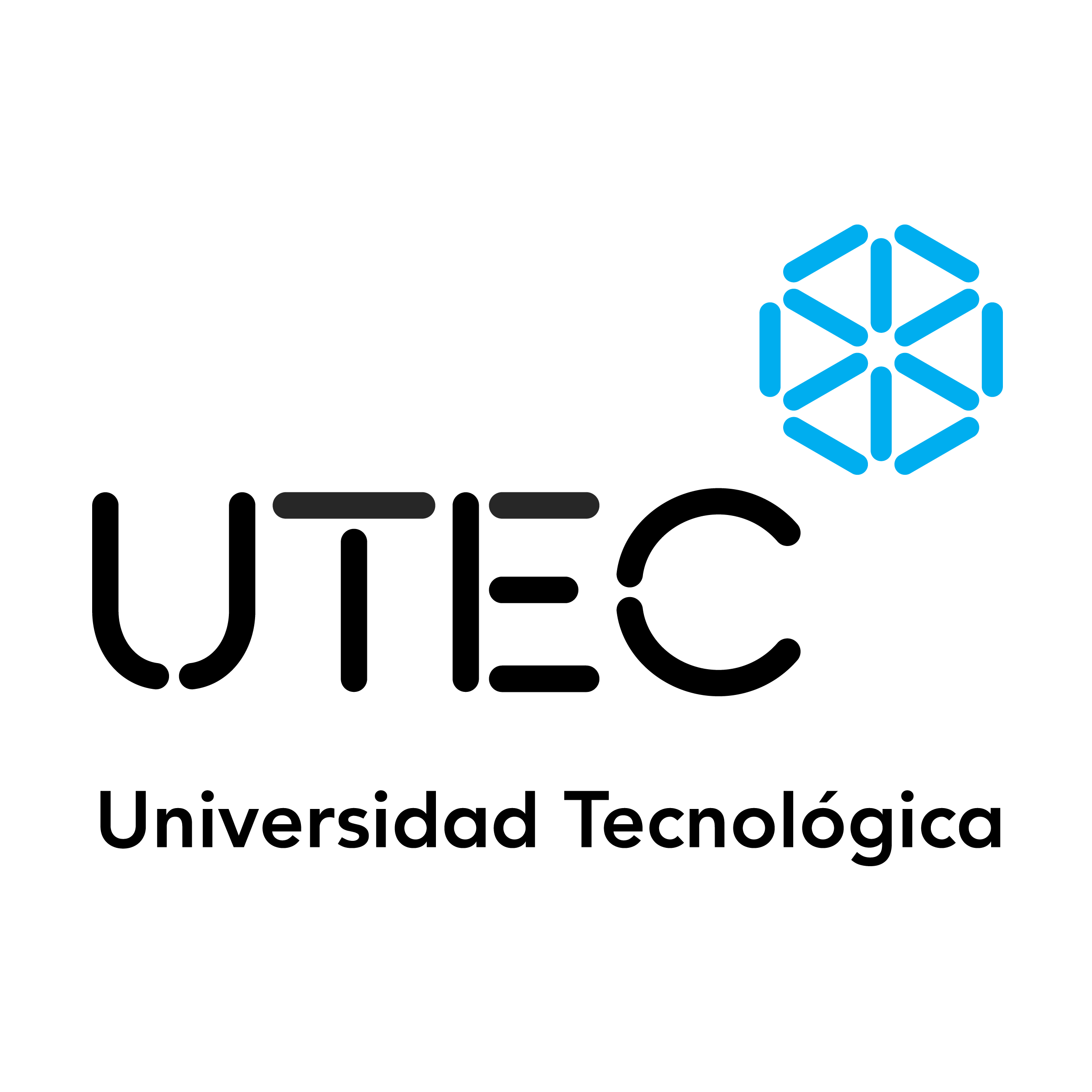
# Laboratorio AWS EKS



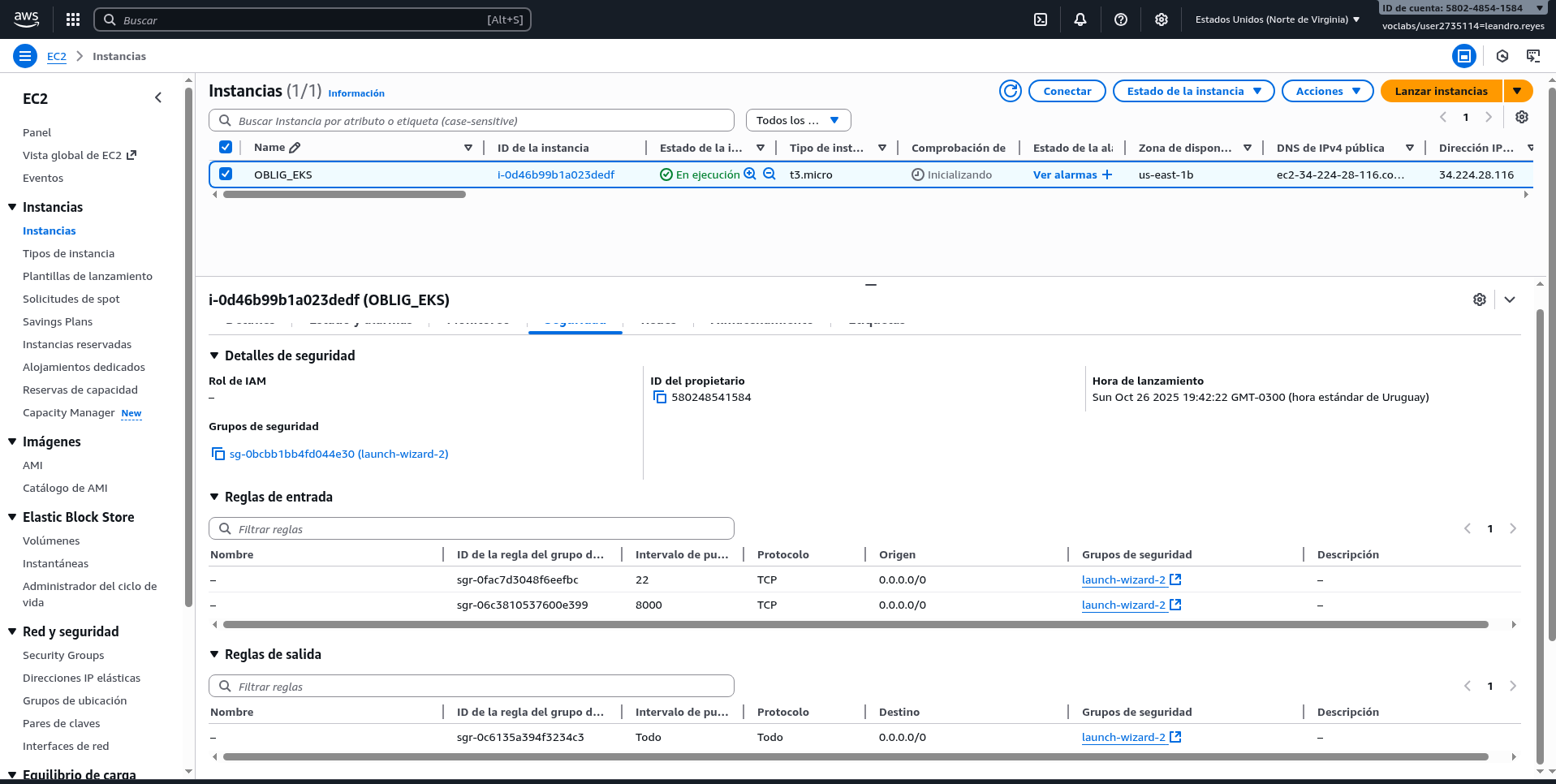
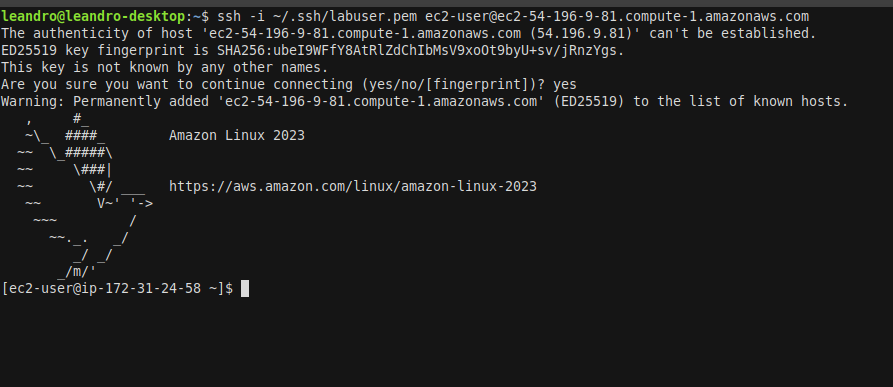
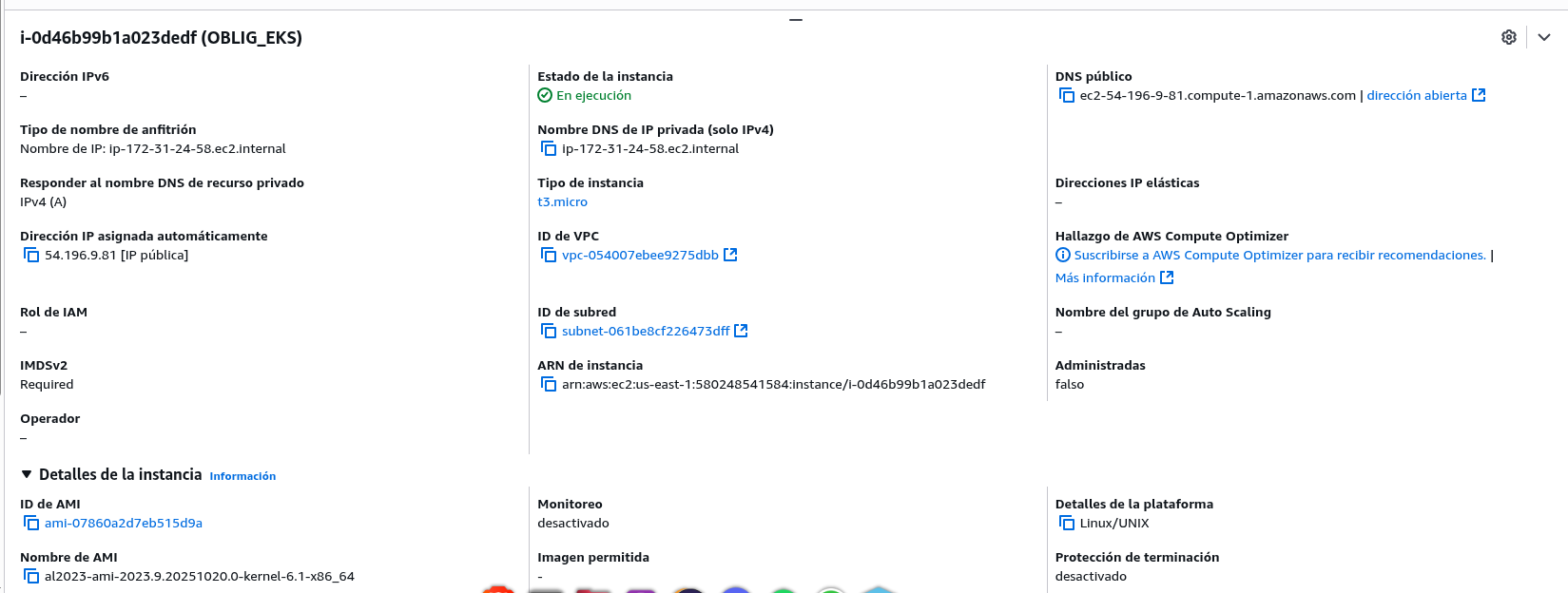
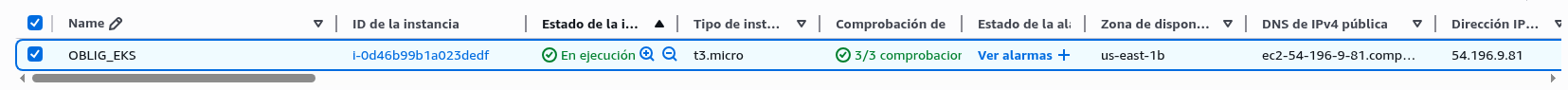
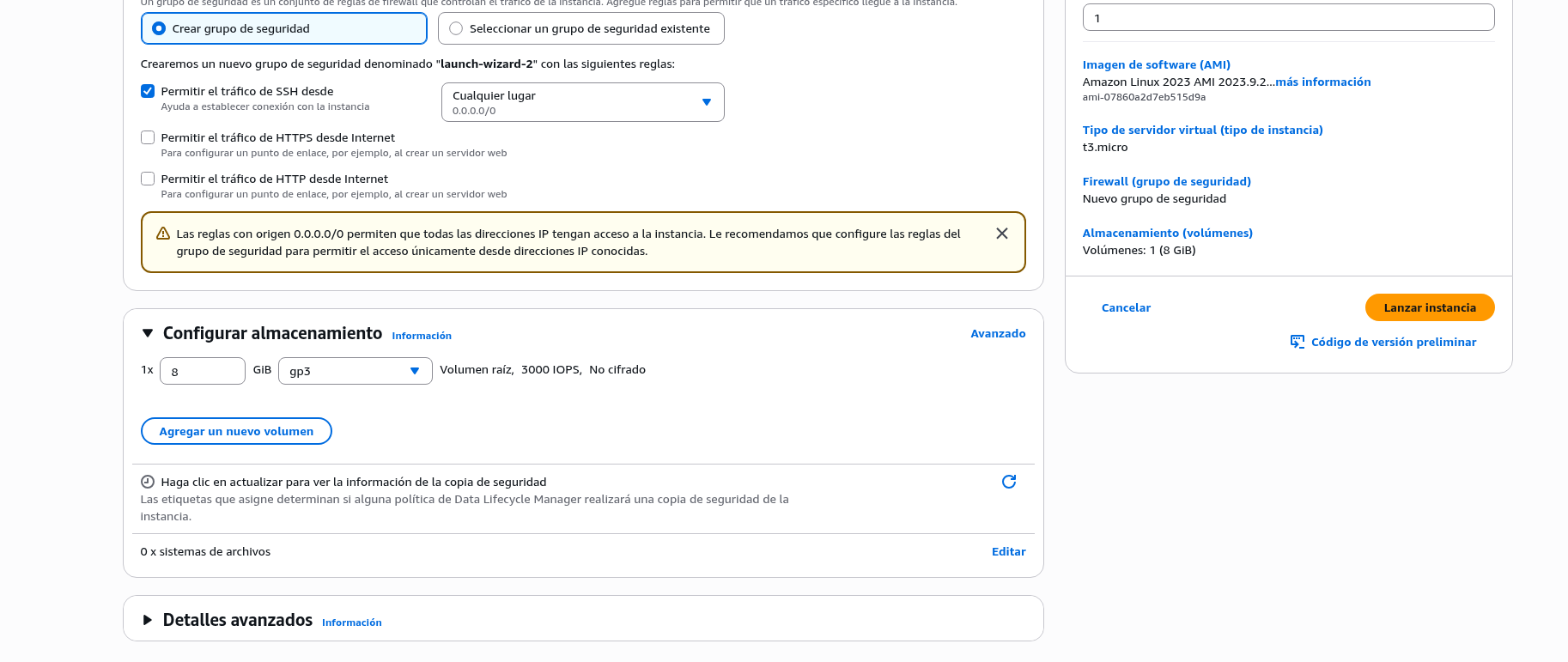
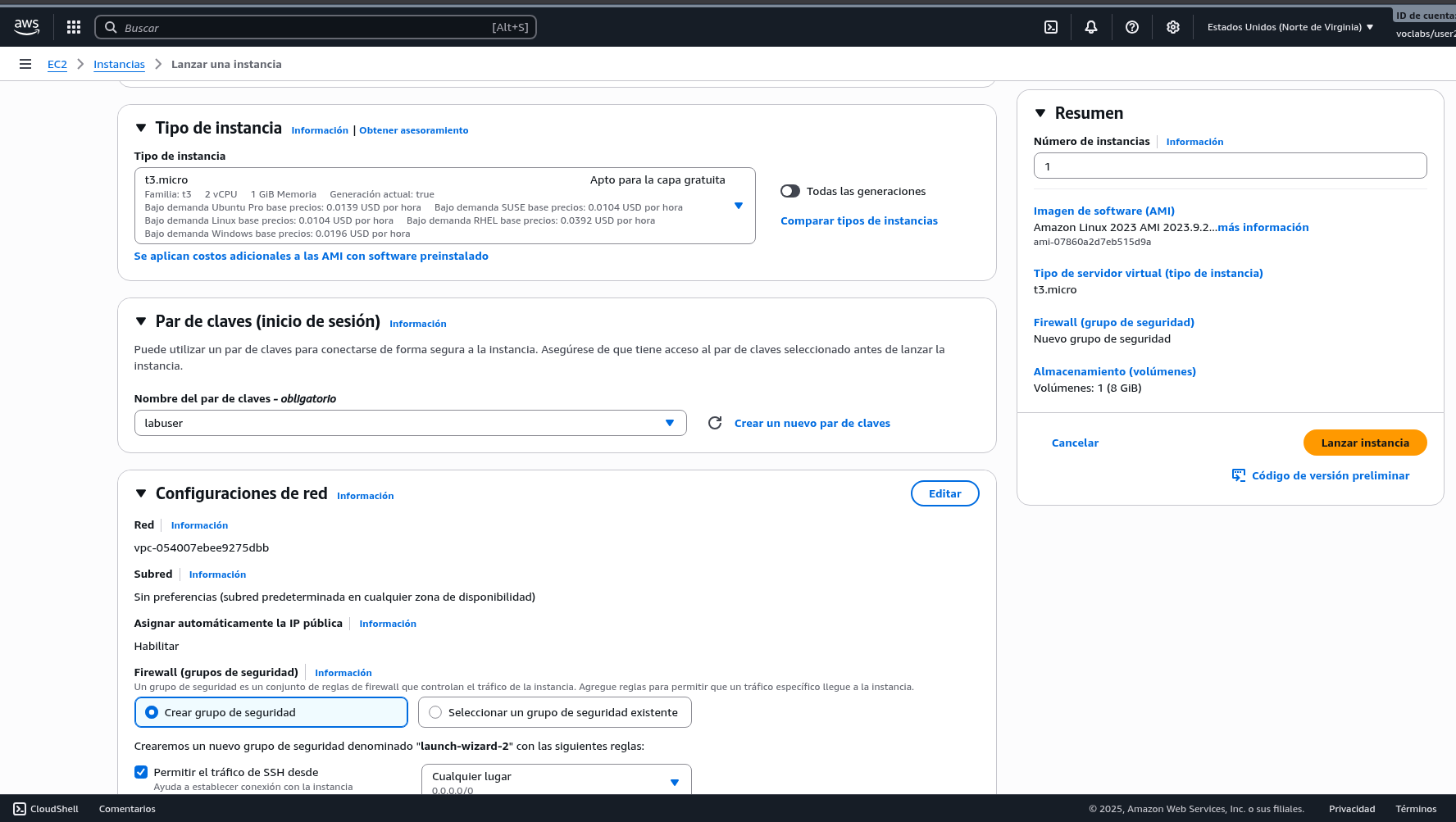
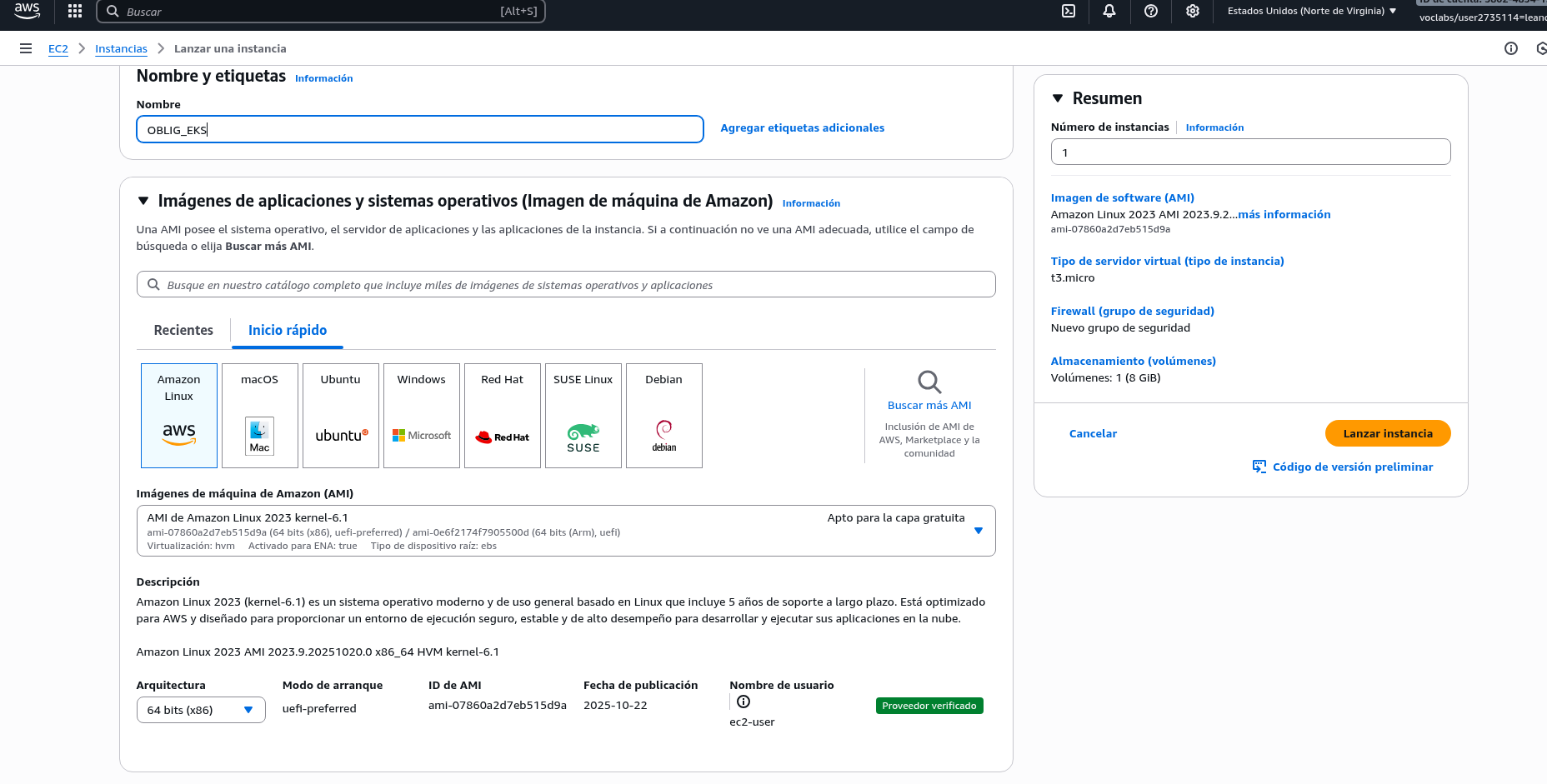
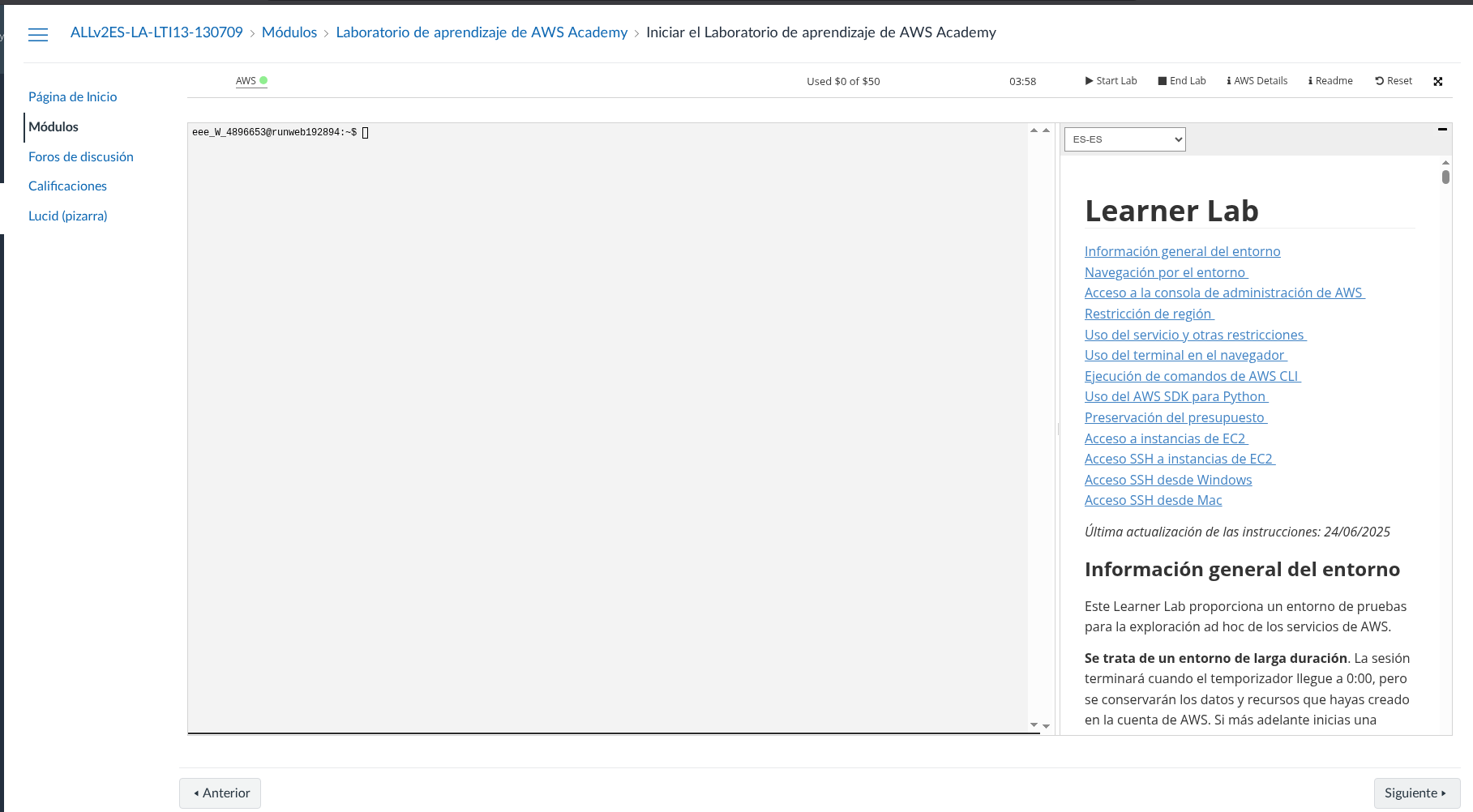
**Administración de Infraestructura**



Leandro Reyes  
20/11/2025

## Preparación del Entorno AWS

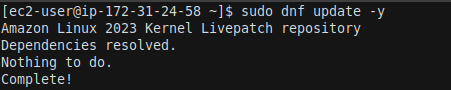
Creación de la instancia EC2, descarga de la clave .pem, configuración del Grupo de Seguridad y conexión SSH.



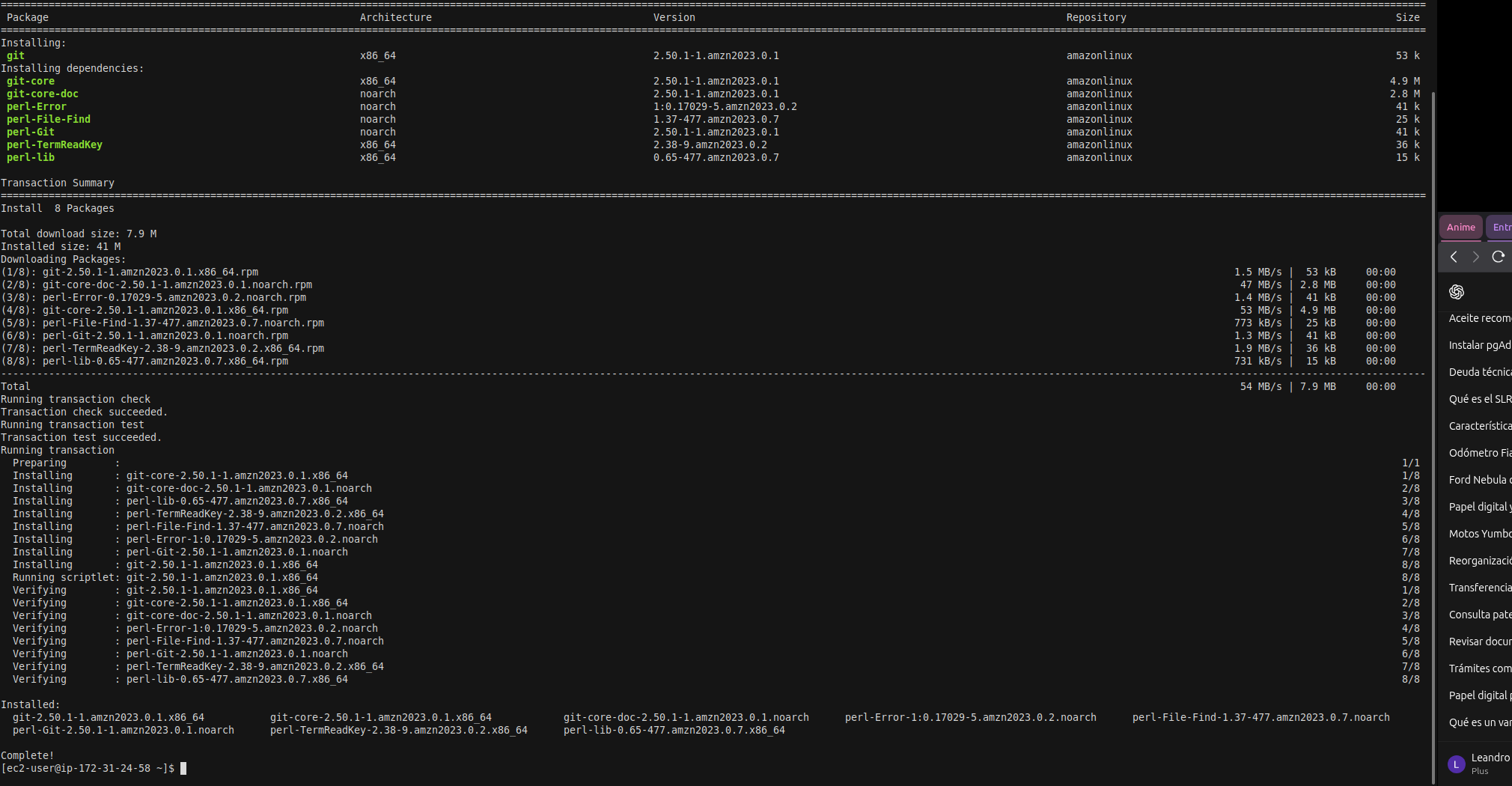
## Configuración del servidor

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

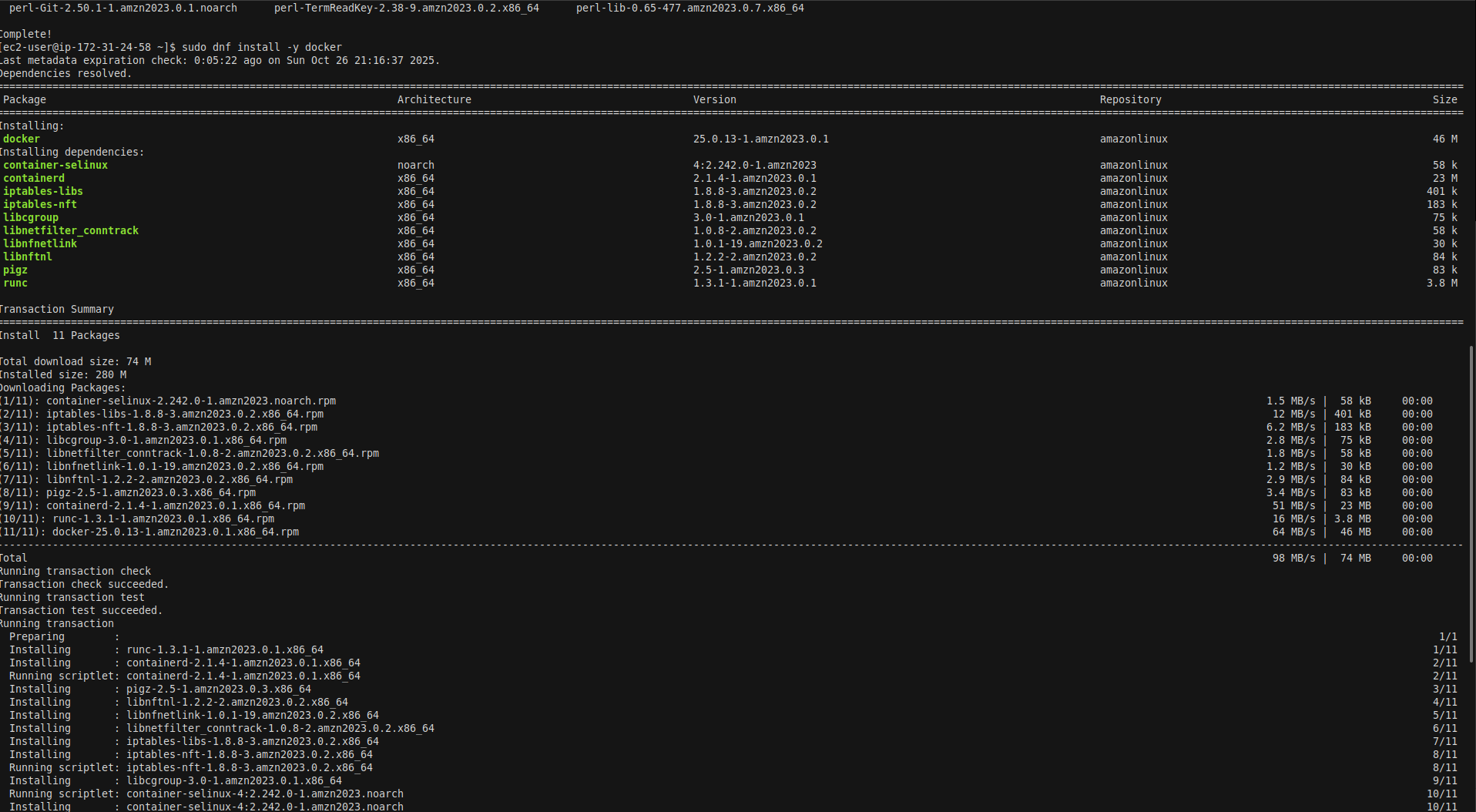
sudo dnf update –y



sudo dnf install –y git

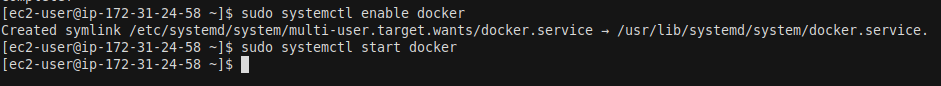


sudo dnf install -y docker

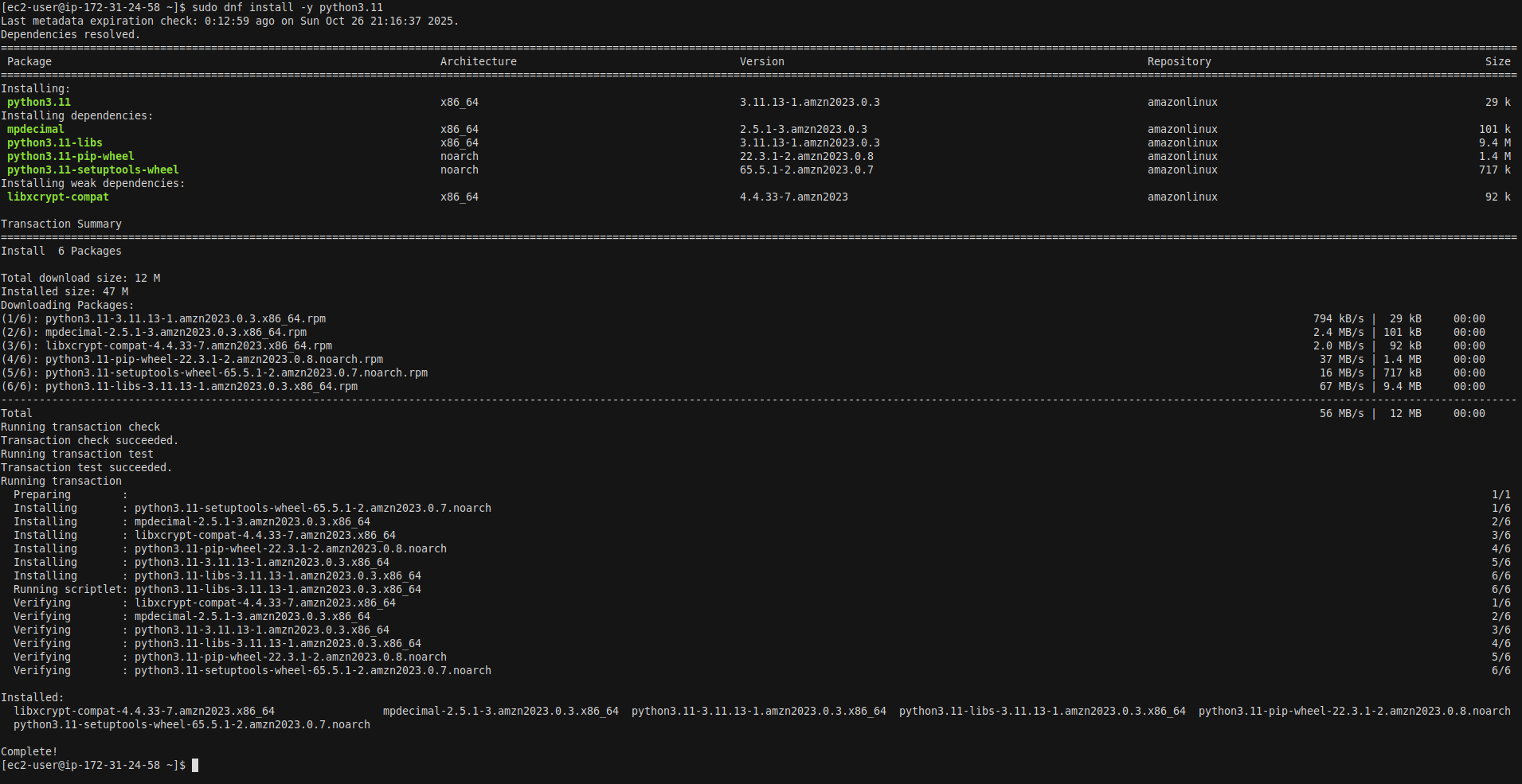


sudo systemctl enable docker

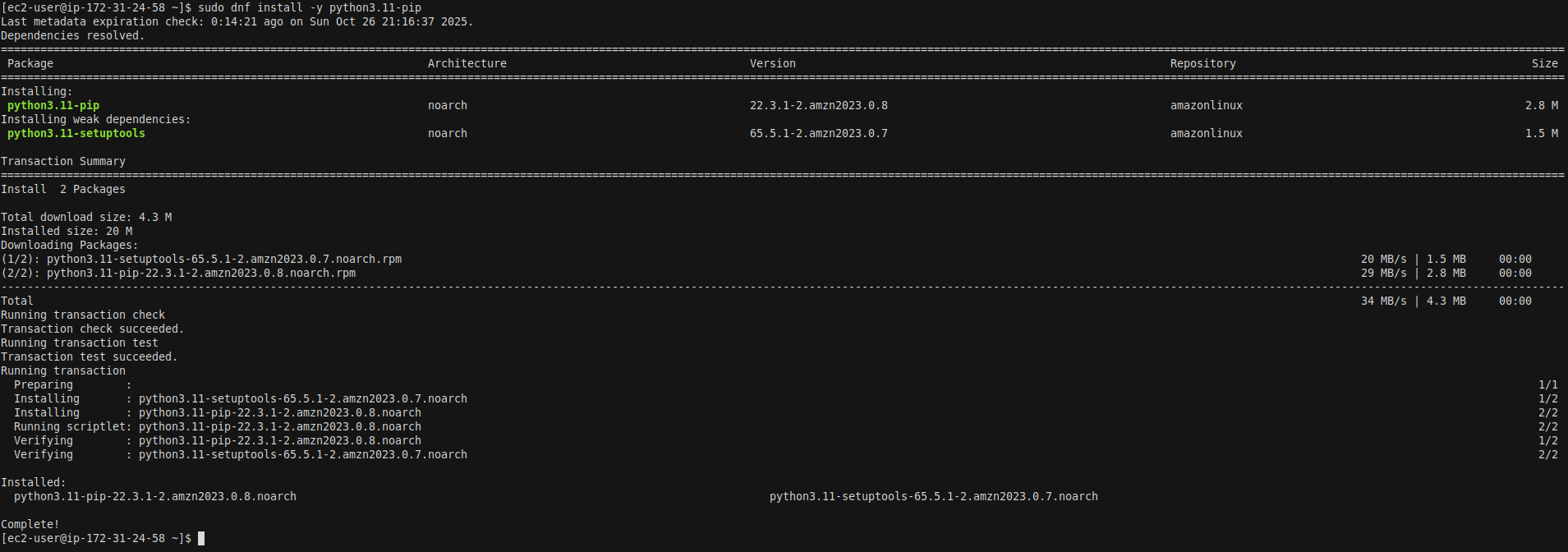
sudo systemctl start docker



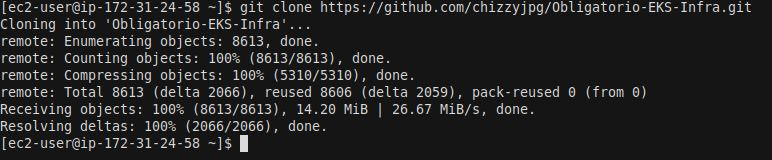
sudo dnf install -y python3.11



sudo dnf install -y python3.11-pip



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

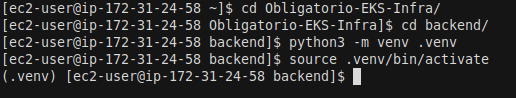


## cd Obligatorio-EKS-Infra

cd backend

python3 -m venv .venv

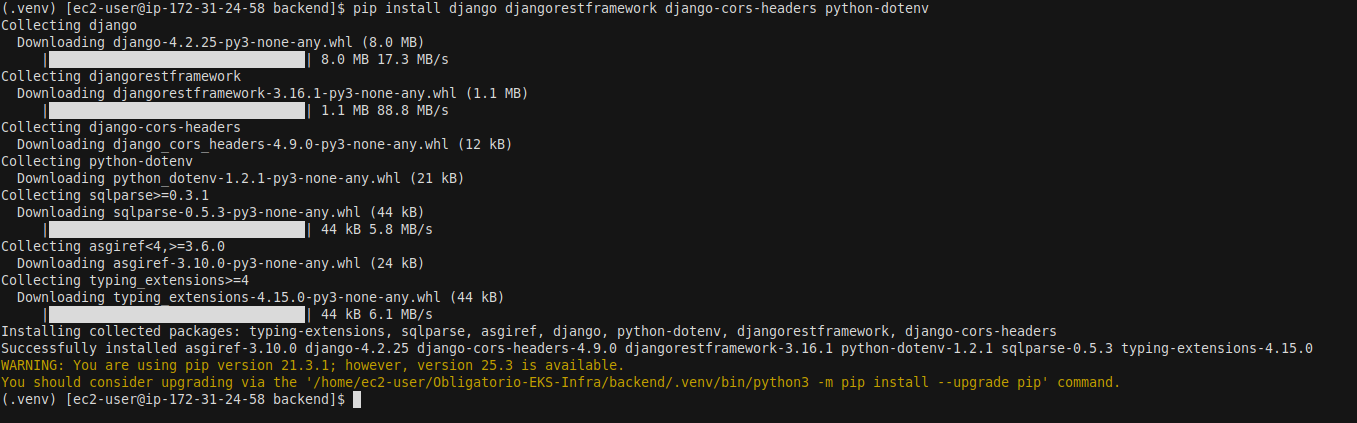
source .venv/bin/activate



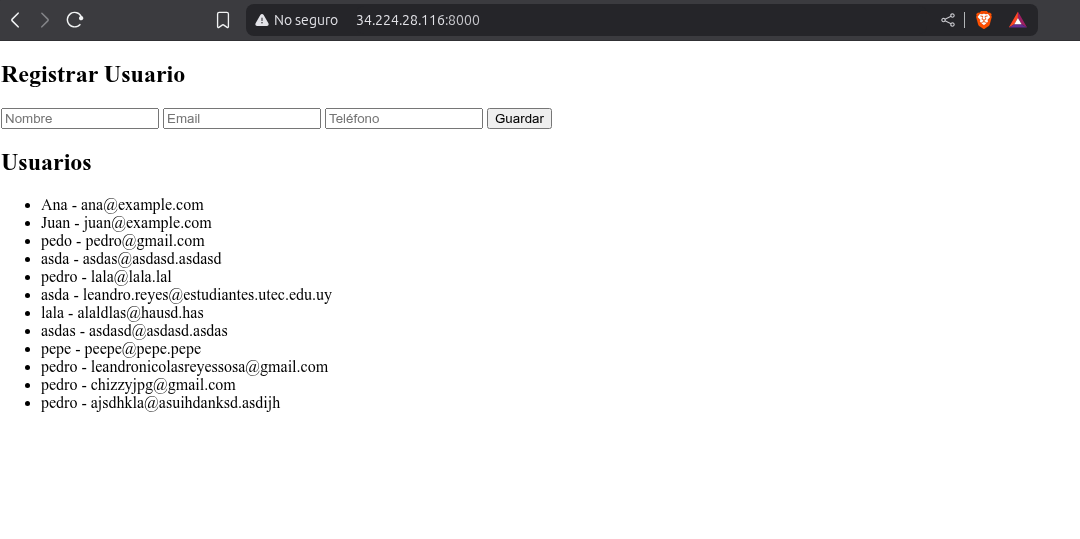
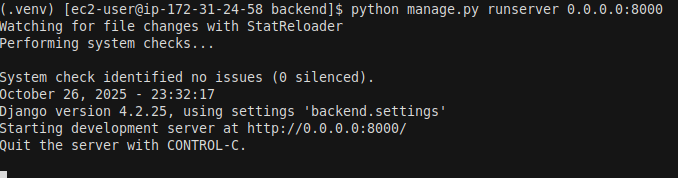
## Despliegue de la Aplicación Monolítica

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



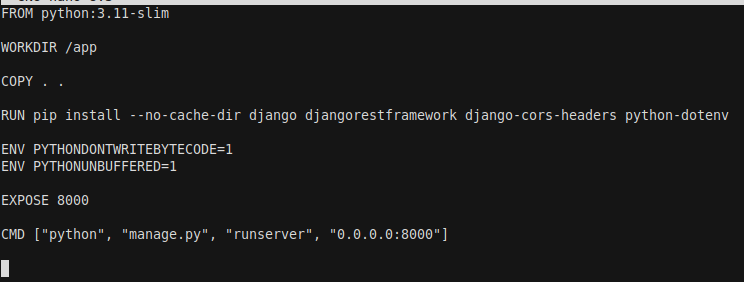
## Conversión a Microservicios

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:

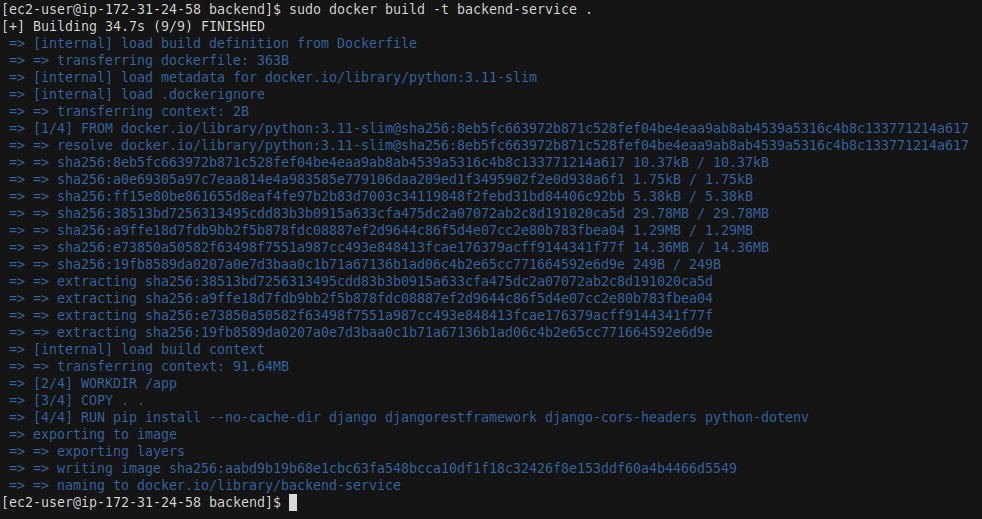
Con el comando groups verificas si el usuario pertenece al grupo de docker, en el caso de que no de una salida ejemplo



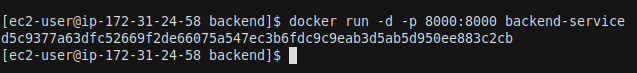
Usando el comando

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

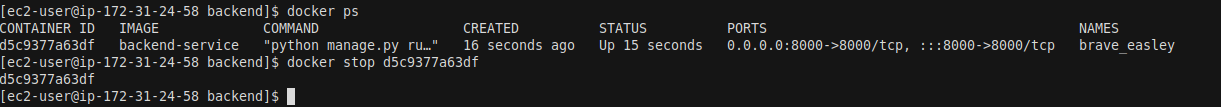


docker run -d -p 8000:8000 backend-service



docker ps

## docker stop d5c9377a63df

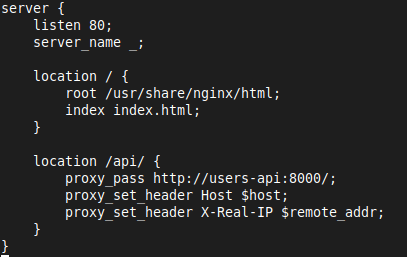


docker rm d5c9377a63df # borra la imagen del docker

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

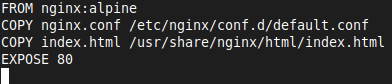
proxy\_set\_header Host $host; # mantiene la ip original

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

}

}

nano Dokerfile



FROM nginx:alpine # version liviana de nginx

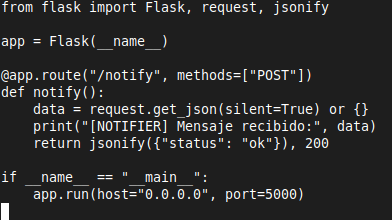
COPY nginx.conf /etc/nginx/conf.d/default.conf #configura el servidor

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

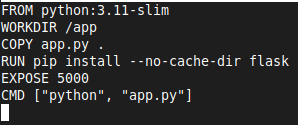
print("[NOTIFIER] Mensaje recibido:", data) # imprime en consola una notificación

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:

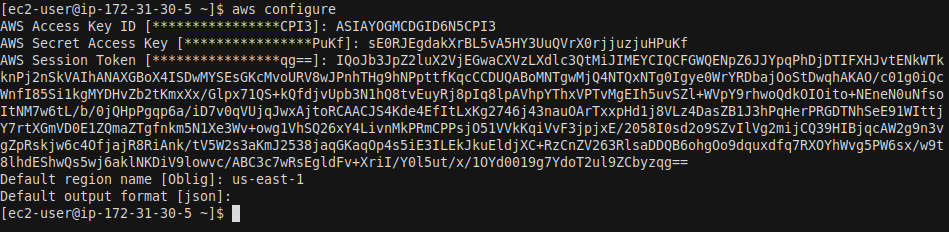
- "80:80"

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

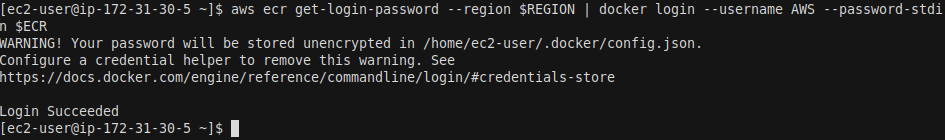
Token: 



ACCOUNT\_ID=580248541584

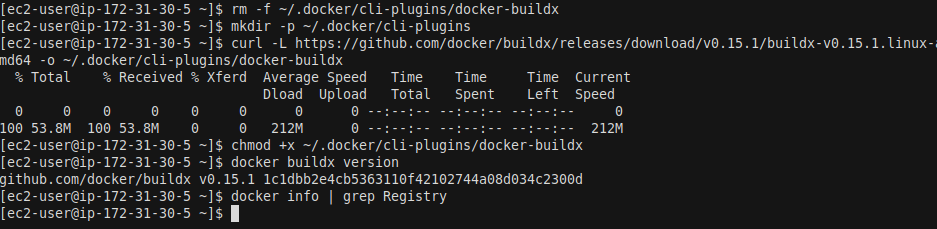
REGION=us-east-1

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



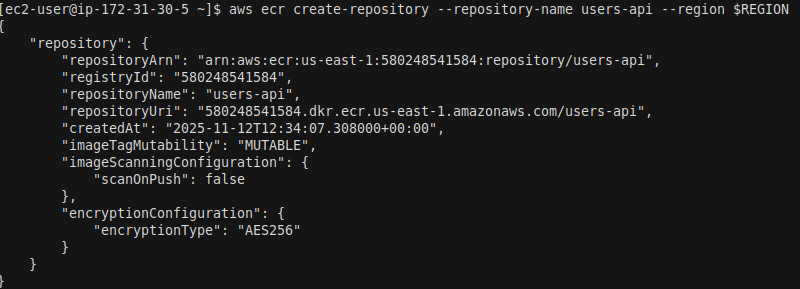
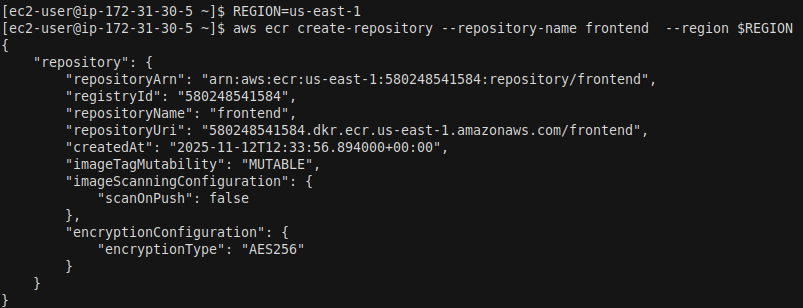
rm -f ~/.docker/cli-plugins/docker-buildx

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

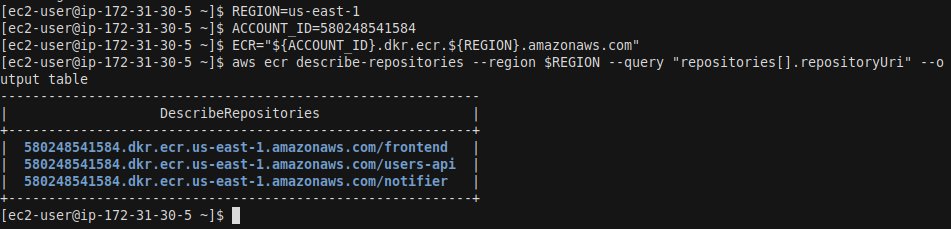


REGION=us-east-1

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

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

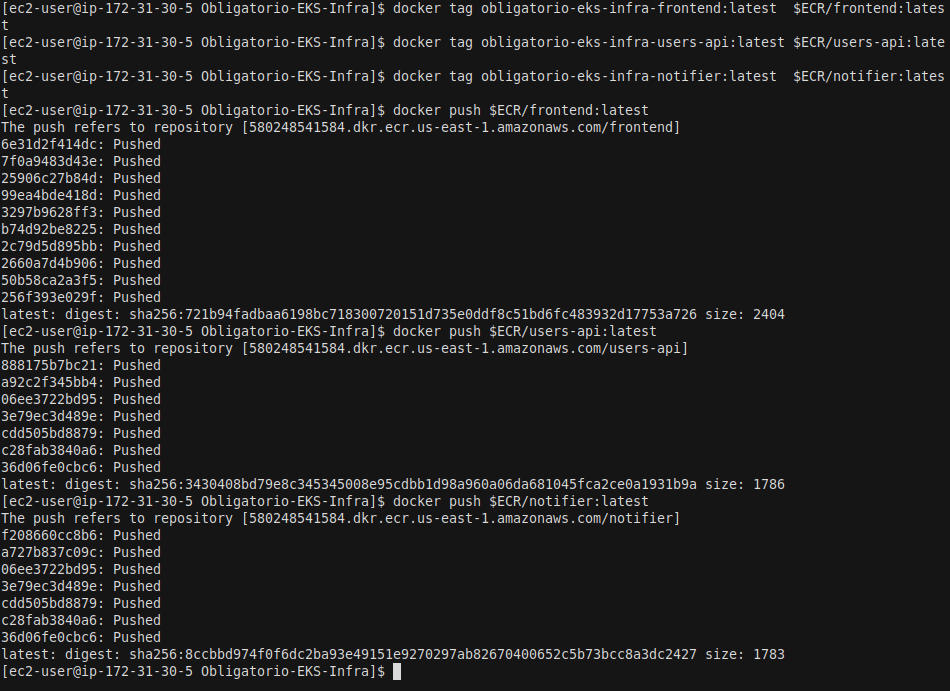
aws ecr create-repository --repository-name notifier --region $REGION



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

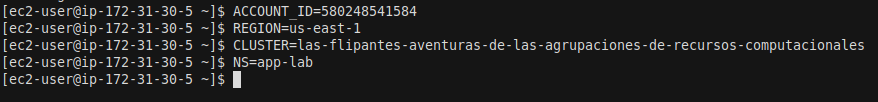
docker tag obligatorio-eks-infra-notifier:latest $ECR/notifier:latest

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



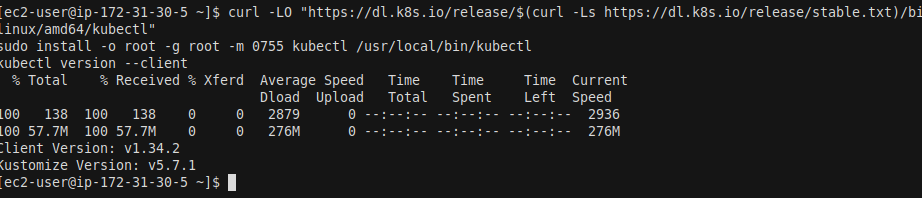
sudo curl --silent --location "https://github.com/eksctl-io/eksctl/releases/latest/download/eksctl\_$(uname -s)\_amd64.tar.gz" -o /tmp/eksctl.tar.gz

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

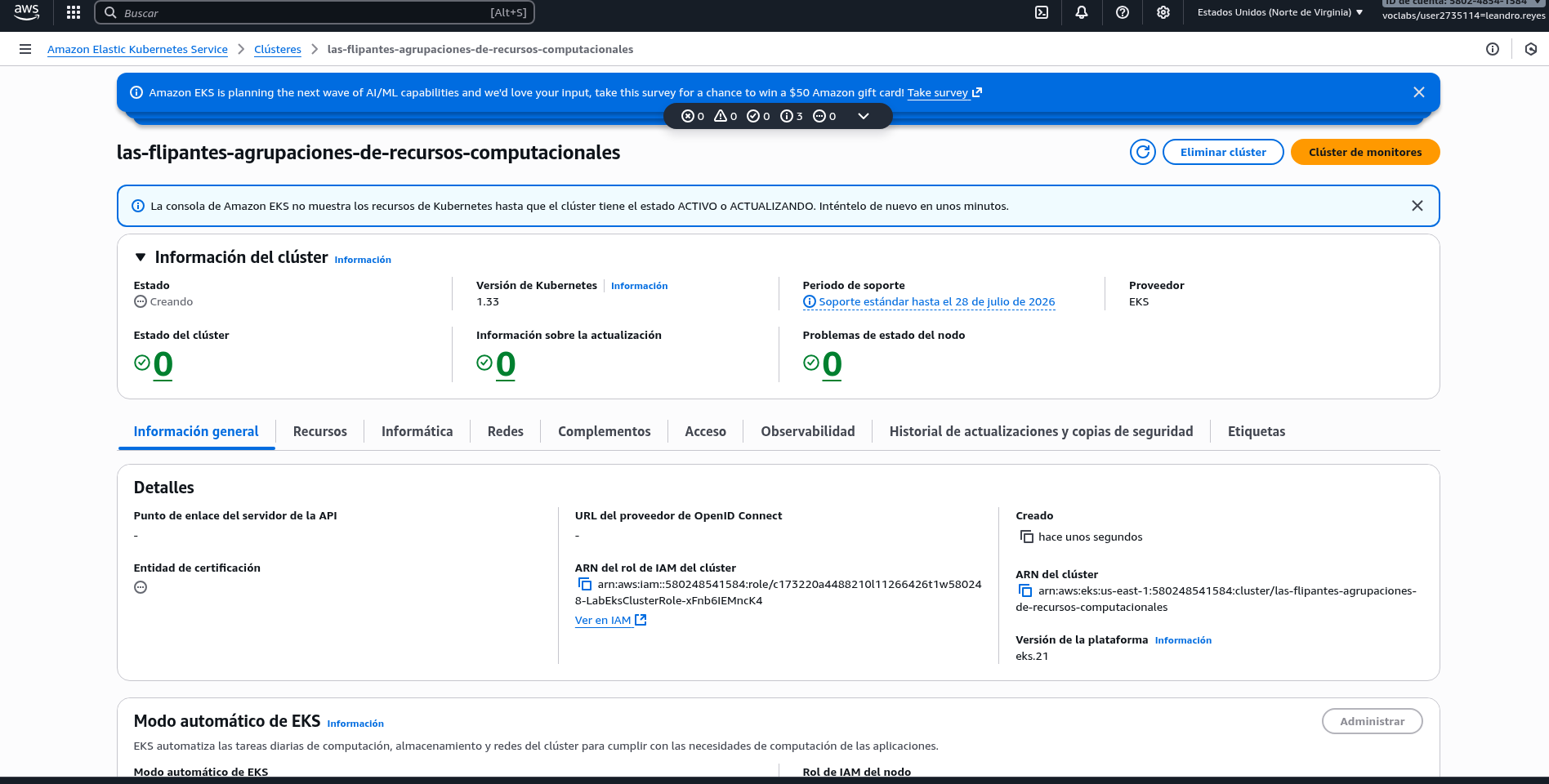
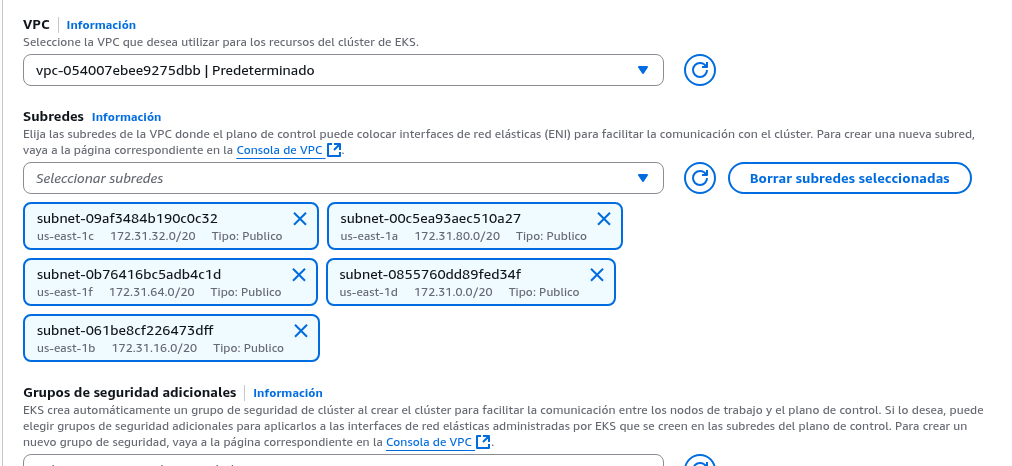
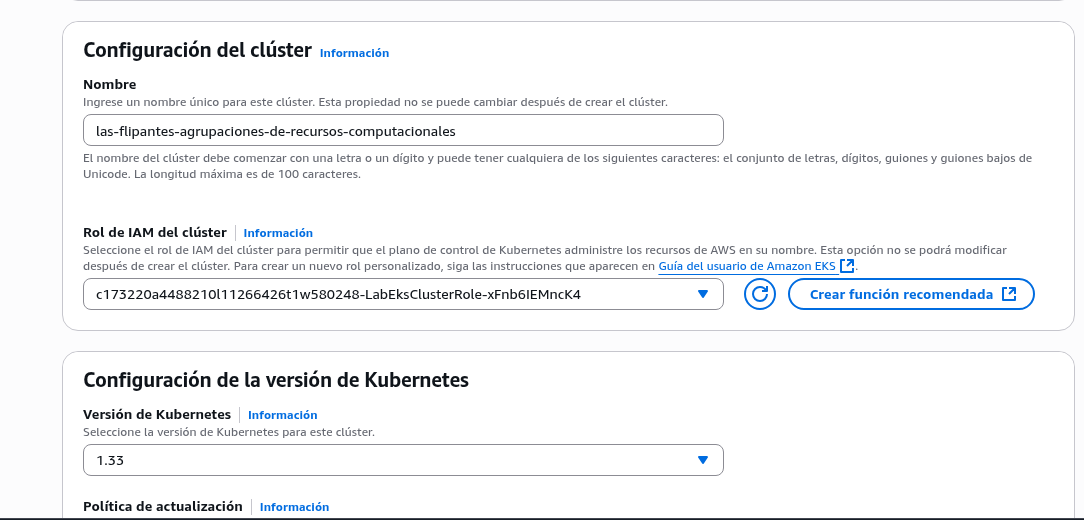


curl -LO "https://dl.k8s.io/release/$(curl -Ls https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

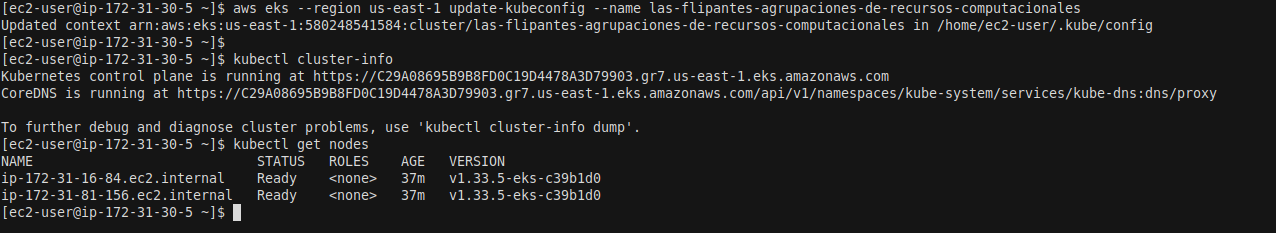
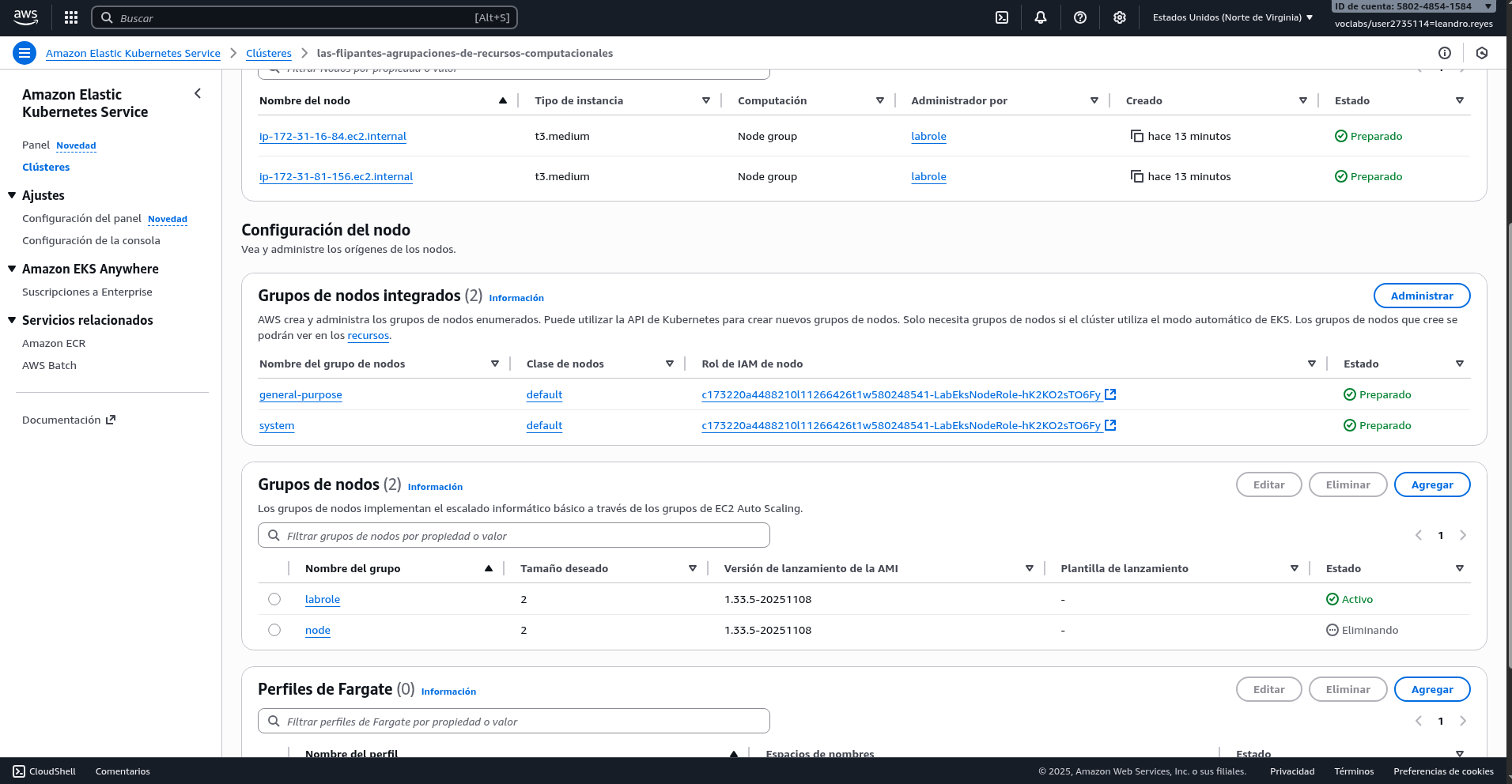
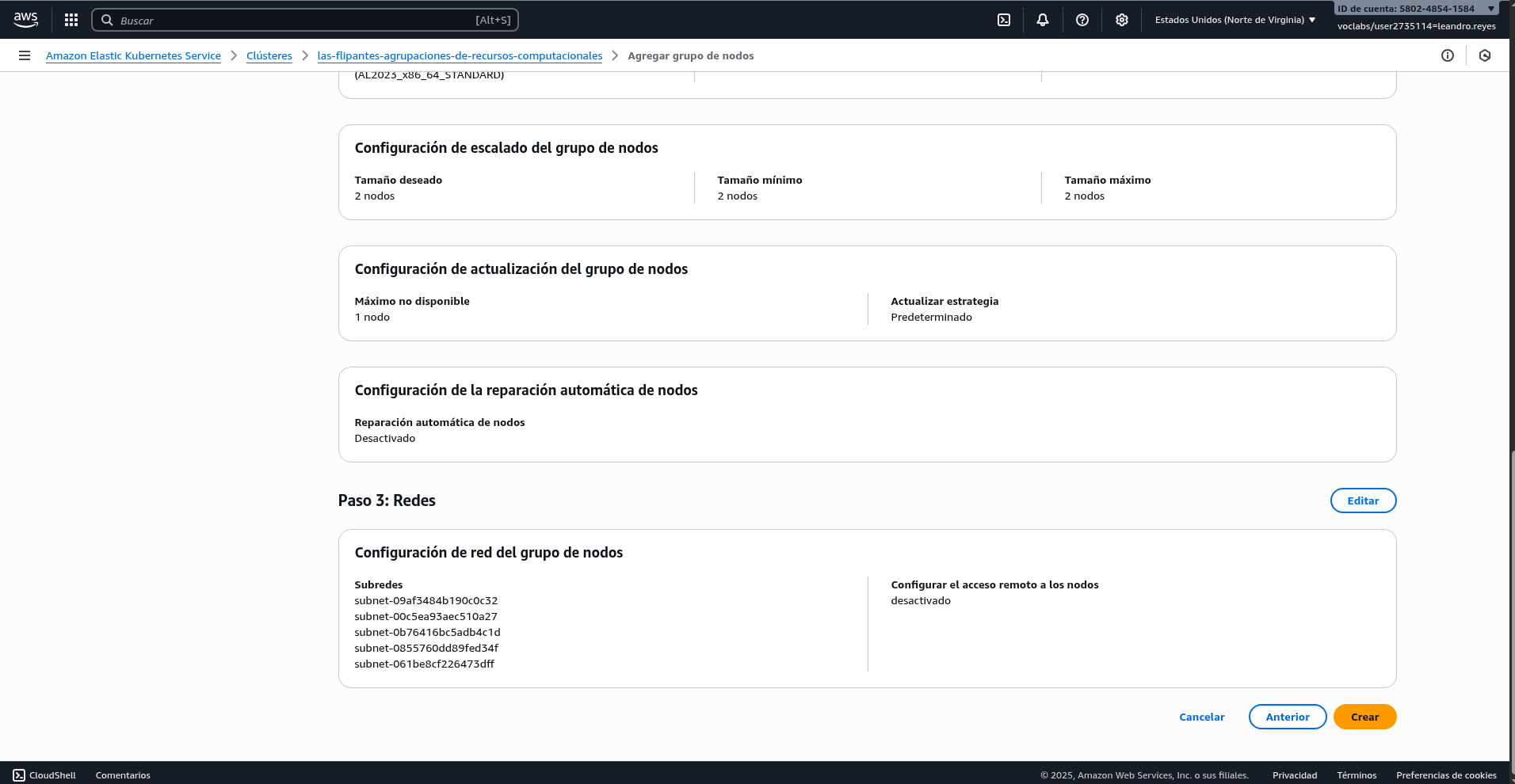
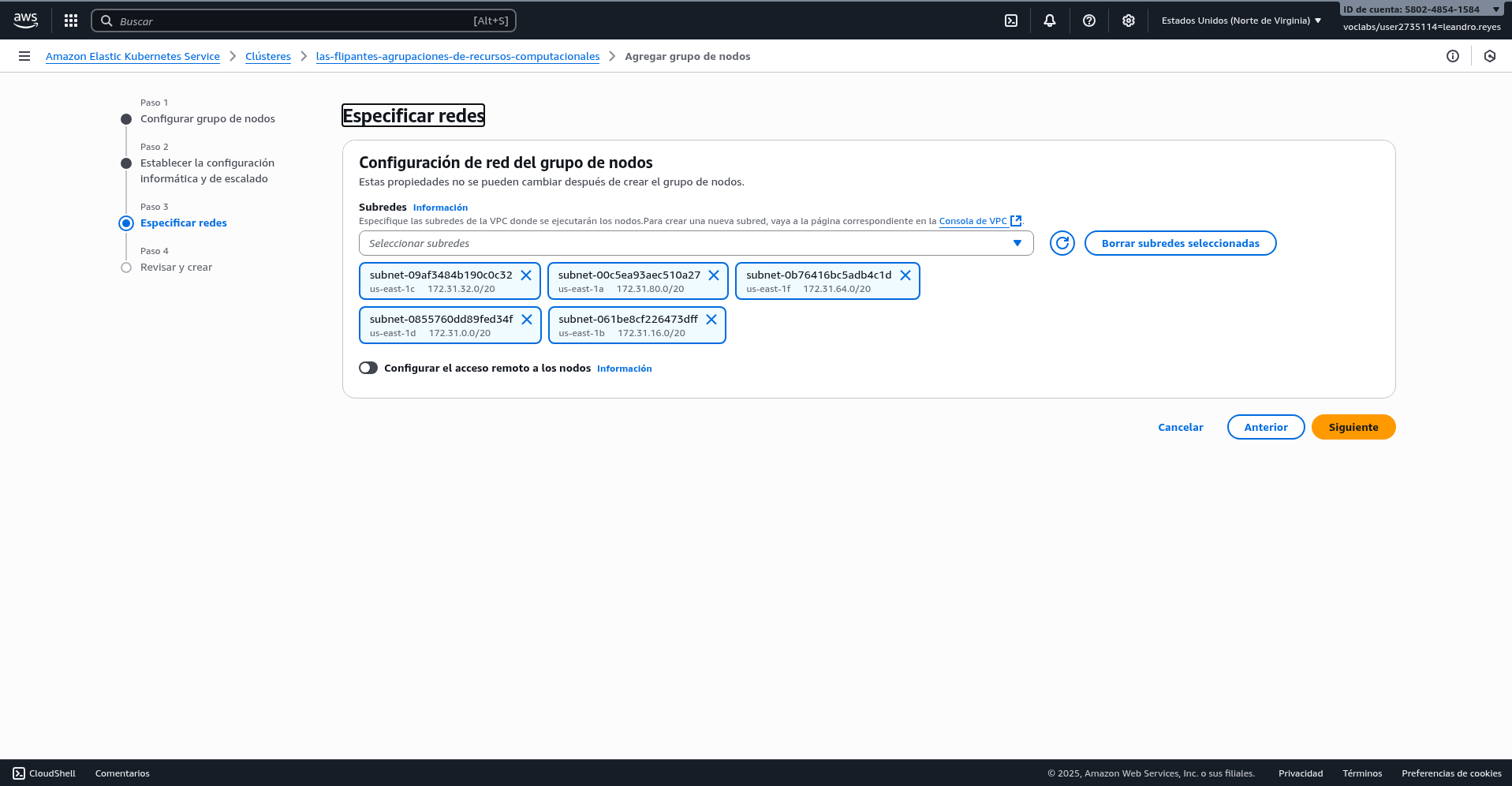
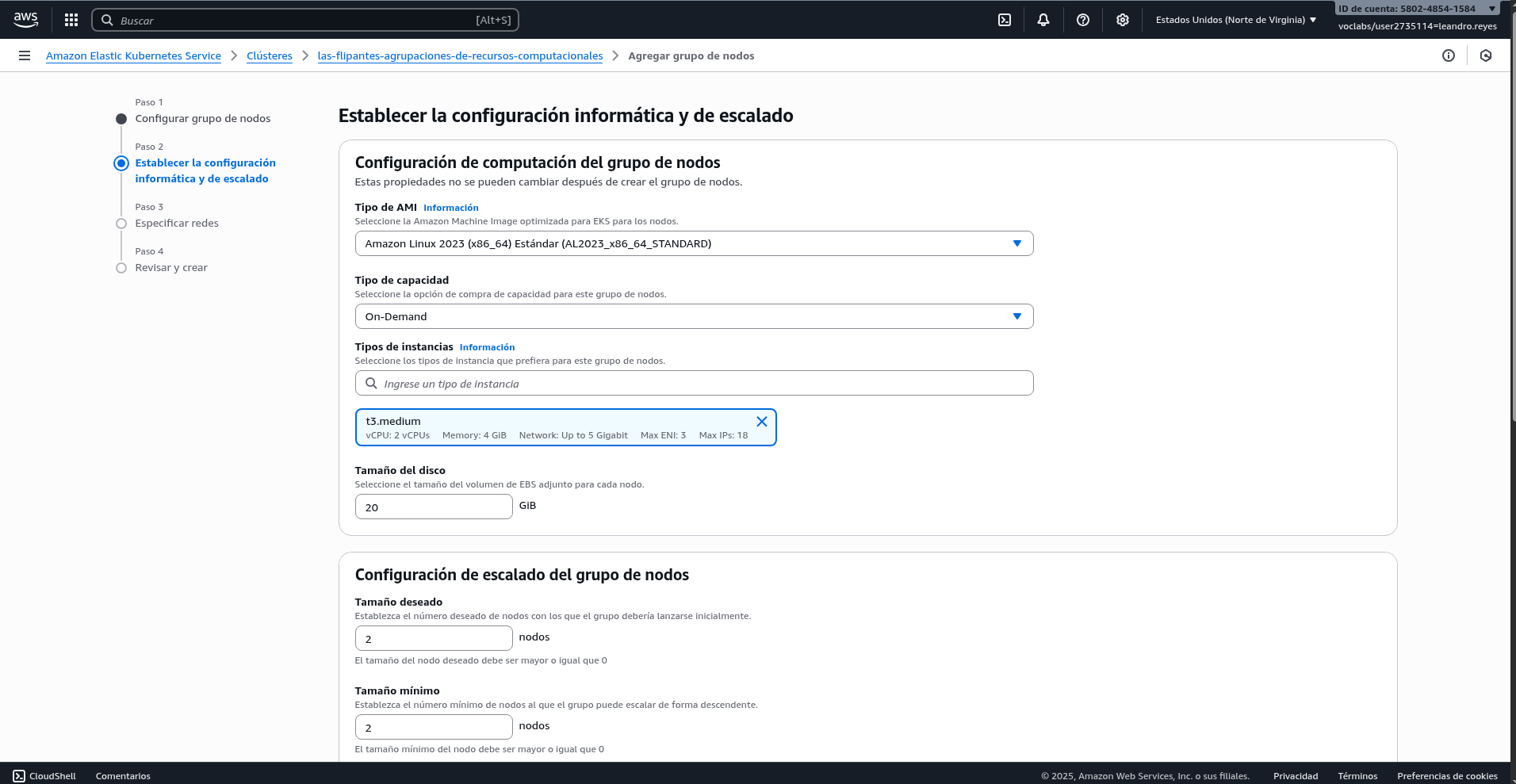
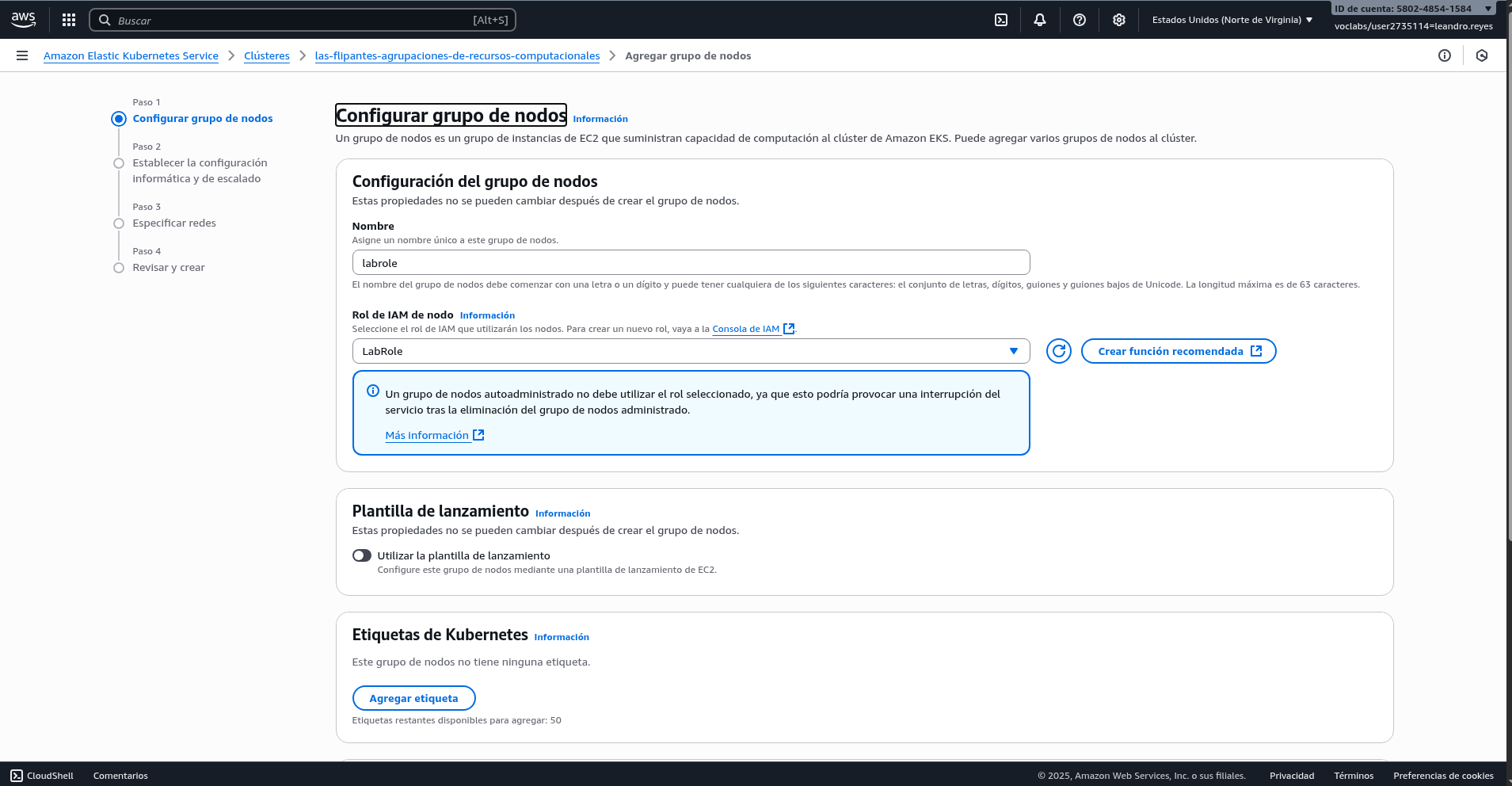
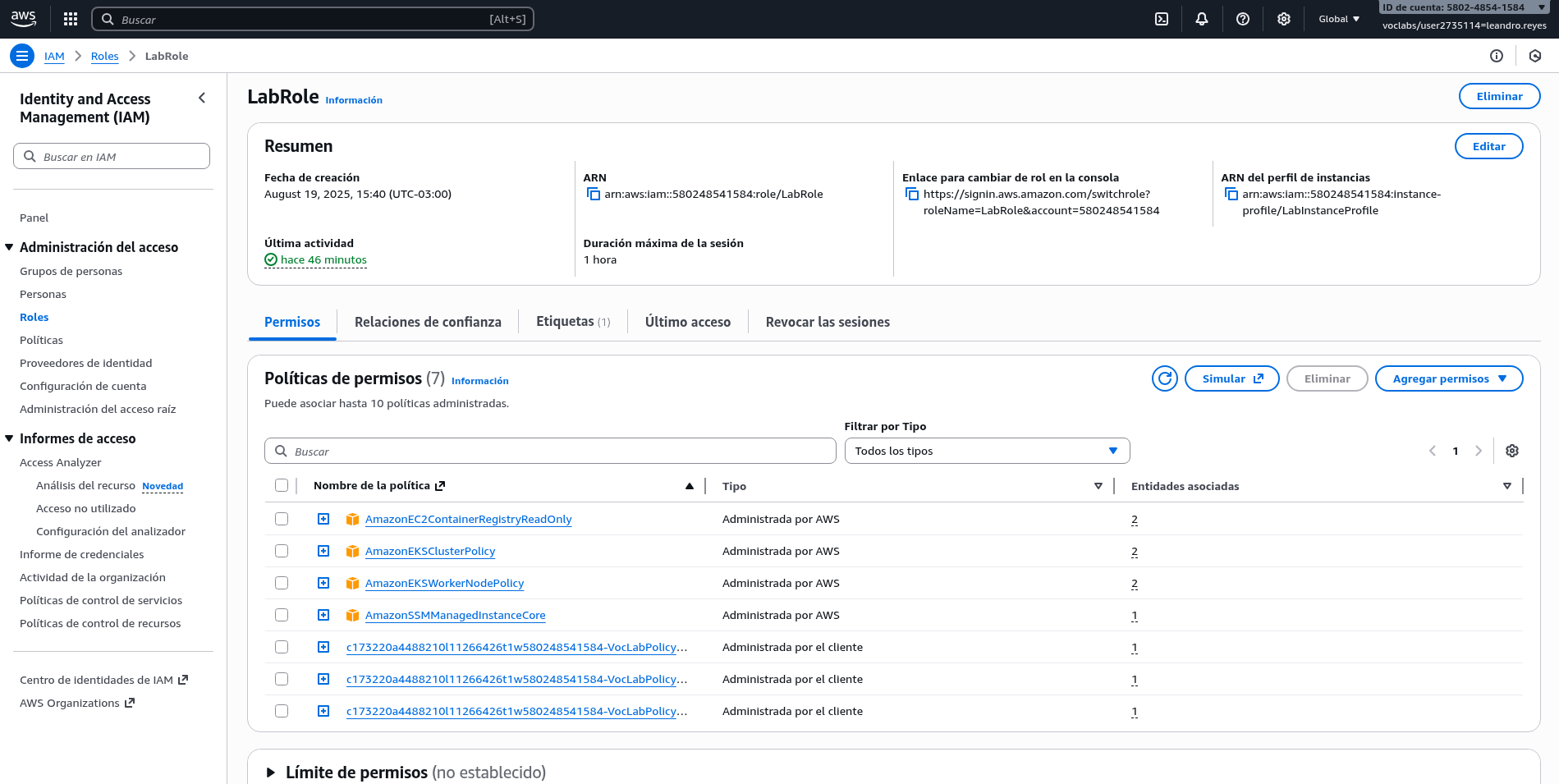
sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

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

## Evidencias y Capturas

Registro de imágenes del funcionamiento