ITESO

Universidad Jesuita de Guadalajara

Gaming Hub

Equipo número: 5

Presentado por: <Luis Angel Santana Hernandez, Juan bernardo orozco quirarte, luis daniel arellano núñez>

Correo Electrónico: <luis.santana, bernardo.orozco, luis.arellano>@iteso.mx

Para el curso de: Cloud Computing (Development)

Impartida por: Mtro. Ojeda Orozco Miguel Angel

Fecha: <31/29/04/2025>

Nombre del proyecto

Gaming Hub

Marco Teórico

Amazon, junto con su alojamiento de servidores AWS, ha creado Amazon Gamelift, un método de alojamiento de servidores de videojuegos. Gamelift leyendo un poco son instancias de EC2 y EC2 Graviton, con las cuales levantan servidores de videojuegos [1].

A partir de esto surge la pregunta: ¿podemos hacer un servicio similar a este?

Durante este proyecto, hemos decidido hacer un sistema el cual te ayude a alojar servidores e información de estos dentro de servicios de EC2, servicios de almacenamiento, etc. Debido a las necesidades del proyecto que se encuentran en el área de requisitos, decidimos usar las siguientes herramientas:

Amazon EC2: Sera la base para el despliegue y la administración de servidores de videojuegos en la nube, con la ayuda de EC2 podremos ejecutar instancias virtuales con la capacidad de cómputo necesaria para poder correr juegos con baja latencia y buen rendimiento, además EC2 nos brinda flexibilidad en la selección de los tipos de instancia, cosa que nos va a permitir optimizar los costos y el rendimiento dependiendo de la demanda del juego, y bueno para esta propuesta. Para el Demo incluso con un tipo de instancia sin muchos recursos podemos testear el funcionamiento de nuestro servidor [2].

Amazon S3: Para el almacenamiento de archivos y datos que se relacionen con nuestros servidores. S3 nos permite guardar datos, en este caso siendo más específicos los tipos de archivos y datos que podemos guardar son configuraciones de juegos, logs, backups de servidores, entre otras cosas. Su alta durabilidad y también escalabilidad nos garantizan que la información esté disponible 24/7 sin comprometer el rendimiento de nuestro sistema [3].

Amazon CloudWatch: Para monotireo y generación de alertas sobre el estado de los servidores, CloudWatch es una herramienta estrechamente relacionada con el pilar de Excelencia Operativa, nos permitirá monitorear métricas como el uso del CPU, memoria y tráfico de red a las instancias, automatizar acciones en caso de fallas [4].

Amazon Cloud Front: La latencia en la comunicación es un factor de enorme importancia, Cloud Front nos permitirá optimizar la distribución de contenido, como lo pueden ser actualizaciones del juego, datos de configuración, animaciones como videos, asegurando que los jugadores puedan acceder de forma rápida a los recursos desde diferentes regiones [5].

Amazon DYNAMO: Lo utilizaremos para gestionar la base de datos, lo que nos permitira almacenar y administrar información de las cuentas de cada usuario [6].

AWS Lambda: Para automatizar tareas dentro de nuestro sistema, Lambda nos permitirá ejecutar algunas funciones sin necesidad de mantener servidores adicionales, cosa que nos reducirá costos y facilitará la implementación de procesos como el reinicio automático de servidores y procesamiento de logs y métricas [7].

API Gateway: Nos permitirá exponer una API REST segura y escalable, por medio de esta API los usuarios podrán interactuar con los servidores de manera sencilla y eficiente sin preocuparse por el acceso directo a la consola de AWS o configuraciones [8].

AWS IAM: La seguridad en la nube es un aspecto fundamental, un pilar, con IAM definiremos roles y permisos específicos para controlar el acceso a los servicios, de esta forma podremos garantizar que cada usuario tenga acceso a los recursos que le correspondan, aplicaremos el principio del menor privilegio para minimizar riesgos de seguridad [9].

Amazon VPC: Para garantizar un entorno de producción seguro, con Amazon VPC podemos gestionar dominios y resolver nombres de servidores de juegos que operen en una infraestructura separada de otros recursos públicos, nos permitirá configurar reglas de firewall para mejorar la seguridad y rendimiento [10].

Route 53: Para facilitar la conexión e los jugadores a sus servidores Route 53 nos permitirá gestionar dominios y resolver nombres de servidor, esto asegura que los usuarios puedan conectarse a sus servicios favoritos y servidores sin la necesidad de recordar direcciones IP directamente [11].

REQUERIMIENTOS

Principales

- Página web para que un usuario pueda generar y crear servidores de juegos (por el momento pensamos Minecraft, left for death, etc. [En si juegos con bajos recursos para no acabarnos los recursos de la plataforma educativa de AWS]
- EL usuario puede crear máximo 3 servidores
- Panel para parar, eliminar, actualizar servicios en la nube

Secundarios

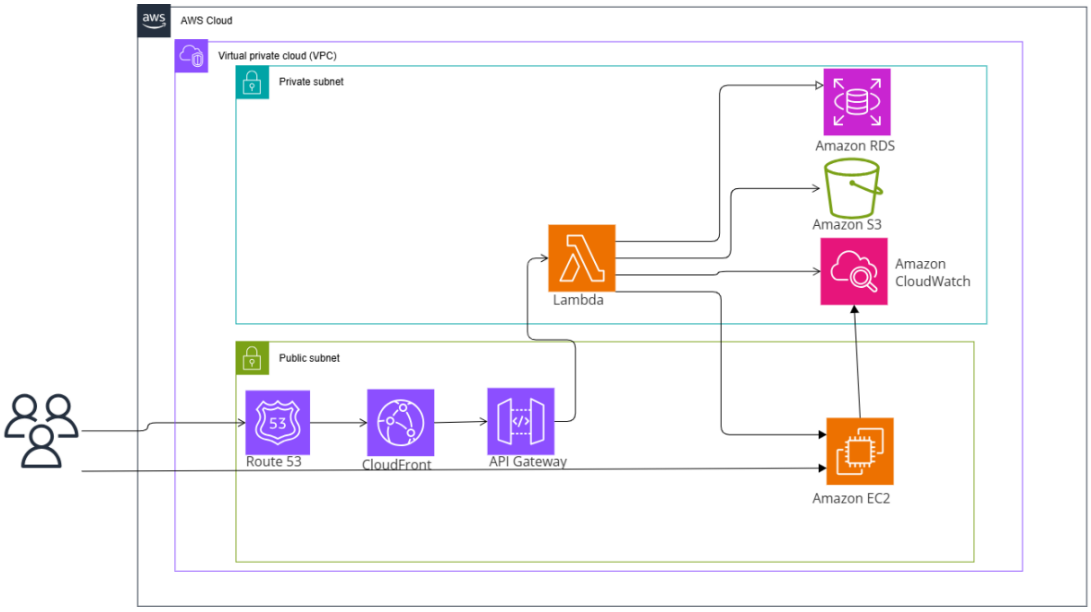
- Se pueden generar servidores públicos (visibles en la página web)
- Alguna forma de que el usuario pueda ver en la página web el porcentaje de uso de los recursos
- Validación de login del usuario

Opcionales u otros

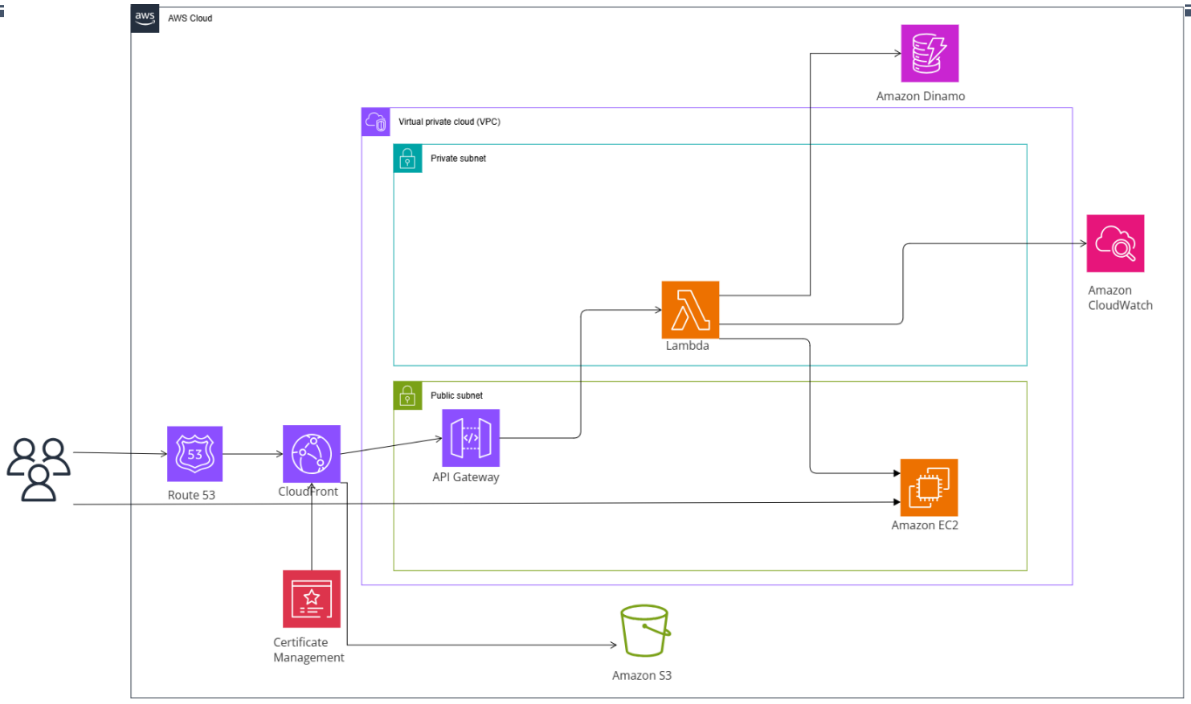
- Diseño de la página bonita
- Auto escalable

Diagrama

ANTES:



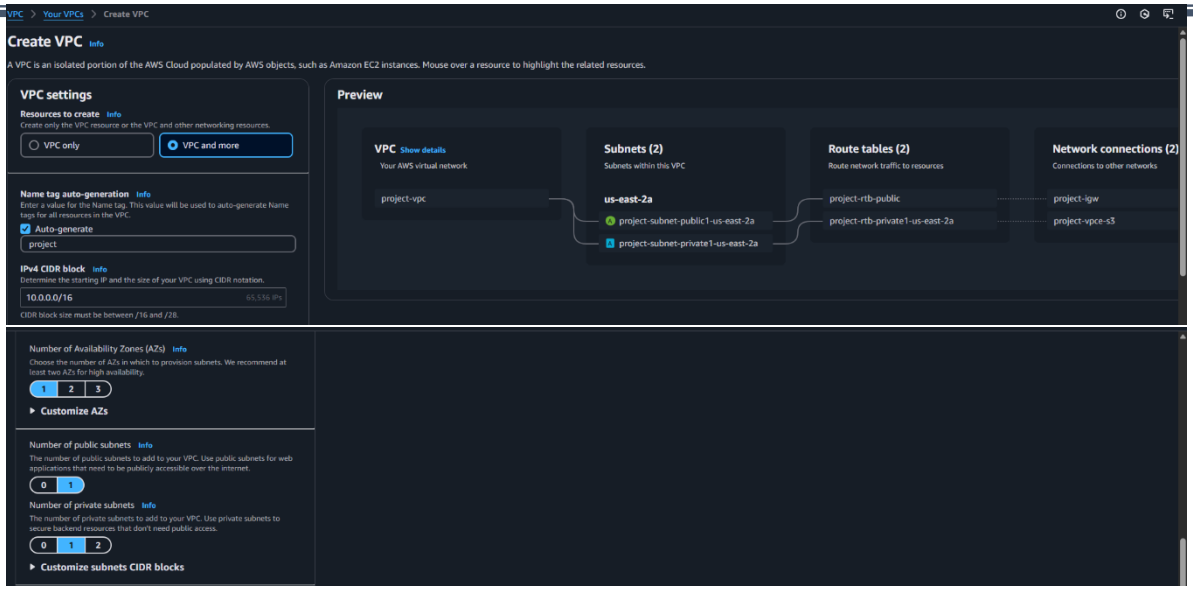
DESPUES:



DESAROLLO:

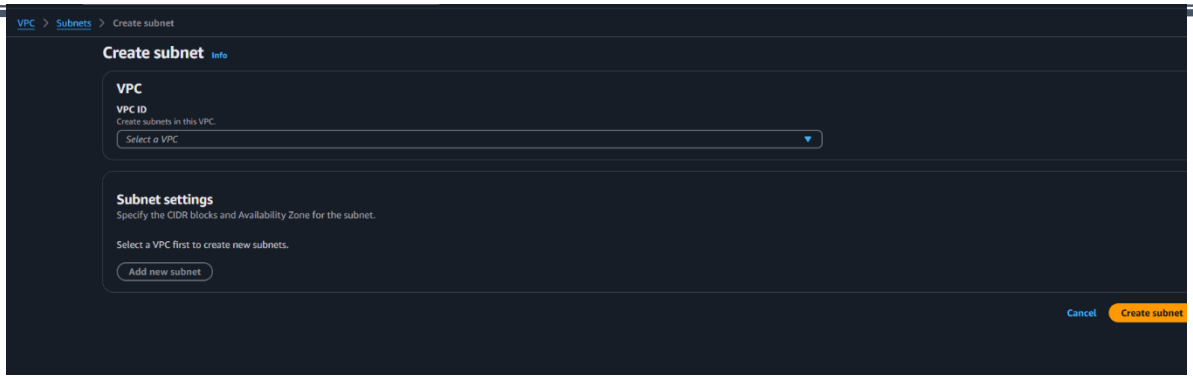
VPC:

1. Desde la consola de AWS acceder al servicio VPC
2. Creamos un VPC, cambiando Availability Zones en 1 en ves de 2(default). Antes de continuar debo de explicar que debes de dejar 2, porque varias aplicaciones te avisan que necesitan 2 availability zones, entonces luego vamos a poner el segundo AZ

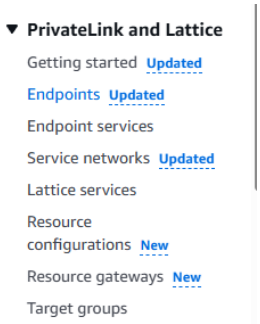


3. No le ponemos NAT Gateway

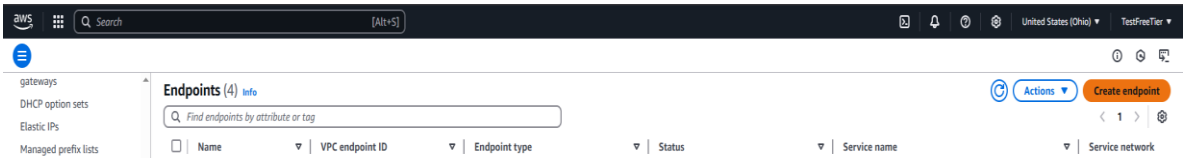
4. Una vez creada nos vamos a subntes y vamos a crear una subntet(COn esto ya deberamos cuplir con 2 AZ, que te piden aplicaciones como amazor RDS (Al final no lo usamos pero en su momento era necesario))



5. Añadir endpoints de conexión para comunicación entre subnets privadas y públicas



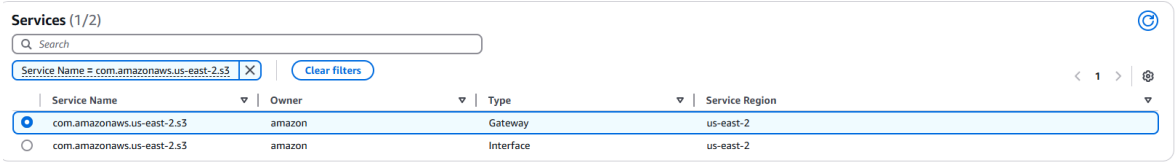
6. Dar clic en créate endpoint



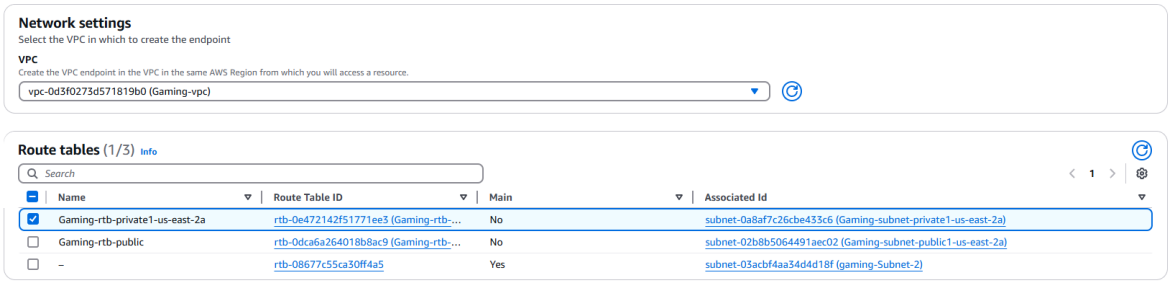
7. Agregar los siguientes endpoints 1 por servicio:

Servicio s3

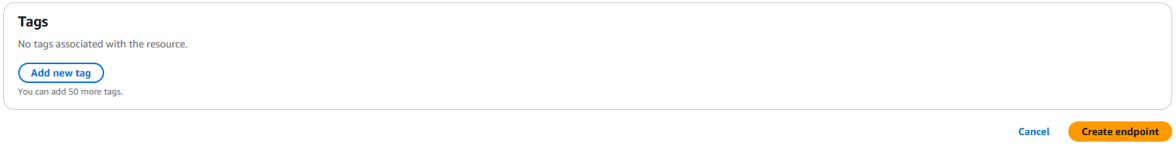
i) buscar el servicio: *com.amazonaws.us-east-2.s3* de tipo gateway



ii) Añadir el VPC creado y en la IP table seleccionar la subnet privada

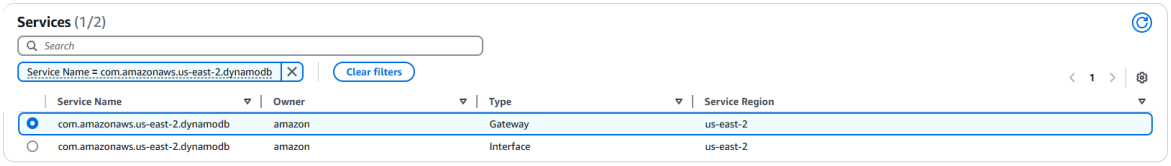


iii) Creamos el endpoint

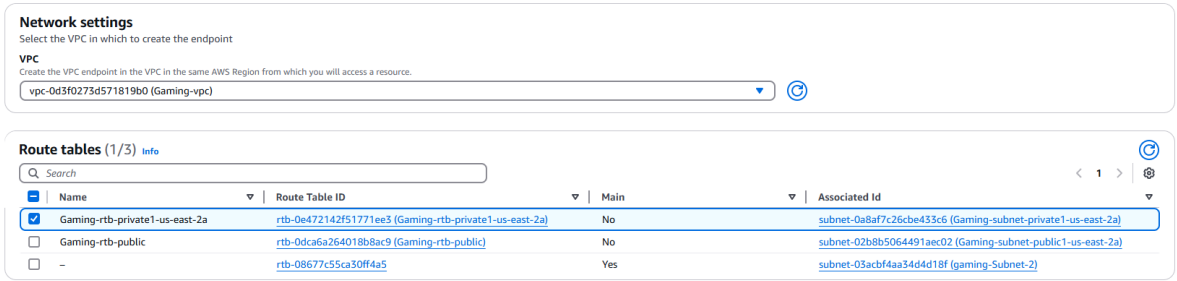


DYNAMO DB

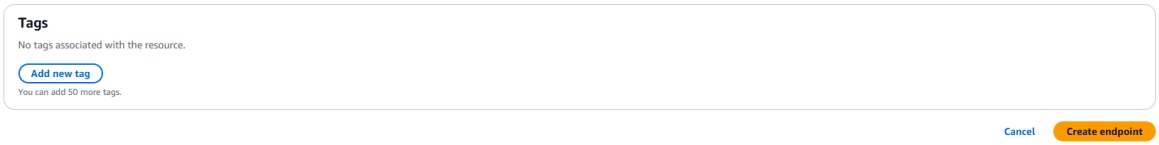
i) Buscar el servicio *com.amazonaws.us-east-2.dynamodb* de tipo Gateway



ii) Añadir VPC y en la IP table seleccionar la subnet privada

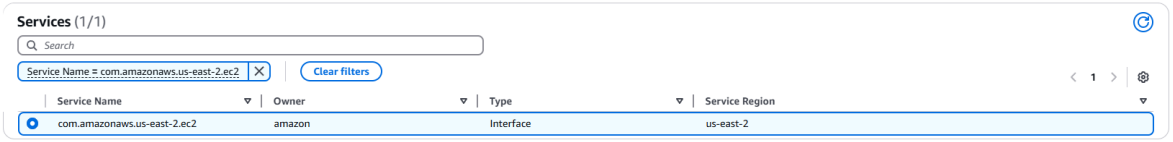


iii) Creamos el endpoint

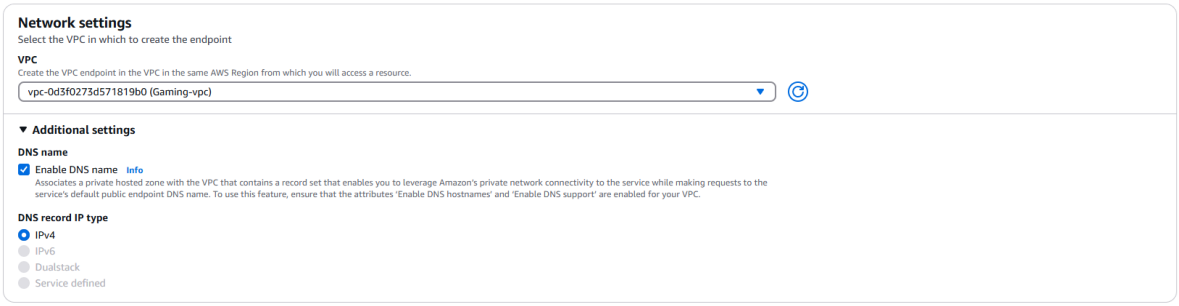


Servicio EC2

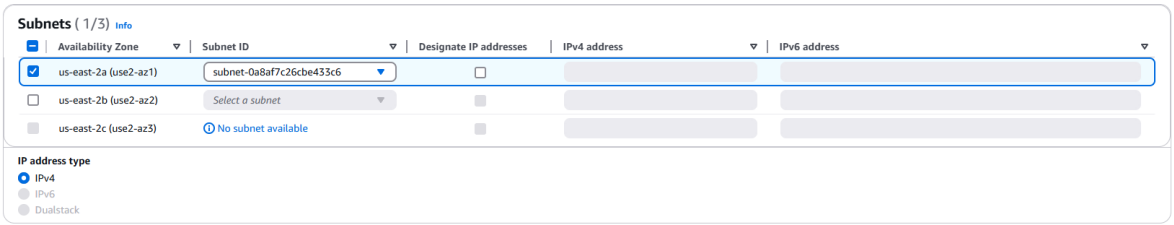
i) Buscamos el servicio *com.amazonaws.us-east-2.ec2*



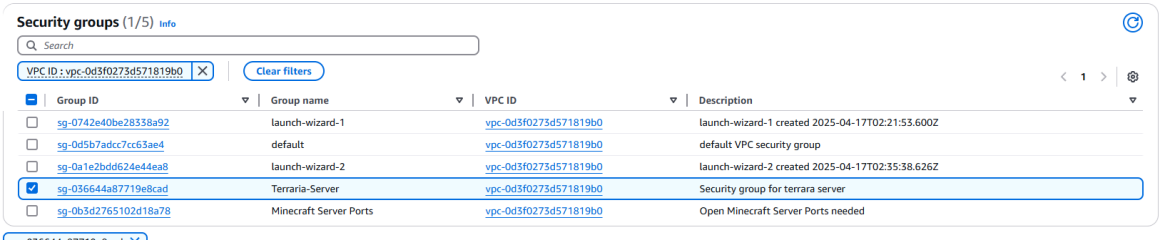
ii) Seleccionamos el VPC, habilitamos DNS y ipv4



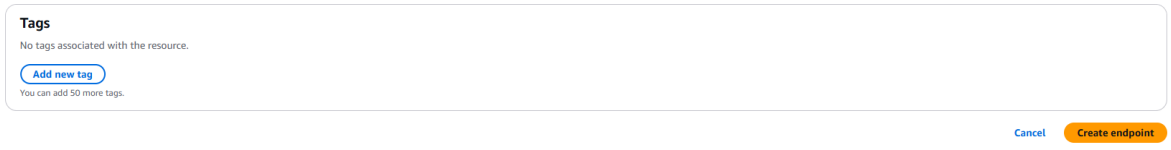
iii) Seleccionamos la zona y la subnet privada



iv) Seleccionamos el SG Terraria-server

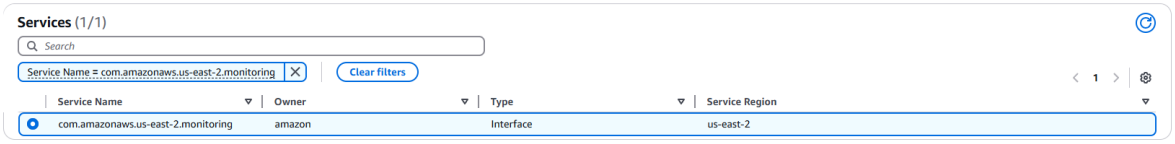


v) Creamos el endpoint

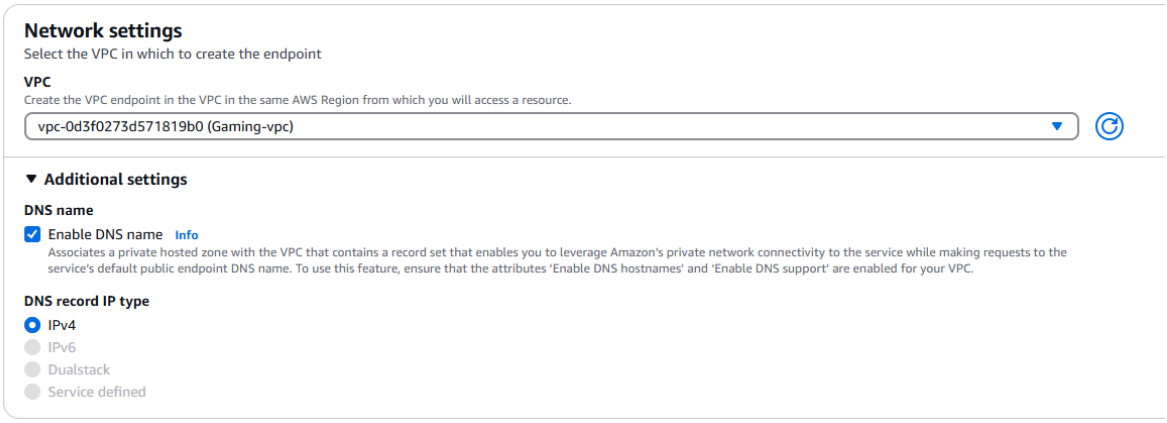


Servicio CloudWatch

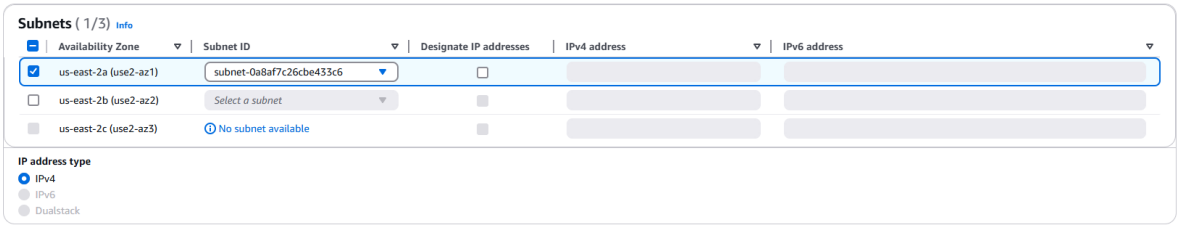
i) Buscamos el servicio *com.amazonaws.us-east-2.monitoring*



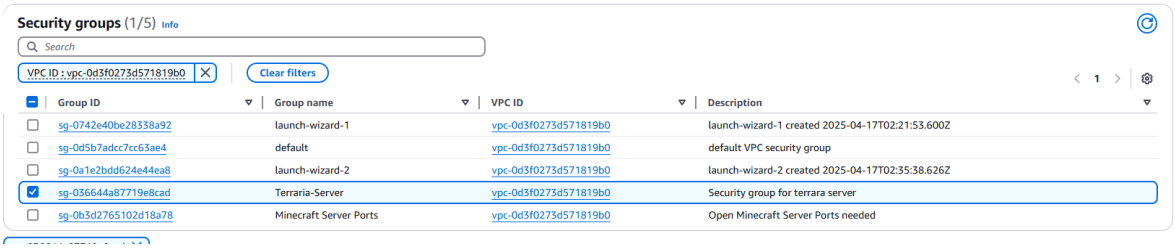
ii) Elejimos el VPC, DNS y ipv4



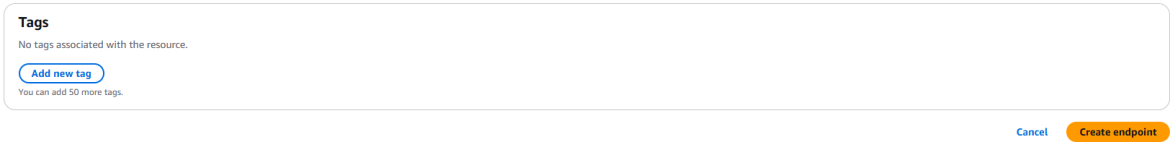
iii) Elejimos la subnet 2a y su rama privada



iv) Elejimos el security group Terraria Server

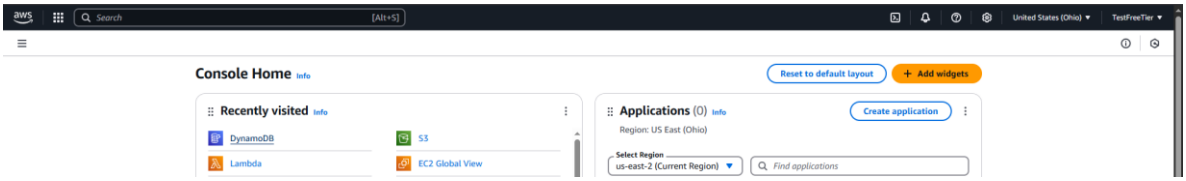


v) Creamos el endpoint

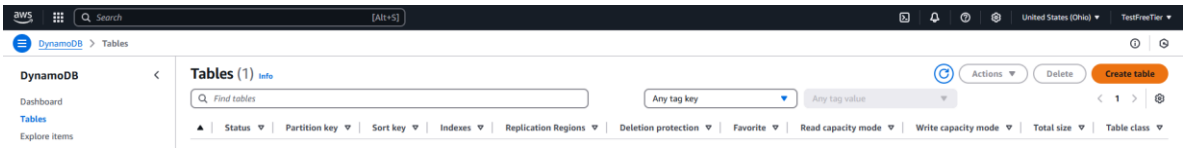


AMAZON DYNAMO:

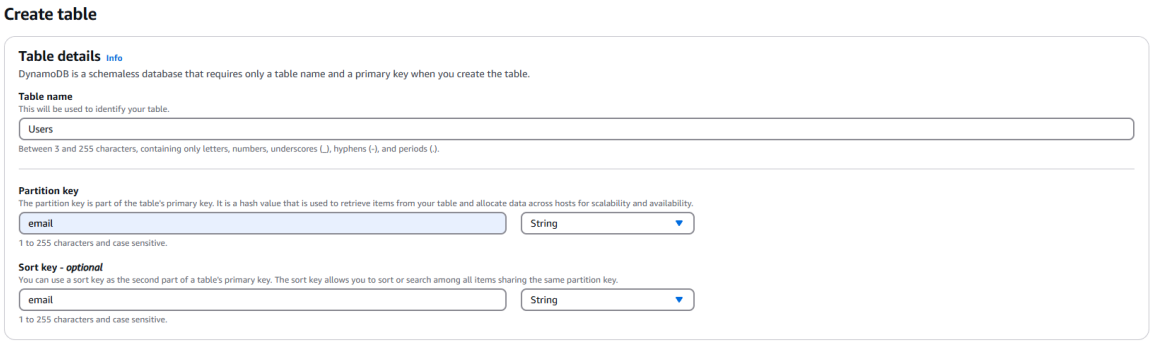
1.- Desde la consola de AWS acceder al servicio de DYNAMO DB



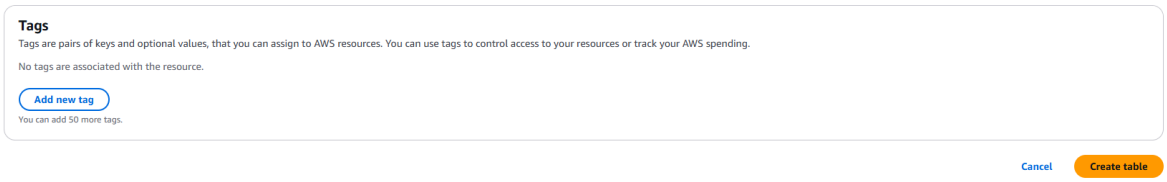
2.- Dentro del menú de tablas dar clic en crear tabla



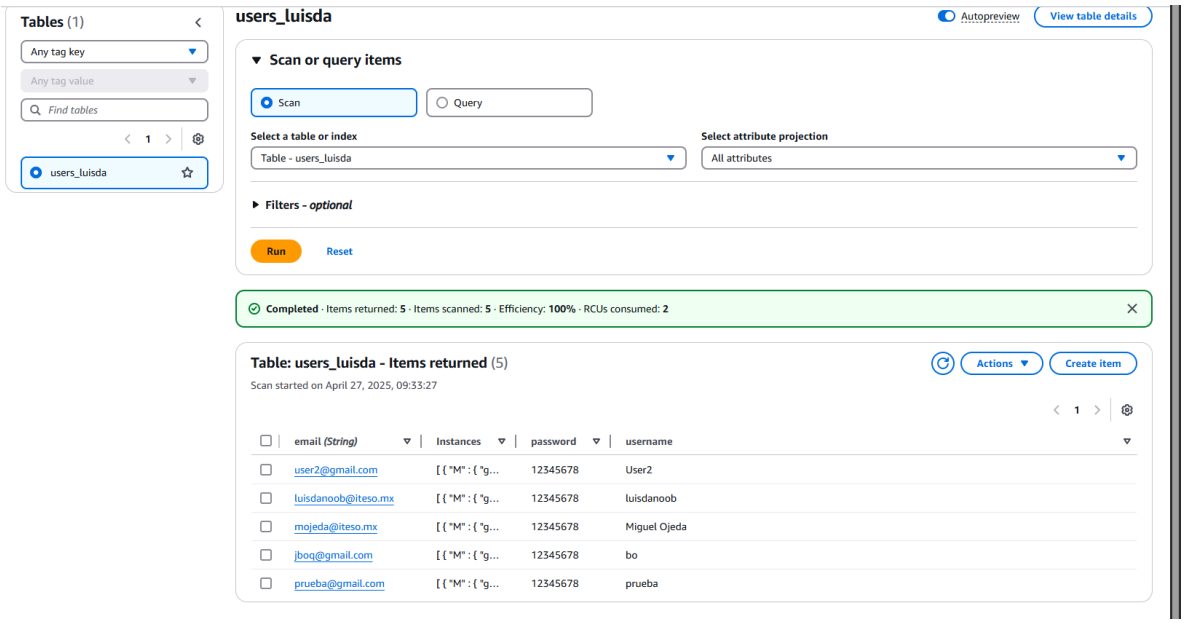
3.- Especificamos el nombre de la tabla, su llave primaria y su sort key



4.- Creamos la tabla

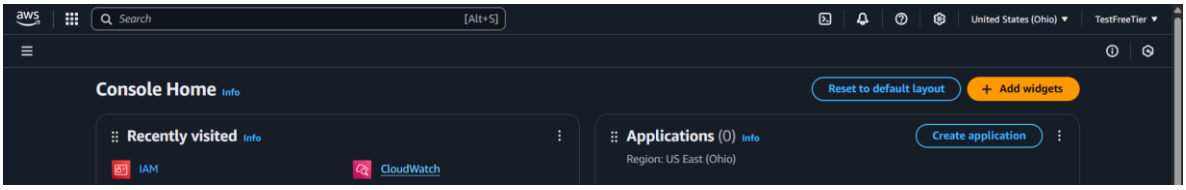


5.- Podemos acceder a los ítems de nuestra nueva tabla

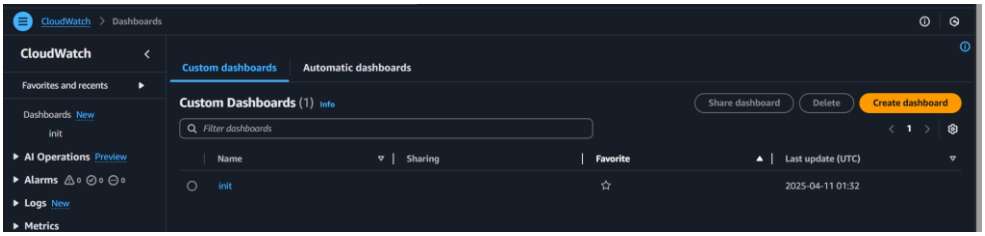


CLOUD WATCH:

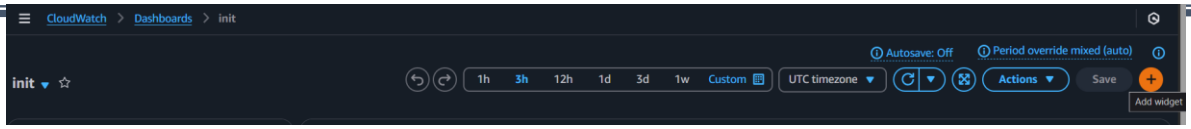
1.- Desde la consola de AWS acceder al servicio CloudWatch



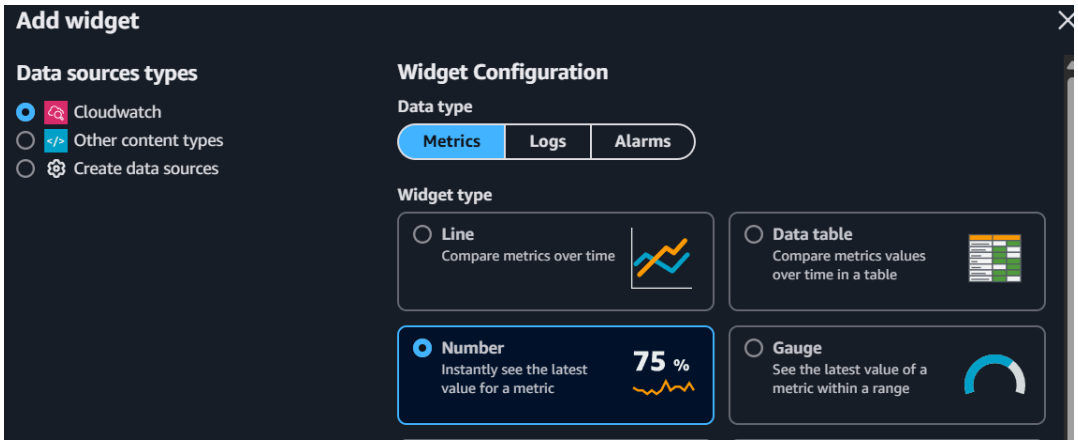
2.- Dar create DashBoard

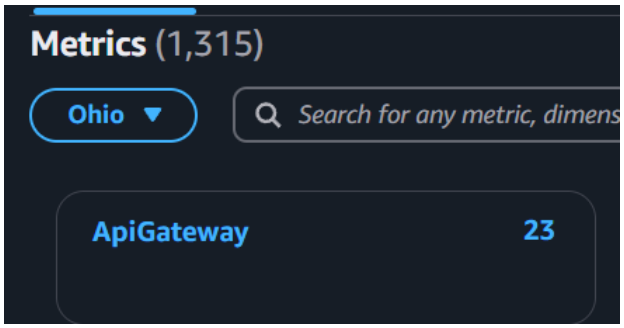


3.- Añadir métricas

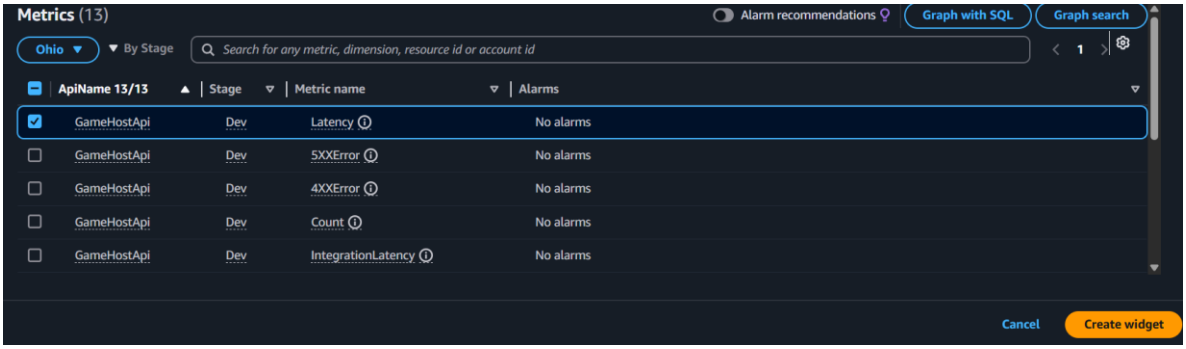


4.- Elegir la instancia y lo que se va a medir y el tipo de grafica

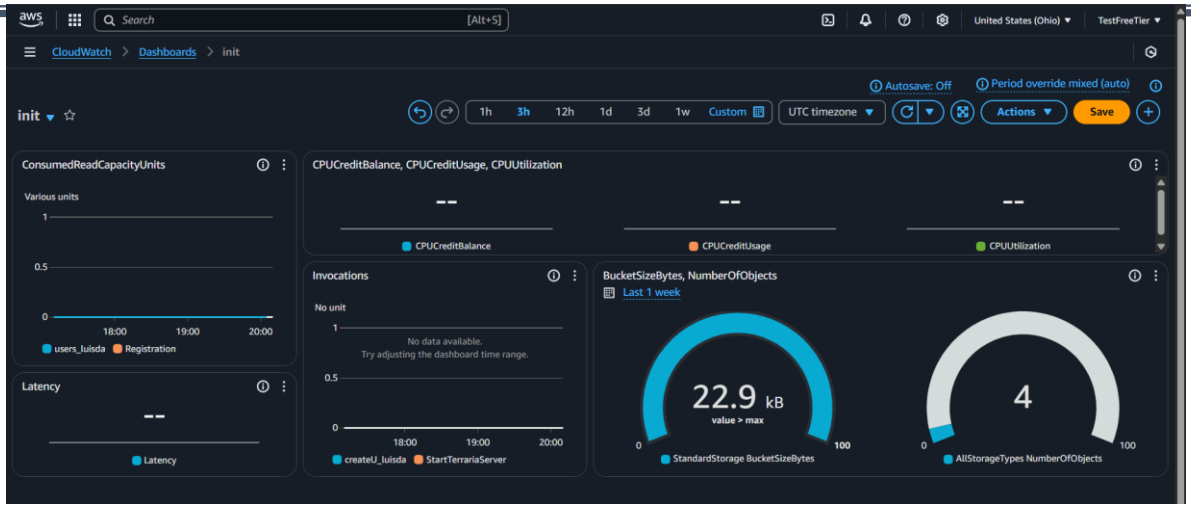




5.- Elegir métrica y creamos widget

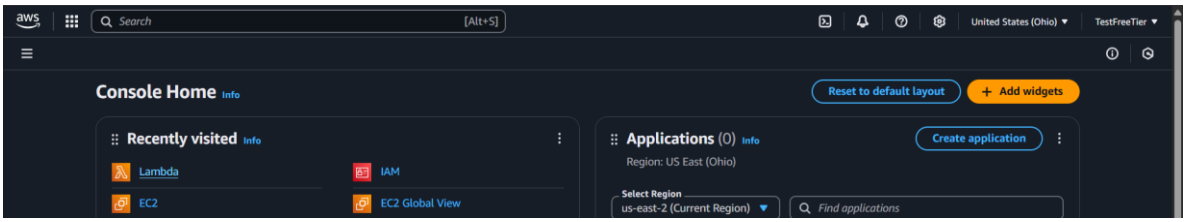


6.- Mostrar el dashboard

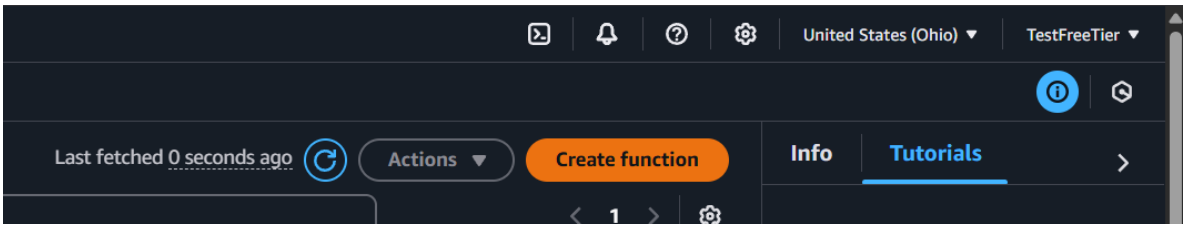


LAMBDA:

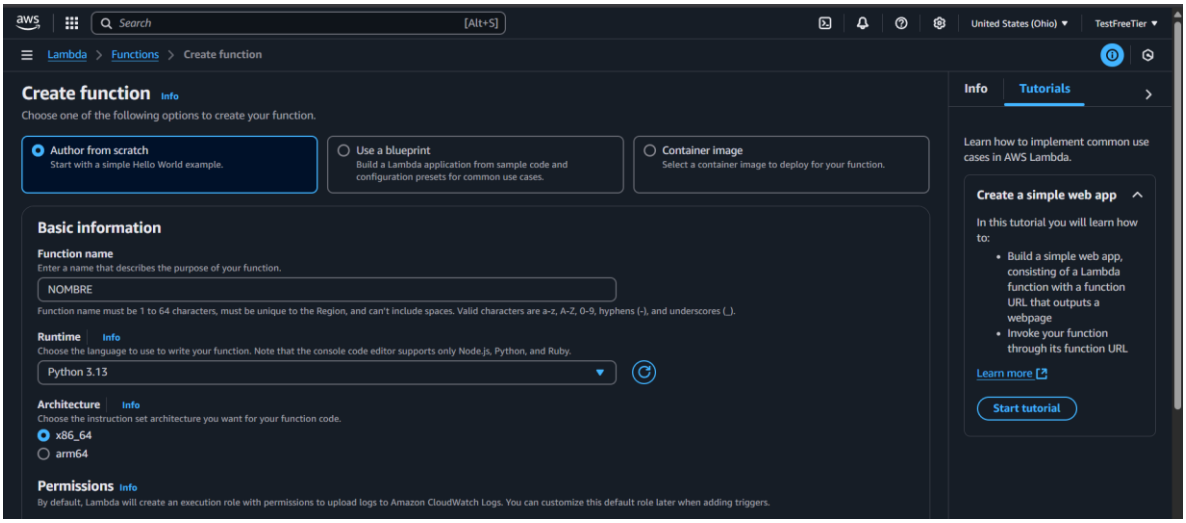
1.- Desde la consola de AWS acceder al servicio LAMBDA



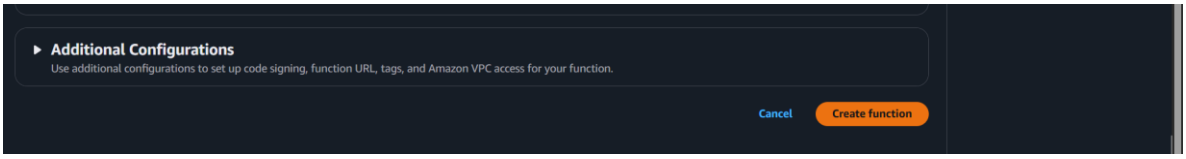
2.- Dentro del servicio LAMBDA crear una función nueva



3.- Dejamos el autor from scratch, el nombre que queramos de la función, usando Python 3.13, y con un rol nuevo



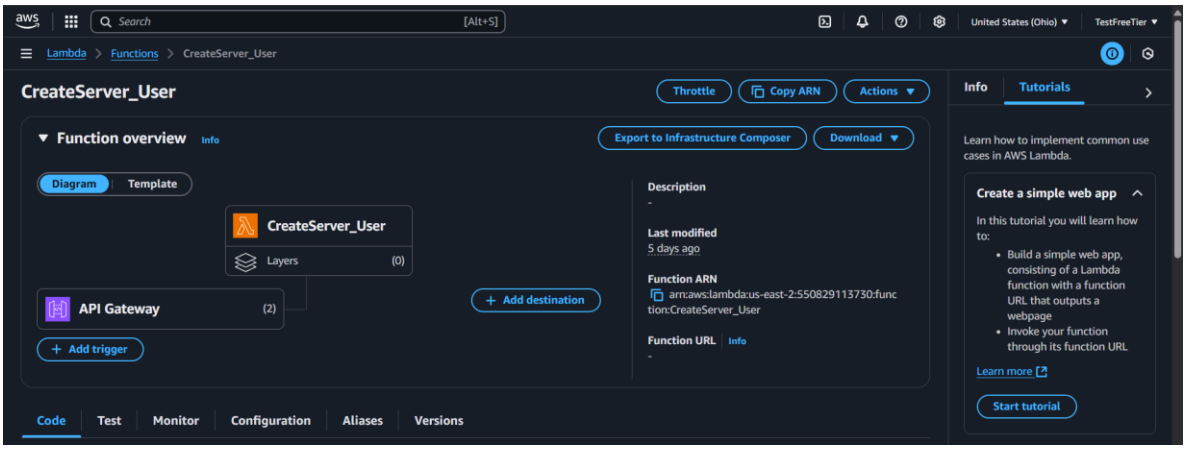
4.- Creamos la función:



5.- Funciones:

FUNCIÓN Crear Instancias desde lambda:

i) En el menú principal de nuestra función



ii) Código

```
import boto3
import time
import json
from botocore.exceptions import ClientError

dynamodb = boto3.resource('dynamodb') #conexión con DYNAMO
tabla = dynamodb.Table('users_luisda') #conexión con la tabla de DYNAMO

def lambda_handler(event, context):
    # Manejo de CORS preflight
    if event['httpMethod'] == 'OPTIONS':
        return {
            'statusCode': 200,
            'headers': {        # Estos headers nos sirven para permitir el
acceso desde el navegador al API desde cualquier dominio y evitar problemas
con los CORS
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            },
            'body': json.dumps('CORS preflight OK')
        }

    # Obtener el cuerpo de la solicitud (ahora el cuerpo es un JSON)
    body = json.loads(event["body"])

    if 'User' not in body: #Manejo de error en la solicitud
        return {
            'statusCode': 400,
            'headers': {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            },
            'body': json.dumps({'error': 'No se proporcionó un cuerpo en la
solicitud'})
        }

    email = body["User"]["email"] #obtenemos el email
    password = body["User"]["password"] #obtenemos el password
```

```
    #Manejamos error de falta de parámetros
    if not email or not password:
        return {
            'statusCode': 400,
            'headers': {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            },
            'body': json.dumps({'error': 'Faltan datos requeridos'})
        }

    game = body["game"] #Obtenemos el juego que se va a levantar
    try:
        response = tabla.get_item(Key={'email': email})
        if 'Item' not in response or response['Item']['password'] !=
password: #Validamos contraseña
            return {
                'statusCode': 400,
                'headers': {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "*",
                    "Access-Control-Allow-Methods": "OPTIONS,POST"
                },
                'body': json.dumps({'error': 'El usuario no existe o la
contraseña es incorrecta'})
            }

        #Dependiendo el juego se mapea con el ID de la imagen con el
servidor ya instalado
        ami_map = {
            "terraria": "ami-02af08be27104ae31"
        }

        #Si el juego no existe en nuestros servidores
        if game not in ami_map:
            return {
                'statusCode': 400,
                'headers': {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "*",
                    "Access-Control-Allow-Methods": "OPTIONS,POST"
                },
```

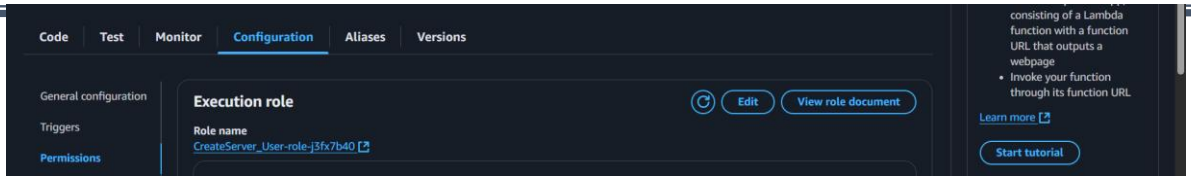
```
                'body': json.dumps({'error': f"Juego '{game}' no
soportado"})
            }


        nombre_instancia = f"Servidor-{game}-{int(time.time())}" #Creamos el
nombre del servidor
        ec2 = boto3.client('ec2') #Nos conectamos con EC2

        #Ejecutamos run_instances que crea las instancias
        response = ec2.run_instances(
            ImageId=ami_map[game], #Imagen
            InstanceType='t2.micro', #Tipo de instancia
            MinCount=1,
            MaxCount=1,
            KeyName='vockey', #LLave
            SecurityGroupIds=['sg-036644a87719e8cad'],
            SubnetId='subnet-02b8b5064491aec02',
            TagSpecifications=[{
                'ResourceType': 'instance',
                'Tags': [{'Key': 'Name', 'Value': nombre_instancia}]
            }]
        )

        #Obtenemos los parámetros de la instancia para mandarsela al usuario
        instance_id = response['Instances'][0]['InstanceId']
        ec2_resource = boto3.resource('ec2')
        instance = ec2_resource.Instance(instance_id)
        instance.wait_until_running()
        instance.load()
        ip = instance.public_ip_address

        # Actualizar la base de datos con la nueva instancia y asociar el
juego
        tabla.update_item(
            Key={'email': email},
            UpdateExpression='SET Instances = list_append(Instances,
:new_instance)',
            ExpressionAttributeValues={
                ':new_instance': [{
                    'instance_id': instance_id,
                    'game': game,  # Aquí guardamos el nombre del juego
junto con la instancia
```

```
                    'public_ip': ip
                }]
            }
        )

        #Regresamos la informción vital para el usuario
        return {
            "statusCode": 200,
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
                "Access-Control-Allow-Methods": "OPTIONS,POST",
                "Content-Type": "application/json"
            },
            "body": json.dumps({
                "game": game, #EL juego con el servidor
                "instance_id": instance_id, #El ID de la instancia
                "public_ip": ip #Su ip para poderse conectar
            })
        }

    #Manejamos exception de error con el cliente
    except ClientError as e:
        return {
            'statusCode': 500,
            'headers': {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "*",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            },
            'body': json.dumps({'error': 'Error en la base de datos',
'details': str(e)})
        }
```

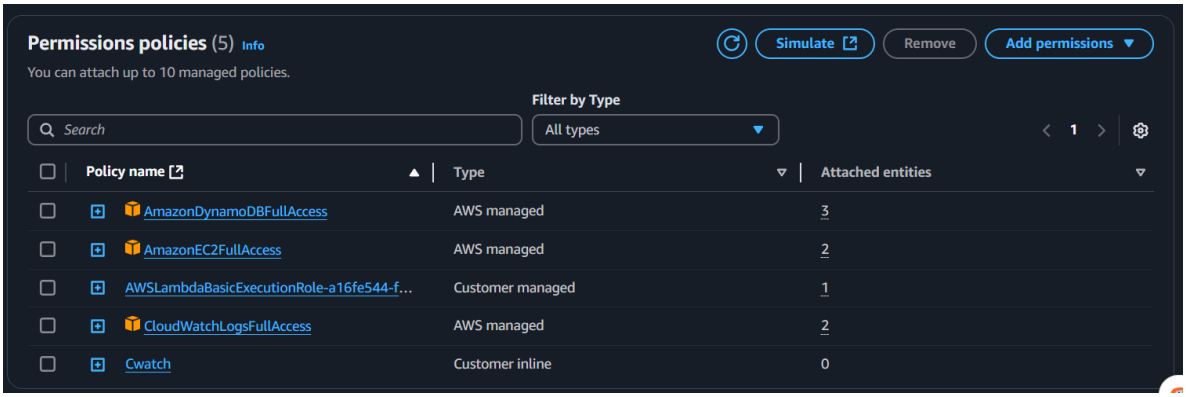
iii) Para el test

```
{
  "httpMethod": "POST",
  "body":
"{\"User\":{\"email\":\"user2@gmail.com\",\"password\":\"12345678\"},\"game\
":\"terraria\"}"
}
```

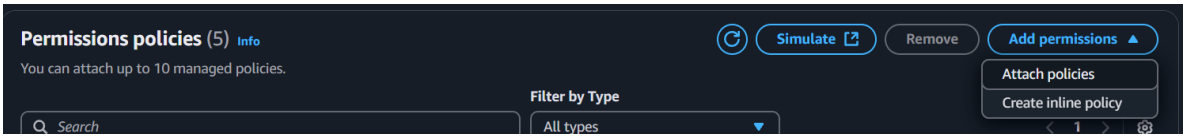
iv) Configurar permisos



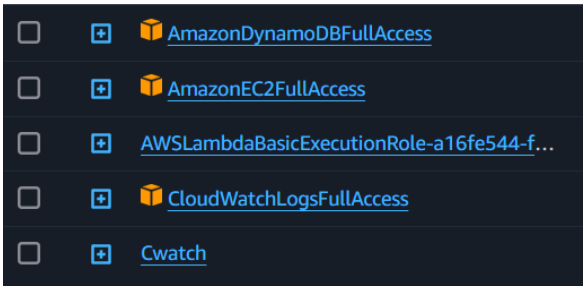
Agregar permisos desde IAM en el rol



Dar click en Add permissions y Attach policies

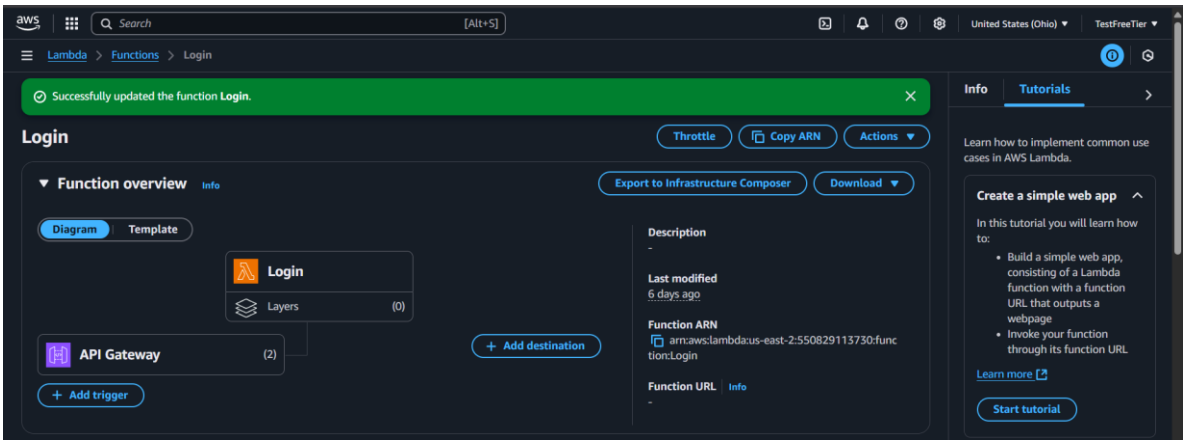


Agregar estos permisos:



FUNCIÓN Login:

i) Consola principal de nuestra función



ii) Código

```
import boto3
import json
from botocore.exceptions import ClientError
import time

# Inicializar recurso DynamoDB
dynamodb = boto3.resource('dynamodb')
tabla = dynamodb.Table('users_luisda')

# Cabeceras CORS comunes
cors_headers = {
    'Access-Control-Allow-Origin': '*',
    'Access-Control-Allow-Headers': 'Content-Type',
    'Access-Control-Allow-Methods': 'OPTIONS,POST'
}

def lambda_handler(event, context):
    start_time = time.time()  # Registrar el tiempo de inicio
    print("Evento recibido:", event)

    # Validar entrada
    if 'User' not in event:
        print("Falta el campo 'User'")
        return {
            'statusCode': 400,
            'headers': cors_headers,
            'body': json.dumps({'error': 'No se proporcionó el campo "User"
en la solicitud'})
```

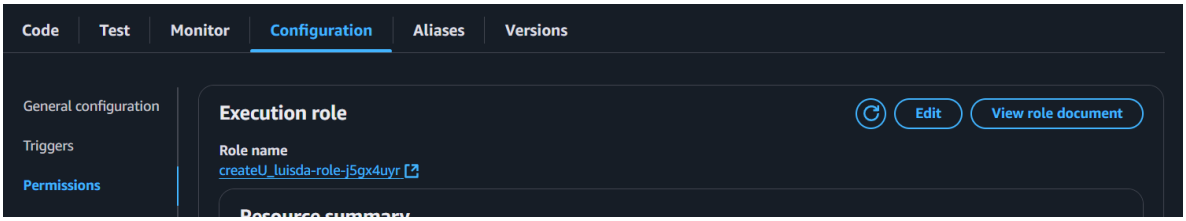
```
        }

    email = event["User"].get("email") #Obtenemos email
    password = event["User"].get("password") #Obtenemos password

    #Validamos que si existan
    if not email or not password:
        print("Faltan email o password")
        return {
            'statusCode': 400,
            'headers': cors_headers,
            'body': json.dumps({'error': 'Faltan datos requeridos: email o
password'})
        }

    try:
        print(f"Consultando usuario con email: {email}")
        response = tabla.get_item(Key={'email': email}) #Obtenemos el email
de la tabla
        print("Consulta a DynamoDB completada en", time.time() - start_time,
"segundos")

        #Sino encuentra el email
        if 'Item' not in response:
            print("Usuario no encontrado")
            return {
                'statusCode': 404,
                'headers': cors_headers,
                'body': json.dumps({'error': 'El usuario no existe'})
            }

        user_data = response['Item'] #Si encuentra el email regresa sus
datos
        print("Datos obtenidos del usuario:", user_data)

        if user_data.get('password') != password: #Ahora validamos si la
contraseña es correcta
            print("Contraseña incorrecta")
            return {
                'statusCode': 401,
                'headers': cors_headers,
                'body': json.dumps({'error': 'Contraseña incorrecta'})
            }
```

```
        print("Autenticación exitosa") #Validamo el login
        return {
            'statusCode': 200,
            'headers': cors_headers,
            #Regresamos la información personal del usuario para las
variables globales
            'body': json.dumps({
                'username': user_data.get('username'),
                'email': user_data.get('email'),
                'instances': user_data.get('Instances', []),
                'message': 'Usuario autenticado correctamente'
            })
        }

    #Manejamos excepciones de conexión

    except ClientError as e:
        print("Error al consultar DynamoDB:", str(e))
        return {
            'statusCode': 500,
            'headers': cors_headers,
            'body': json.dumps({'error': 'Error en la base de datos',
'details': str(e)})
        }

    except Exception as e:
        print("Error inesperado:", str(e))
        return {
            'statusCode': 500,
            'headers': cors_headers,
            'body': json.dumps({'error': 'Error inesperado', 'details':
str(e)})
        }
```

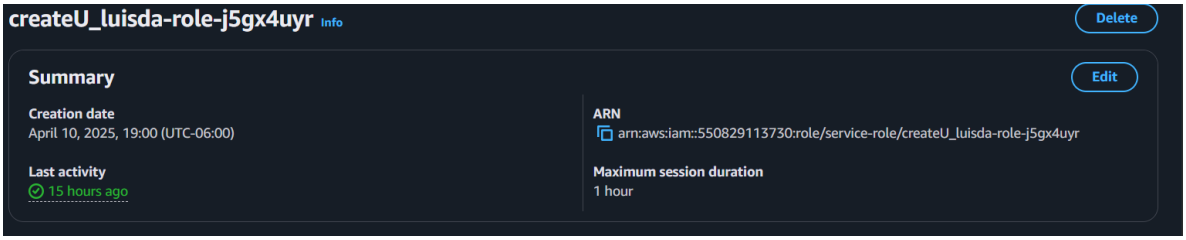
iii) Test

```
{
    "User": {
    "email": "user2@gmail.com",
    "password": "12345678"
  }
}
```

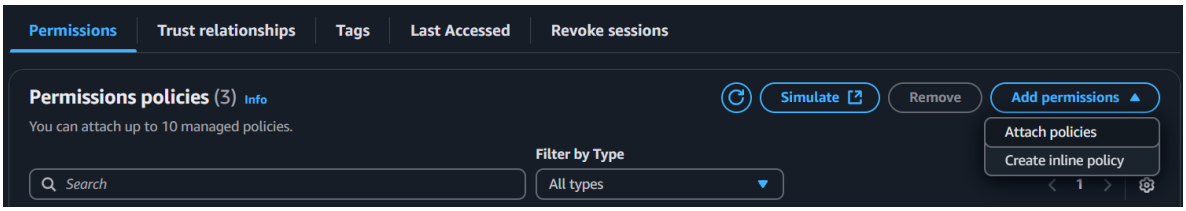
iv) Configurar permisos



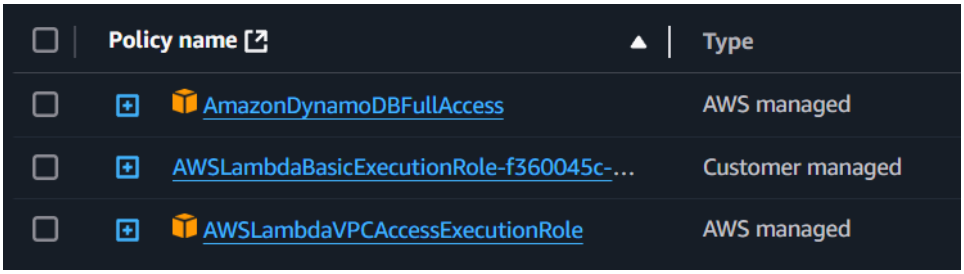
v) Ir al rol de IAM



vi) Agregar permisos y attach policies

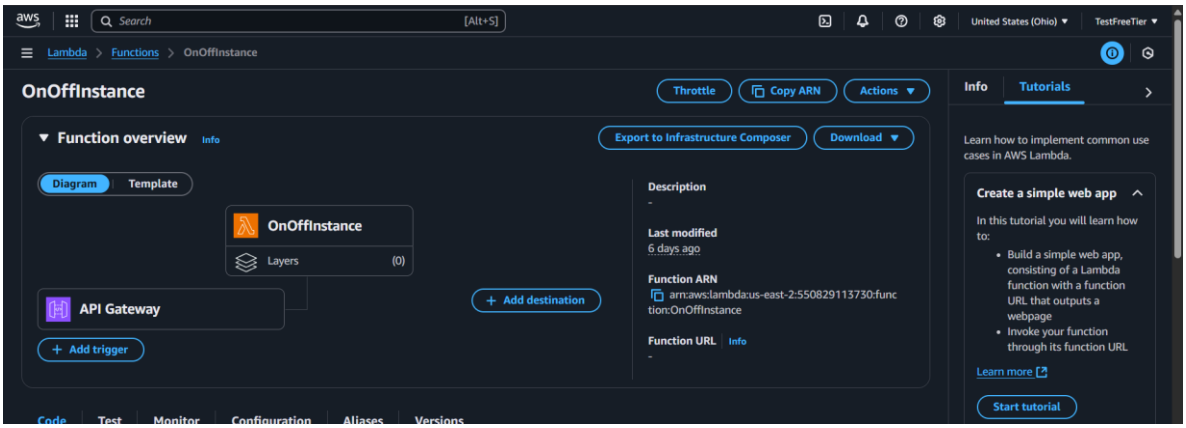


vii) Agregar estos permisos



FUNCIÓN Apagar y Prender Servidor

i) Menú principal de nuestra función



ii) Código

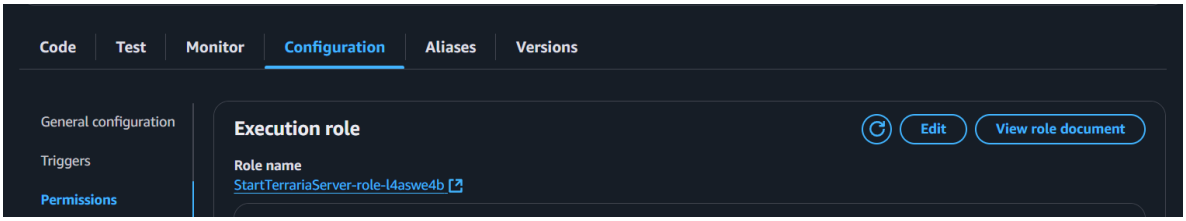
```
import boto3
import json

#Conexión con EC2
ec2 = boto3.client('ec2')

def lambda_handler(event, context):
    try:
        body = json.loads(event["body"])

        #Si el parametro instance no esta en el JSON
        if 'Instance' not in body:
            return {
                'statusCode': 400,
                'headers': {
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({'error': 'No se proporcionó una
instancia'})
            }

        instance_id = body["Instance"]["id"] #Obtenemos el id de la
instancia
        status = body["Instance"]["status"] # Obtenemos su estatus al que se
va a cambiar

        #Dependiendo de su estatus start o stop instance
        if status == "ON":
```

```
            ec2.start_instances(InstanceIds=[instance_id])
            message = f'Instancia {instance_id} encendida'
        elif status == "OFF":
            ec2.stop_instances(InstanceIds=[instance_id])
            message = f'Instancia {instance_id} apagada'
        else:
            return {
                'statusCode': 400,
                'headers': {
                    'Access-Control-Allow-Origin': '*'
                },
                'body': json.dumps({'error': 'Estado inválido. Usa ON u
OFF'})
            }

        #Regresamos si prendimos o apagamos la instancia
        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': '*'
            },
            'body': json.dumps({'message': message})
        }

    #Mala conexión

    except Exception as e:
        return {
            'statusCode': 500,
            'headers': {
                'Access-Control-Allow-Origin': '*'
            },
            'body': json.dumps({'error': str(e)})
        }
```

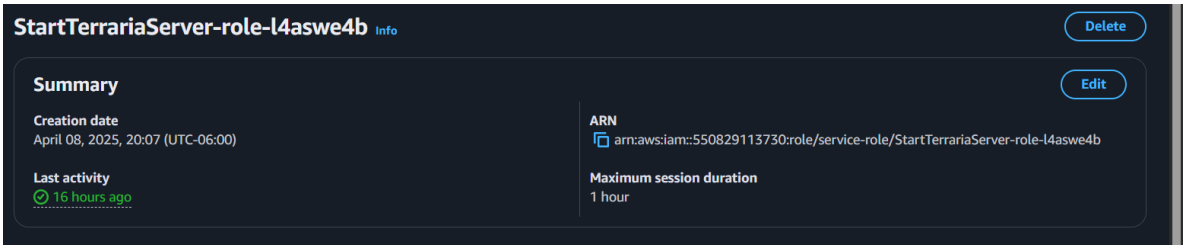
iii) Test

```
{
    "body":
    "{\"Instance\":{\"id\":\"i-0415d3cfdae1048f1\",\"status\":\"OFF\"}}"
}
```

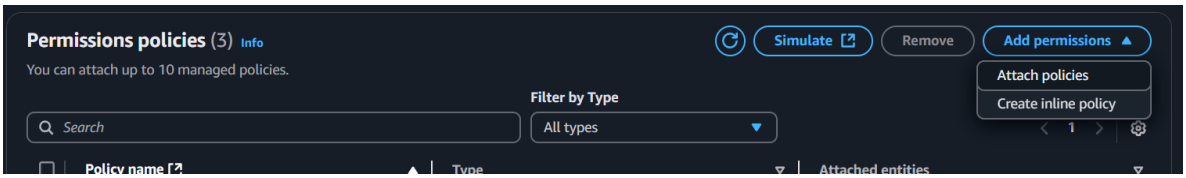
iv) En la configuración ir a permisos



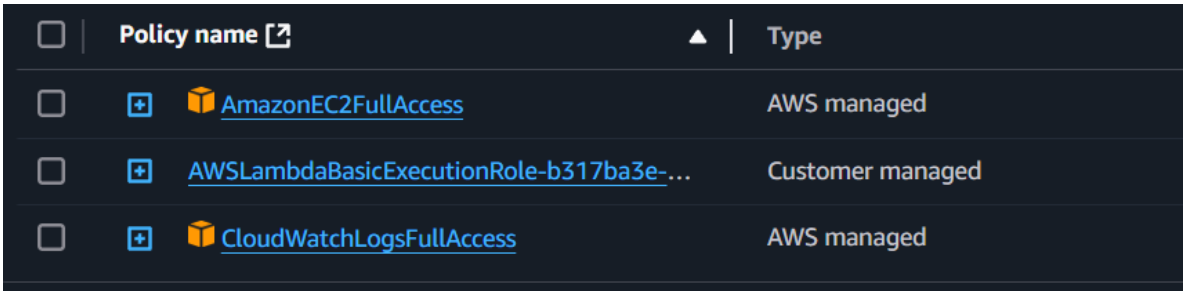
v) Ir al rol de IAM



vi) Add permissions y Attach policies

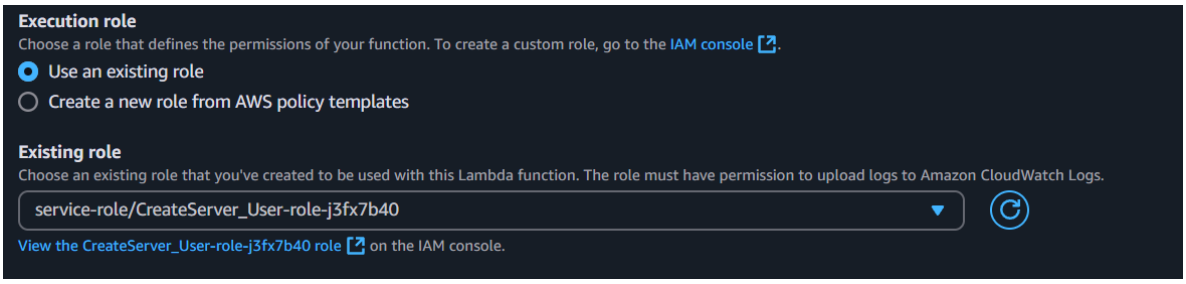


vii) Añadir estos pemisos

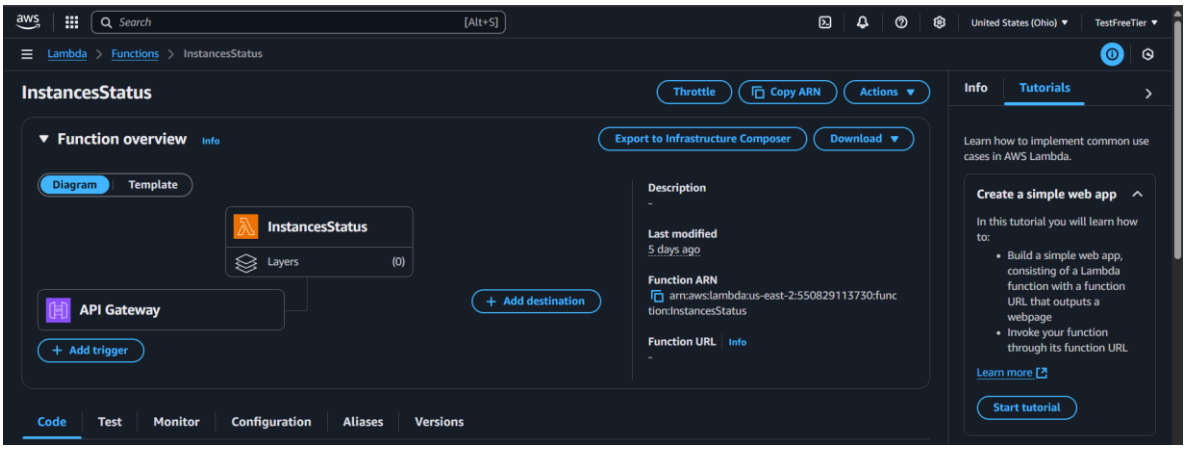


FUNCIÓN Estatus de la instancia

i) ESTA FUNCIÓN USA EL MISMO ROL CreateServer_User-role



ii) Menú principal de nuestra función



iii) Código

```
import boto3
import json
import time
from datetime import datetime, timedelta
from botocore.exceptions import ClientError

# Clientes de AWS
dynamodb = boto3.resource('dynamodb') #DYNAMO
tabla = dynamodb.Table('users_luisda') #Tabla de dynamo
ec2 = boto3.client('ec2') #EC2
cloudwatch = boto3.client('cloudwatch') #nos conectamos a CLOUD WATCH para
métricas

# Función para obtener promedio de CPU
def get_metric_average(metric_name, namespace, dimensions):
    end_time = datetime.utcnow()
```

```
    start_time = end_time - timedelta(hours=12) #Especificamos un tiempo
(ultimas 12 horas)

    #Llamamos a CloudWatch para que nos regrese las metricas deseadas
    response = cloudwatch.get_metric_statistics(
        Namespace=namespace,
        MetricName=metric_name,
        Dimensions=dimensions,
        StartTime=start_time,
        EndTime=end_time,
        Period=300,
        Statistics=['Average']
    )

    #filtramos los puros datos de la respuesta
    datapoints = response.get('Datapoints', [])
    if not datapoints:
        return None

    #Retornamos los datos
    return round(sorted(datapoints, key=lambda x: x['Timestamp'],
reverse=True)[0]['Average'], 2)

def lambda_handler(event, context):
    try:
        body = json.loads(event['body'])
        user = body.get('User') #Obtenemos el usuario

        #Si no estamos logeados
        if not user:
            return {
                "statusCode": 400,
                "body": json.dumps({"error": "No se proporcionó un usuario
válido"}),
                "headers": {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type",
                    "Access-Control-Allow-Methods": "OPTIONS,POST"
                }
            }

        email = user.get("email") #Obtenemos el email
        password = user.get("password") #Obtenemos password
```

```
        #Validamos la existencia del email y password
        if not email or not password:
            return {
                'statusCode': 400,
                'body': json.dumps({'error': 'Faltan datos requeridos'}),
                'headers': {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type",
                    "Access-Control-Allow-Methods": "OPTIONS,POST"
                }
            }

        response = tabla.get_item(Key={'email': email}) #Obtenemos si el
usuario si esta en la tabla
        if 'Item' not in response or response['Item']['password'] !=
password: #Validamos si la contraseña es correcta y el usuario existe
            return {
                'statusCode': 400,
                'body': json.dumps({'error': 'El usuario no existe o la
contraseña es incorrecta'}),
                'headers': {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type",
                    "Access-Control-Allow-Methods": "OPTIONS,POST"
                }
            }

        # Obtenemos las instancias del usuario
        isinstances = response['Item'].get('Instances', [])
        if not isinstances:
            return {
                'statusCode': 400,
                'body': json.dumps({'error': 'No hay instancias asociadas al
usuario'}),
                'headers': {
                    "Access-Control-Allow-Origin": "*",
                    "Access-Control-Allow-Headers": "Content-Type",
                    "Access-Control-Allow-Methods": "OPTIONS,POST"
                }
            }
```

```
        instance_ids = [inst['instance_id'] for inst in isinstances]
#Obtenemos su id
        response = ec2.describe_instances(InstanceIds=instance_ids)

        instances_info = []
        for reservation in response['Reservations']:
            for instance in reservation['Instances']:
                instance_id = instance['InstanceId']
                game_name = next((inst['game'] for inst in isinstances if
inst['instance_id'] == instance_id), 'No Game') #Obtenemos el servidor

                # Obtener rendimiento CPU
                cpu_utilization = get_metric_average(
                    metric_name='CPUUtilization',
                    namespace='AWS/EC2',
                    dimensions=[{'Name': 'InstanceId', 'Value':
instance_id}]
                )

                #Obtenemos la información completa de la instancia
                info = {
                    'InstanceId': instance_id,
                    'game': game_name,
                    'PublicIpAddress': instance.get('PublicIpAddress', 'No
Public IP'),
                    'State': instance['State']['Name'],
                    'cpuUtilization': f"{cpu_utilization}%" if
cpu_utilization is not None else 'No data'
                }
                instances_info.append(info)

        #Si todo función regresamos 200 y la información de todas las
instancias del usuario
        return {
            'statusCode': 200,
            'headers': {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            },
            'body': json.dumps({'instances': instances_info})
        }
```

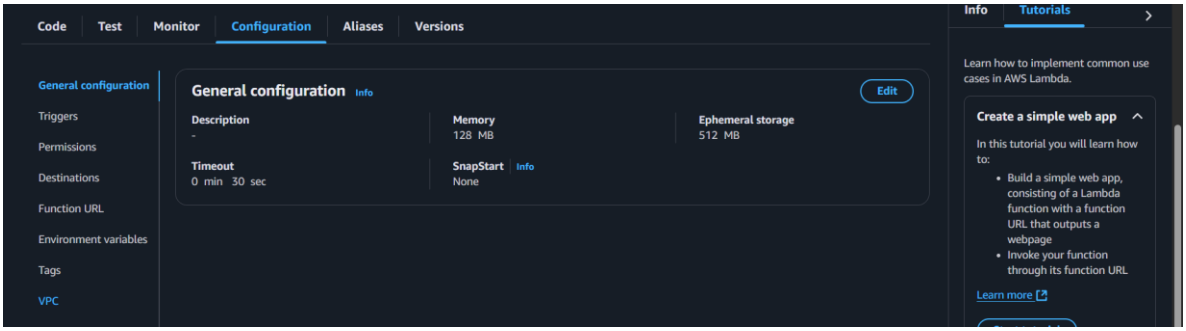
```
    #Excepción de conexion con DYNAMO
    except ClientError as e:
        return {
            'statusCode': 500,
            'body': json.dumps({'error': 'Error en la base de datos',
'details': str(e)}),
            'headers': {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            }
        }

    #Error de conexión
    except Exception as e:
        return {
            "statusCode": 500,
            "body": json.dumps({"error": str(e)}),
            "headers": {
                "Access-Control-Allow-Origin": "*",
                "Access-Control-Allow-Headers": "Content-Type",
                "Access-Control-Allow-Methods": "OPTIONS,POST"
            }
        }
```

iv) Test

```
{
  "body":
    "{\"User\":{\"email\":\"user2@gmail.com\",\"password\":\"12345678\"}}"
```

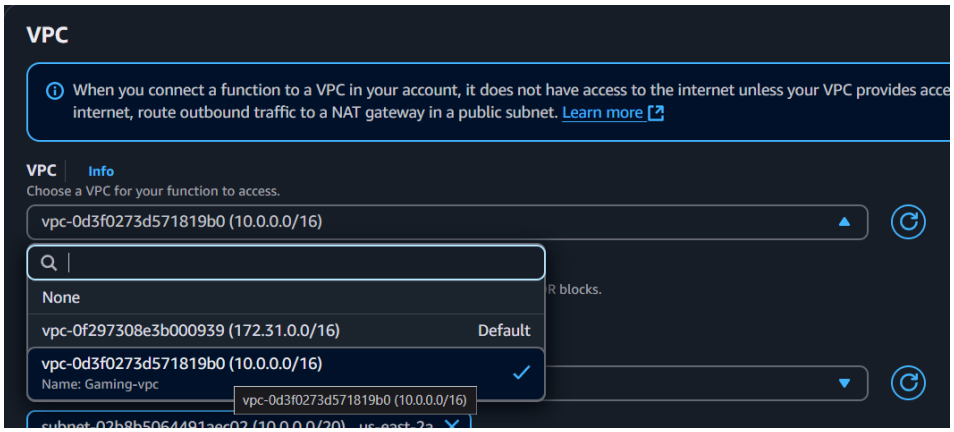
6.- Conectar funciones lambda con el VPC subnet privada

i.- En la configuración de la función irnos al apartado del VPC

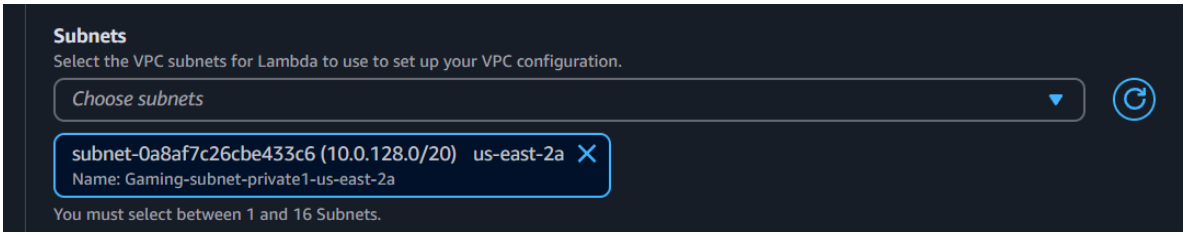


ii.- Dar click a editar

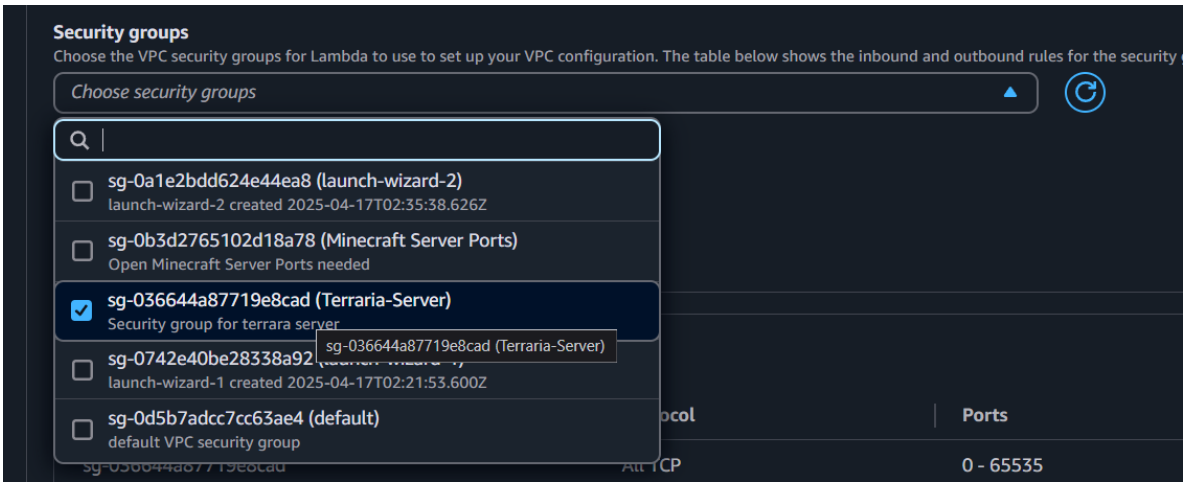
iv.- Elegimos el VPC



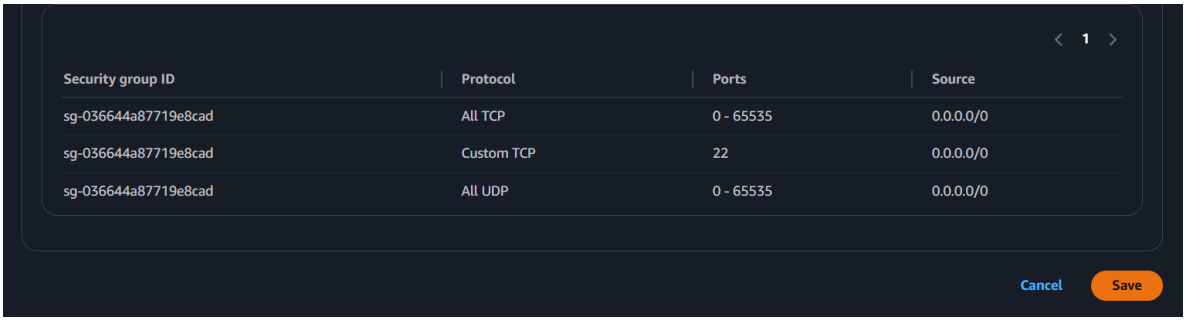
v.- Agregamos la subnet privada del VPC



vi.- Elegir el security group establecido

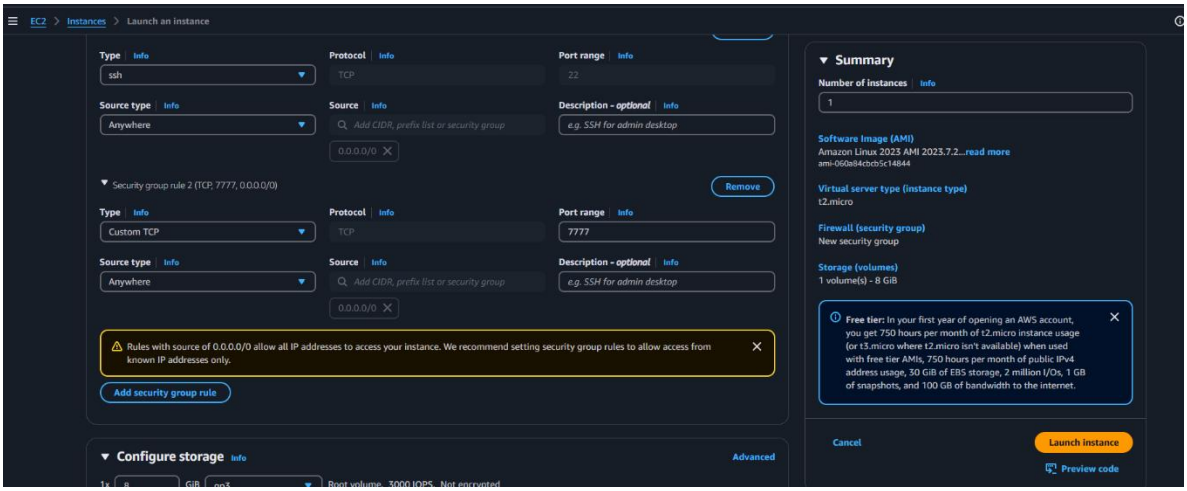


vii.- Damos click en save



AMAZON EC2:

1. Vamos a levantar un EC2 t2.micro, con amazon Linux y en las reglas de seguridad, aparte de permitir ssh, vamos a permitir la entrada el puerto 7777 TCP (Puerto que se necesita para el servidor). Esta instnacia se pondra en el VPC creado anteriormente, en la subnet publica



2. Script de creación y hacer que el servidor funcione, aunque se reinicie

```
# descargamos los componentes necesarios
sudo yum update -y
sudo yum install -y unzip wget screen tmux libcurl


#descargamos el servidor y descomprimimos
wget  https://terraria.org/api/download/pc-dedicated-server/terraria-server-1449.zip

unzip terraria-server-1449.zip

# entramos al archivo descomprimido
cd 1449/Linux


#damos permisos de ejecucion a los archivos necesarios
chmod +x TerrariaServer.bin.x86_64


#Corremos y creamos un mundo
./TerrariaServer.bin.x86_64
```

```
#new World
n

#small
1

#classic
1

#Random
1

#Name
TerrariaEC2

#enter para llave random

#Cerramos servidor con CRTL+C
# vamos a hacer un serverconfig file para que no tengamos que volver a hacer el mundo cada vez que reiniciamos el servidor
nano serverconfig.txt

#codigo a escribir adentro
    # Nombre del mundo (usa el nombre exacto del archivo .wld)
    world=/home/ec2-user/.local/share/Terraria/Worlds/TerrariaEC2.wld

# guardamos el achivo y salimos de nano con CTRL+X, Y, ENTER


# ahora vamos a generar un script para que levante el servidor automaticamente
cd /home/ec2-user/1449/Linux
nano startserver.sh


# en si solamente corre el servidor pero con el archivo de configuracion que hicimos antes
    #!/bin/bash
    cd /home/ec2-user/1449/Linux
    export TERM=xterm
```

```
    ./TerrariaServer.bin.x86_64              -config              /home/ec2-
user/1449/Linux/serverconfig.txt
# guardamos el achivo y salimos de nano con CTRL+X, Y, ENTER


#damos permisos de ejecucion al script
chmod +x startserver.sh

# ahora vamos a levantar el servicio para que el servidor se inicie
automaticamente al reiniciar la instancia
sudo nano /etc/systemd/system/terraria.service

# lo que vas a escribir adentro

    [Unit]
    Description=Servidor de Terraria
    After=network.target

    [Service]
    User=ec2-user
    WorkingDirectory=/home/ec2-user/1449/Linux
    ExecStart=/home/ec2-user/1449/Linux/startserver.sh
    Restart=on-failure
    RestartSec=10
    StandardOutput=journal
    StandardError=journal

    [Install]
    WantedBy=multi-user.target

# guardamos el achivo y salimos de nano con CTRL+X, Y, ENTER

#ahora agregamos el servicio a systemd para que lo reconozca y lo inicie al
reiniciar la instancia
sudo systemctl daemon-reexec
sudo systemctl daemon-reload
sudo systemctl enable terraria.service
sudo systemctl start terraria.service
```

```
# revisamos el estado del servicio
sudo systemctl status terraria.service
```

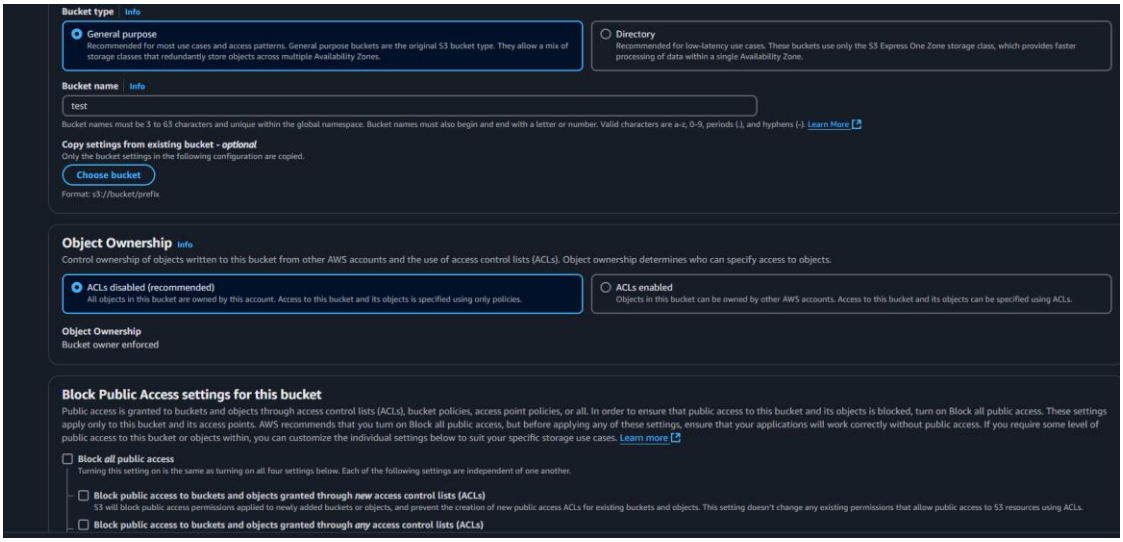
3. Revisamos el estatus

```
[root@ip-172-31-9-144 Linux]# sudo systemctl status terraria
● terraria.service - Servidor de Terraria
     Loaded: loaded (/etc/systemd/system/terraria.service; enabled; preset: disabled)
     Active: active (running) since Sat 2025-04-26 02:27:05 UTC; 5s ago
   Main PID: 4912 (startserver.sh)
      Tasks: 11 (limit: 1111)
     Memory: 514.3M
        CPU: 5.920s
     CGroup: /system.slice/terraria.service
             ├─4912 /bin/bash /home/ec2-user/1449/Linux/startserver.sh
             └─4914 ./TerrariaServer.bin.x86_64 -config /home/ec2-user/1449/Linux/serverconfig.txt

Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Settling liquids 39%
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Settling liquids 40%
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Settling liquids 41%
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Settling liquids 44%
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Settling liquids 48%
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Settling liquids 50%
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Terraria Server v1.4.4.9
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Listening on port 7777
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: Type 'help' for a list of commands.
Apr 26 02:27:11 ip-172-31-9-144.us-east-2.compute.internal startserver.sh[4914]: : Server started
```

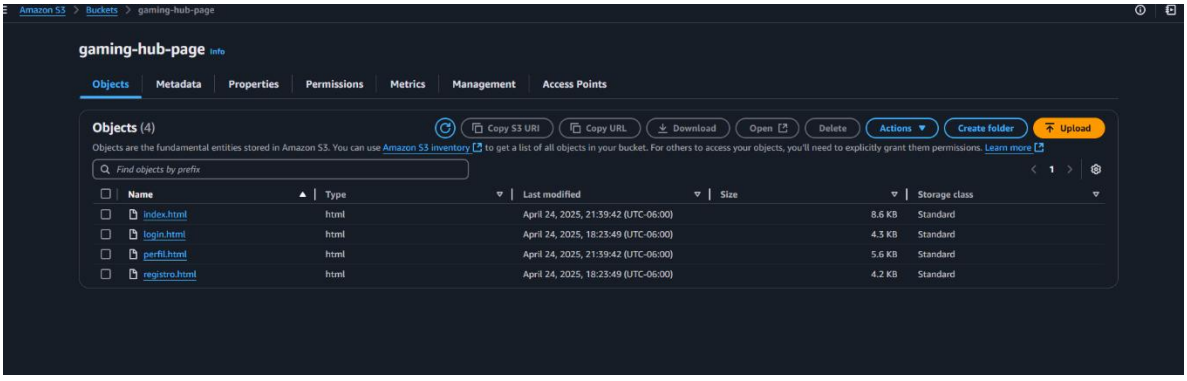
4. Ahora podemos entrar al servidor sin problemas

AMAZON S3:

1. Generamos un bucket eliminando la prohibición a TODO acceso público de esta bucket, ya que va a ser referenciada por Cloud Front



2. Metemos los archivos html al bucket

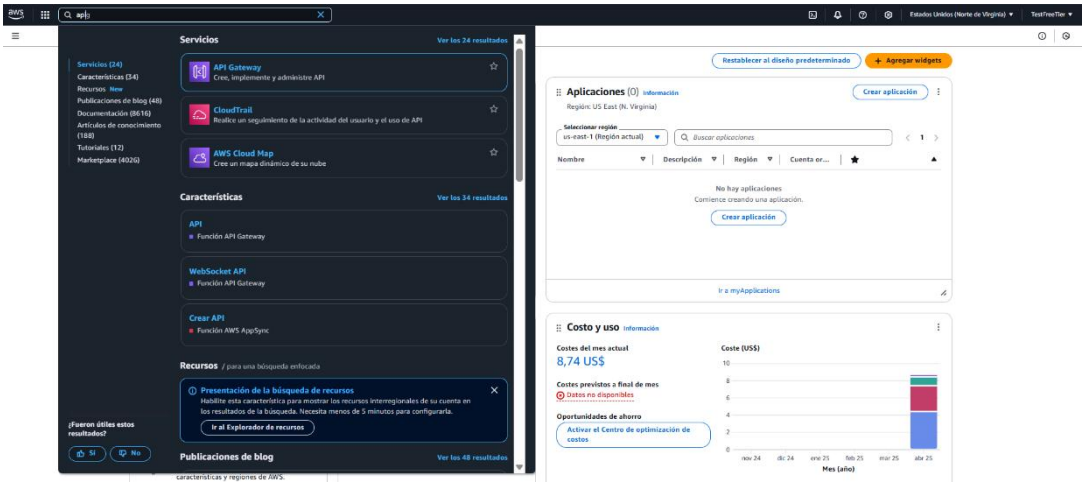


3. Nos metemos a properties y configuramos el static web hosting para el EC2

API GATEWAY:

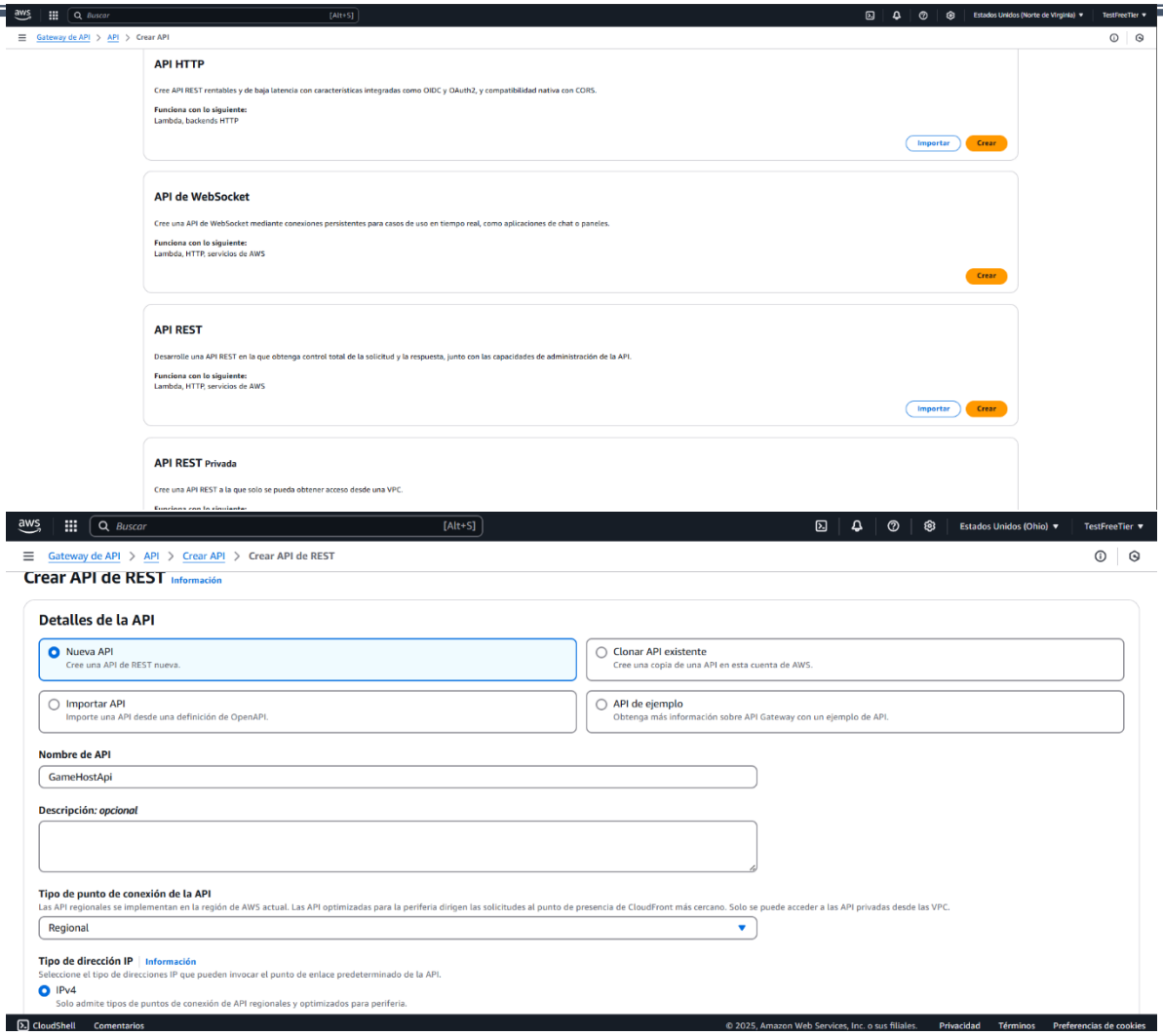
1. Crear API Gateway

- En la barra de búsqueda de AWS buscar API Gateway.



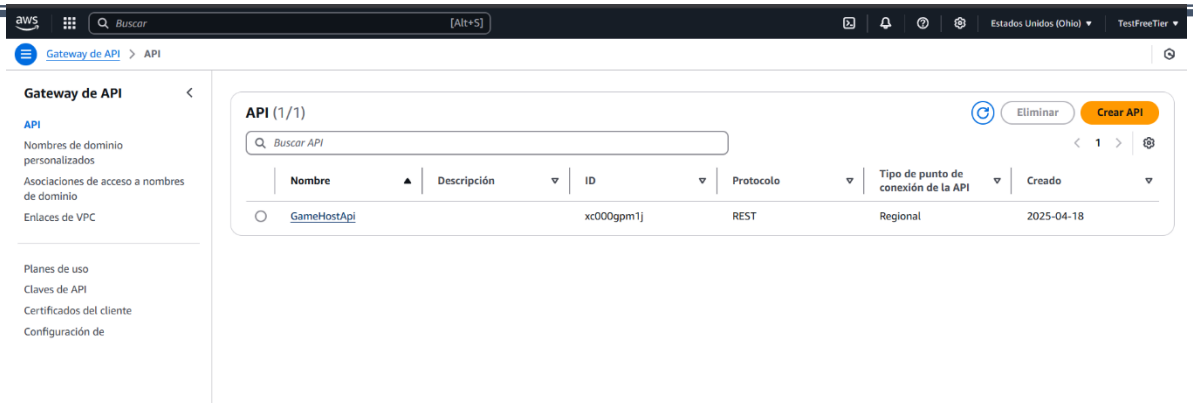
- Entra a **API Gateway**, selecciona **Crear API.**

Elige **REST API pública**, Poner en nombre: GameHostApi, tipo Regional, y crea la API.

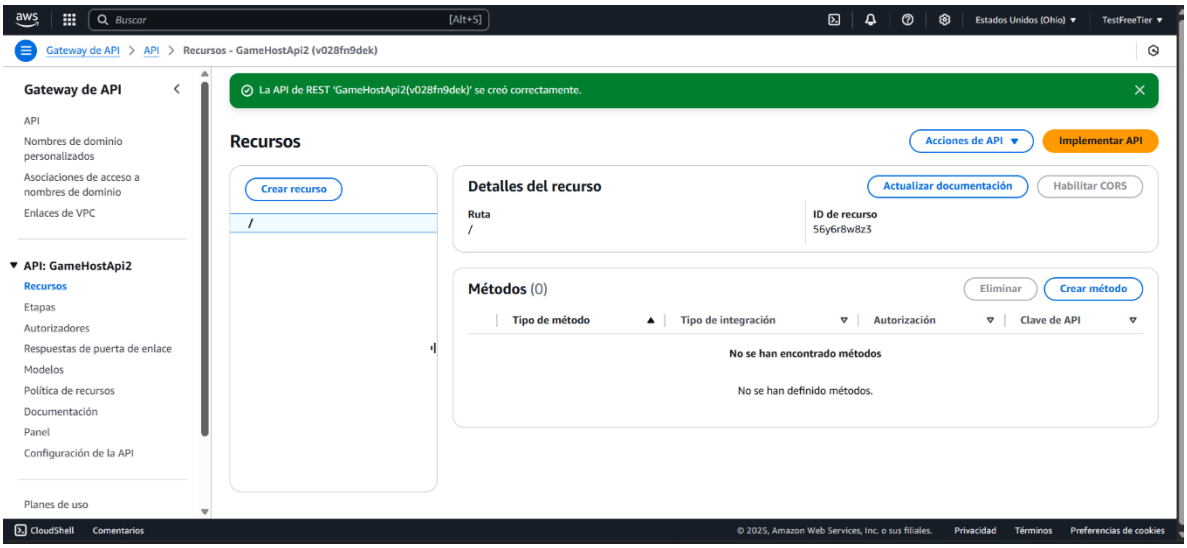


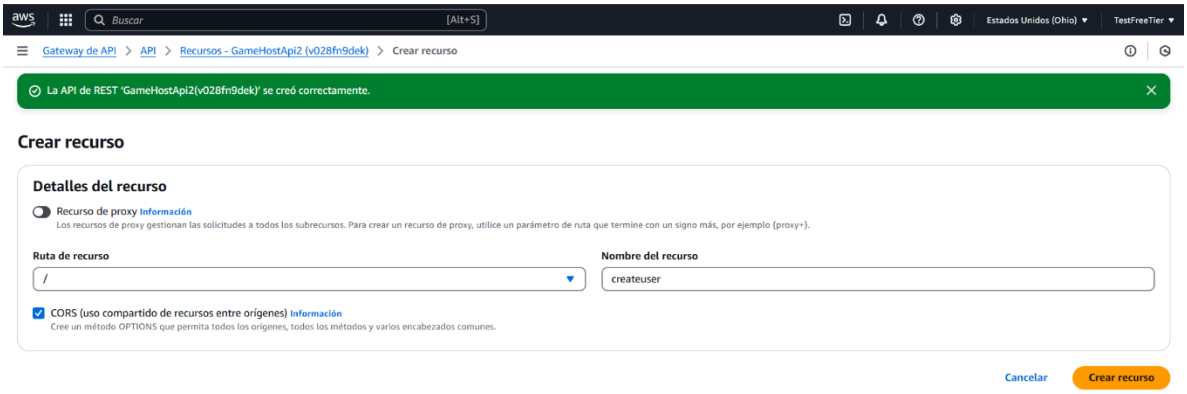
2. Crear Recursos y Métodos

- En nuestra API GameHostApi, dar clic derecho en para configurarla, podremos crear recursos y añadir configuraciones adicionales.

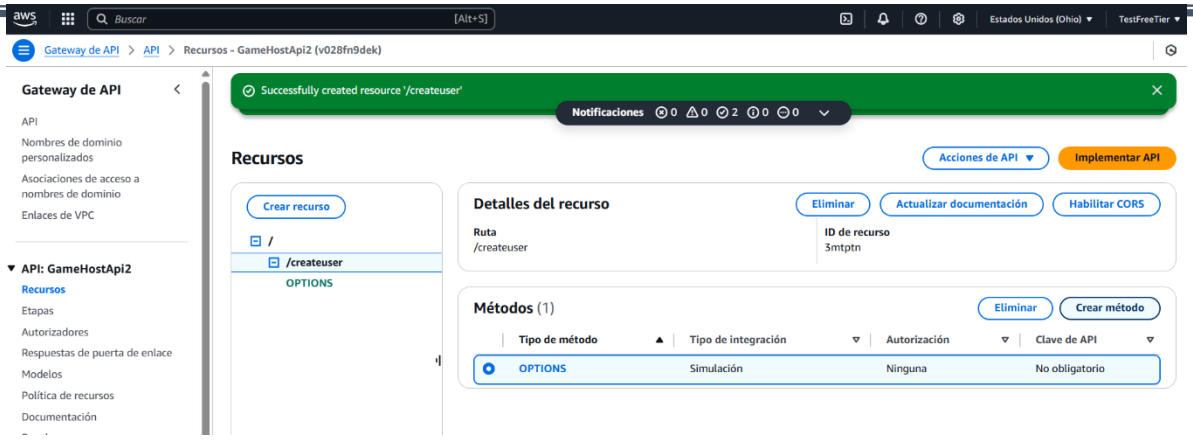


- Damos click en **Crear recurso** (le asignamos el nombre /createuser), habilita **CORS** si sale la opción, y crea.

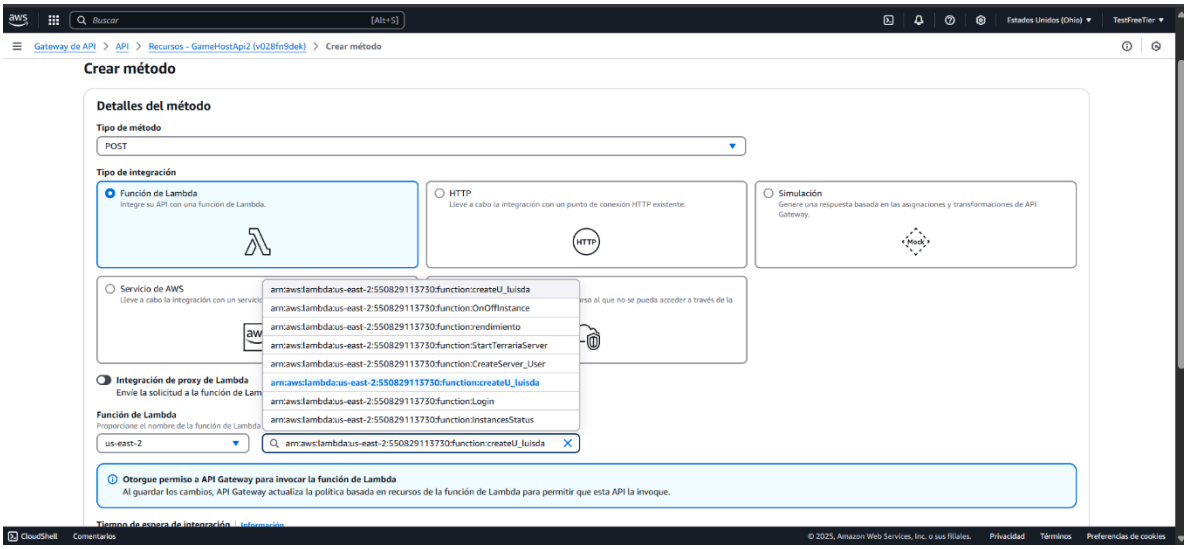




- Luego en /createuser, damos click en Crear método en el apartado **Métodos.**



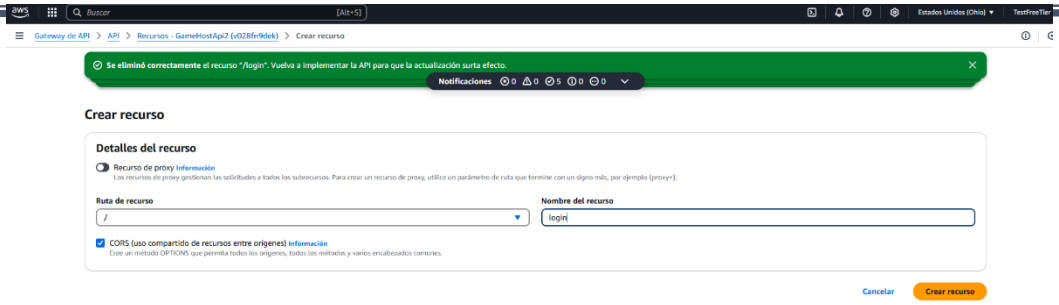
- Creamos un método de tipo post, lo integramos con Función de Lambda y especificamos la función de lambda para crear usuario, en este caso CreateU_Luisda. Aqui por como manejamos nuestra función lambda, no usamos Lambda Proxy porque mandaremos el JSON directo y sin envoltorio, lo mismo en login.



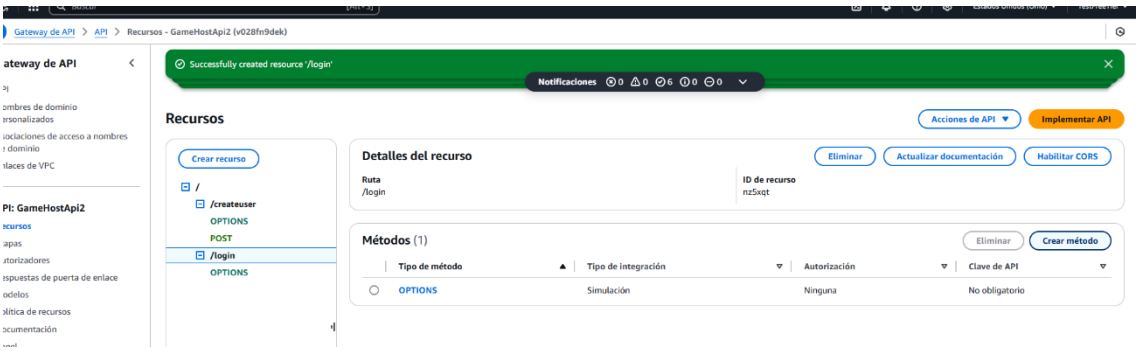
- **Repite** este proceso para:
  - /login → vinculado a Login
  - /onoff → vinculado a OnOffInstance
  - /createserver → vinculado a CreateServer_User
  - /instancestatus → vinculado a InstancesStatus.

(todos usando POST y Lambda Proxy menos en login)

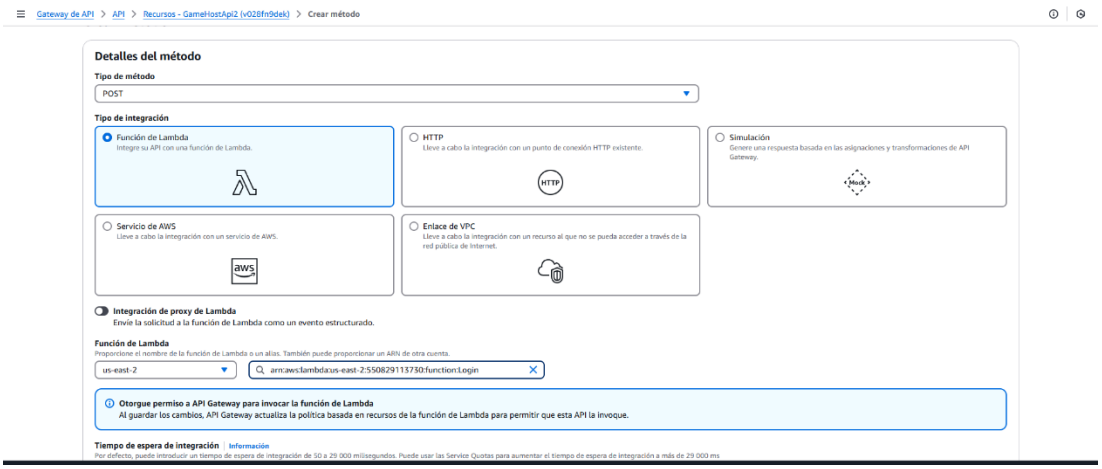
- Proceso para login: Creamos recurso /login habilitando CORS.



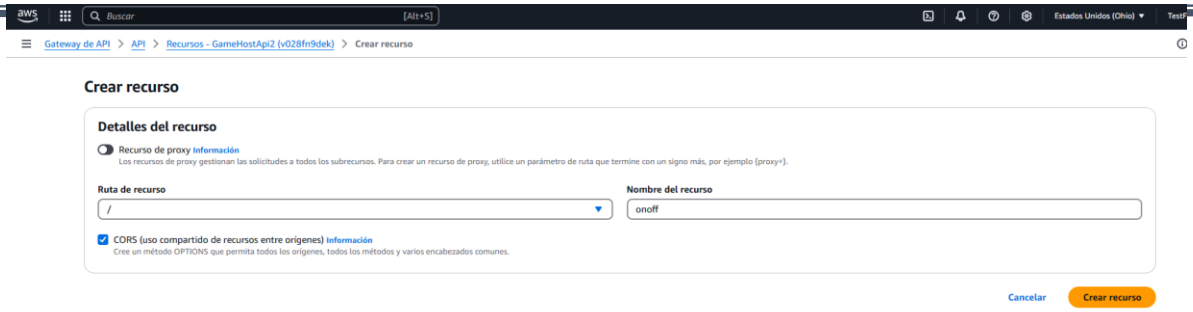
- Damos Click en crear método para login.



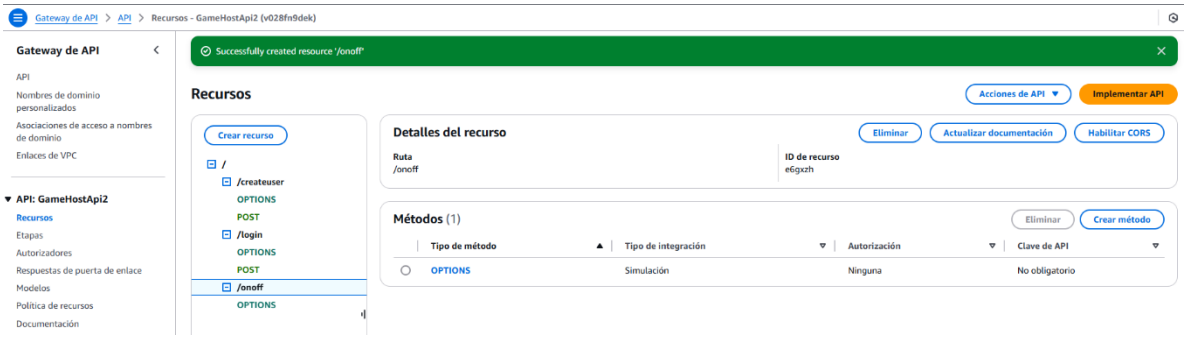
- Tipo Post, integración con Lambda y lo ligamos a la función Login, creamos método.



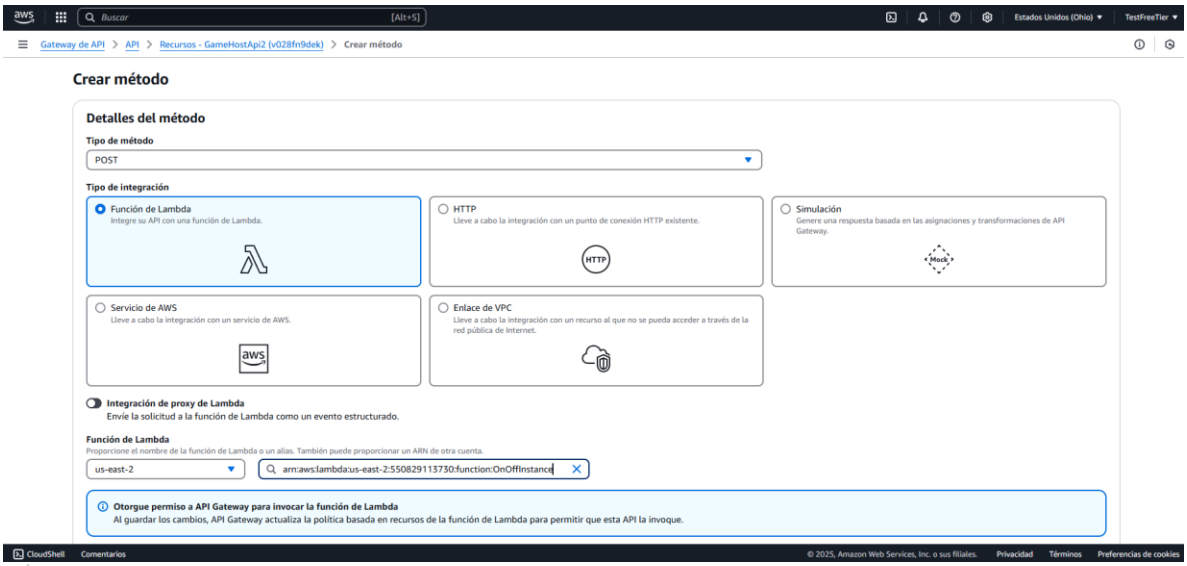
- Proceso para onoff: Creamos recurso /onoff habilitando CORS.

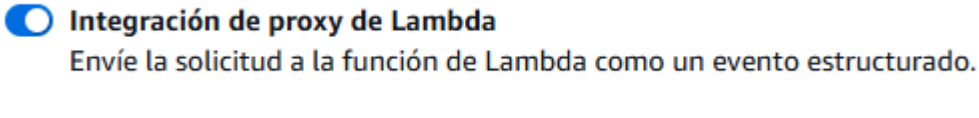


- Damos Click en crear método para onoff.

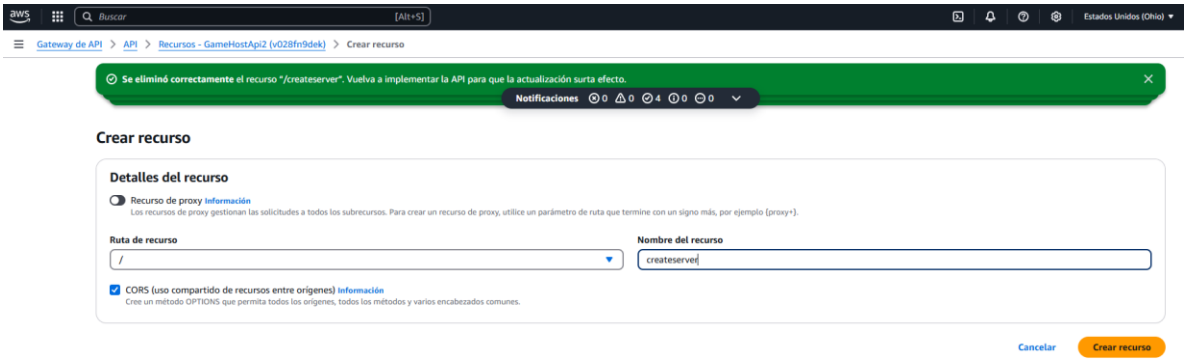


- Tipo Post, integración con Lambda, activamos integración de proxy de Lambda y lo ligamos a la función OnOffInstance, creamos método.

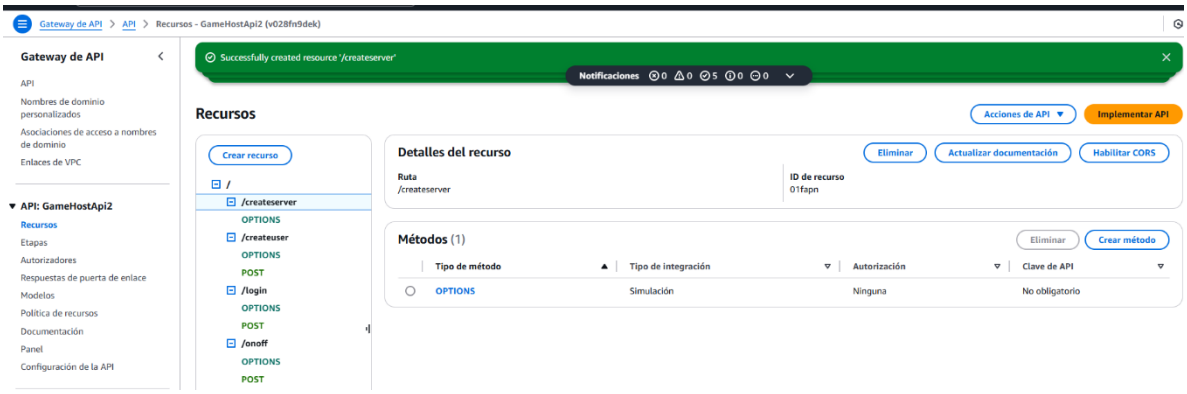




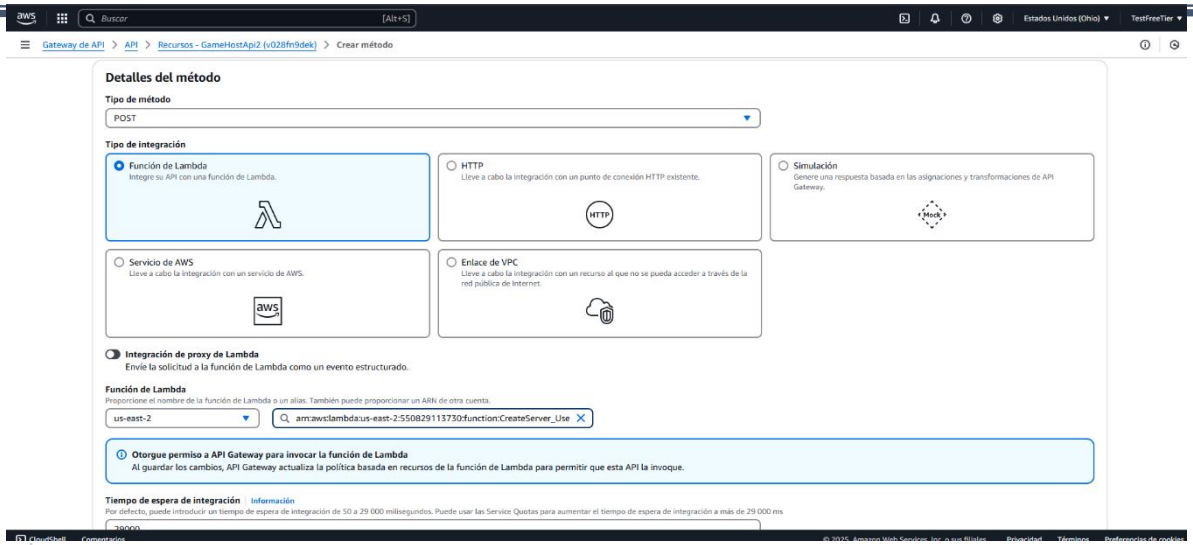
- Proceso para Create Server: Creamos recurso /createserver habilitando CORS.

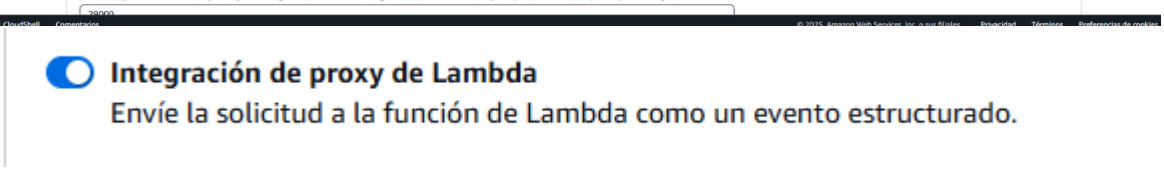


- Damos Click en crear método para createserver.

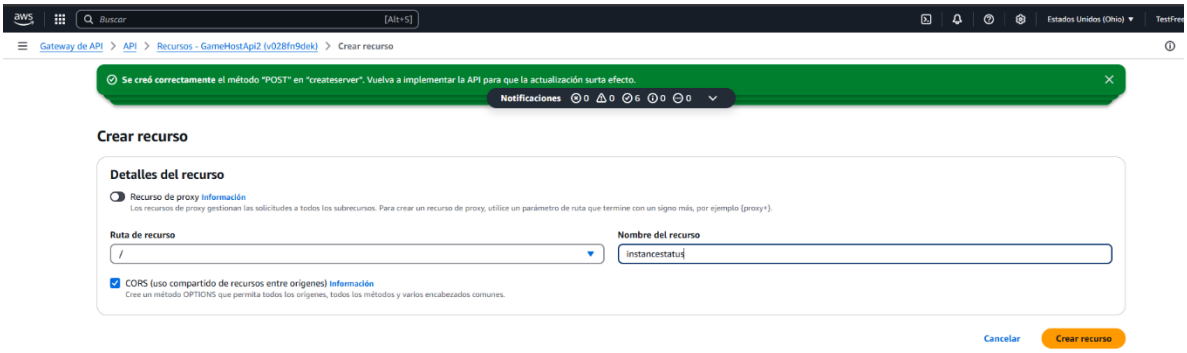


- Tipo Post, integración con Lambda, activamos integración de proxy de Lambda y lo ligamos a la función CreateServer_User, creamos el método.

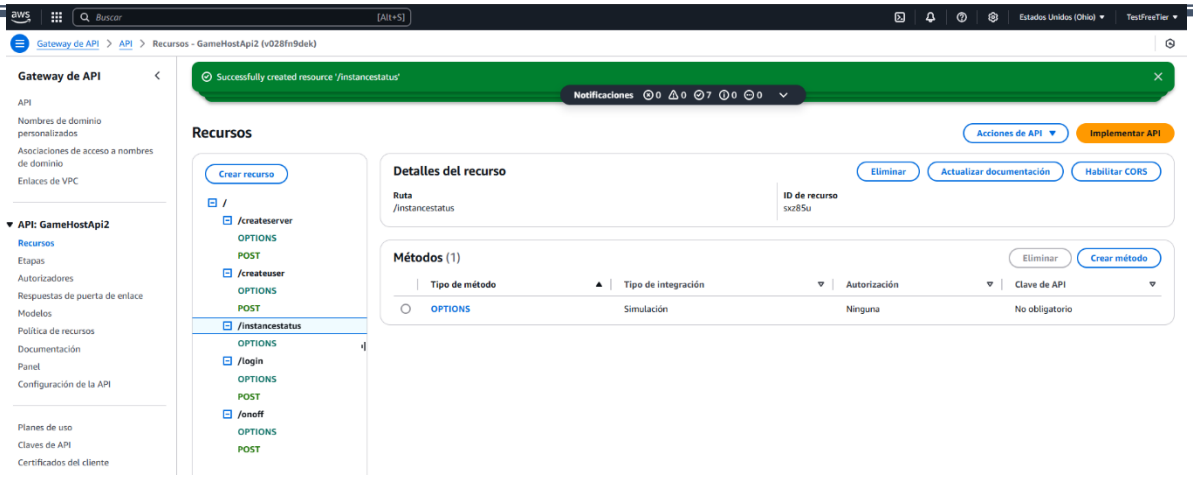




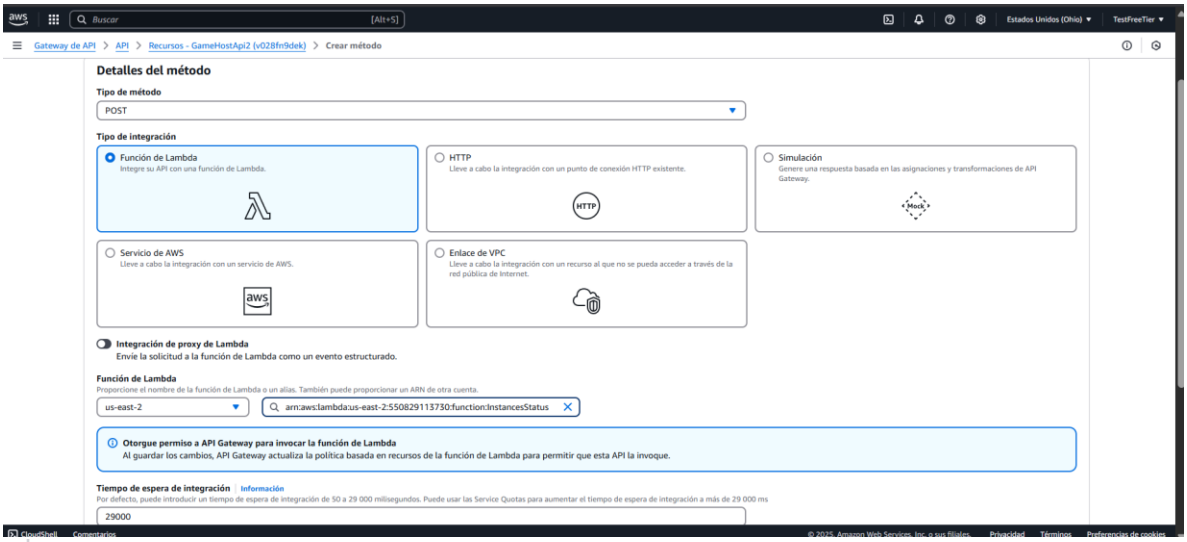
- Proceso para Instance Status: Creamos recurso /instancestatus habilitando CORS.

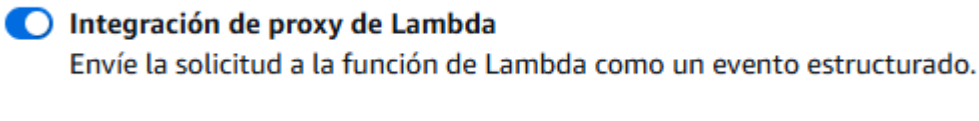


- Damos Click en crear método para instancestatus.



- Tipo Post, integración con Lambda, activamos integración de proxy de Lambda y lo ligamos a la función InstancesStatus, creamos método.

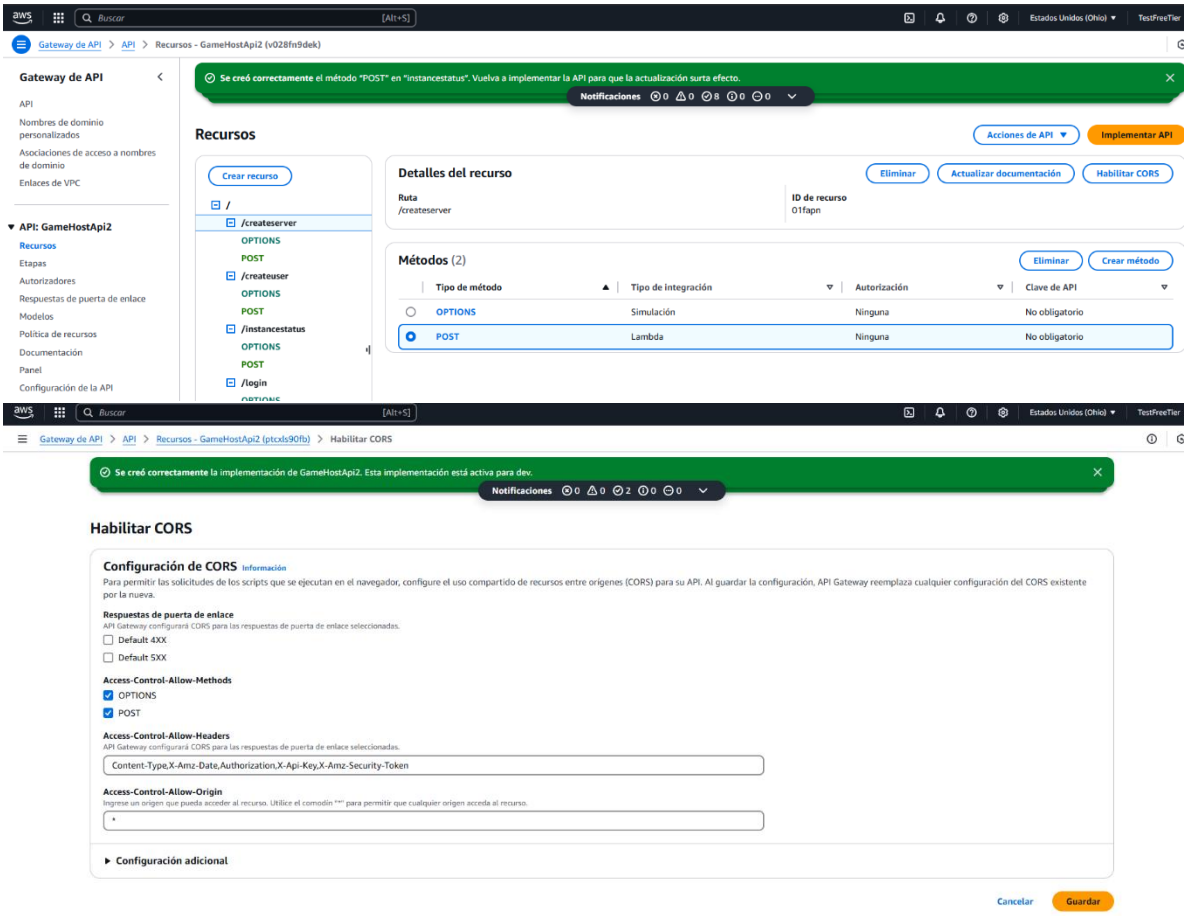




3. Habilitar CORS en cada recurso

- En cada recurso (/instancestatus, /createserver, /login, /onoff, /createuser):
  - Selecciona el POST, luego **Detalles del recurso** > **Habilitar CORS**. (Enable CORS).
  - Configura:
    - Allow-Headers: Content-Type,X-Amz-Date,Authorization,X-Api-Key,X-Amz-Security-Token
    - Allow-Methods: OPTIONS, POST

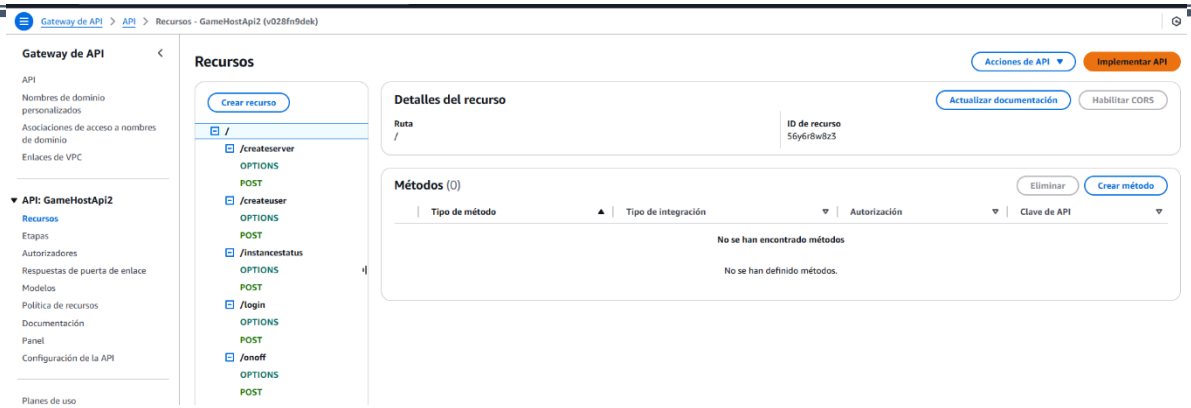
- Allow-Origin: *
- Aplica los cambios.



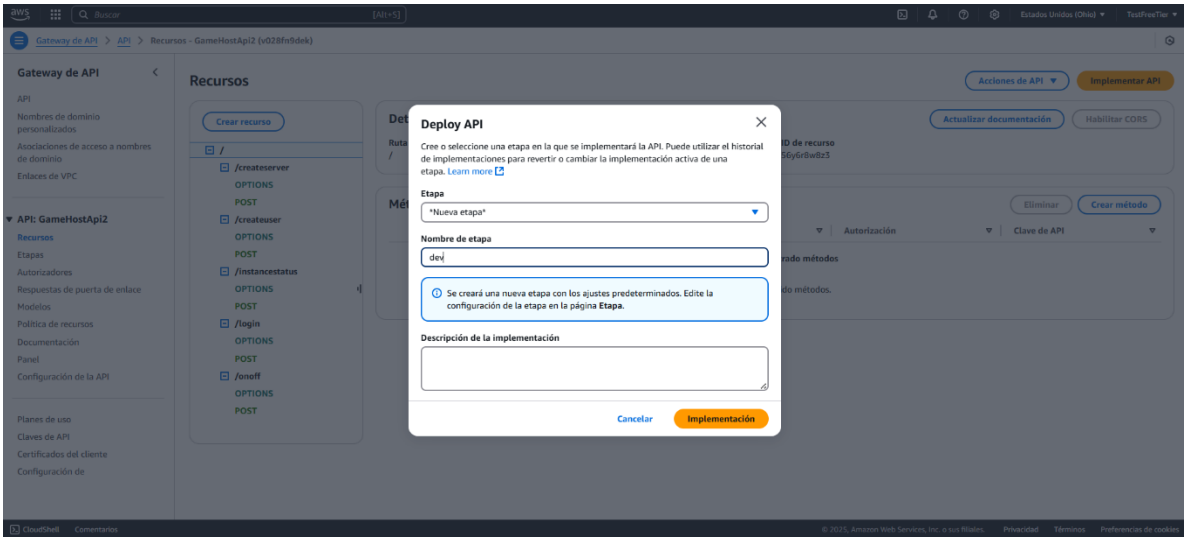
Repetimos el proceso para cada recurso como se mencionó previamente.

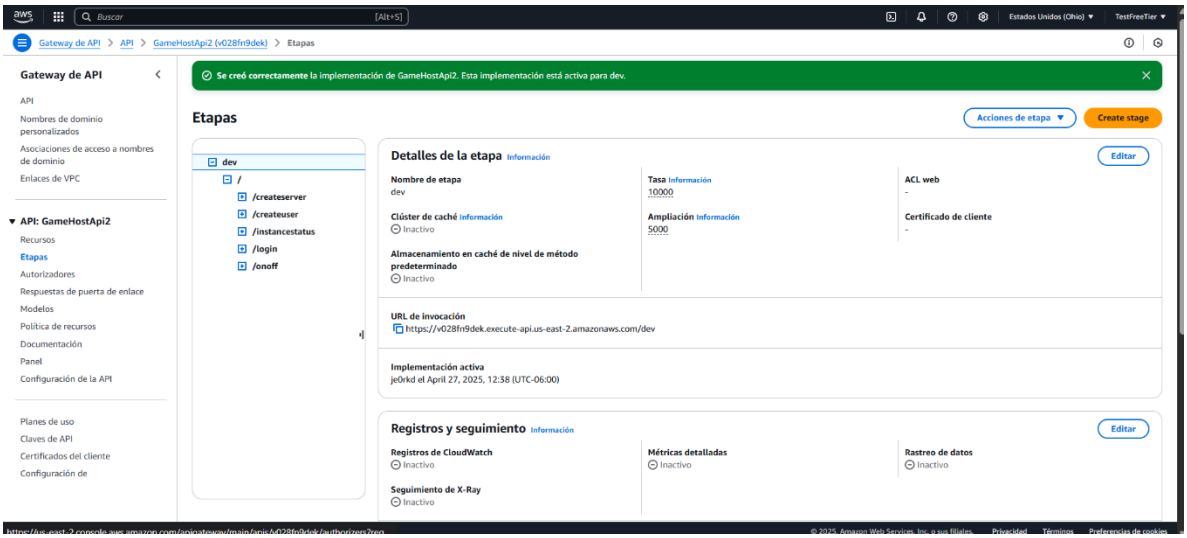
4. Deploy del API

- En GameHostoApi, ve a **Acciones** > **Implementar API (el botón amarillo en la esquina superior derecha).**



- Crea un **nuevo stage** o etapa (por ejemplo dev en este caso), y despliega.





5. Implementación

- En GameHostoApi, ve a **Estapas o Stages**> damos click al método que queramos invocar, copiamos la URL de invocación, y ya podemos manejarla y enviarle solicitudes de tipo POST desde nuestro script de JS.
- Para el login copiamos el URL de invocación de /login POST, y lo ponemos en el siguiente FETCH en nuestro login.html.

Código completo de login (HTML, CSS) con Script JS para manejar las solicitudes y respuestas:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>GameHost - Iniciar Sesión</title>
  <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
  <style>
    body {
      background-color: #0f0f0f;
      color: #00ffcc;
      font-family: 'Press Start 2P', cursive;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
      margin: 0;
    }

    .login-container {
      background-color: #1a1a1a;
      border: 2px solid #00ffcc;
      border-radius: 12px;
      padding: 40px;
```

```
    width: 400px;
    box-shadow: 0 0 15px #00ffcc;
    animation: fadeIn 1s ease-in-out;
  }

  .login-container h2 {
    text-align: center;
    color: #fff;
    margin-bottom: 30px;
  }

  label {
    display: block;
    margin: 15px 0 5px;
    font-size: 10px;
    color: #00ffff;
  }

  input[type="text"],
  input[type="password"] {
    width: 100%;
    padding: 12px;
    background-color: #0f0f0f;
    border: 2px solid #00ffcc;
    border-radius: 5px;
    color: #00ffcc;
    font-family: 'Press Start 2P', cursive;
    font-size: 10px;
    box-sizing: border-box;
  }

  .btn {
    margin-top: 25px;
    width: 100%;
    padding: 15px;
    background-color: #00ffcc;
    color: #000;
    border: none;
    font-size: 12px;
    border-radius: 5px;
    cursor: pointer;
    transition: transform 0.2s, background 0.3s;
```

```
    }

    .btn:hover {
      background-color: #00ffff;
      transform: scale(1.05);
    }

    @keyframes fadeIn {
      from { opacity: 0; transform: scale(0.9); }
      to { opacity: 1; transform: scale(1); }
    }

    .back-link {
      display: block;
      text-align: center;
      margin-top: 20px;
      font-size: 10px;
    }

    .back-link a {
      color: #00ffff;
      text-decoration: none;
    }

    .back-link a:hover {
      text-decoration: underline;
    }
  </style>
</head>
<body>
  <div class="login-container">
    <h2>🔐 Iniciar Sesión</h2>
    <form id="login-form">
      <label for="username">Usuario</label>
      <input type="text" id="username" name="username" required>

      <label for="password">Contraseña</label>
      <input type="password" id="password" name="password" required>

      <button type="submit" class="btn">Conectar</button>
    </form>
```

```
    <div class="back-link">
      <a href="index.html">⬅ Volver al inicio</a> |
      <a href="registro.html">📝 Registrarse</a>
    </div>
  </div>

  <script>
    document.getElementById('login-form').addEventListener('submit', async
function(e) {
      e.preventDefault();

      // Obtenemos email y password
      const email = document.getElementById('username').value;
      const password = document.getElementById('password').value;

      // Post para registrar usuario
      try {
        const response = await fetch('https://xc000gpm1j.execute-api.us-
east-2.amazonaws.com/Dev/login', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json'
          },
          body: JSON.stringify({
            User: { email, password }
          })
        });

        const text = await response.text();
        let data = {};

        try {
          data = JSON.parse(text);
        } catch (e) {
          console.warn('No se pudo parsear el JSON:', text);
          alert('Error al procesar la respuesta del servidor');
          return;
        }

        // Si no hubo errores, guardamos las credenciales del usuario en el
LocalStorage
```

```
        if (data.statusCode === 200) {
          const parsedBody = JSON.parse(data.body);
          localStorage.setItem('email', parsedBody.email);
          localStorage.setItem('password', password);
          localStorage.setItem('username', parsedBody.username);
          alert('Usuario autenticado correctamente');
          window.location.href= 'index.html';

        } else {
          const errorMessage = data.body ? JSON.parse(data.body).error :
'Error al iniciar sesión';
          alert(errorMessage);
        }
      } catch (err) {

      }
    });
  </script>
</body>
</html>
```

- Para el registro copiamos el URL de invocación de /registroPOST, y lo ponemos en el siguiente FETCH en nuestro registro.html.

```
        // Post a API gateway
        try {
          const response = await fetch('https://xc000gpm1j.execute-api.us-east-2.amazonaws.com/Dev/createuser', {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(payload)
          });
```

Código completo de registro (HTML, CSS) con Script JS para manejar las solicitudes y respuestas:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>GameHost - Registrarse</title>
  <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
  <style>
    body {
      background-color: #0f0f0f;
```

```
    color: #00ffcc;
    font-family: 'Press Start 2P', cursive;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    margin: 0;
  }

  .register-container {
    background-color: #1a1a1a;
    border: 2px solid #00ffcc;
    border-radius: 12px;
    padding: 40px;
    width: 400px;
    box-shadow: 0 0 15px #00ffcc;
    animation: fadeIn 1s ease-in-out;
  }

  .register-container h2 {
    text-align: center;
    color: #fff;
    margin-bottom: 30px;
  }

  label {
    display: block;
    margin: 15px 0 5px;
    font-size: 10px;
    color: #00ffff;
  }

  input[type="text"] {
    width: 100%;
    padding: 12px;
    background-color: #0f0f0f;
    border: 2px solid #00ffcc;
    border-radius: 5px;
    color: #00ffcc;
    font-family: 'Press Start 2P', cursive;
    font-size: 10px;
    box-sizing: border-box;
```

```
  }

  .btn {
    margin-top: 25px;
    width: 100%;
    padding: 15px;
    background-color: #00ffcc;
    color: #000;
    border: none;
    font-size: 12px;
    border-radius: 5px;
    cursor: pointer;
    transition: transform 0.2s, background 0.3s;
  }

  .btn:hover {
    background-color: #00ffff;
    transform: scale(1.05);
  }

  @keyframes fadeIn {
    from { opacity: 0; transform: scale(0.9); }
    to { opacity: 1; transform: scale(1); }
  }

  .back-link {
    display: block;
    text-align: center;
    margin-top: 20px;
    font-size: 10px;
  }

  .back-link a {
    color: #00ffff;
    text-decoration: none;
  }

  .back-link a:hover {
    text-decoration: underline;
  }
</style>
```

```
</head>
<body>
  <div class="register-container">
    <h2>📝 Registrarse</h2>
    <form id="register-form">
      <label for="username">Usuario</label>
      <input type="text" id="username" name="username" required>

      <label for="email">Correo electrónico</label>
      <input type="text" id="email" name="email" required>

      <label for="password">Contraseña</label>
      <input type="text" id="password" name="password" required>

      <button type="submit" class="btn">Crear Cuenta</button>
    </form>
    <div class="back-link">
      <a href="index.html">⬅ Volver al inicio</a>
    </div>
  </div>

  <script>
    document.getElementById('register-form').addEventListener('submit',
async function(e) {
      e.preventDefault();

      const username = document.getElementById('username').value.trim();
      const email = document.getElementById('email').value.trim();
      const password = document.getElementById('password').value.trim();

      // Validacion de email
      const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
      if (!emailRegex.test(email)) {
        alert('⚠️ El correo electrónico no es válido');
        return;
      }

    // Validar longitud password
      if (password.length < 8) {
        alert('⚠️ La contraseña debe tener al menos 8 caracteres');
        return;
```

```
      }

      const payload = {
        User: { username, email, password }
      };

      // Post a API gateway
      try {
        const response = await fetch('https://xc000gpm1j.execute-api.us-
east-2.amazonaws.com/Dev/createuser', {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify(payload)
        });

        const data = await response.json();

        if (response.ok && data.statusCode === 201) {
          alert('☑ Usuario registrado con éxito');
          window.location.href = 'index.html';
        } else {
          alert('⚠ Error: ' + JSON.parse(data.body).error);
        }
      } catch (err) {
        console.error(err);
        alert('❌ Error al conectar con el servidor');
      }
    });
  </script>
</body>
</html>
```

- Para el index copiamos el URL de invocación POST de /createserver, /instancestatus y /onoff y lo ponemos en las siguientes variables declaradas en nuestro código index.html ya que por medio de estas se hacen las solicitudes requeridas para el proceso de crear servidor, checar su estatus y apagarlo o encenderlo en nuestro index.html.

```
// API Gateways
const API_URL = "https://v028fn9dek.execute-api.us-east-2.amazonaws.com/dev/createserver";
const CONSULTAR_ESTADO_URL = "https://v028fn9dek.execute-api.us-east-2.amazonaws.com/dev/instancestatus";
const ENCENDER_URL = "https://v028fn9dek.execute-api.us-east-2.amazonaws.com/dev/onoff";
```

Código completo de index (HTML, CSS) con Script JS para manejar las solicitudes y respuestas:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1" />
  <title>GameHUB - Servidores Minecraft y Terraria</title>
  <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
  <style>
    body {
      margin: 0;
      padding: 0;
      background: #0f0f0f;
      color: #00ffcc;
      font-family: 'Press Start 2P', cursive;
      text-shadow: 1px 1px #000;
    }
    header {
      background: linear-gradient(90deg, #00ffcc, #0066ff);
      padding: 30px;
      text-align: center;
      color: #000;
    }
    nav {
      background: #111;
      padding: 15px;
      text-align: center;
    }
    nav a {
      color: #00ffcc;
      text-decoration: none;
      margin: 0 20px;
      font-size: 12px;
    }
    nav a:hover {
      color: #fff;
```

```
    }
    section {
      padding: 40px 20px;
      max-width: 1000px;
      margin: auto;
    }
    .card {
      background: #1a1a1a;
      border: 2px solid #00ffcc;
      border-radius: 10px;
      padding: 30px;
      margin-bottom: 40px;
      box-shadow: 0 0 10px #00ffcc;
    }
    .card h2 {
      color: #fff;
    }
    .btn {
      display: inline-block;
      background: #00ffcc;
      color: #000;
      padding: 15px 25px;
      font-size: 12px;
      border: none;
      border-radius: 5px;
      margin-top: 15px;
      cursor: pointer;
      transition: transform 0.2s, background 0.3s;
    }
    .btn:hover {
      background: #00ffff;
      transform: scale(1.05);
    }
    footer {
      background: #111;
      text-align: center;
      padding: 30px;
      font-size: 10px;
      color: #555;
    }
  </style>
</head>
```

```
<body>
  <header>
    <h1>🎮 GameHUB</h1>
    <p>Servidores dedicados para Minecraft y Terraria</p>
  </header>

  <nav>
    <a href="#minecraft">Minecraft</a>
    <a href="#terraria">Terraria</a>
    <a href="#precios">Precios</a>
    <a href="#contacto">Contacto</a>
    <a href="login.html" class="btn">Log In</a>
    <a href="perfil.html" class="btn">Profile</a>
  </nav>

  <section id="minecraft">
    <div class="card">
      <h2>🌍 Servidores Minecraft</h2>
      <p id="pMinecraft">Juega con amigos en mundos personalizados. Java y
Bedrock, soporte 24/7.</p>
      <button class="btn" id="minecraftButton">Levantar Nuevo
Servidor</button>
      <button class="btn reload" onclick="location.reload()">↻</button>
      <div id="minecraftInstances"></div>
    </div>
  </section>

  <section id="terraria">
    <div class="card">
      <h2>🌓 Servidores Terraria</h2>
      <p id="p1">Explora, construye y sobrevive con mods o clásicos.
Servidores estables y rápidos.</p>
      <button class="btn" id="terrariaButton">Levantar Nuevo
Servidor</button>
      <button class="btn reload" onclick="location.reload()">↻</button>
      <div id="terrariaInstances"></div>
    </div>
  </section>

  <section id="precios">
    <div class="card">
```

```
      <h2>💰 Planes</h2>
      <ul>
        <li>Básico: 2GB RAM - $5 USD/mes</li>
      </ul>
    </div>
  </section>

  <section id="contacto">
    <div class="card">
      <h2>📬 Contáctanos</h2>
      <p>¿Dudas? Escríbenos: <a href="mailto:soporte@gamehub.com"
style="color: #00ffff;">soporte@gamehub.com</a></p>
    </div>
  </section>

  <footer>
    <p>&copy; 2025 GameHUB. Powered by gamers, for gamers.</p>
  </footer>

  <script>
    const user = localStorage.getItem("email");
    const passwd = localStorage.getItem("password");

    // API Gateways
    const API_URL = "https://xc000gpm1j.execute-api.us-east-
2.amazonaws.com/Dev/createserver";
    const CONSULTAR_ESTADO_URL = "https://xc000gpm1j.execute-api.us-east-
2.amazonaws.com/Dev/instancestate";
    const ENCENDER_URL = "https://xc000gpm1j.execute-api.us-east-
2.amazonaws.com/Dev/onoff";

    const terrariaBtn = document.getElementById("terrariaButton");
    const minecraftBtn = document.getElementById("minecraftButton");
    const p1 = document.getElementById("p1");
    const pMinecraft = document.getElementById("pMinecraft");

    const terrariaInstances = document.getElementById("terrariaInstances");
    const minecraftInstances =
document.getElementById("minecraftInstances");

    // Consultar estado al cargar
```

```
    consultarEstado("minecraft");
    consultarEstado("terraria");

    // Funcion para consultar el estado de los servidores minecraft o
terraria
    async function consultarEstado(game) {
      try {
        const res = await fetch(CONSULTAR_ESTADO_URL, {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ User: { email: user, password: passwd } })
        });

        const data = await res.json();
        const instances = data.instances?.filter(i => i.game === game);

        const container = game === "minecraft" ? minecraftInstances :
terrariaInstances;
        container.innerHTML = '';  // Limpiar el contenedor

        const instanceDiv = document.createElement("div");

        if (!instances || instances.length === 0) {
          instanceDiv.innerHTML = `<p>💤 No hay instancias activas.</p>`;
          container.appendChild(instanceDiv);
        } else {
          instances.forEach(instancia => {
            const instanceDiv = document.createElement("div");
            instanceDiv.innerHTML = `
              <p>🆔 ID: ${instancia.InstanceId} - ${instancia.State ===
'running' ? '☑️ En ejecución' : '🟢 Detenido'}<br>
                🌐 IP: ${instancia.PublicIpAddress}<br>
                📊 Uso del CPU: ${instancia.cpuUtilization}</p>
                <button class="btn"
onclick="toggleServer('${instancia.InstanceId}', '${game}')">
                  ${instancia.State === 'running' ? 'Detener' : 'Levantar'}
Servidor
                </button>
              `;
            container.appendChild(instanceDiv);
          });
```

```
        }

      } catch (err) {
        console.error(err);
        const container = game === "minecraft" ? minecraftInstances :
terrariaInstances;
        container.innerHTML = "<p>❌ Error al consultar estado.</p>";
      }
    }


    // Funcion para levantar o apagar servidores
    async function toggleServer(id, game) {
      const boton = event.target;
      const container = game === "minecraft" ? minecraftInstances :
terrariaInstances;
      const instanciaDiv = boton.parentNode;

      boton.disabled = true;
      boton.textContent = "Procesando...";

      try {
        const estado = boton.textContent.includes("Levantar") ? "ON" :
"OFF";
        const res = await fetch(ENCENDER_URL, {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            Instance: { id: id, status: estado }
          })
        });

        if (res.ok) {
          consultarEstado(game); // Actualizar el estado
        } else {
          instanciaDiv.innerHTML = "<p>❌ Error al cambiar el estado del
servidor.</p>";
        }
      } catch (err) {
        console.error(err);
        instanciaDiv.innerHTML = "<p>❌ Error de red.</p>";
```

```
      }
    }

    // Levantar nuevo servidor
    async function levantarServidor(game) {
      const container = game === "minecraft" ? minecraftInstances :
terrariaInstances;
      const nuevoServidorDiv = document.createElement("div");
      nuevoServidorDiv.innerHTML = `<p>⏳ Levantando nuevo servidor...</p>`;
      container.appendChild(nuevoServidorDiv);

      try {
        const res = await fetch(API_URL, {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({
            User: { email: user, password: passwd },
            game: game
          })
        });

        const data = await res.json();

        if (res.ok && data.public_ip) {
          nuevoServidorDiv.innerHTML = `
            ☑ Servidor de ${game} activo<br>🌐 IP: ${data.public_ip}<br>
            <button class="btn" onclick="toggleServer('${data.instance_id}',
'${game}')">Detener Servidor</button>
          `;
        } else {
          nuevoServidorDiv.innerHTML = `<p>❌ Error al levantar el
servidor.</p>`;
        }
      } catch (err) {
        console.error(err);
        nuevoServidorDiv.innerHTML = `<p>❌ Error de red.</p>`;
      }
    }

    minecraftBtn.addEventListener("click", () =>
levantarServidor("minecraft"));
```

```
    terrariaBtn.addEventListener("click", () =>
levantarServidor("terraria"));

  </script>
</body>
</html>
```

- Para el perfil copiamos el URL de invocación de /instancestatus POST, y lo ponemos en la siguiente variable (CONSULTAR_ESTADO_URL) en nuestro perfil.html.

```
document.addEventListener('DOMContentLoaded', ()=>{
  const CONSULTAR_ESTADO_URL = "https://v028fn9dek.execute-api.us-east-2.amazonaws.com/dev/instancestatus";
  emailText=document.getElementById('email');
  perfilText=document.getElementById('username');
  instancesText=document.getElementById('instances');
  emailText.textContent=localStorage.getItem('email');
  perfilText.textContent=localStorage.getItem('username');
```

Código completo de perfil (HTML, CSS) con Script JS para manejar las solicitudes y respuestas:

```
<!DOCTYPE html>
<html lang="es">
<head>
  <meta charset="UTF-8">
  <title>GameHost - Perfil de Usuario</title>
  <link
href="https://fonts.googleapis.com/css2?family=Press+Start+2P&display=swap"
rel="stylesheet">
  <style>
    body {
      background-color: #0f0f0f;
      color: #00ffcc;
      font-family: 'Press Start 2P', cursive;
      margin: 0;
      padding: 0;
    }

    header {
      padding: 20px;
      text-align: center;
      background-color: #1a1a1a;
      border-bottom: 2px solid #00ffcc;
    }
```

```css
header h1 {
  font-size: 20px;
  margin: 0;
  text-shadow: 0 0 5px #00ffcc;
}

.logout {
  position: absolute;
  top: 20px;
  right: 20px;
  font-size: 10px;
  text-decoration: none;
  color: #ff0066;
}

.section {
  padding: 30px;
}

.section h2 {
  font-size: 14px;
  margin-bottom: 20px;
  color: #00ffff;
}

.gallery {
  display: flex;
  flex-wrap: wrap;
  gap: 20px;
  justify-content: center;
}

.gallery img,
.gallery video {
  border: 2px solid #00ffcc;
  border-radius: 10px;
  max-width: 200px;
  max-height: 150px;
  object-fit: cover;
}
```

```
    .upload-form {
      text-align: center;
      margin-top: 40px;
    }

    .upload-form input[type="file"] {
      margin: 10px 0;
      background: #1a1a1a;
      border: 1px solid #00ffcc;
      color: #00ffcc;
      font-family: 'Press Start 2P', cursive;
      font-size: 10px;
    }

    .btn {
      margin-top: 10px;
      padding: 10px 20px;
      background-color: #00ffcc;
      color: #000;
      border: none;
      font-size: 10px;
      border-radius: 5px;
      cursor: pointer;
      transition: transform 0.2s, background 0.3s;
    }

    .btn:hover {
      background-color: #00ffff;
      transform: scale(1.05);
    }

    footer {
      text-align: center;
      padding: 20px;
      font-size: 8px;
      color: #555;
    }
  </style>
</head>
<body>
  <header>
```

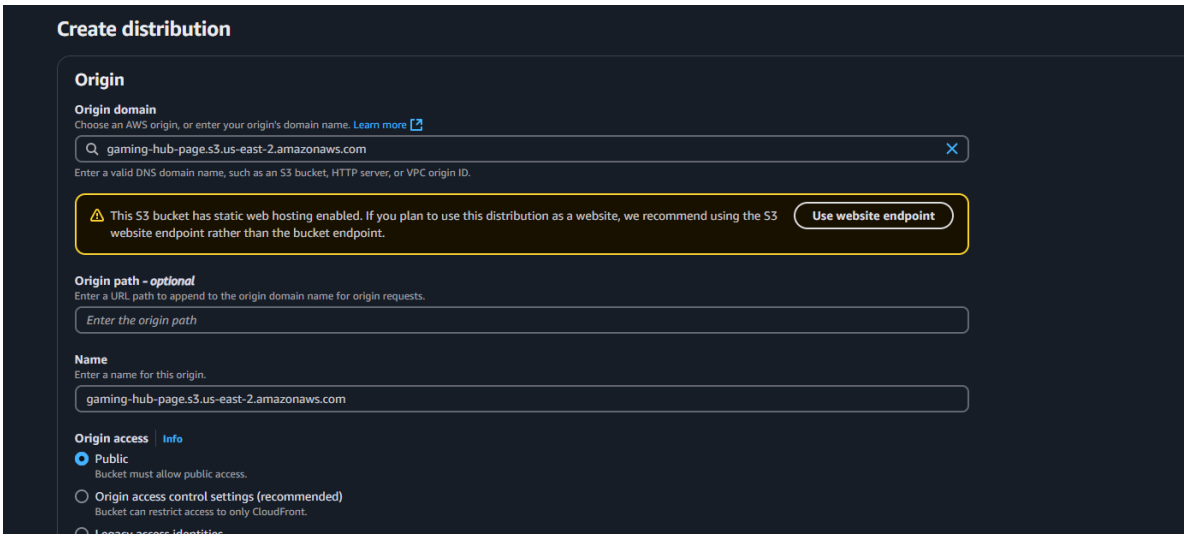
```
    <h1>👤 Email: <span id="email"></span></h1>
    <h1>👤 Username: <span id="username"></span></h1>
    <hr>
    <a href="index.html" class="btn">Home</a>
  </header>

  <div class="section">
    <h2>💻 Instancias del usuario</h2>
    <div class="instan">
      <p id="instances"></p>
    </div>
    <h2>🖼️ Galería de Imágenes</h2>
    <div class="gallery">
      <!-- Imágenes subidas -->
      <img src="img/sample1.jpg" alt="Imagen 1">
      <img src="img/sample2.jpg" alt="Imagen 2">
    </div>

    <div class="upload-form">
      <form action="/upload-image" method="post" enctype="multipart/form-
data">
        <label for="image">Subir imagen:</label><br>
        <input type="file" name="image" id="image" accept="image/*"
required><br>
        <button type="submit" class="btn">Subir Imagen</button>
      </form>
    </div>
  </div>

  <div class="section">
    <h2>🎥 Galería de Videos</h2>
    <div class="gallery">
      <!-- Videos subidos -->
      <video controls>
        <source src="video/sample1.mp4" type="video/mp4">
        Tu navegador no soporta videos.
      </video>
      <video controls>
        <source src="video/sample2.mp4" type="video/mp4">
        Tu navegador no soporta videos.
      </video>
```

```
    </div>

    <div class="upload-form">
      <form action="/upload-video" method="post" enctype="multipart/form-
data">
        <label for="video">Subir video:</label><br>
        <input type="file" name="video" id="video" accept="video/*"
required><br>
        <button type="submit" class="btn">Subir Video</button>
      </form>
    </div>
  </div>

  <footer>
    GameHost © 2025 - Servidores épicos para gamers épicos 🎮🔥
  </footer>

  <script>

    document.addEventListener('DOMContentLoaded', ()=>{
      const CONSULTAR_ESTADO_URL = "https://xc000gpm1j.execute-api.us-east-
2.amazonaws.com/Dev/instancestate";
      emailText=document.getElementById('email');
      perfilText=document.getElementById('username');
      instancesText=document.getElementById('instances');
      emailText.textContent=localStorage.getItem('email');
      perfilText.textContent=localStorage.getItem('username');

      const user= localStorage.getItem('email');
      const passwd= localStorage.getItem('password');
      async function consultarEstado() {
      try {
        const res = await fetch(CONSULTAR_ESTADO_URL, {
          method: "POST",
          headers: { "Content-Type": "application/json" },
          body: JSON.stringify({ User: { email: user, password: passwd } })
        });

        const data = await res.json();
        console.log(JSON.stringify(data));
        const instances=data.instances;
```

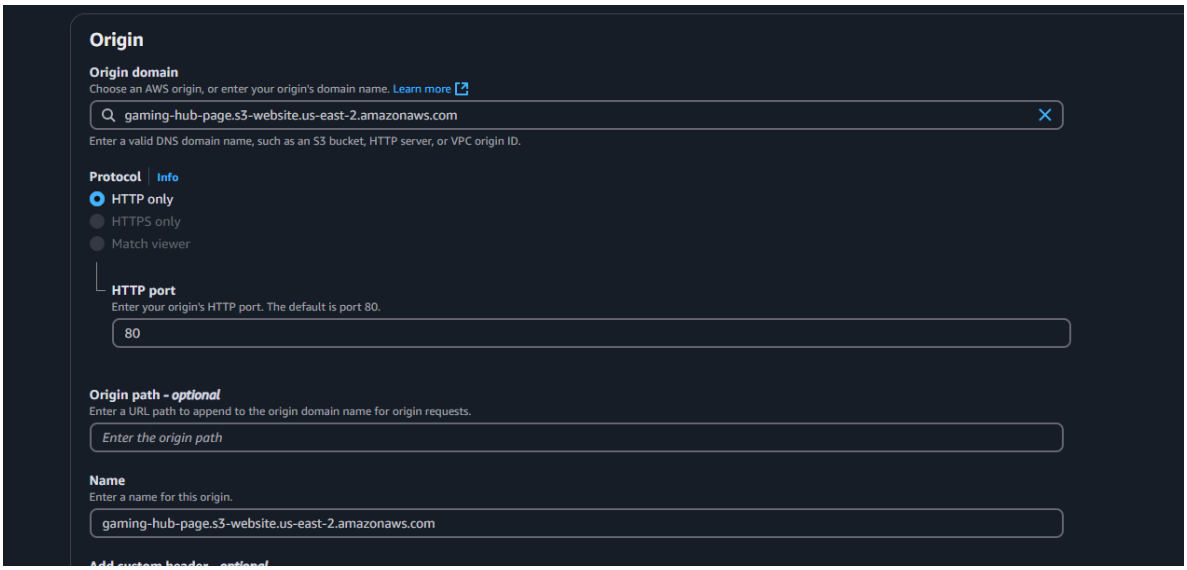
```
        if (!instances || instances.length === 0) {
          instancesText.textContent = "💤 No hay instancias activas de
Terraria.";
          return;
        }

        let instancesContent ='';
        instances.forEach(instancia => {
            if (instancia.State === "running") {
              instancesContent += `✅ Servidor en ejecución<br>🆔 ID:
${instancia.InstanceId}<br>🌐 IP: ${instancia.PublicIpAddress}<br>📊 Uso
del CPU: ${instancia.cpuUtilization}<br><br>`;
            } else {
              instancesContent  += "🛑 Servidor detenido<br><br>";
            }
        });

        instancesText.innerHTML = instancesContent;

      } catch (err) {
        console.error(err);
        instancesText.textContent = "❌ Error al consultar estado.";
      }
    };
    consultarEstado();

    });
  </script>
</body>
</html>
```

CLOUD FRONT:

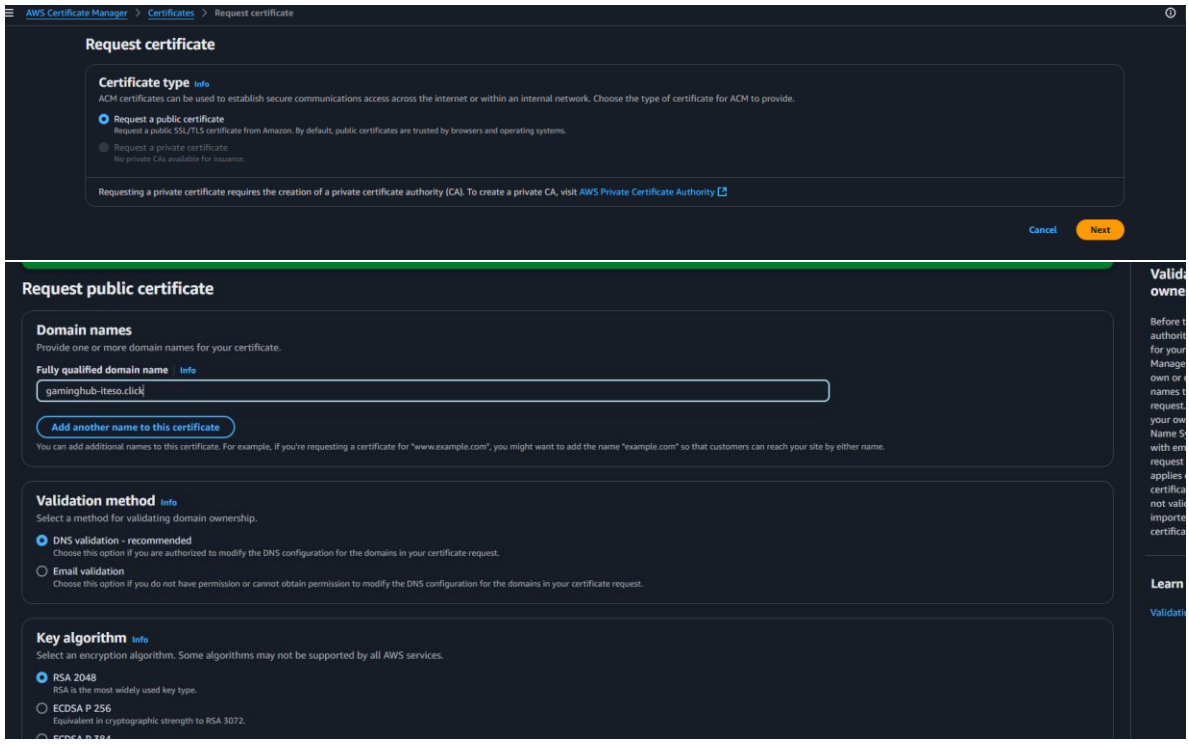
1. Nos metemos a crear un cloud front y en origin ponemos el bucket de ec2



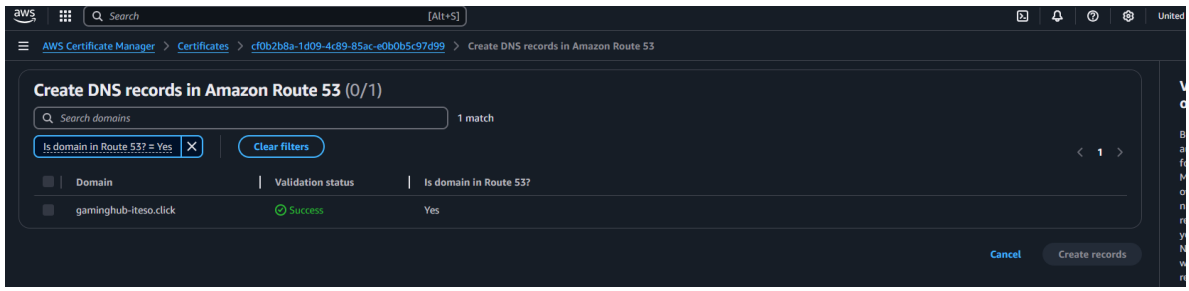
2. Aceptamos al uso del webside endpoint



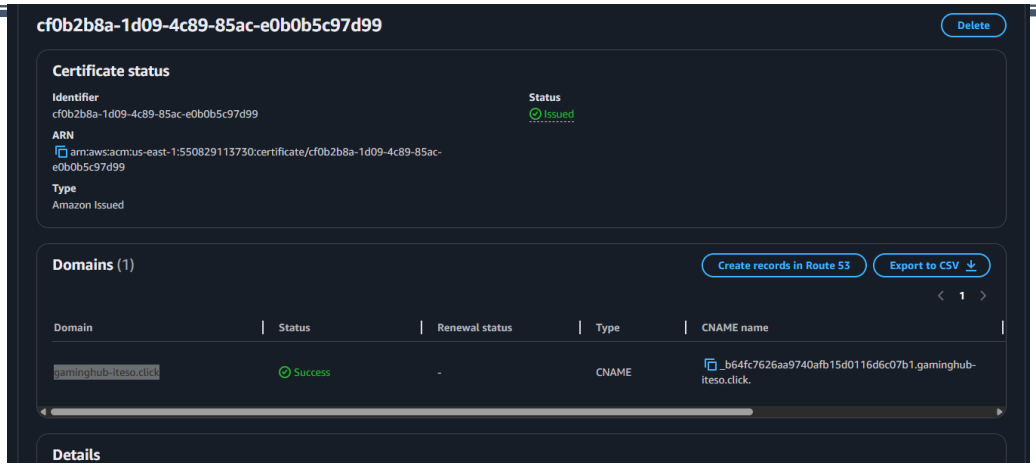
3. Tenemos que ingresar a certificate para generar un certificado de ACM apra el dominio con el mismo nombre que le vas a poner al route 53



4. Vamos a pedir un record para mostrar que de echo somo duenos del dominio de Route 53

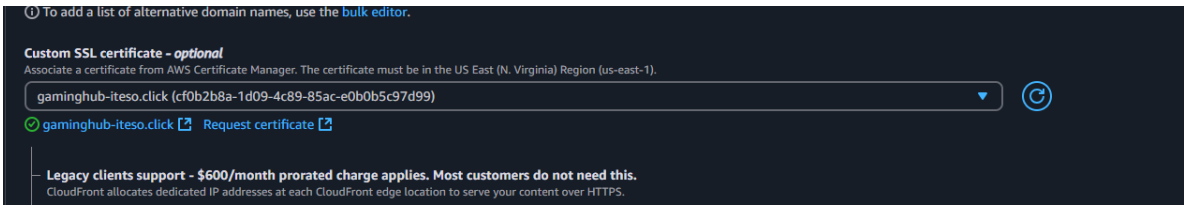


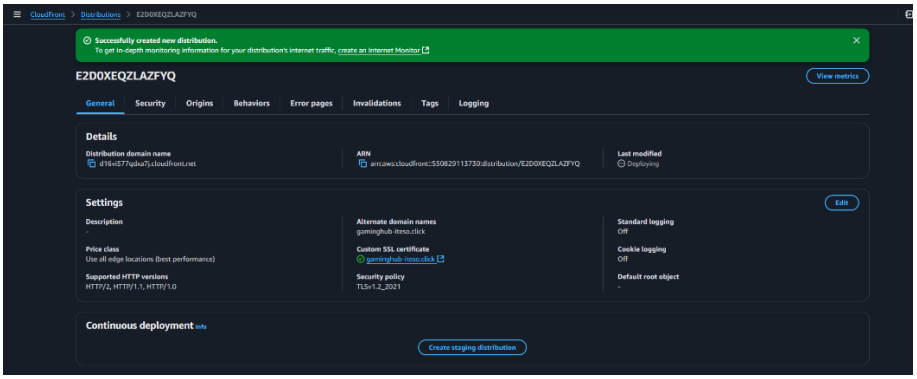
5. Con eso deberíamos tener el certificado



6. Ahora podemos ponerle el nombre alternativo al cloudfront

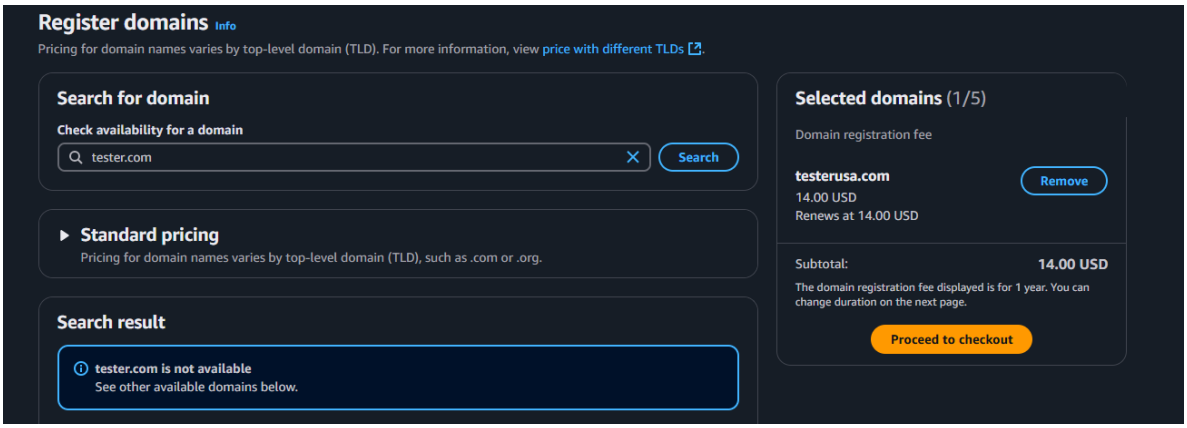
7. Y ponemos el certificado para mostrar que tenemos el dominio



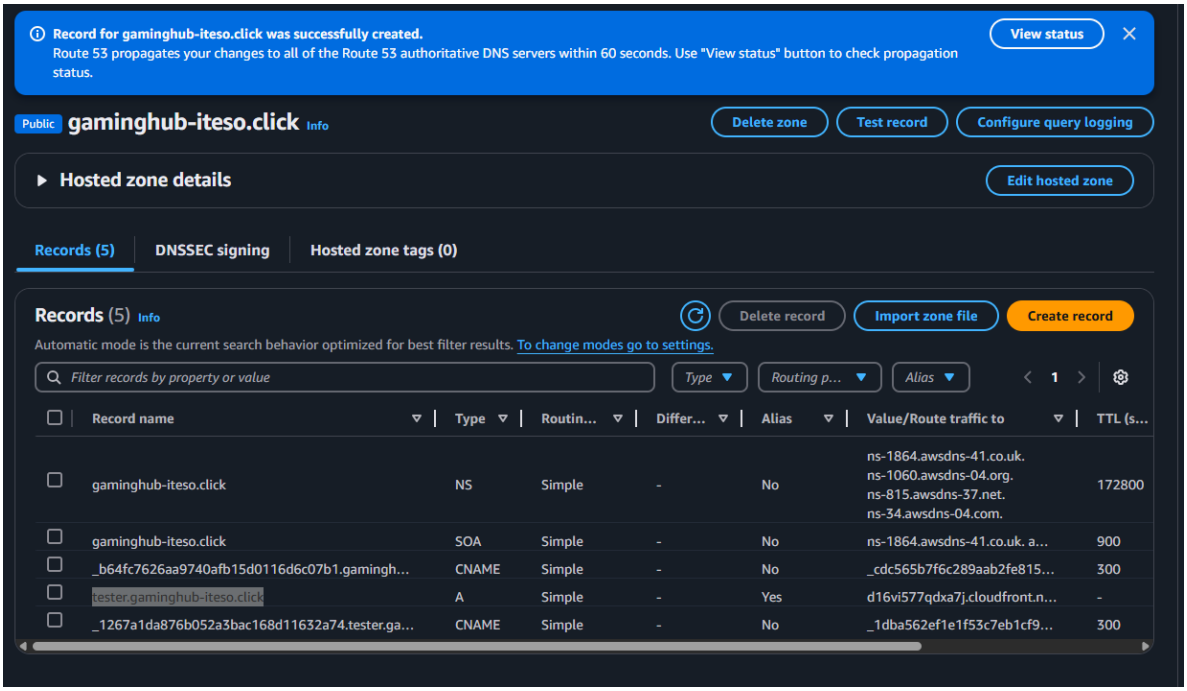


ROUTE 53:

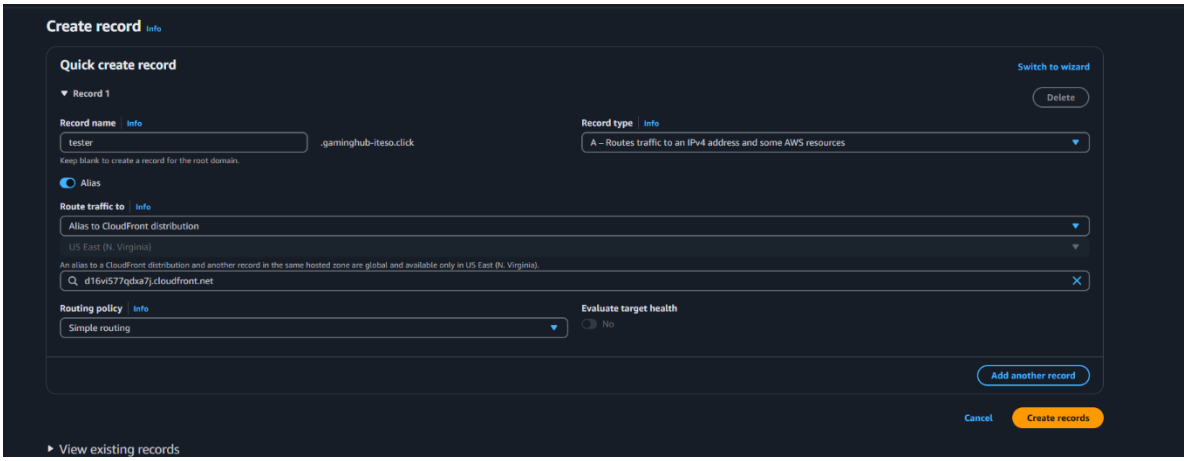
1. Registramos un domino y pagamos por su uso



2. Dentr de hosted zones, nos metemos al dominio y crearemos un nuevo récord



3. Creamos el récord con la instancia de CloudFront



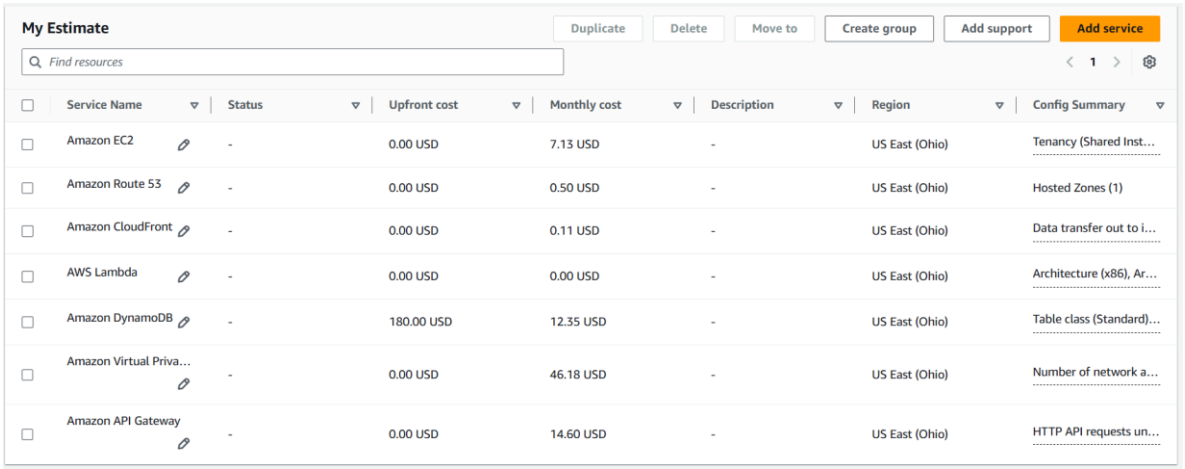
4. Listo

NOTA: CloudFront(sobre todo el certificado) y Route 53 se deben de hacer al mismo tiempo y pensando en el nombre del subdominio desde el inicio. si la páguina es tester.gaminghub.com, tienes que llamarle así en el Certificate y ponerlo exactamente igual en el Route53 como récord o te va a dar error 53[yo cometí ese error y tuve que repetir el proceso del certifícate con tester(en Route 53 era tester.gaminghub-iteso.click y en el certificado era gaminghub-iteso.click)]

Costo

Gracias a aws y su cuenta gratis en total este proyecto nos ha costado 13.64 dólares mensuales(avg), lo cual es muy bajo a comparación de los valores que sacamos de la calculadora. AWS nos regaló 750 horas de EC2, 5 GB de almacenamiento en los s3, 25 GB de almacenamiento de DynamoDB, 1 millón de llamadas a él API Gateway y 1TB gratis de CLoudFront [13]. Aparte de los precios de servicios, pagamos 3 dólares por el dominio de gaming Hub Iteso, por lo cual cree que si utilizamos bien la versión gratis de aws. Obviamente este precio va a escalar con el tiempo ya que se levantarían más instancias de EC2, aumentando el precio mensual del producto.

Checando con la calculadora de precios



Conclusiones Parte 1

La propuesta de Gaming Hub presenta un enfoque innovador para la gestión y alojamiento de servidores de videojuegos en la nube utilizando los servicios de AWS. A través de herramientas como Amazon EC2, S3, CloudWatch, Lambda y API Gateway, se plantea una solución escalable, segura y eficiente para que los usuarios puedan crear y administrar sus propios servidores. Además, se consideran aspectos clave como el monitoreo del rendimiento, el almacenamiento de datos y la distribución de tráfico con Elastic Load Balancing, garantizando una experiencia óptima.

Este proyecto busca ofrecer una plataforma accesible e intuitiva, con opciones de personalización y seguridad mediante AWS IAM y Amazon VPC. La propuesta incluye características como la escalabilidad automática y la gestión flexible de servidores, con posibilidades de crecimiento a futuro. Como siguiente paso, se evaluará la viabilidad técnica y se definirán los detalles de implementación para desarrollar un sistema funcional que responda a las necesidades de los jugadores y optimice el uso de los recursos en la nube.

Conclusiones Parte 2

Este proyecto demuestra cómo una función Lambda puede integrarse con Amazon services para controlar instancias EC2, y conexión con otros servicios como DYNAMO, CloudWatch, etc., enfocándose en ofrecer una solución accesible y automatizada para obtener métricas clave como el uso de CPU y memoria. La implementación considera buenas prácticas de seguridad mediante roles IAM, así como la posibilidad de extender la solución dentro de una arquitectura escalable en Amazon VPC.

A través de API Gateway, logramos integrar funciones Lambda para acciones como el despliegue de instancias y el registro de usuarios, entre otras funcionalidades. Configuramos el método de solicitud (en este caso, principalmente POST) para facilitar el envío de solicitudes utilizando JavaScript. En el código de JavaScript manejamos correctamente las respuestas devueltas por el API para de esta forma poder mostrar mensajes a los usuarios o actualizar divs con información relevante como lo puede ser el uso del CPU, instancias asociadas al usuario y muchas cosas más.

Aparte de esto, nos gustaría hablar un poco de Route 53, ya que esto se complicó un poco más de lo que se creía. Route 53 trabaja en conjunto con AWS Certificate Manager, ya que se necesita otorgar certificados a CloudFront para que este pueda reconocer su nombre en la web (pruebas.gaminhub-iteso.click), lo cual no teníamos claro al momento de crear nuestro CloudFront. Por esta razón, tuvimos que recrear todo el CloudFront y generar varios certificados hasta que entendimos cómo funcionaban y cómo se relacionaban con Route 53.

Finalmente, algo que nos impactó de forma negativa y al mismo tiempo positiva fue el firewall de AWS, el cual se configuró automáticamente sin avisar cuando creamos el CloudFront con Route 53. Sin embargo, fue el responsable del 30 % de nuestro costo total.

Bibliografía

[1] Amazon Web Services, “Amazon GameLift – Dedicated Game Server Hosting,” 2024. [Online]. Available: https://aws.amazon.com/gamelift/

[2] Amazon Web Services, Inc. "¿Qué es Amazon EC2?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AWSEC2/latest/UserGuide/concepts.html. [Accedido: 24-mar-2025].

[3] Amazon Web Services, Inc. "¿Qué es Amazon S3?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AmazonS3/latest/userguide/Welcome.html. [Accedido: 24-mar-2025].

[4] Amazon Web Services, Inc. "¿Qué es Amazon CloudWatch?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html. [Accedido: 24-mar-2025].

[5] Amazon Web Services, Inc. "¿Qué es Amazon CloudFront?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AmazonCloudFront/latest/DeveloperGuide/Introduction.html. [Accedido: 24-mar-2025].

[6] Amazon Web Services, Inc. "¿Qué es Amazon RDS?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/AmazonRDS/latest/UserGuide/Welcome.html. [Accedido: 24-mar-2025].

[7] Amazon Web Services, Inc. "¿Qué es AWS Lambda?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/lambda/latest/dg/welcome.html. [Accedido: 24-mar-2025].

[8] Amazon Web Services, Inc. "¿Qué es Amazon API Gateway?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/apigateway/latest/developerguide/welcome.html. [Accedido: 24-mar-2025].

[9] Amazon Web Services, Inc. "¿Qué es AWS IAM?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/IAM/latest/UserGuide/introduction.html. [Accedido: 24-mar-2025].

[10] Amazon Web Services, Inc. "¿Qué es Amazon VPC?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/vpc/latest/userguide/what-is-amazon-vpc.html. [Accedido: 24-mar-2025].

[11] Amazon Web Services, Inc. "¿Qué es Amazon Route 53?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/Route53/latest/DeveloperGuide/Welcome.html. [Accedido: 24-mar-2025].

[12] Amazon Web Services, Inc. "¿Qué es Elastic Load Balancing?" [En línea]. Disponible: https://docs.aws.amazon.com/es_es/elasticloadbalancing/latest/application/introduction.html. [Accedido: 24-mar-2025].

[13] "Capa gratuita de AWS | Cloud computing gratis |AWS," *Amazon Web Services, Inc.* https://aws.amazon.com/es/free/?trk=2590dac4-c7c1-4106-8878-5344251a0baf&sc_channel=ps&ef_id=Cj0KCQjw8cHABhC-ARIsAJnY12xHcKEhsiN4rRT4lHntsTu1ChC0ohGXLb37dM3LTQuyer43IrIQussaAlOJEALw_wcB:G:s&s_kwcid=AL!4422!3!646879396464!e!!g!!aws%20free%20tier!19636893018!145054767999&gad_campaignid=19636893018&gbraid=0AAAADjHtp_tKvL5v7h-zyn9dz82_p7TG&gclid=Cj0KCQjw8cHABhC-ARIsAJnY12xHcKEhsiN4rRT4lHntsTu1ChC0ohGXLb37dM3LTQuyer43IrIQussaAlOJEALw_wcB&all-free-tier.sort-by=item.additionalFields.SortRank&all-free-tier.sort-order=asc&awsf.Free%20Tier%20Types=*all&awsf.Free%20Tier%20Categories=*all