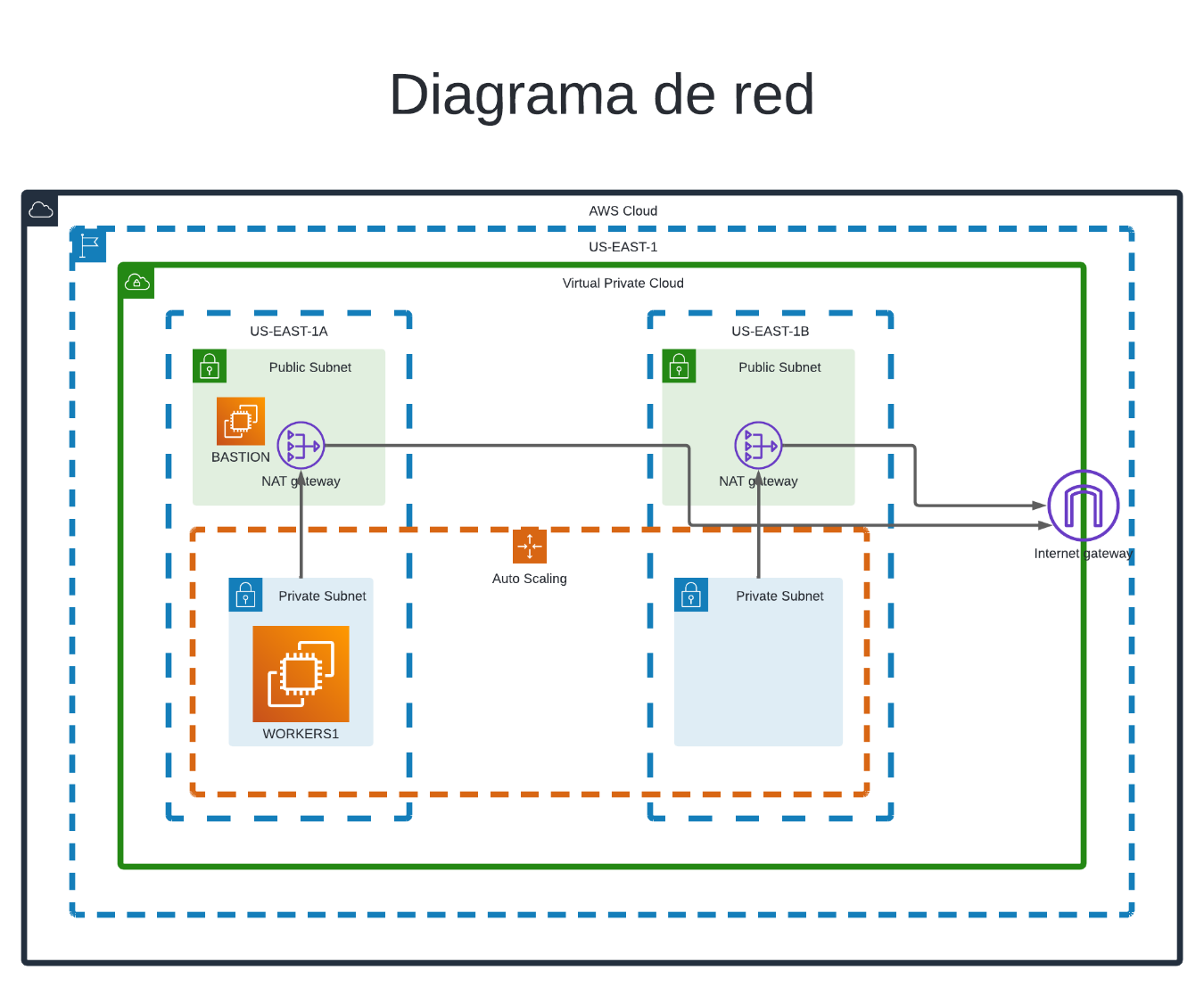
DOCUMENTACION PRUEBA SMART

Introducción

Este documento contiene el proceso que se llevo a cabo para dar solución con la prueba técnica donde su utilizo Terraform para crear la infraestructura a nivel de VPC, Grupos de Seguridad e Instancia bastión o pública, Se uso el servicio EKS, ECR, un ELB y un pipeline dentro de una cuenta demo de GitHub.

**Diagrama de red**

Debido a los requisitos para montar un cluster de EKS, el diagrama de red cuenta con dos subredes públicas y privadas, con sus dos respectivos nat Gateway en cada sub red publica

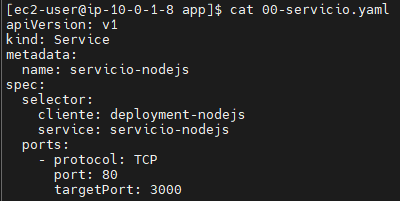


**Infraestructura eks**

Se montó un cluster de EKS sencillo, en su interior tiene:

A nivel de deployment:

1. Servicio

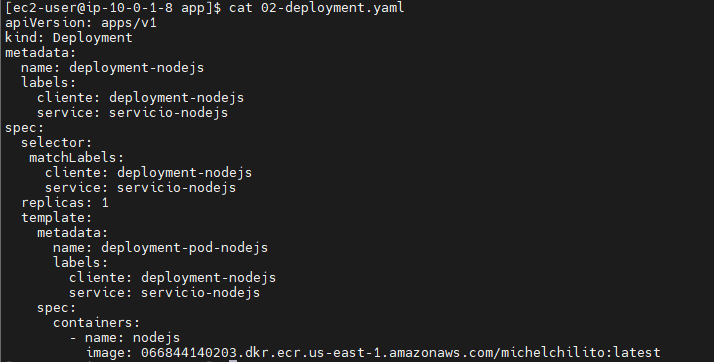


1. Ingress

Texto

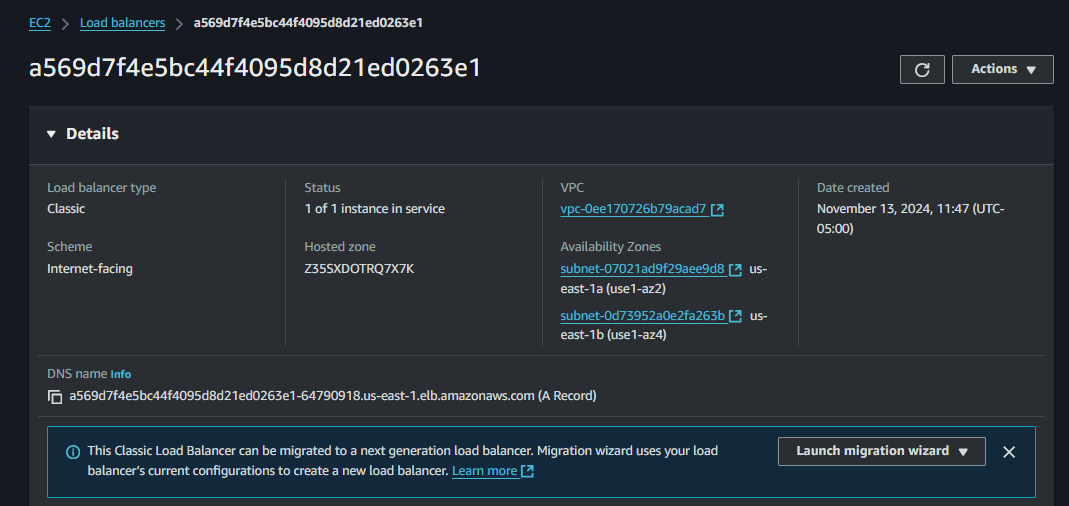
Descripción generada automáticamente

1. Deployment

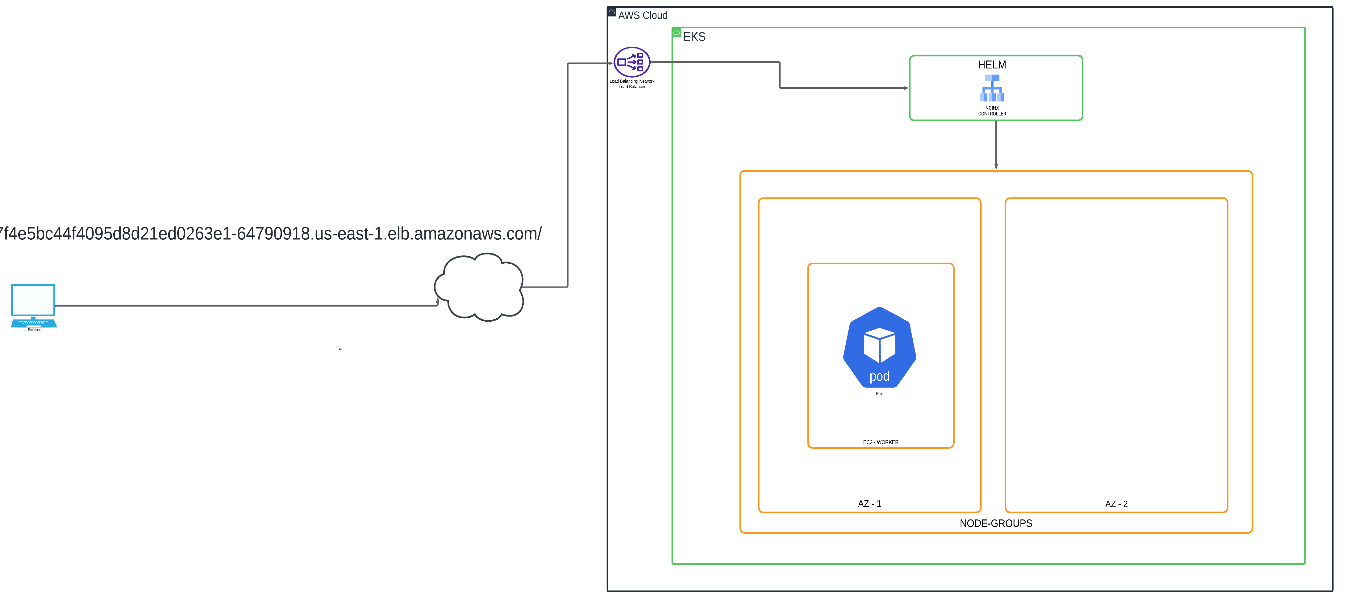


Otras tecnologías usadas dentro del cluster eks:

1. Se instalo HELM el cual se utiliza para crear un nginx-controller, donde este a su vez crea un ELB para el acceso a los servicios que tengamos en nuestro cluster, en este caso nuestro deployment nodejs



**Diagrama representativo de lo mencionado anteriormente.**

****

**Fase 1** Creación de la infraestructura como código.

Se crearon 5 archivos para crear el diagrama de red.

Texto

Descripción generada automáticamente

El archivo siguiente trata de:

**Fase1-1-red.tf**

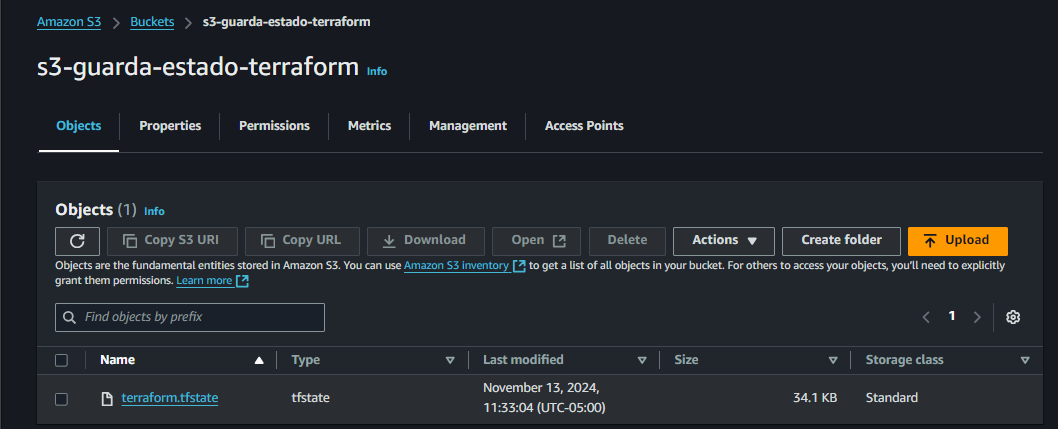
Tiene el código para crear todo lo necesario a nivel de VPC

**Fase1-2-ec2.tf**

Tiene el código para crear una sola instancia, la instancia bastion.

**Fase1-3-s3-guarda-estado.tf**

Tiene el código para consumir el s3 que se creo anteriormente con el objetivo de guardar el estado de terraform



**Proveedor-coud.tf**

Tiene el código donde especificamos que nube usaremos y su region, en este caso aws – us-east-1

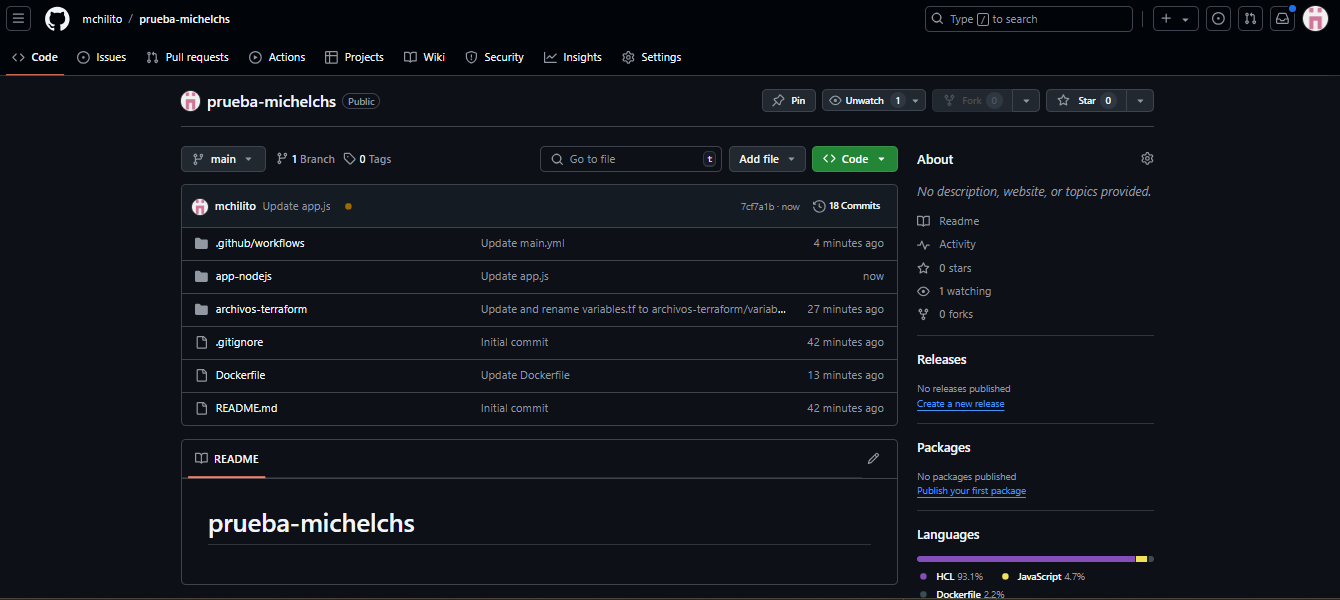
**Variables.tf**

Tiene el código donde creamos las variables para la región de aws

**Fase 2** Crear un repositorio Git y subir un proyecto de aplicación

1. Para el repositorio, se creo un repositorio publico temporal en GitHub

<https://github.com/mchilito/prueba-michelchs>



1. Para la aplicación, se uso NodeJS con un ejemplo básico.
2. Para las pruebas del uso de Docker, se creo un servidor de pruebas local donde se instalo Docker y se creo la siguiente estructura

Texto

Descripción generada automáticamente

1. Para almacenar las imágenes de los contenedores, se uso el servicio ECR

Captura de pantalla de un celular

Descripción generada automáticamente

**Fase 3** – Automatización del despliegue – Pipeline

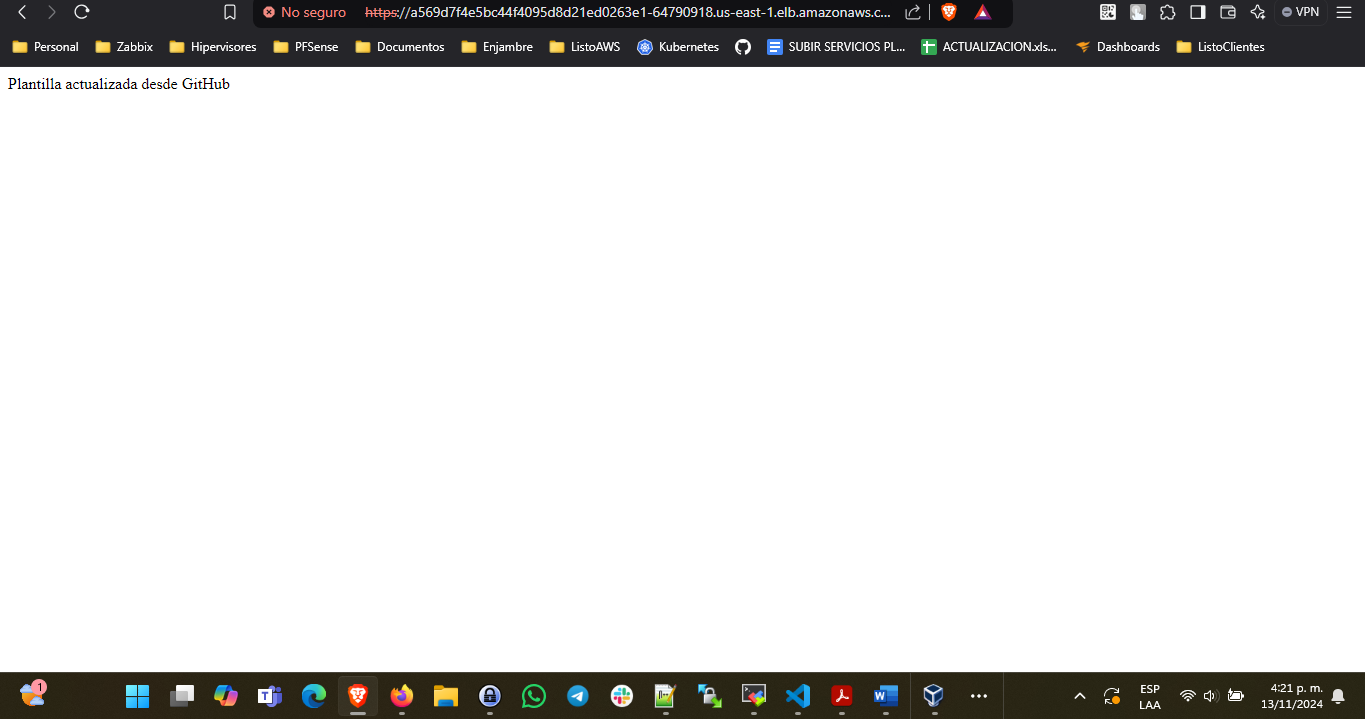
**NOTA**: El pipeline se encuentra en la ruta prueba-michelchs/github/workflows del repositorio

Se uso Github Actions para esta tarea.

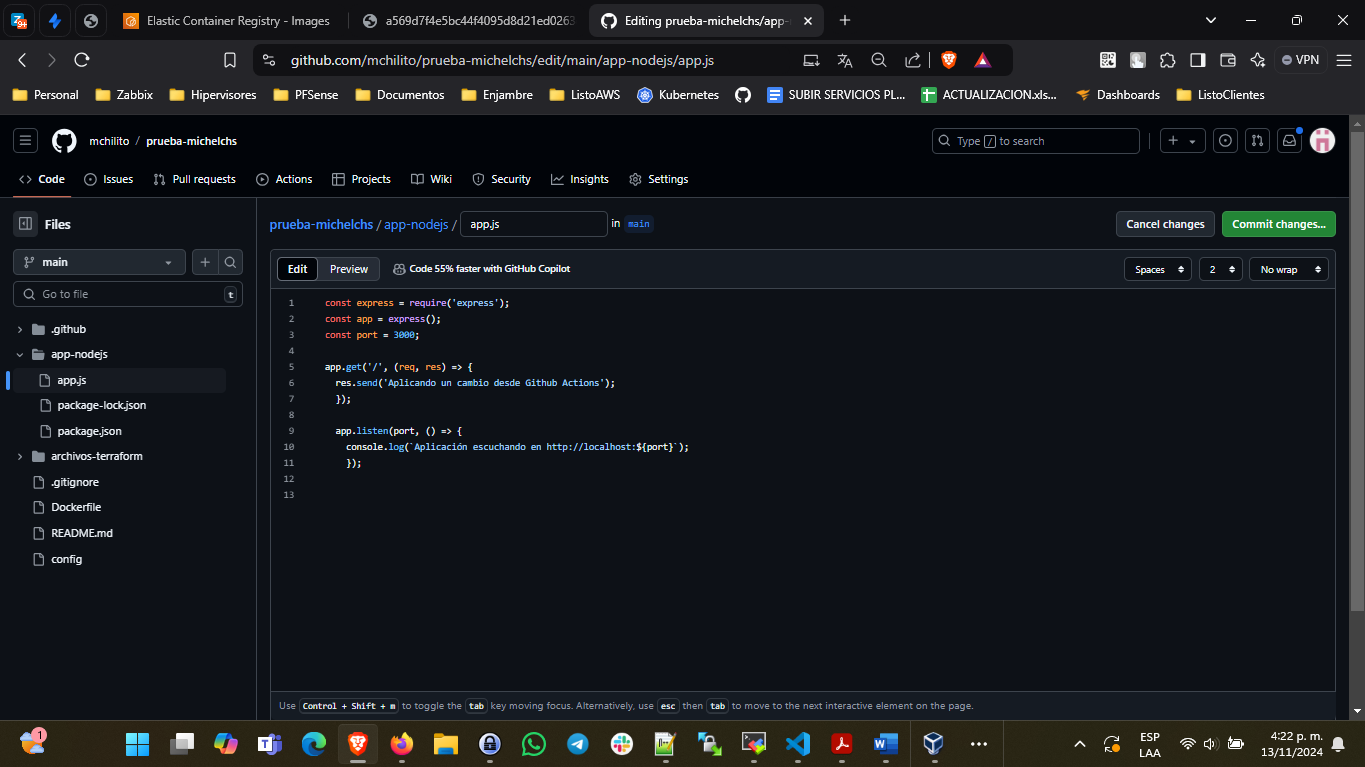
Simulando un cambio

Primeramente, se tiene este servicio desde la siguiente url

<https://a569d7f4e5bc44f4095d8d21ed0263e1-64790918.us-east-1.elb.amazonaws.com/>



Vamos a editar el código del archivo



Damos commit

Captura de pantalla de computadora

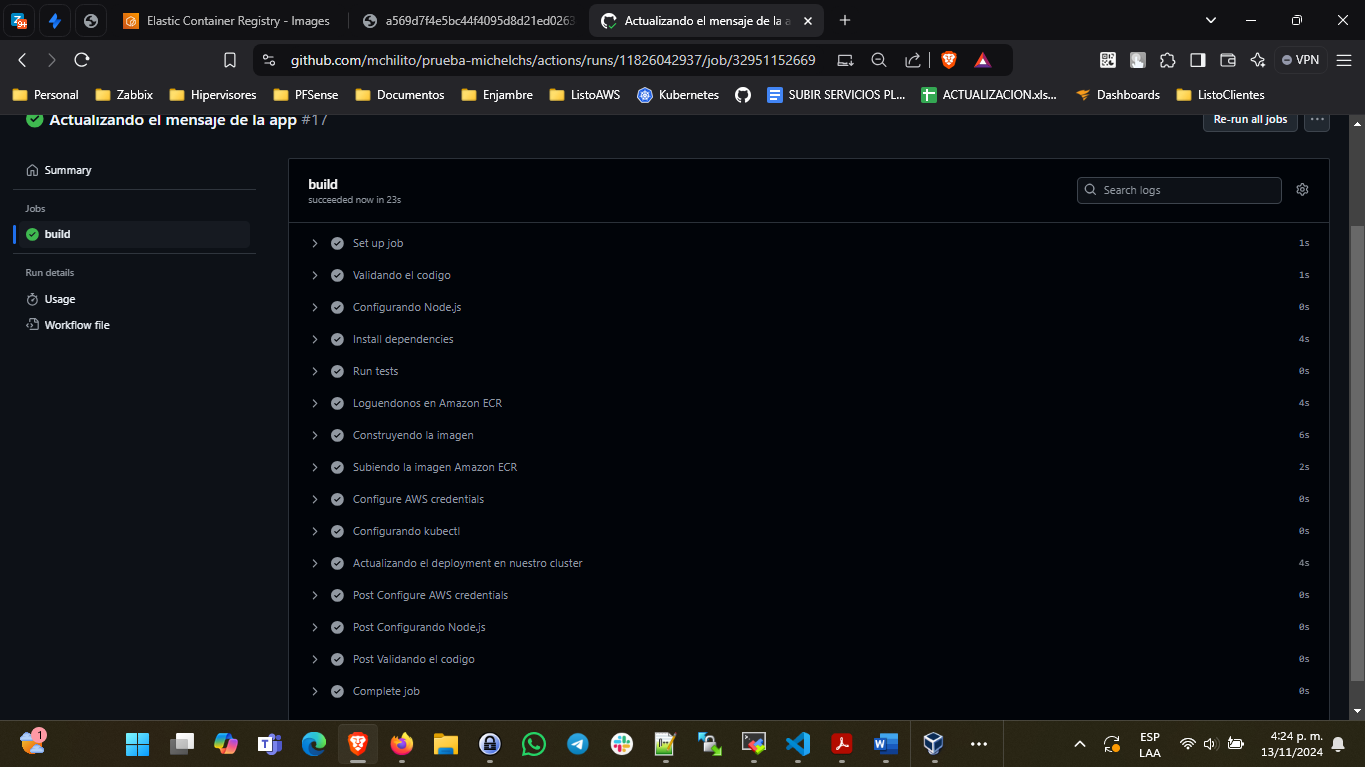
Descripción generada automáticamente

Vemos el flujo o Actions

Pantalla de un video juego

Descripción generada automáticamente con confianza media

Aquí solo tenemos un stage llamado build y podemos apreciar que salió todo bien

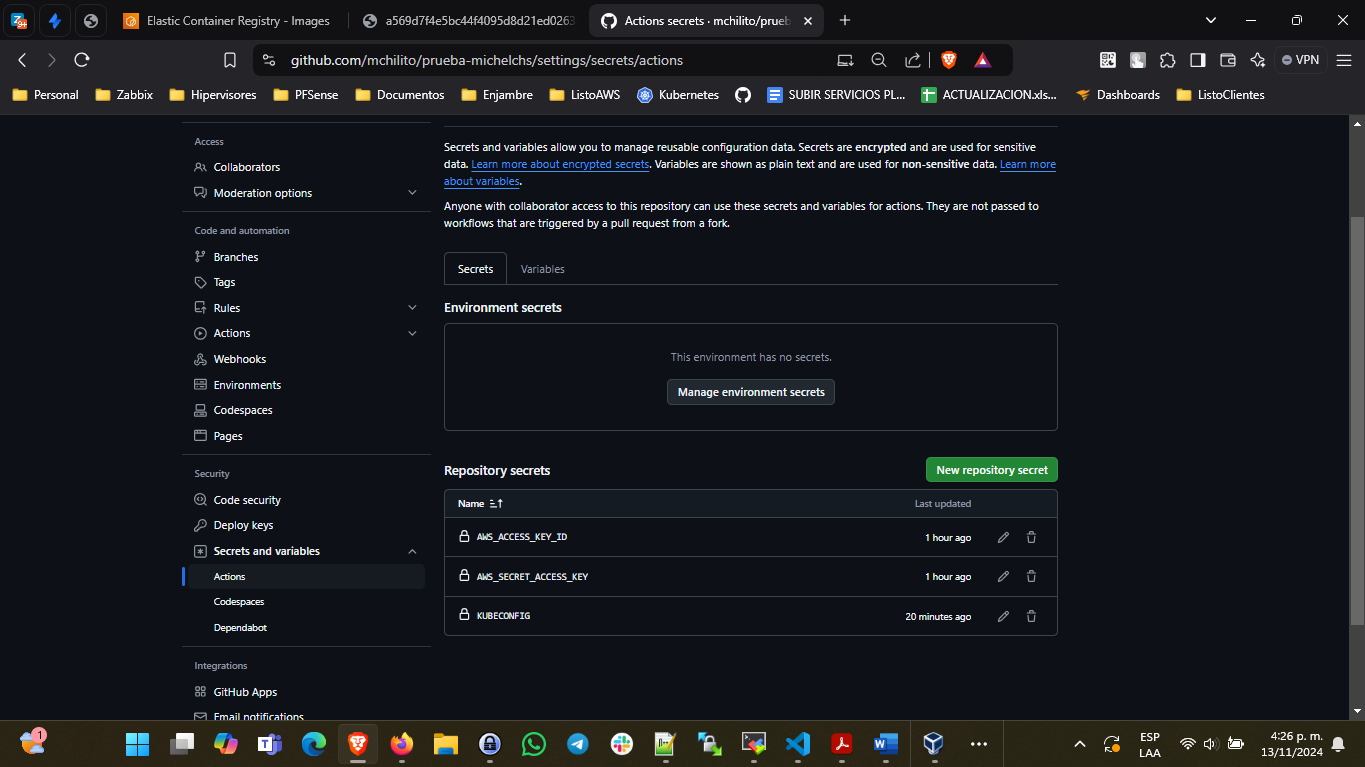


Refrescamos la pagina y obtenemos el nuevo mensaje

Interfaz de usuario gráfica, Texto, Aplicación, Word

Descripción generada automáticamente

Para lograr esto, se crearon unos secrets en el repo



**CONSIDERACIONES A MEJORAR**

1. Agregarle o dividir el proceso que realiza el pipeline y no dejarle solo un stage (build).
2. Crear el cluster EKS desde Terraform y no manualmente.
3. Añadir pruebas de integración
4. Implementar alertas para monitorear los fallos de los despliegues.
5. Implementar un proceso de limpieza de imágenes en ECR mediante una lambda.