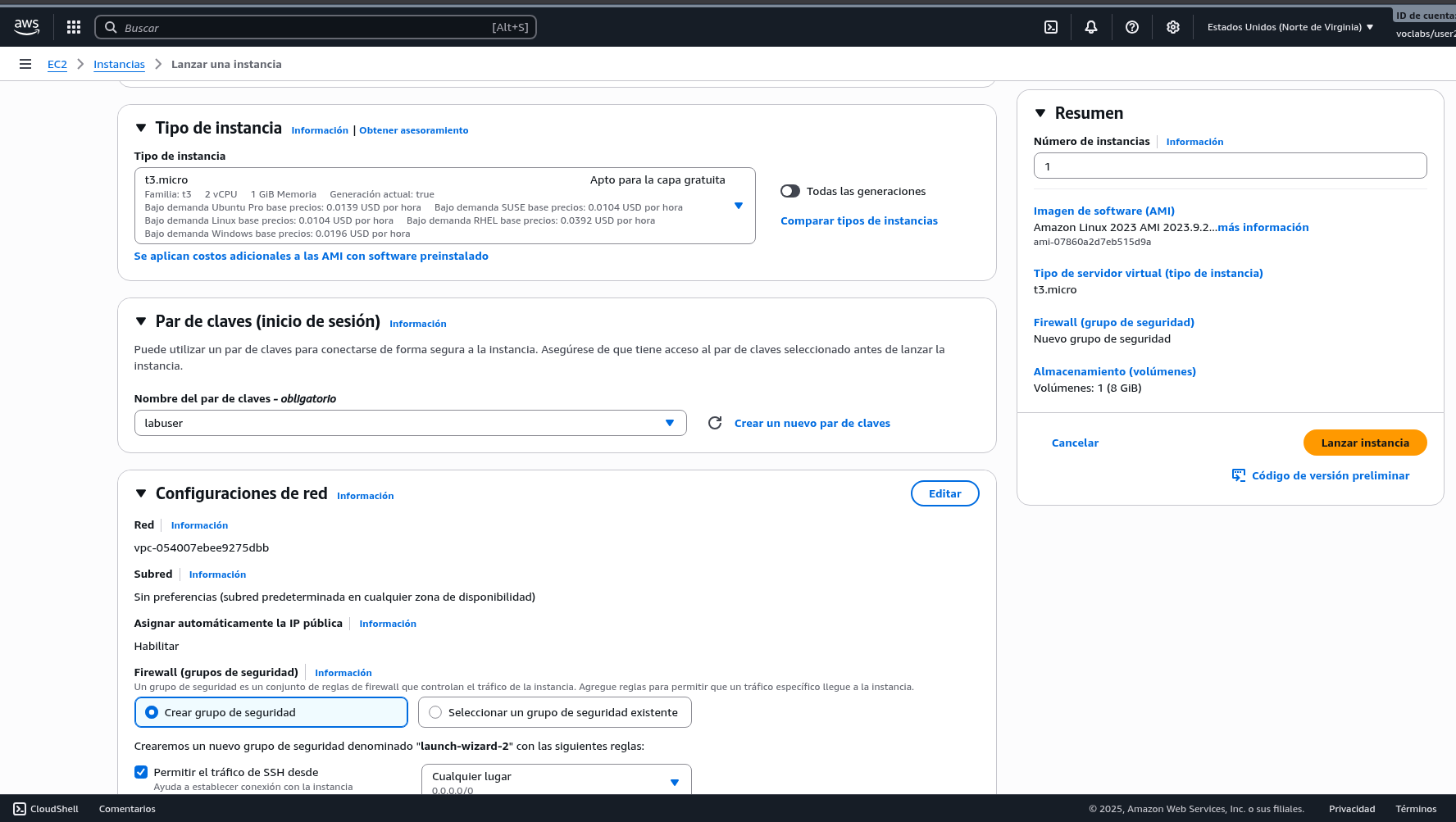
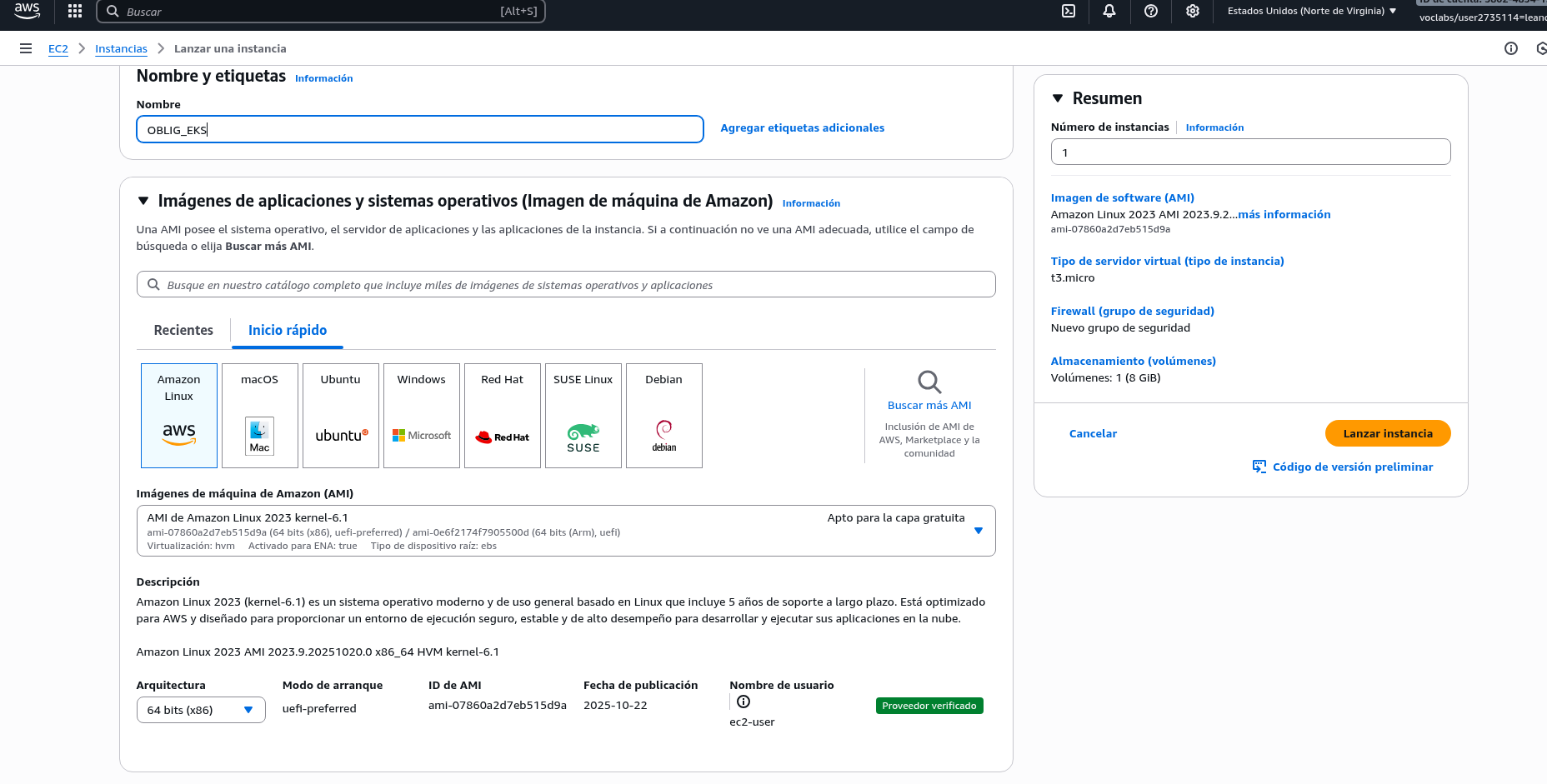
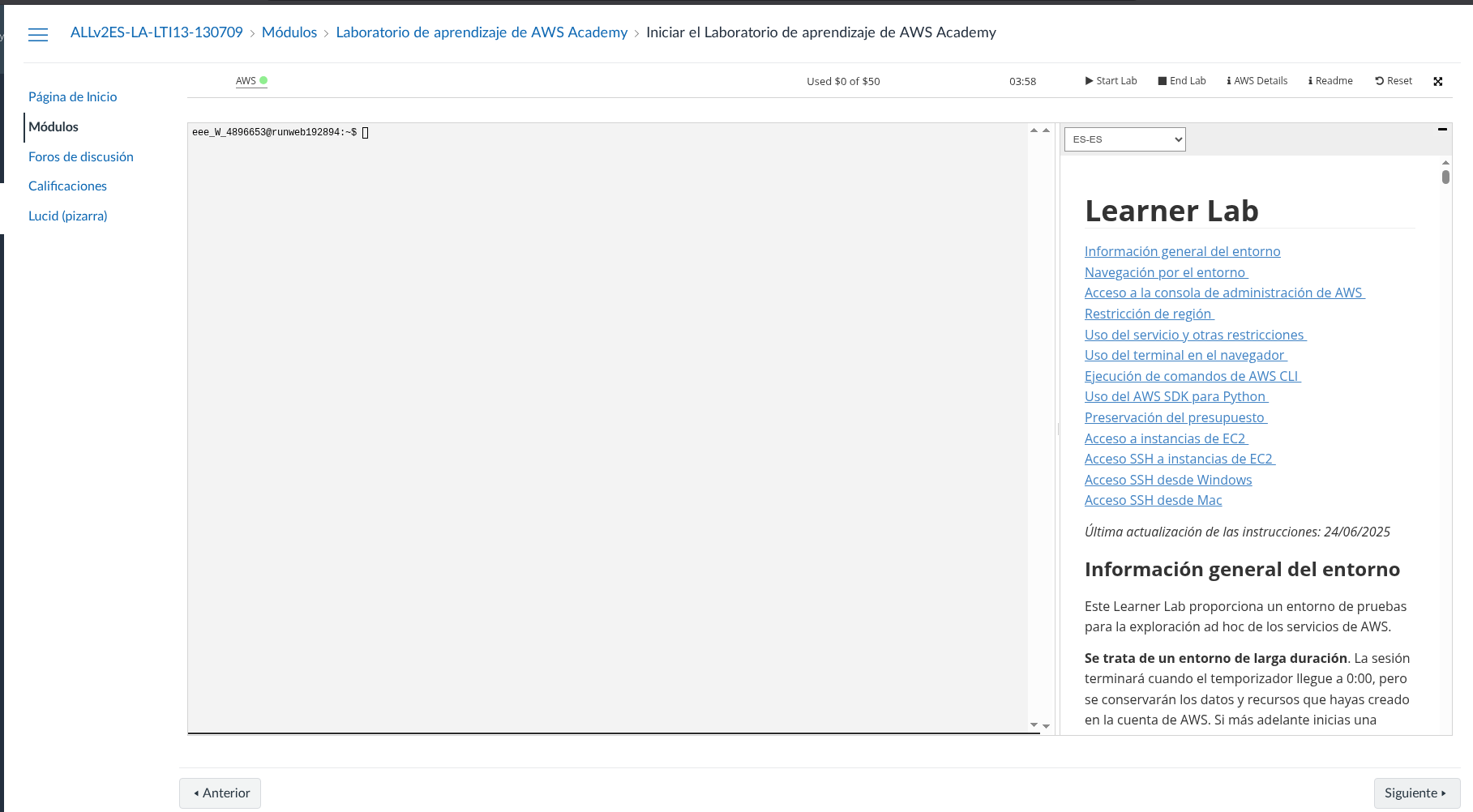
**Administración de Infraestructura**



Leandro Reyes  
20/11/2025

## Preparación del Entorno AWS

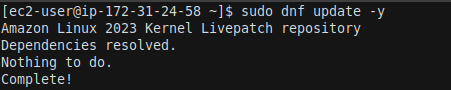
Creación de la instancia EC2, descarga de la clave .pem, configuración del Grupo de Seguridad y conexión SSH.



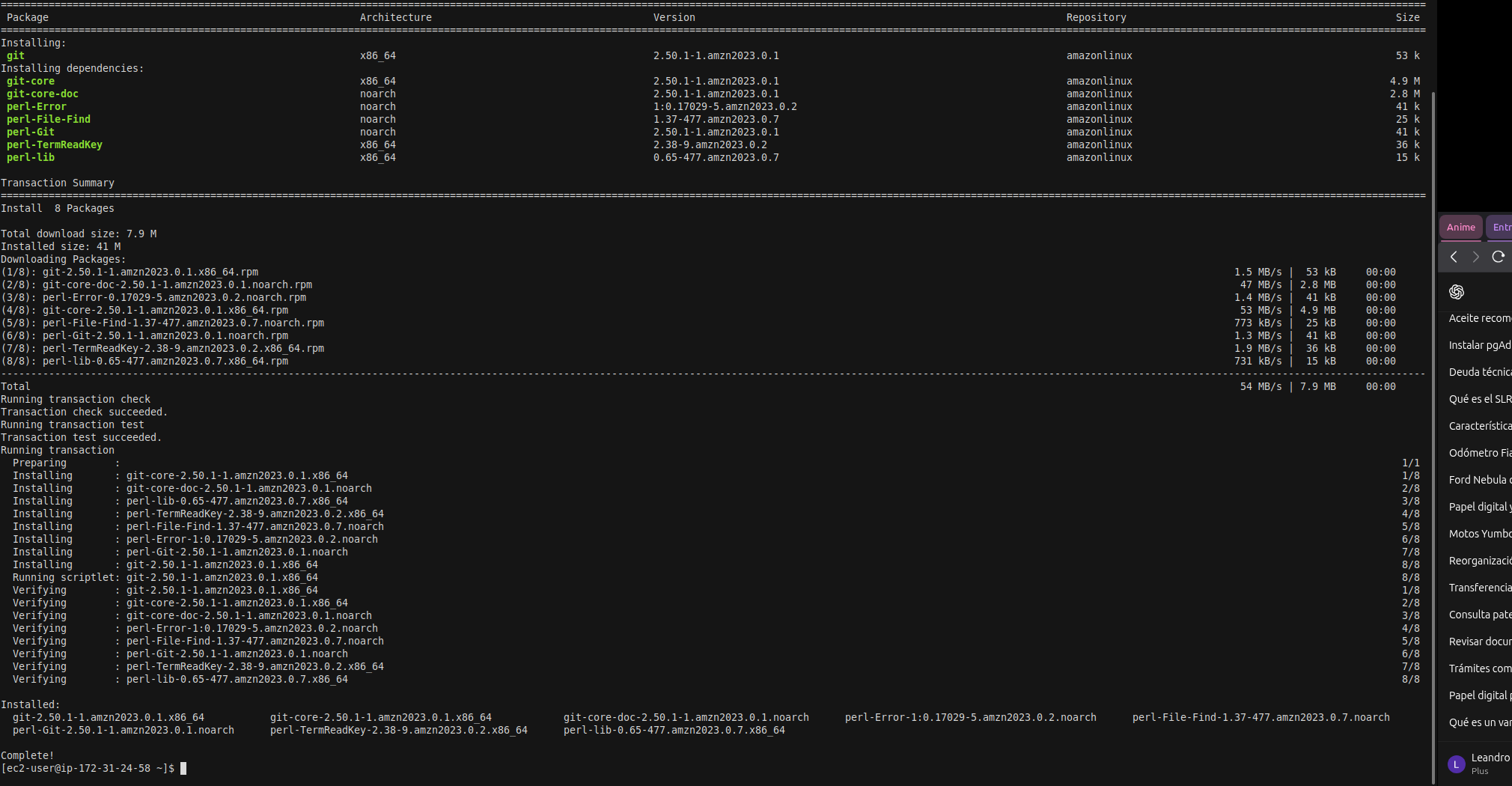
## Configuración del servidor

Instalación de Docker, Git, Python, etc. y verificación del entorno.

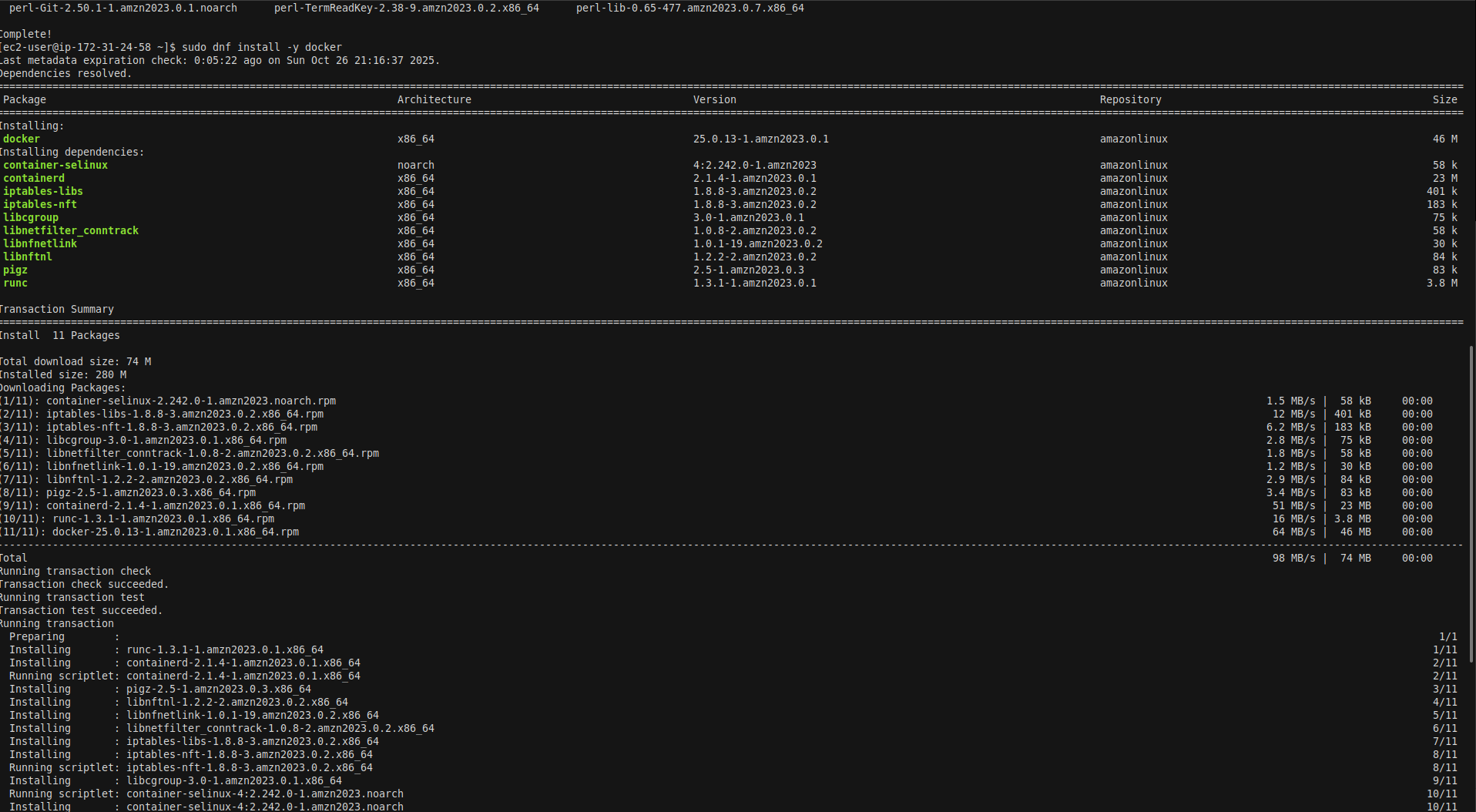
sudo dnf update –y



sudo dnf install –y git

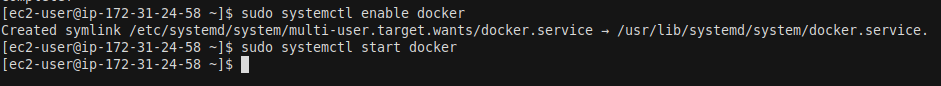


sudo dnf install -y docker

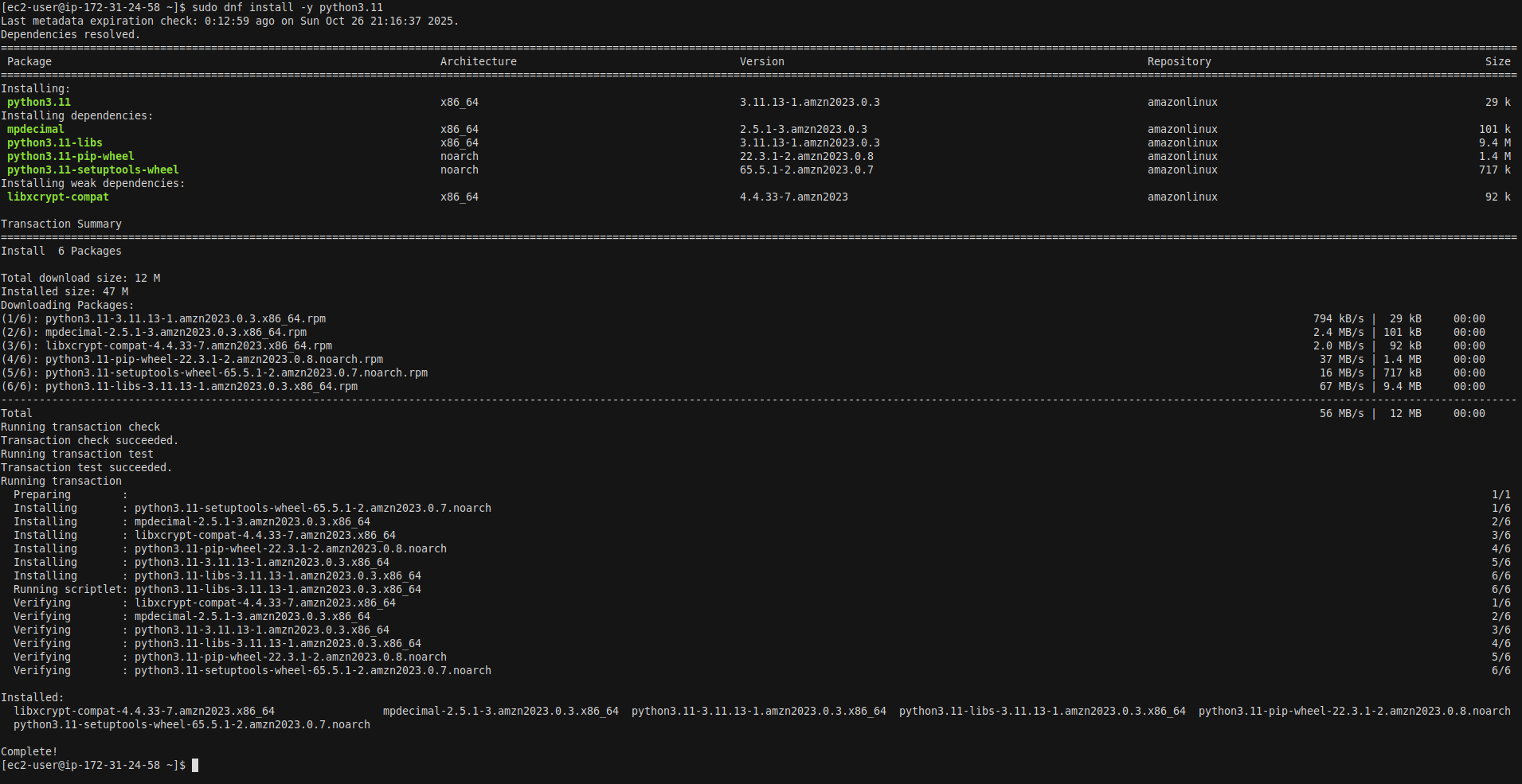


sudo systemctl enable docker

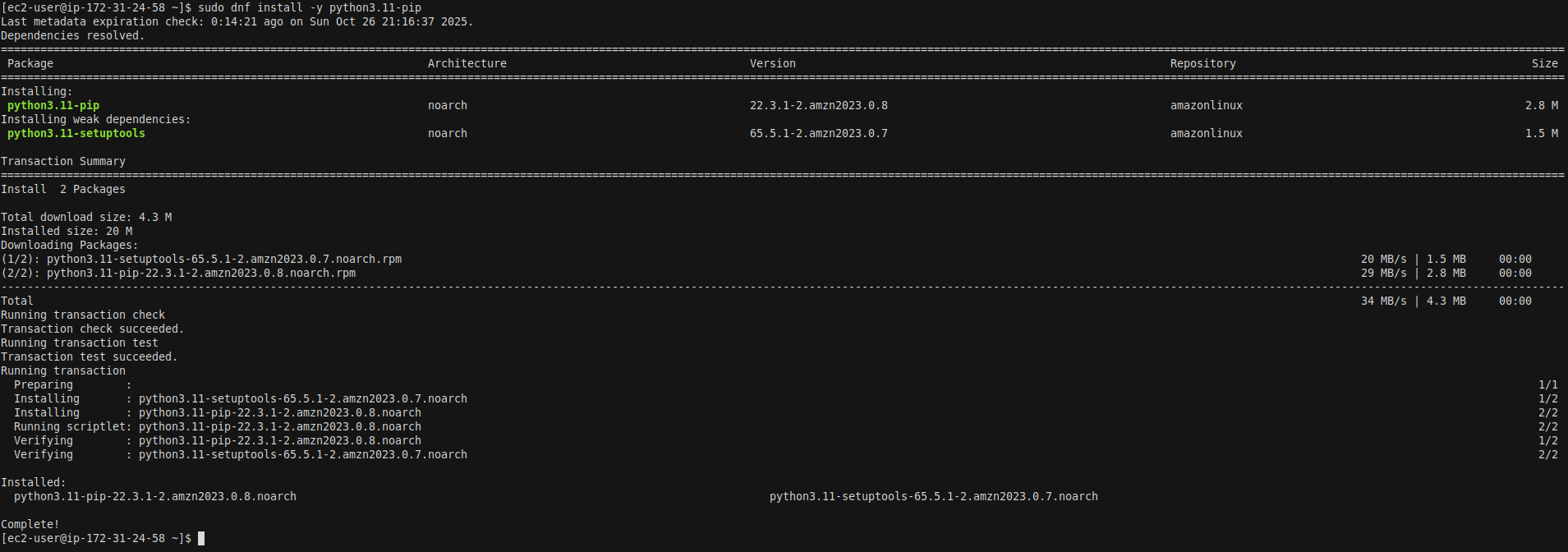
sudo systemctl start docker



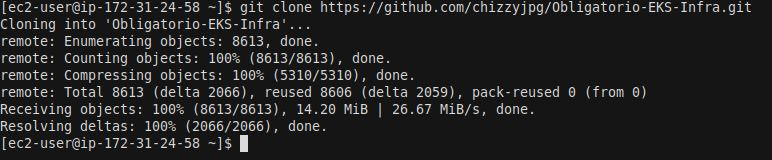
sudo dnf install -y python3.11



sudo dnf install -y python3.11-pip



git clone https://github.com/chizzyjpg/Obligatorio-EKS-Infra.git

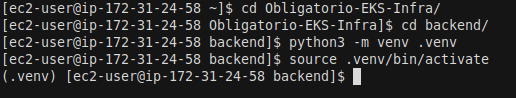


## cd Obligatorio-EKS-Infra

cd backend

python3 -m venv .venv

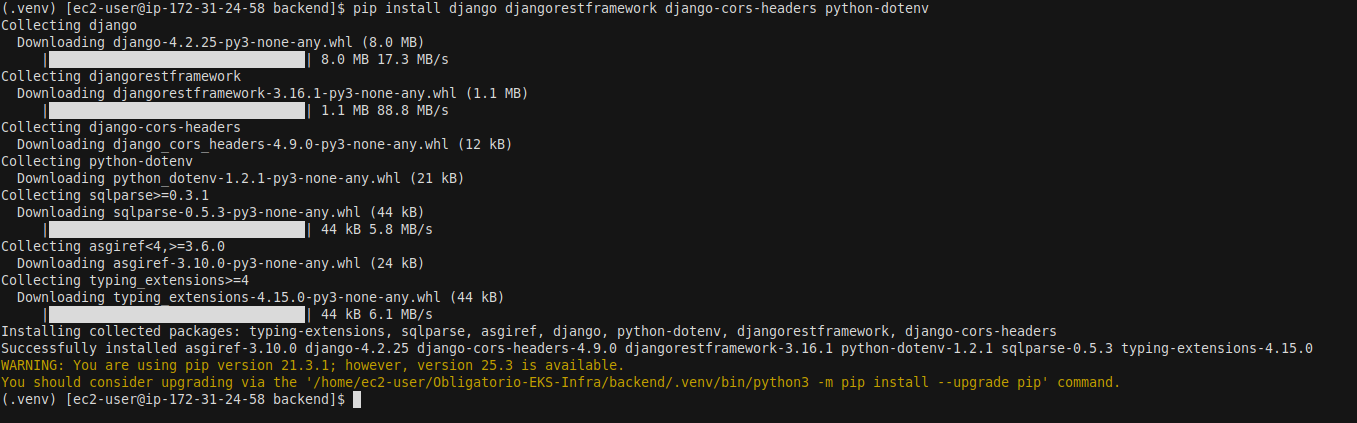
source .venv/bin/activate



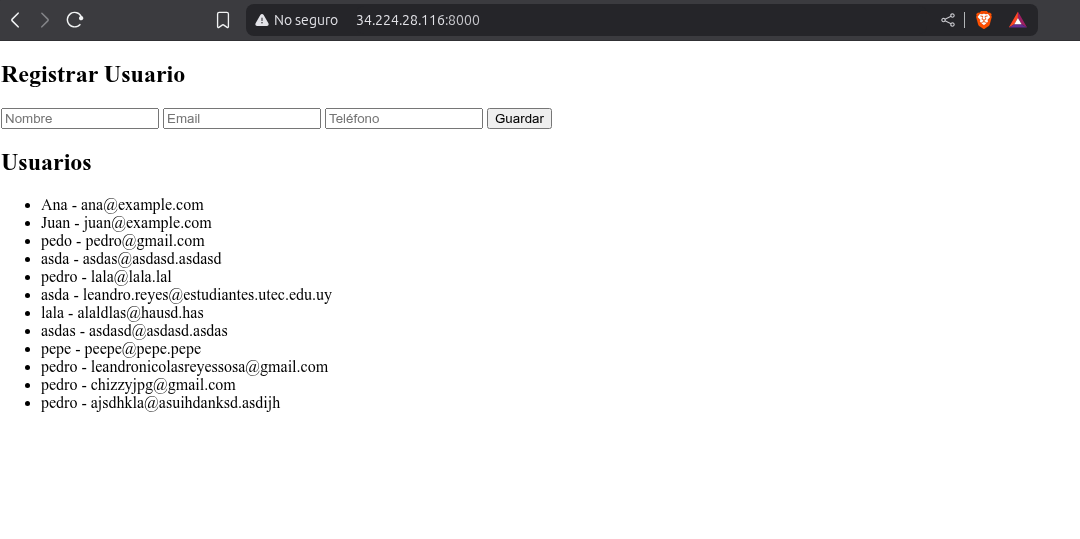
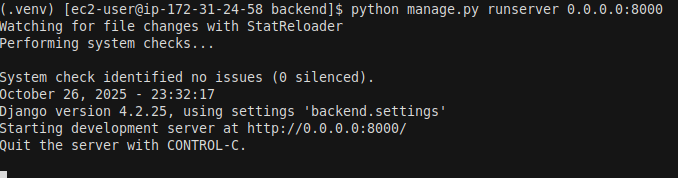
## Despliegue de la Aplicación Monolítica

Clonación del repositorio, instalación de Django y ejecución del backend monolítico.

pip install django djangorestframework django-cors-headers python-dotenv



python manage.py runserver 0.0.0.0:8000



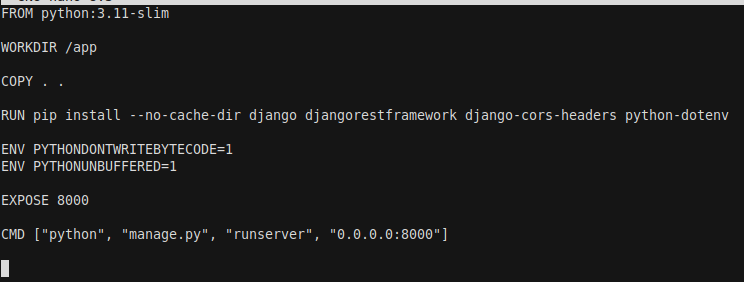
## Conversión a Microservicios

Creación de contenedores Docker para cada componente y verificación local.

Tengo todo separado por carpetas, en una tengo el backend, frontend y notificaciones

Backend

nano Dockerfile



FROM python:3.11-slim # mini version de python

WORKDIR /app # trabajamos en esta carpeta

COPY . . # copia todos los archivos del proyecto

RUN pip install --no-cache-dir django djangorestframework django-cors-headers python-dotenv # instala django

ENV PYTHONDONTWRITEBYTECODE=1 # evita que python cree archivos

ENV PYTHONUNBUFFERED=1 # muestra las notificaciones en tiempo real

EXPOSE 8000 # escucha el puerto 8000

CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"] # comando para ejecutar el contenedor

docker build -t backend-service .



Solución:

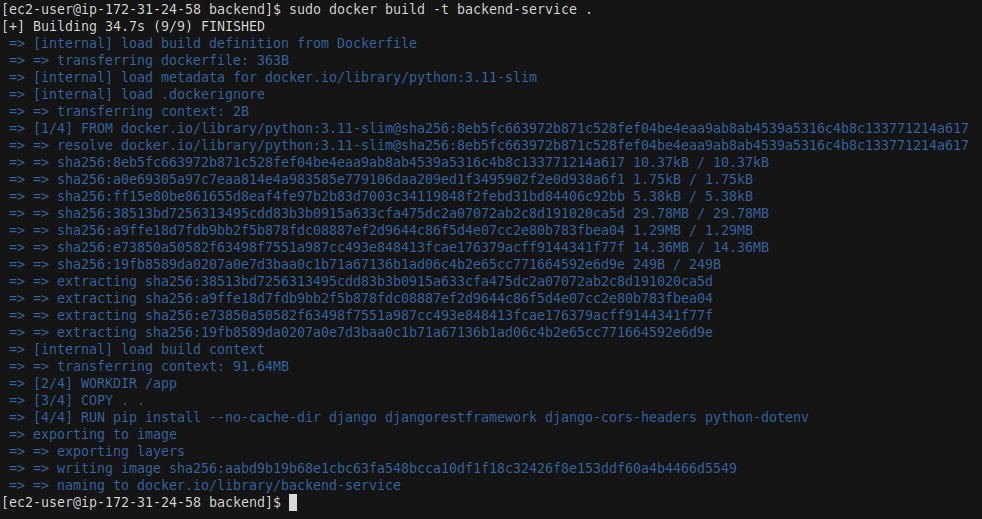
Con el comando groups verificas si el usuario pertenece al grupo de docker, en el caso de que no de una salida ejemplo



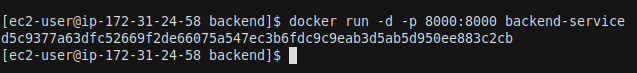
Usando el comando

sudo usermod –aG docker ec2-user

Ingresas el usuario al grupo de docker y al salir y volver a entrar estárías dentro del grupo eso ya debería solucionar el problema

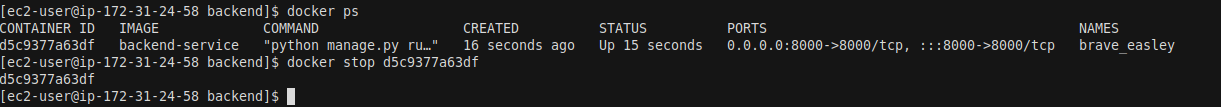


docker run -d -p 8000:8000 backend-service



docker ps

## docker stop d5c9377a63df

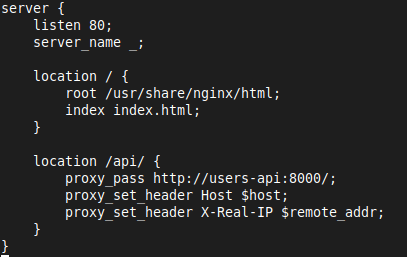


docker rm d5c9377a63df # borra la imagen del docker

Frontend

Dentro de forntend debemos crear un archivo

nano nginx.conf



server {

listen 80; # Escucha el puerto 80

server\_name \_; # todos los posibles nombre de dominio

location / {

root /usr/share/nginx/html; #ubicacion del .html

index index.html; # nombre del index

}

location /api/ {

proxy\_pass <http://users-api:8000/;> #redirige todo al backend

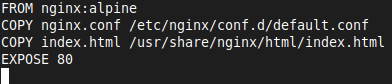
proxy\_set\_header Host $host; # mantiene la ip original

proxy\_set\_header X-Real-IP $remote\_addr; # envia al backend

}

}

nano Dokerfile



FROM nginx:alpine # version liviana de nginx

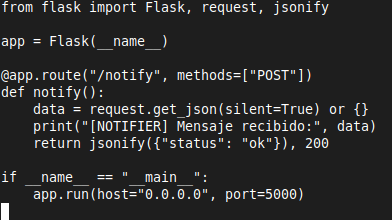
COPY nginx.conf /etc/nginx/conf.d/default.conf #configura el servidor

COPY index.html /usr/share/nginx/html/index.html #usa html estatico

EXPOSE 80 #escucha en el puerto 80

Notifier

nano app.py



from flask import Flask, request, jsonify # flask crea la web, request escucha http y jsonify crea respuestas http

app = Flask(\_\_name\_\_) # instancia la aplicacion

@app.route("/notify", methods=["POST"]) # define la ruta

def notify():

data = request.get\_json(silent=True) or {} # lee el post si no hay datos deja vasio

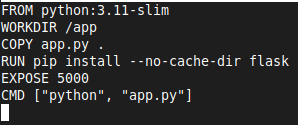
print("[NOTIFIER] Mensaje recibido:", data) # imprime en consola una notificación

return jsonify({"status": "ok"}), 200 # devuelve notificacion ok con codigo 200

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host="0.0.0.0", port=5000) # escucha en todas las interfaces en el puerto 5000

nano Dockerfile



FROM python:3.11-slim #base liviana de python

WORKDIR /app

COPY app.py .

RUN pip install --no-cache-dir flask # instala flask

EXPOSE 5000 #escucha puerto 5000

CMD ["python", "app.py"] # ejecuta en consola app.py

Ahora creamos el compositor de la pagina

nano docker-compose.yml



version: "3.9" #version del compositor

services:

users-api:

build:

context: ./backend #levanta el docker del backend

container\_name: users-api

ports:

- "8000:8000"

environment:

- DJANGO\_ALLOWED\_HOSTS=\*

- NOTIFIER\_URL=http://notifier:5000/notify # llama al microservicio de notificaciones

depends\_on:

- notifier

notifier:

build:

context: ./notifier # levanta el notifier

container\_name: notifier

ports:

- "5000:5000"

frontend:

build:

context: ./frontend # levanta el frontend

container\_name: frontend

ports:

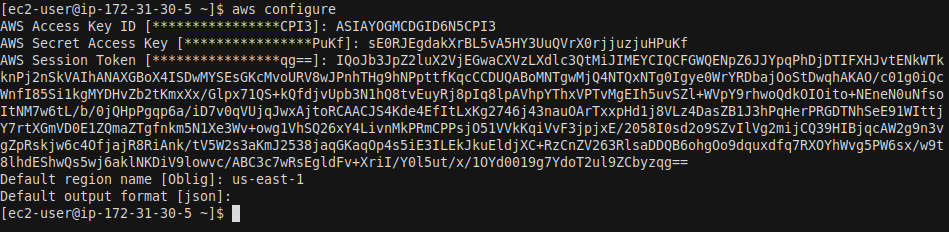
- "80:80"

depends\_on:

- users-api

## Publicación en AWS ECR

Carga de las imágenes Docker a los repositorios ECR para su uso en EKS.



aws configure

Key: ASIAYOGMCDGID6N5CPI3

secret\_access\_Key: sE0RJEgdakXrBL5vA5HY3UuQVrX0rjjuzjuHPuKf

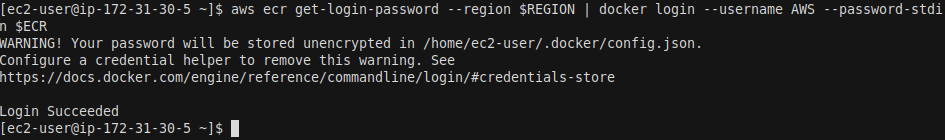
Token: 



ACCOUNT\_ID=580248541584

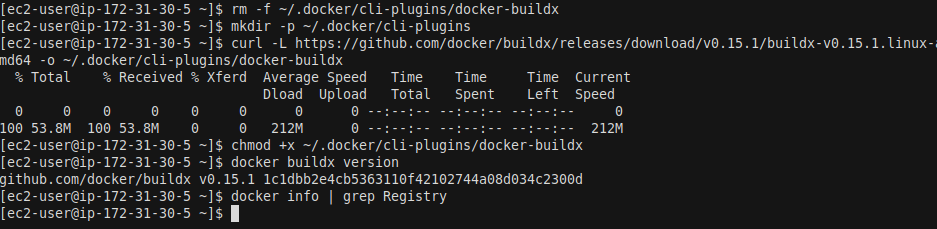
REGION=us-east-1

ECR="${ACCOUNT\_ID}.dkr.ecr.${REGION}.amazonaws.com"



aws ecr get-login-password --region $REGION | docker login --username AWS --password-stdin $ECR

El error dice que tengo una version de buildx incorrecto en mi sistema operativo esto se arregla borrando el que ya tengo e instalando uno nuevo



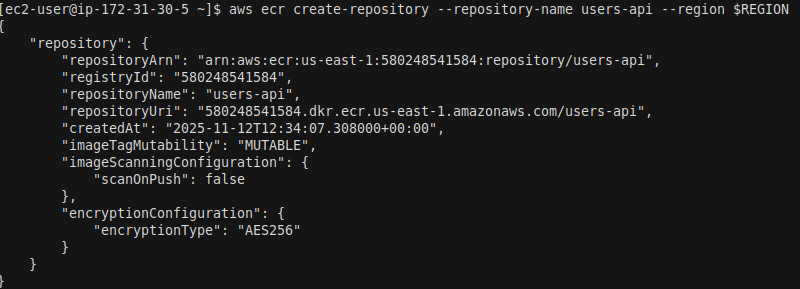
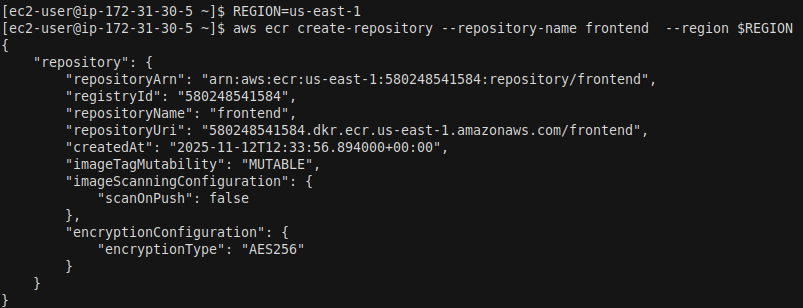
rm -f ~/.docker/cli-plugins/docker-buildx

mkdir -p ~/.docker/cli-plugins

curl -L https://github.com/docker/buildx/releases/download/v0.15.1/buildx-v0.15.1.linux-amd64 -o ~/.docker/cli-plugins/docker-buildx

chmod +x ~/.docker/cli-plugins/docker-buildx

docker buildx version

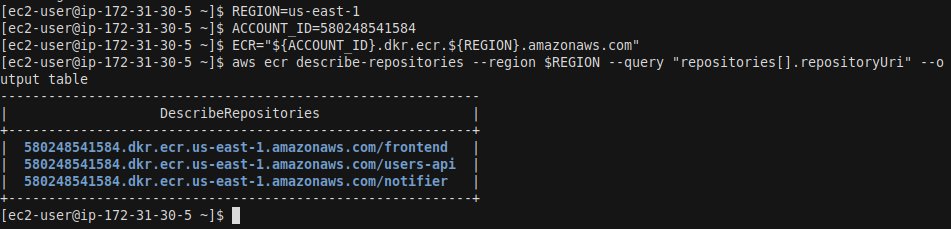


REGION=us-east-1

aws ecr create-repository --repository-name frontend --region $REGION

aws ecr create-repository --repository-name users-api --region $REGION

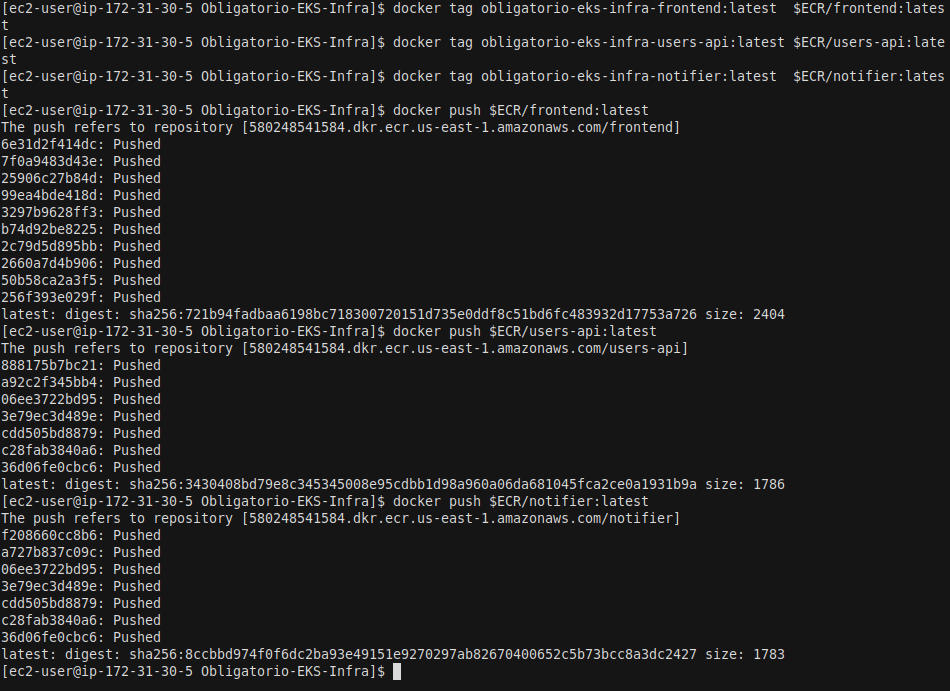
aws ecr create-repository --repository-name notifier --region $REGION



ACCOUNT\_ID=580248541584

ECR="${ACCOUNT\_ID}.dkr.ecr.${REGION}.amazonaws.com"

aws ecr describe-repositories --region $REGION --query "repositories[].repositoryUri" --output table



docker tag obligatorio-eks-infra-frontend:latest $ECR/frontend:latest

docker tag obligatorio-eks-infra-users-api:latest $ECR/users-api:latest

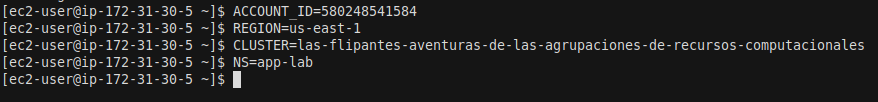
docker tag obligatorio-eks-infra-notifier:latest $ECR/notifier:latest

docker push $ECR/frontend:latest

docker push $ECR/users-api:latest

docker push $ECR/notifier:latest

## Despliegue en Kubernetes (EKS)



CLUSTER=las-flipantes-agrupaciones-de-recursos-computacionales

NS=app-lab

Instalamos eksctl para crear los clústers de EKS



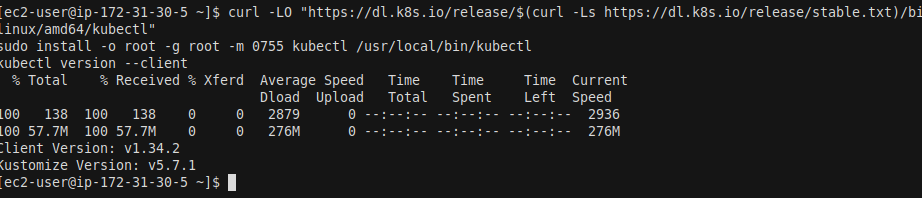
sudo curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" -o /tmp/eksctl.tar.gz

sudo tar -xzf /tmp/eksctl.tar.gz -C /tmp

sudo mv /tmp/eksctl /usr/local/bin

eksctl version

Instalamos kubectl para interactuar con los kubernetes de EKS

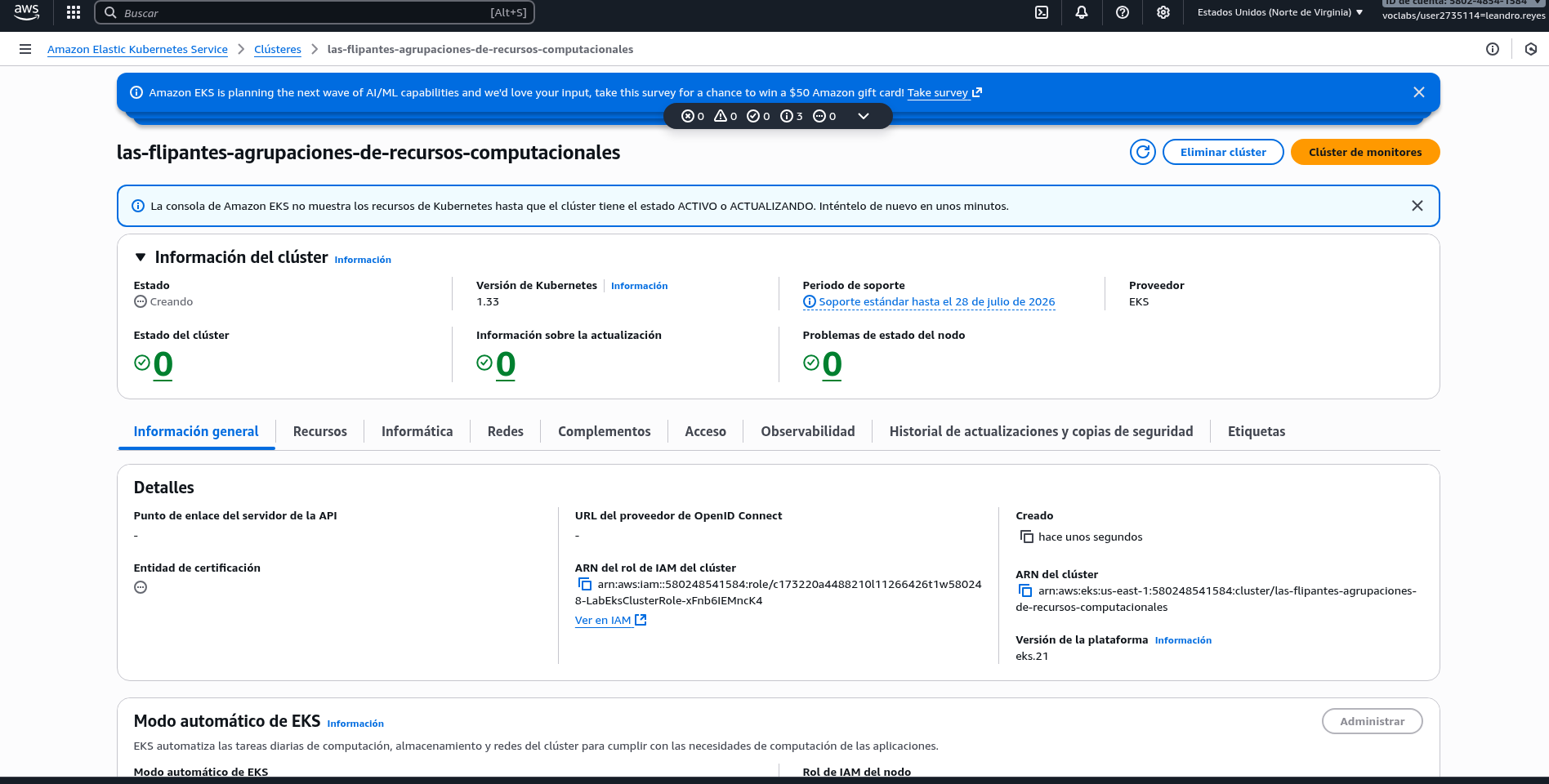
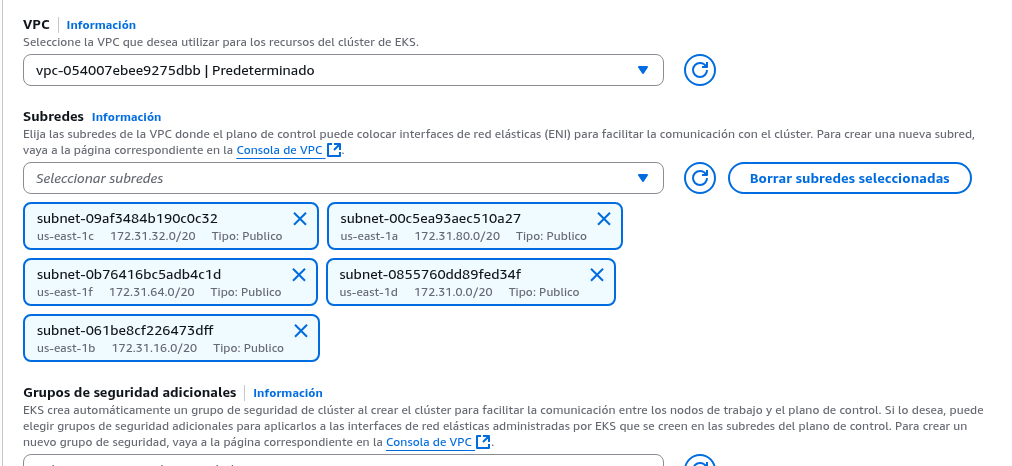
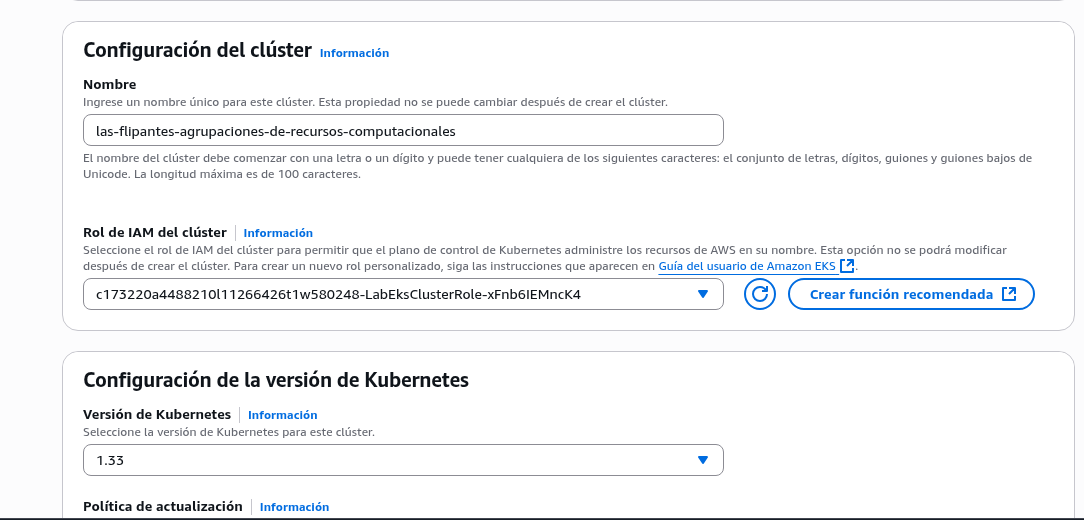


curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

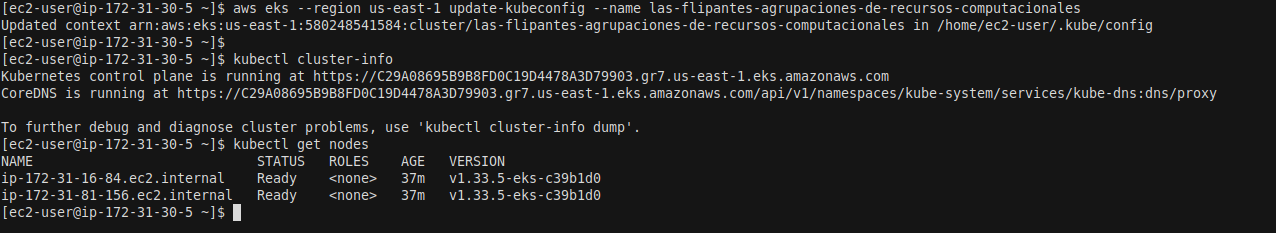
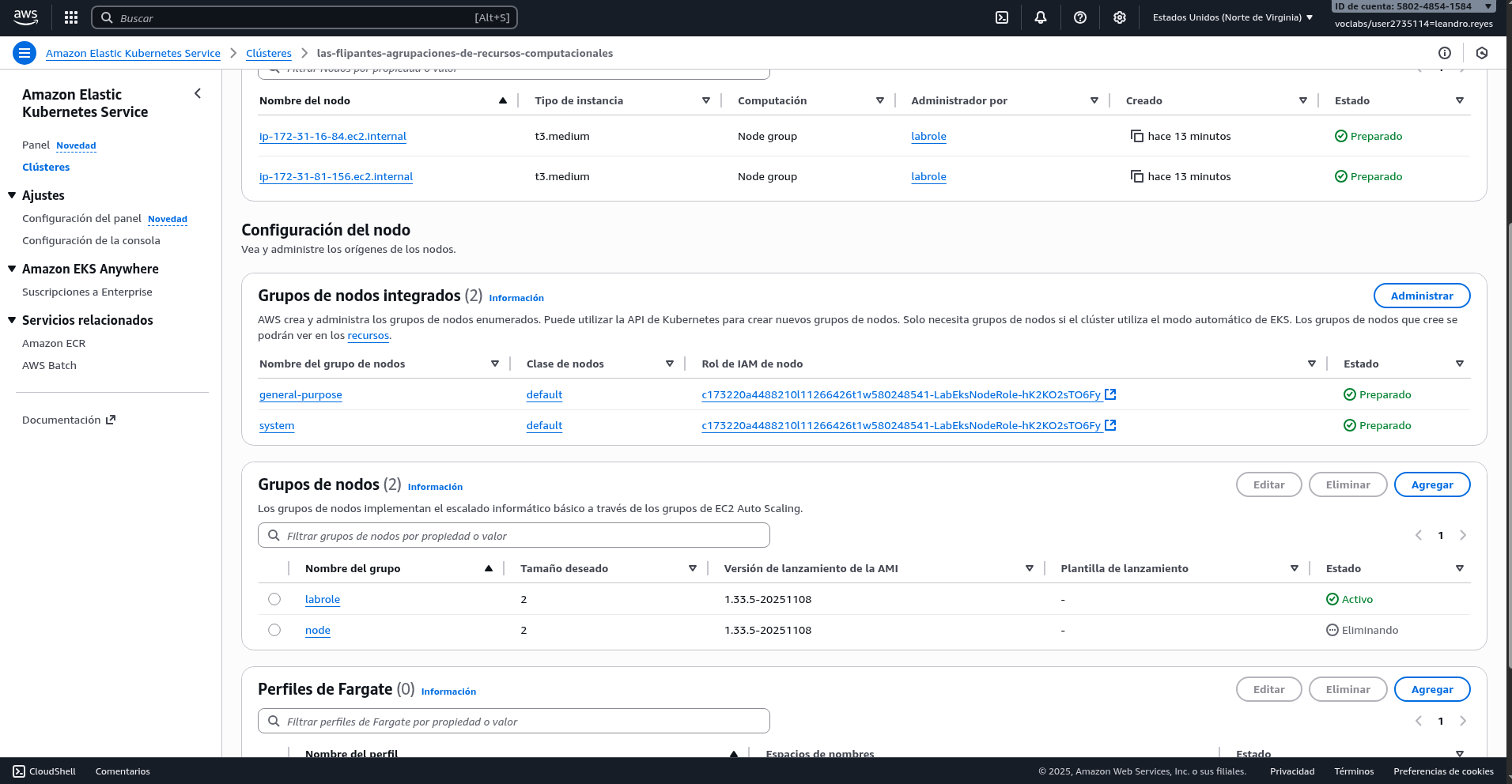
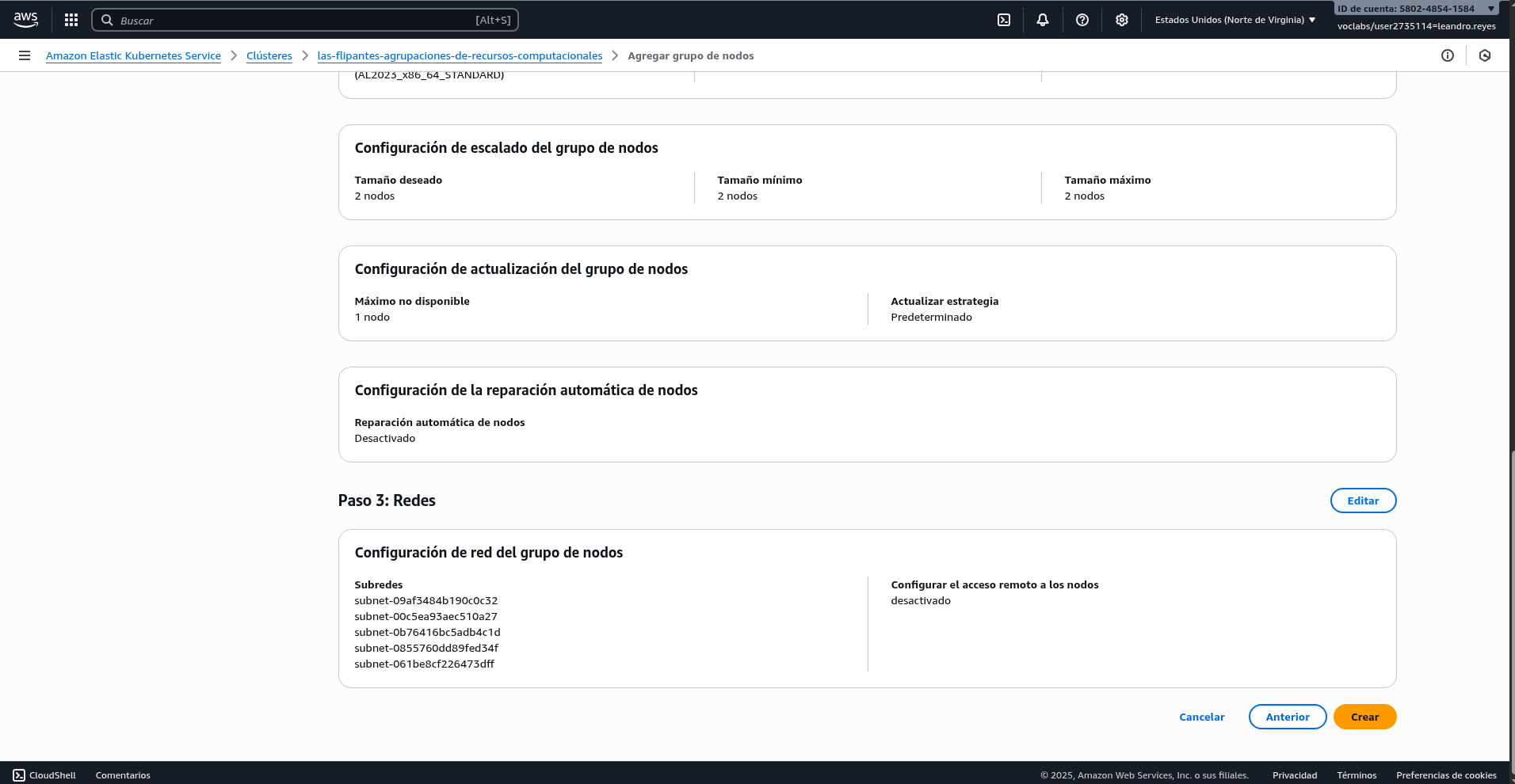
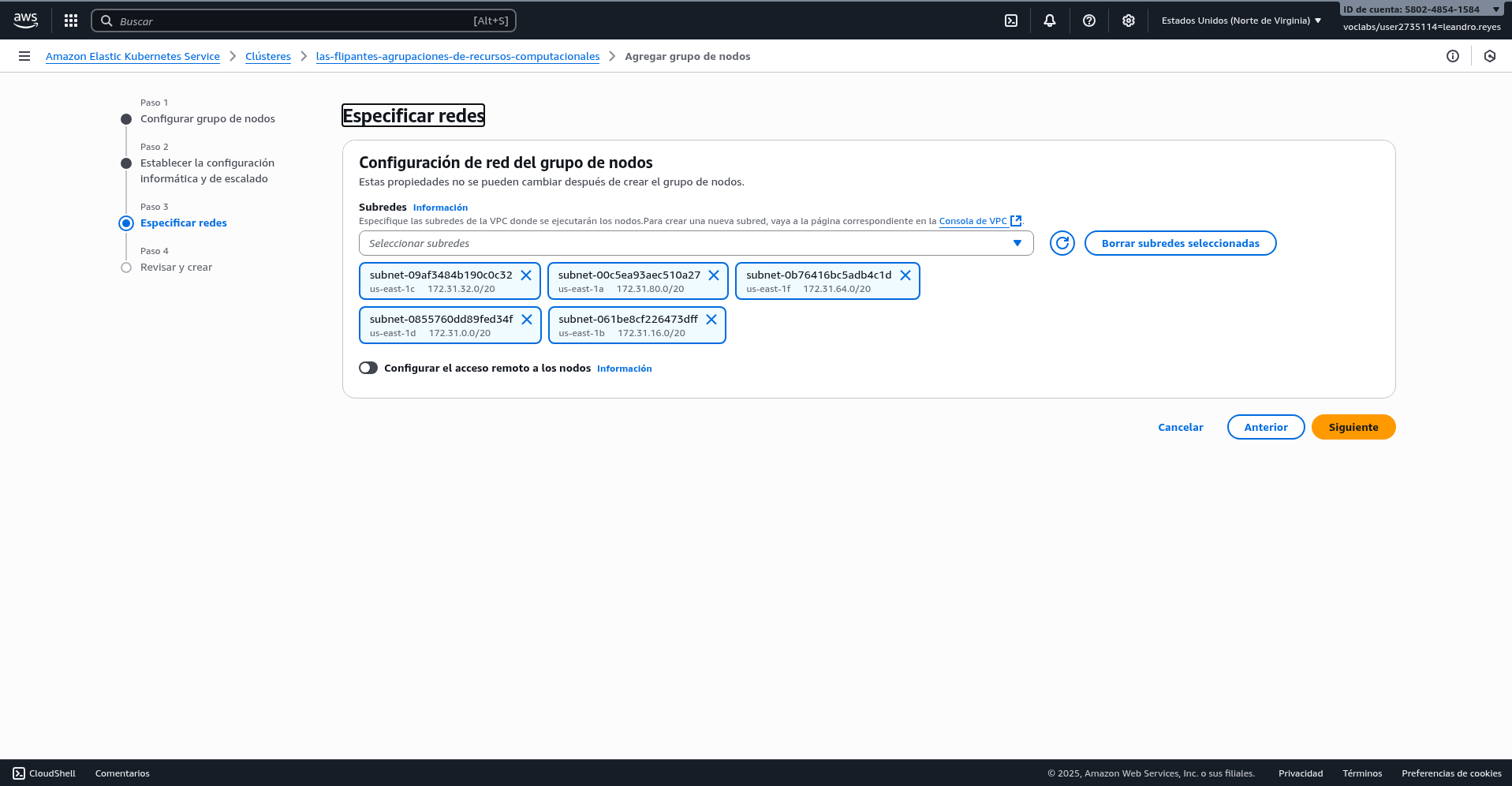
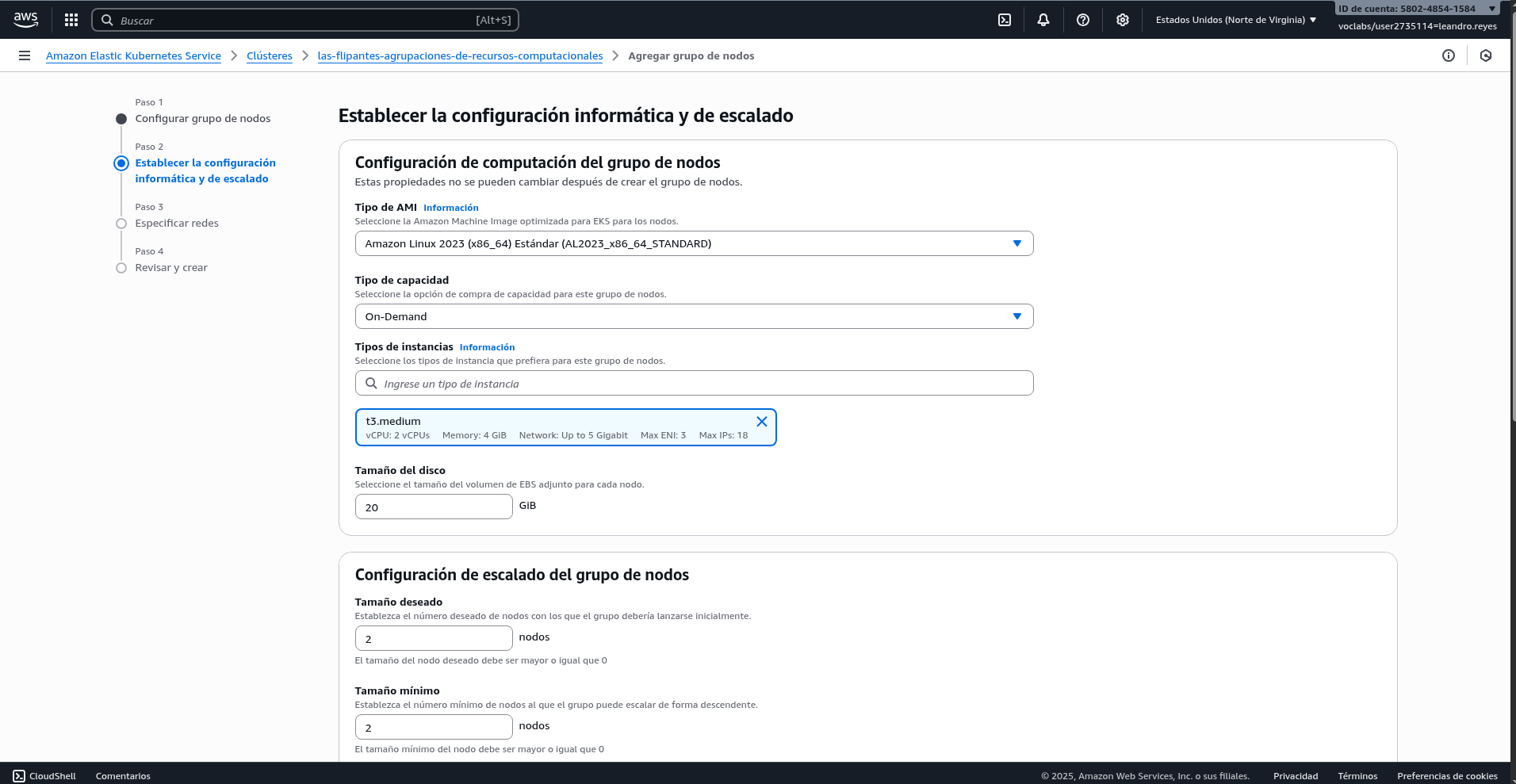
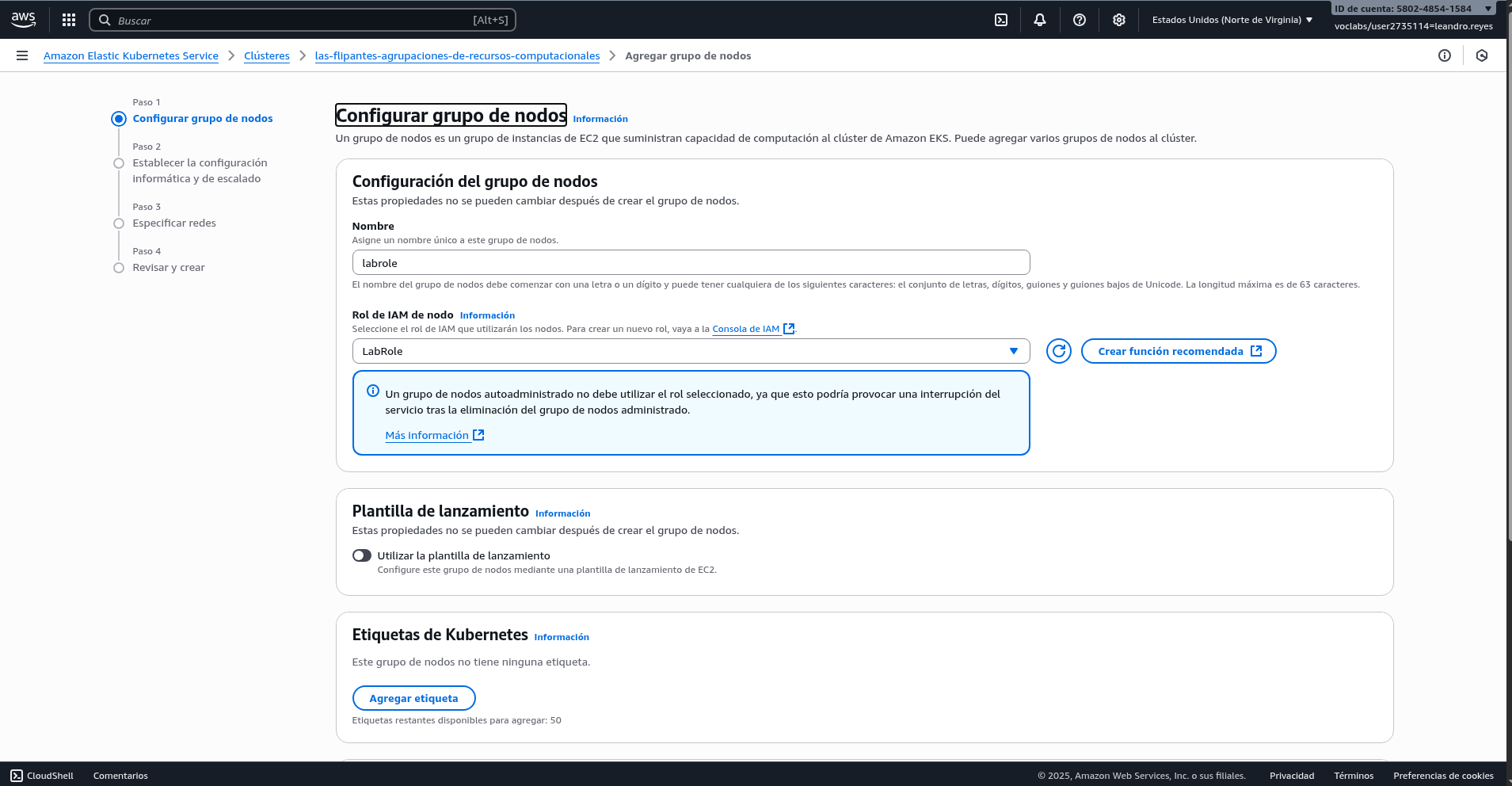
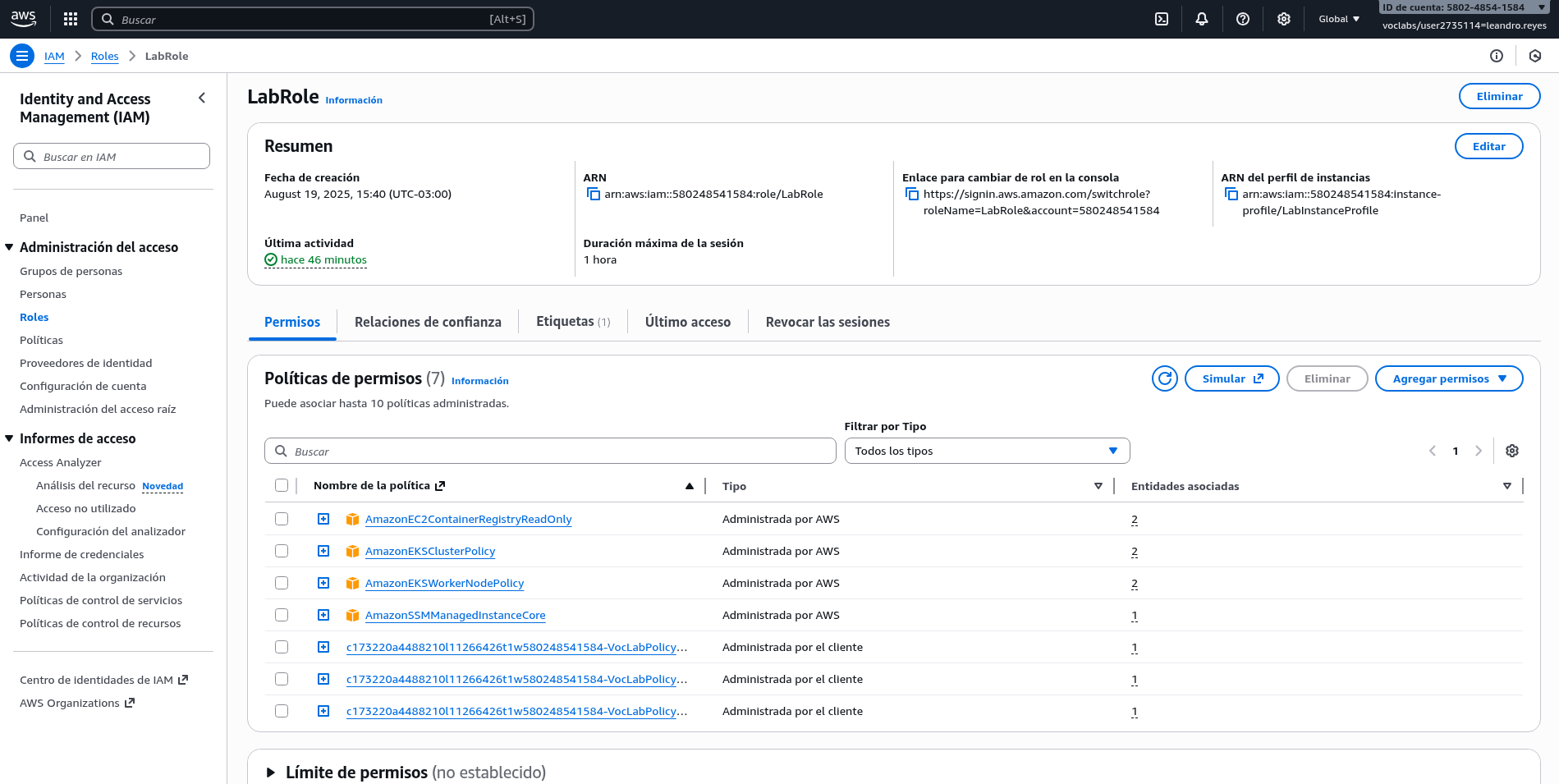
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

kubectl version --client

Crear Cluster



Usando este rol creamos el nodo



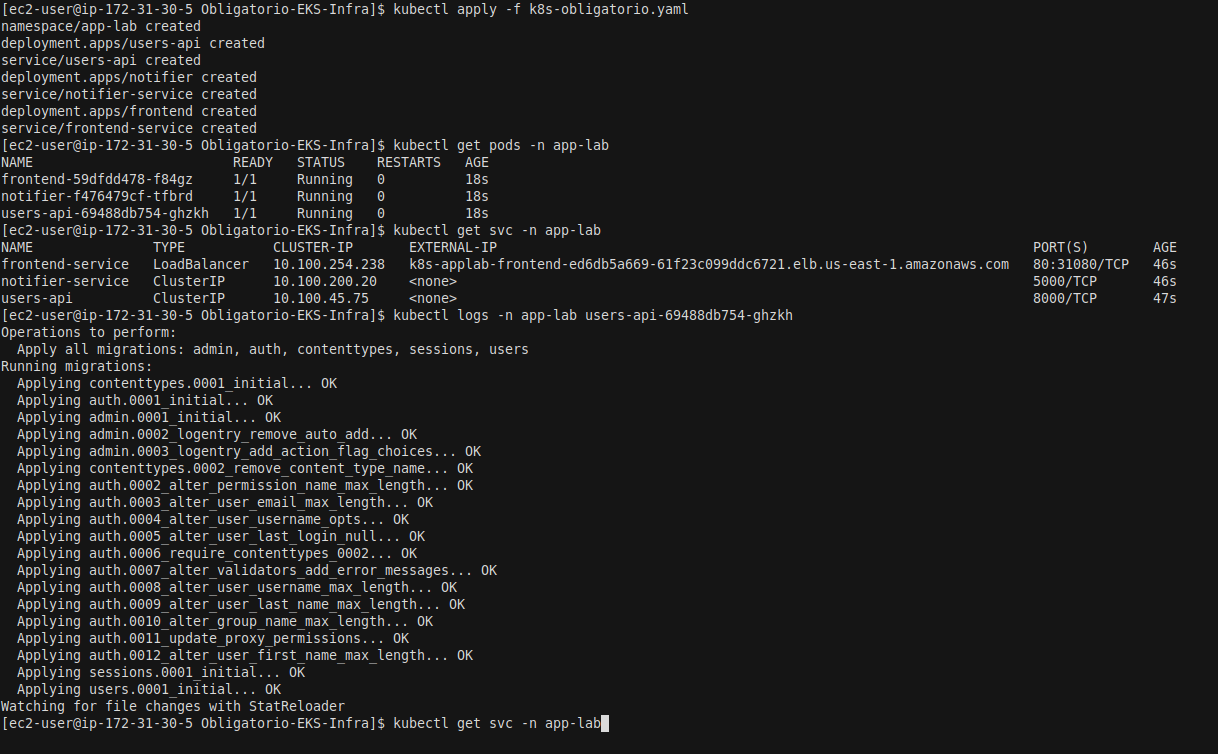
Configuramos el Cluster y verificamos su conectividad

aws eks --region us-east-1 update-kubeconfig --name las-flipantes-agrupaciones-de-recursos-computacionales

kubectl cluster-info

kubectl get nodes

Abrimos la pagina para el uso

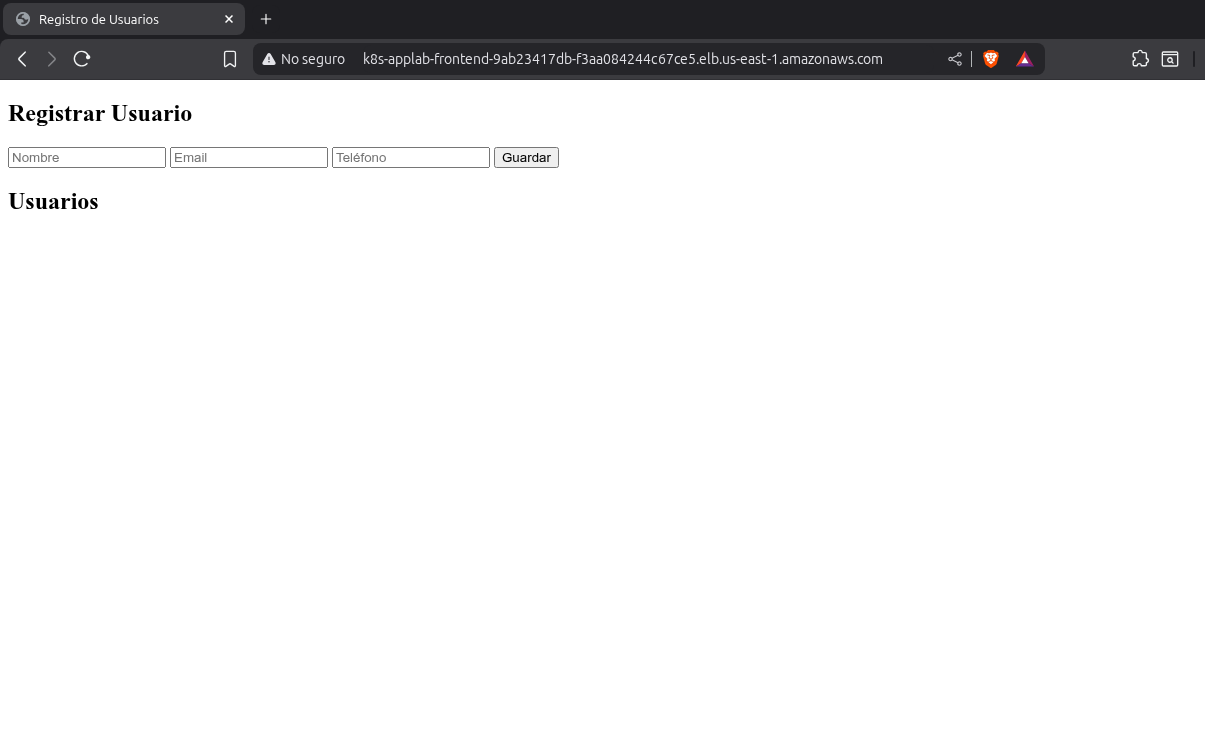


kubectl apply -f k8s-obligatorio.yaml

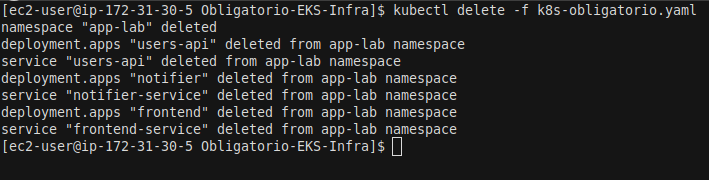
kubectl get pods -n app-lab

kubectl get svc -n app-lab

kubectl logs -n app-lab users-api-69488db754-ghzkh



Bajar la pagina web



kubectl delete -f k8s-obligatorio.yaml

## Arquitectura de Microservicios Implementada

La aplicación fue rediseñada adoptando un modelo de arquitectura basado en microservicios, donde cada componente se ejecuta de manera independiente y se comunica mediante HTTP dentro del clúster de Kubernetes.

Los microservicios implementados fueron:

### 1. Frontend-Service

* Contenedor Nginx que sirve un HTML/JS estático.
* Único componente expuesto públicamente mediante un LoadBalancer.
* Se comunica con el backend enviando solicitudes HTTP a la API REST de users-api.

### 2. Users-API-Service (Backend Django)

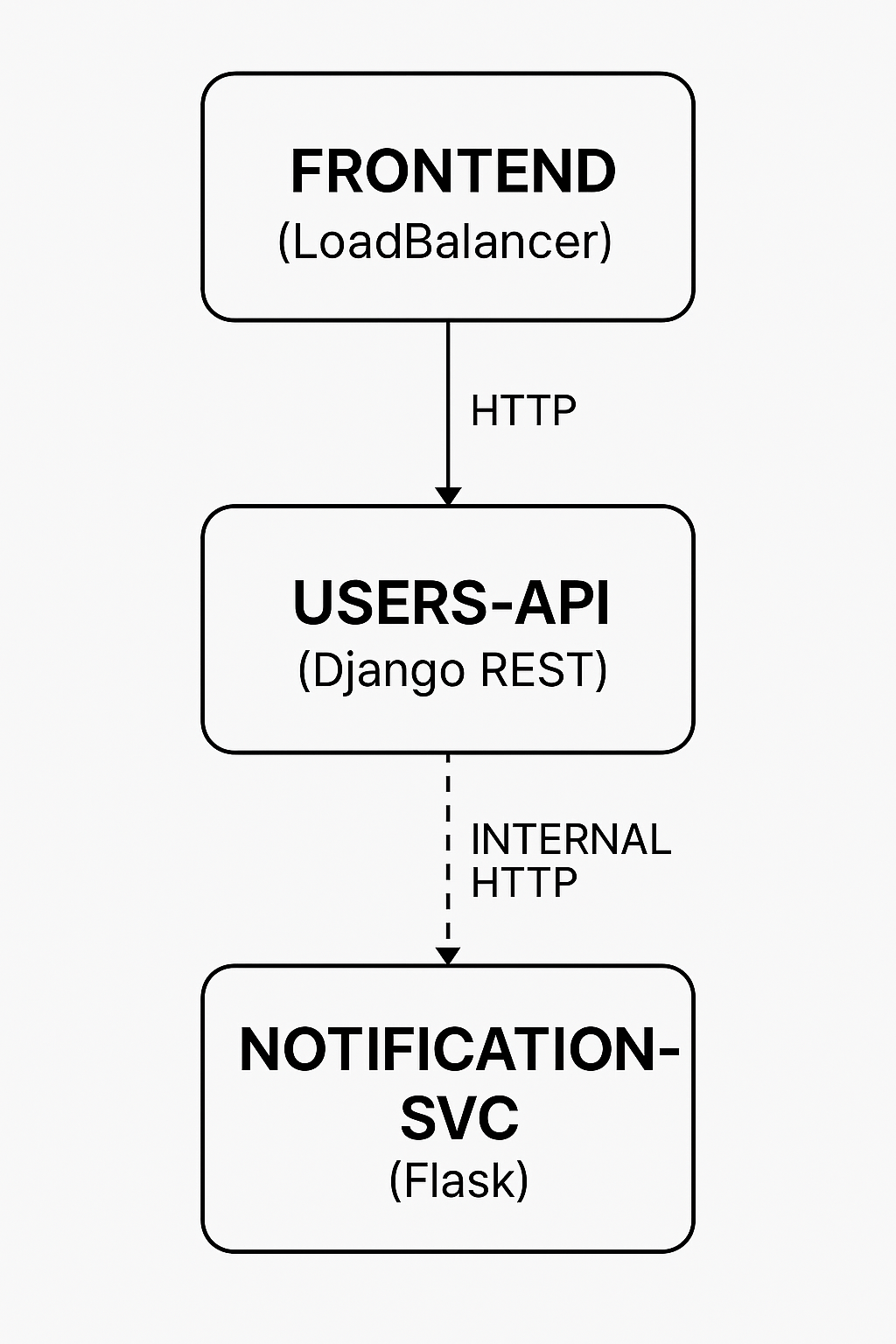
* Microservicio principal encargado de:
  + Crear usuarios
  + Listar usuarios
  + Guardar datos en la base SQLite
  + Enviar eventos de notificación al service “notifier”
* Publicado como ClusterIP, visible solo dentro del clúster.
* Contiene un mecanismo automático para ejecutar migraciones al iniciar el pod.

### 3. Notification-Service

* Desarrollado en Python Flask.
* Expone un endpoint interno /notify.
* Recibe eventos desde users-api cuando se crea un usuario.
* Solo accesible internamente en Kubernetes, nunca público.

### 4. Base de Datos (SQLite dentro del backend)

* Almacena los usuarios creados.
* Para EKS se usó /tmp/db.sqlite3 con migraciones automatizadas.
* No se expone hacia afuera del contenedor.



## Arquitectura de Red – VPC + Subredes + EKS + LoadBalancer

El clúster se desplegó sobre la arquitectura administrada por AWS EKS, que incluye una VPC diseñada específicamente para ejecutar workloads de Kubernetes.

### 1. VPC (Virtual Private Cloud)

EKS crea automáticamente una VPC dedicada para el clúster, incluyendo:

* Tablas de ruteo
* Internet Gateway
* Subredes públicas y privadas
* Seguridad por medio de Security Groups

La VPC permite aislar el clúster y que solo los componentes necesarios queden expuestos al exterior.

### 2. Subredes gestión y nodos

AWS EKS utiliza dos tipos de subredes:

### Subredes privadas

* Aquí se ejecutan los worker nodes del clúster.
* Aquí corren tus pods:
  + frontend
  + users-api
  + notifier
* Los pods no son accesibles directamente desde Internet.

### Subredes públicas

* Solo se utilizan para:
  + Balanceadores de carga (LoadBalancers)
  + Componentes de administración de AWS
  + Interfaz de acceso al frontend

Esto garantiza que **solo el frontend** sea accesible desde fuera de la VPC.

### 3. Worker Nodes de EKS

Son las instancias EC2 administradas por el Node Group.

En nuestro caso el clúster quedó compuesto por:

* 2 worker nodes
* Basados en AWS Linux
* Cada uno ejecutando pods según su scheduler

Los nodos se comunican internamente mediante la red VPC sin exposición directa al exterior.

## 5. Comunicación dentro del clúster

Los servicios se comunican usando la red interna de Kubernetes:

* frontend → users-api via ClusterIP
* users-api → notifier via ClusterIP
* users-api accede a SQLite local dentro del contenedor
* Comunicación manejada por kube-proxy y el CNI de AWS

### 6. Seguridad (Security Groups y Roles)

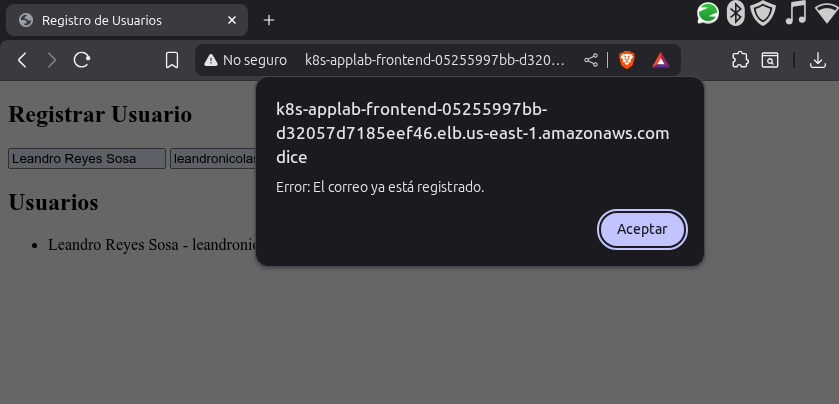
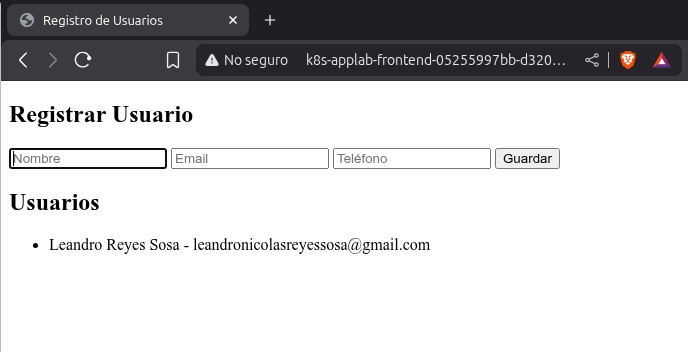
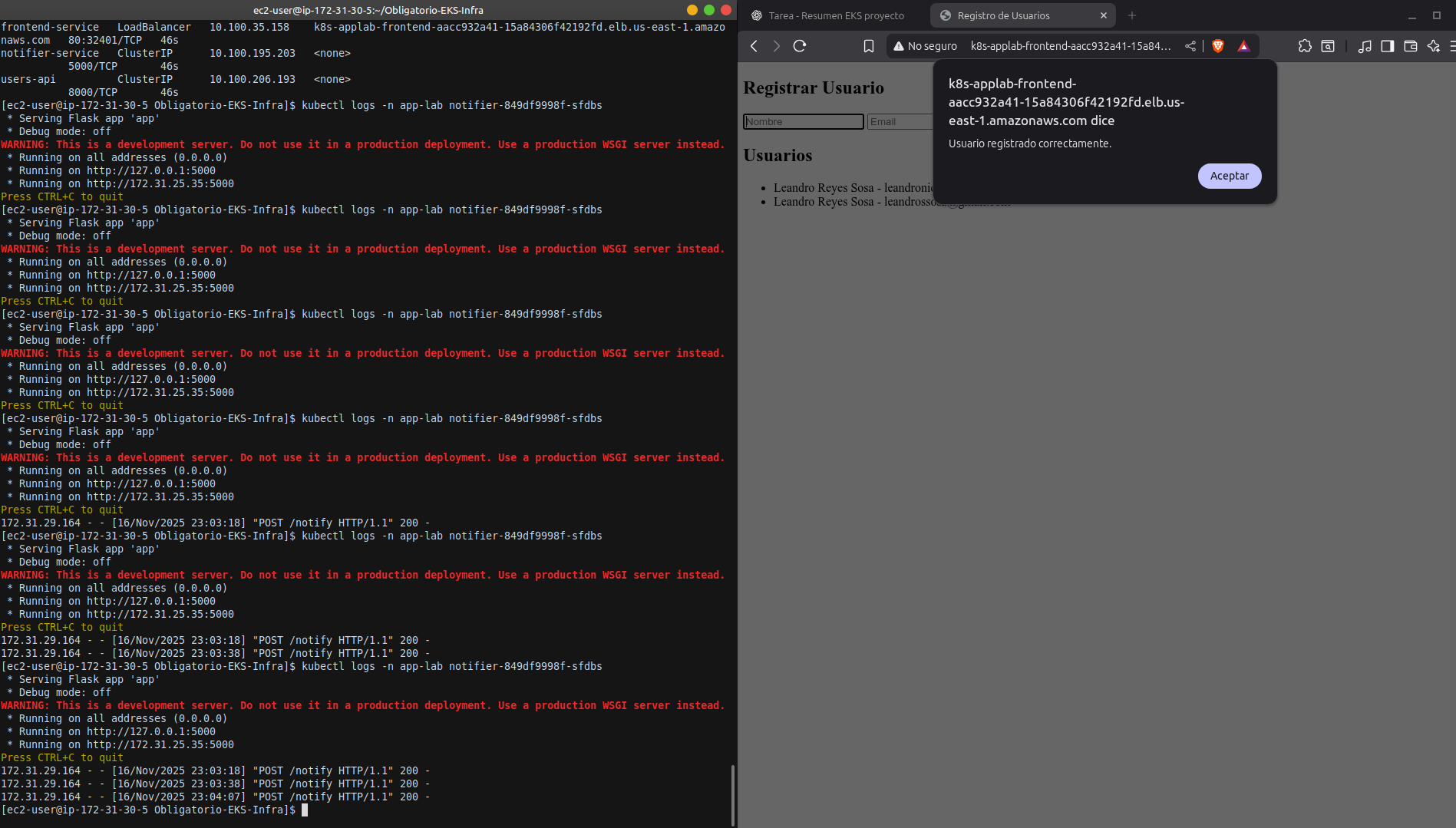
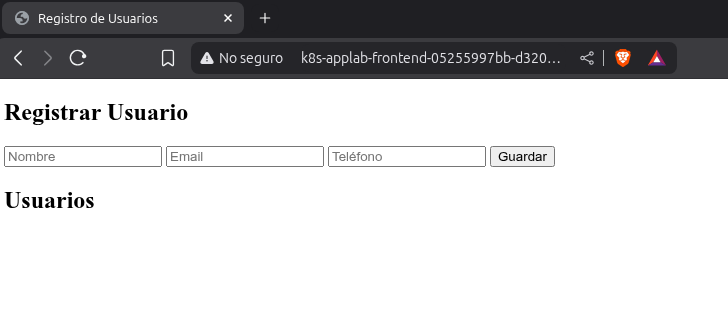
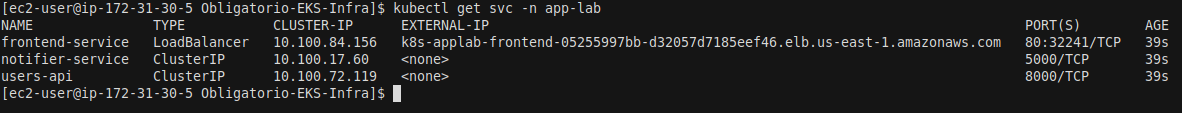
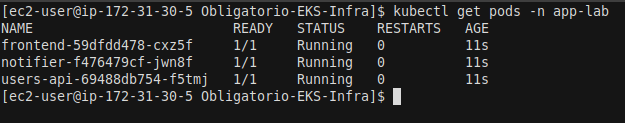
* Solo el LoadBalancer permite ingreso desde Internet (puerto 80).
* Los nodos permiten únicamente el tráfico interno del clúster.
* IAM Roles garantizan que los pods del backend puedan leer imágenes de ECR.
* Nada queda expuesto fuera de los servicios necesarios.

### 7. Flujo general del tráfico

1. Usuario abre el dominio/ELB del frontend
2. El HTML llama a la API interna (/users/)
3. EKS enruta la solicitud al servicio users-api
4. El backend guarda en SQLite y luego POST → notifier
5. Notifier imprime o procesa el evento
6. Respuesta vuelve a frontend
7. Frontend actualiza la tabla de usuarios



## Pruebas de funcionamiento en EKS



## SSDLC – Seguridad aplicada al proyecto

Durante el desarrollo y despliegue de la aplicación se aplicaron prácticas básicas del ciclo de vida seguro del software (SSDLC). Si bien el objetivo principal del laboratorio es la construcción y despliegue de microservicios en EKS, se incorporaron las siguientes medidas de seguridad:

### 1. Validación de datos en la API

El backend en Django valida que los campos de entrada (nombre, email y teléfono) sean correctos antes de almacenarlos.  
 Se evita inyección de datos y entradas mal formadas.

### 2. Separación por microservicios

La separación en:

* frontend
* users-api
* notifier

reduce el impacto de un problema en un servicio, evitando que afecte a todo el sistema.

### 3. Servicios internos no expuestos

Los servicios users-api y notifier están configurados como **ClusterIP**, evitando accesos externos no autorizados.

Solo el frontend está expuesto públicamente mediante un LoadBalancer.

### **4. Gestión segura de imágenes**

Las imágenes se almacenan en **Amazon ECR**, evitando repositorios inseguros.  
 Además, el uso de tags (v1, v2, etc.) permite controlar versiones y evitar imágenes corruptas.

### **5. Análisis básico de vulnerabilidades**

Se ejecutaron pruebas con:

* pip install --upgrade para mantener dependencias actualizadas
* revisión manual de dependencias en Python
* buenas prácticas en Docker (imagen base ligera, no exponer puertos innecesarios)

### **6. Restricciones de acceso a Kubernetes**

El acceso al clúster se realiza únicamente desde la instancia EC2 autorizada mediante:

aws eks update-kubeconfig ...

Con esto se evita acceso externo al cluster.

## Conclusión

El proyecto permitió integrar múltiples tecnologías de forma práctica: desarrollo de microservicios, contenedorización con Docker, publicación de imágenes en AWS ECR y despliegue completo en un clúster EKS.

Se logró implementar una arquitectura funcional compuesta por frontend, backend y un servicio de notificaciones, comprobando la comunicación interna entre los servicios y el acceso externo mediante un LoadBalancer.

Durante el desarrollo surgieron problemas reales, como errores de configuración de imágenes, fallas en la base de datos SQLite y migraciones pendientes, los cuales fueron solucionados aplicando buenas prácticas y herramientas propias de Kubernetes.

Finalmente, el sistema quedó funcionando de extremo a extremo dentro de AWS, cumpliendo con los requerimientos del práctico y reflejando un entorno real de despliegue en la nube.

## Repositorios de github

Backend: <https://github.com/chizzyjpg/backend.git>

Frontend:https://github.com/chizzyjpg/Frontend.git

Notifier: https://github.com/chizzyjpg/Notifier.git

Laboratorio completo: <https://github.com/chizzyjpg/Obligatorio-EKS-Infra.git>  
  
Video de youtube de defenza  
https://youtu.be/W5yQqCDI2b8