Desarrollo de soluciones Cloud
Proyecto Entrega 5

INTEGRANTES:

Jheisson Orlando Cabezas Vera	**j.cabezasv@uniandes.edu.co**
Diego Alberto Rodríguez Cruz	**da.rodriguezc123@uniandes.edu.co**

DOCENTE:
Jesse Padilla Agudelo

UNIVERSIDAD DE LOS ANDES
BOGOTÁ D.C.
2025 – II

CONTENIDO

INTRODUCCIÓN 3
OBJETIVOS 4
Objetivo General 4
Objetivos Específicos 4
Modelo de Despliegue 5
Arquitectura propuesta (AS-IS) 5
Despliegue de infraestructura con Terraform 6
Funcionamiento de la aplicación 15
CONCLUSIONES 20
BIBLIOGRAFÍA 21

INTRODUCCIÓN

La entrega aborda la implementación de soluciones escalables en la capa *batch/worker* del backend, utilizando servicios de AWS como EC2, RDS, S3, SQS/Kinesis, CloudWatch y Auto Scaling para construir una arquitectura flexible y eficiente en la nube. El trabajo integra conocimientos previos del curso de Maestría en Ingeniería de Software y refuerza buenas prácticas de despliegue, monitoreo y administración en entornos distribuidos.

Además, el proyecto contempla varias alternativas de despliegue a elección del equipo, entre ellas: usar ECS, desplegar en EKS, reemplazar la capa *worker* por funciones en AWS Lambda o automatizar completamente el proceso mediante Terraform. Estas opciones permiten evaluar diversas estrategias de escalabilidad y automatización en arquitecturas modernas.

OBJETIVOS

Objetivo General

Implementar una arquitectura backend escalable y eficiente en la nube, utilizando servicios de AWS y técnicas de automatización, con el fin de garantizar un rendimiento óptimo ante cargas variables y asegurar buenas prácticas de despliegue, monitoreo y administración en entornos distribuidos.

Objetivos Específicos

- Diseñar y configurar una capa *batch/worker* escalable empleando servicios de AWS como EC2, SQS/Kinesis y Auto Scaling.
- Optimizar la infraestructura y los costos operativos mediante el uso adecuado de servicios administrados como RDS, S3 y CloudWatch.
- Evaluar y aplicar alternativas de despliegue avanzadas, tales como ECS, EKS o AWS Lambda, según la estrategia seleccionada por el equipo.
- Implementar automatización mediante Terraform, promoviendo la infraestructura como código y la repetibilidad de los entornos.
- Integrar buenas prácticas de monitoreo, despliegue y operación para asegurar la disponibilidad, escalabilidad y mantenibilidad del sistema.
- Consolidar los conocimientos adquiridos en el curso, aplicándolos en un entorno práctico que refleje escenarios reales en la industria del software.

Modelo de Despliegue

El despliegue de la entrega final contempla varios componentes de decisión libre por parte del equipo. Entre las alternativas se incluyen:

- Desplegar la aplicación en **Elastic Container Service (ECS)**, siempre y cuando este servicio no haya sido utilizado previamente.
- Realizar el despliegue sobre **Kubernetes (EKS)**.
- Reemplazar la capa *worker* por funciones en **AWS Lambda**.
- **Automatizar** por completo el proceso de despliegue utilizando **Terraform**

Arquitectura propuesta (AS-IS)

La arquitectura implementada está diseñada para proporcionar **alta disponibilidad, escalabilidad automática y resiliencia** ante fallos, aprovechando los servicios administrados de **Amazon Web Services (AWS)**.

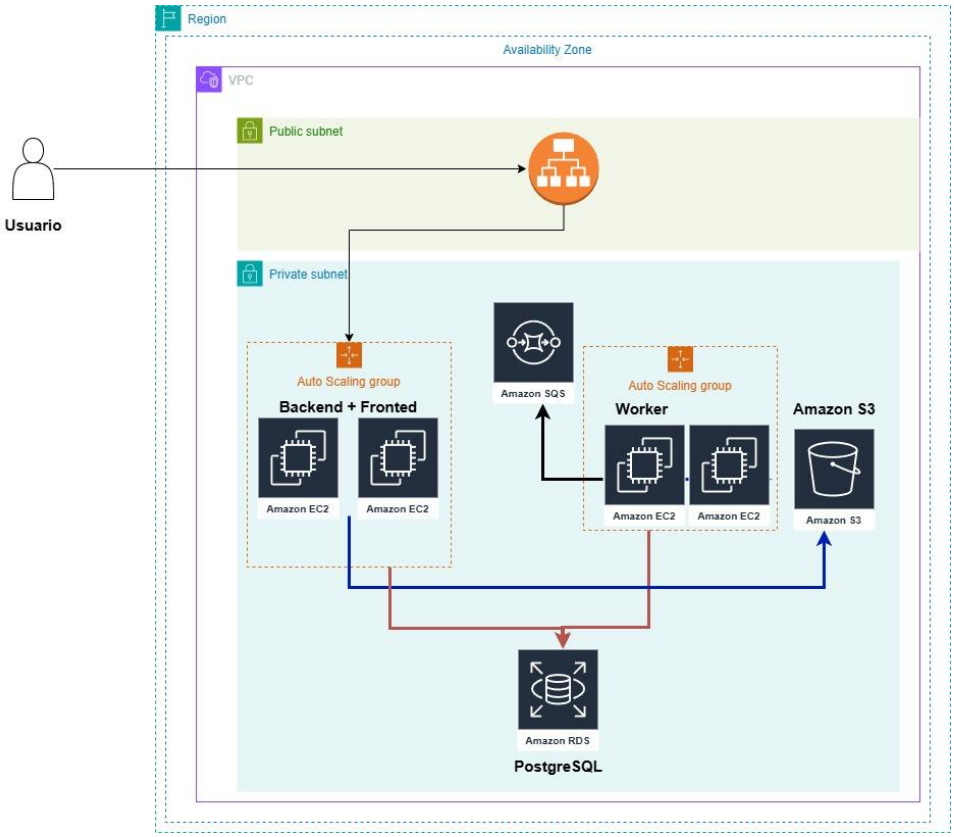

Figura 1. Arquitectura distribuida con escalamiento automático y SQS.

Despliegue de infraestructura con Terraform

Para el despliegue de la infraestructura se optó por utilizar la **AWS CLI** para la ejecución de **CloudFormation**, con el cual se crean los servicios necesarios para el despliegue. A continuación, se describen los pasos que se siguieron para la implementación de la infraestructura:

Paso 1 – Configurar AWS Cli

- ***Opción 1 – Exportación de variables***

➢ `export AWS_ACCESS_KEY_ID=$(aws configure get aws_access_key_id)`
➢ `export AWS_SECRET_ACCESS_KEY=$(aws configure get ws_secret_access_key)`
➢ `export AWS_SESSION_TOKEN=$(aws configure get aws_session_token)`

- ***Opción 2 – Configuración ~/.aws/credentials***

`cat ~/.aws/credentials`
Editar según corresponda las credenciales:
`[default]`
`aws_access_key_id=xxxxxxxxxxxx`
`aws_secret_access_key=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`
`aws_session_token=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx`

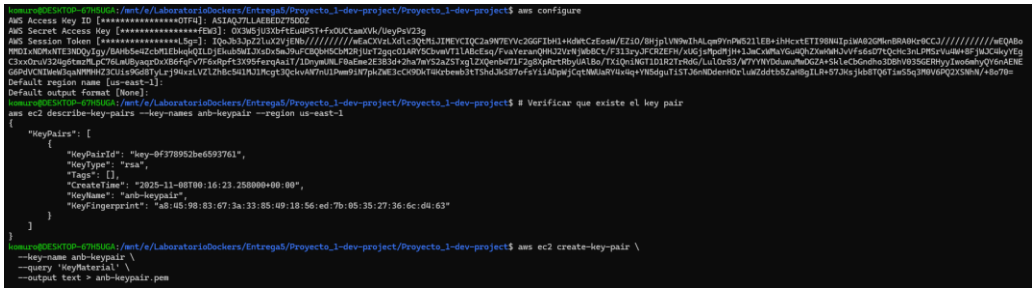

Figura 2. Configuración AWS Cli.

Paso 2 – Clonación del repositorio

Clonar el repositorio: Primero, asegúrese de tener git instalado en su máquina. Luego, clone el repositorio en su entorno local, para el caso de nosotros instalamos la WSL de Ubuntu para poder realizar el despliegue:

`git clone https://github.com/development-cloud-solutions/Proyecto_1.git`

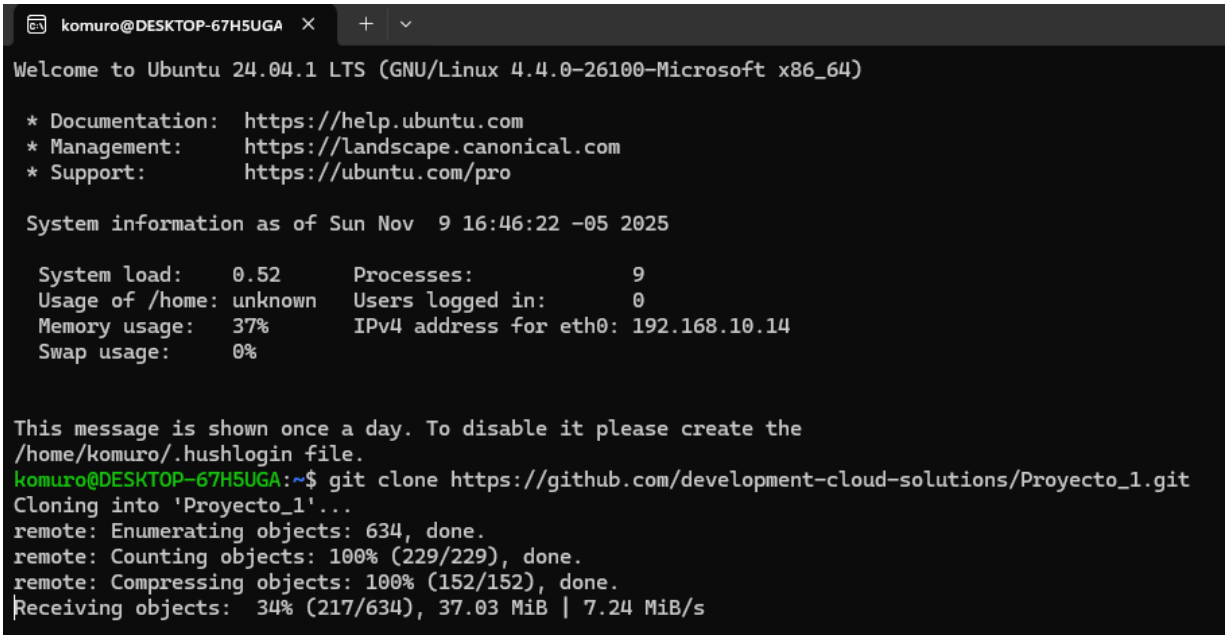

Figura 3. Arquitectura distribuida con escalamiento automático para el backend y fronted.

Paso 3 – Creación de la llave de conexión (anb-keypair.pem)

```
aws ec2 create-key-pair \
--key-name anb-keypair \
--query 'KeyMaterial' \
--output text > anb-keypair.pem
```

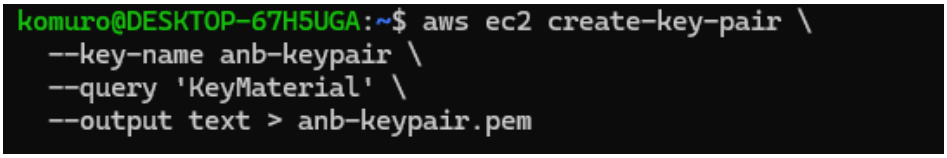

Figura 4. Creación de llave

```
chmod 400 anb-keypair.pem
```

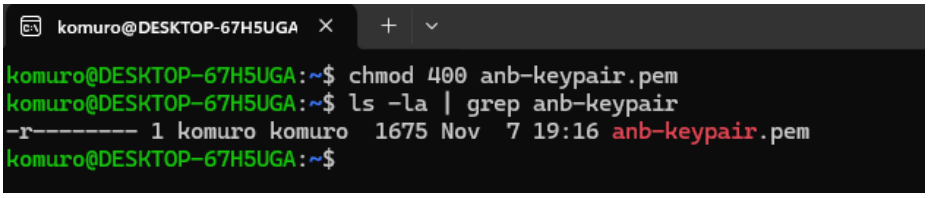

Figura 4. Asignación de permisos a la llave

Paso 4 – Generación del secreto JWT y la contraseña para RDS

En este paso se generan las credenciales necesarias para la autenticación y el acceso seguro a los servicios de la aplicación. Es importante recordar que estos valores deben manejarse mediante variables de entorno y nunca almacenarse directamente en el código fuente, con el fin de proteger la información sensible.

El secreto JWT (*JSON Web Token Secret*) se utiliza para firmar y verificar los tokens de autenticación. Se recomienda generar un valor aleatorio seguro utilizando openssl:

- # Generar JWT Secret de 32 bytes en formato hexadecimal

```
JWT_SECRET=$(openssl rand -hex 32)
```

Generar la contraseña para la base de datos RDS

La contraseña del usuario de la base de datos RDS debe ser segura y cumplir con las políticas de contraseñas de AWS (mínimo 8 caracteres, incluyendo mayúsculas, minúsculas, números y símbolos).
Durante el entorno de desarrollo o pruebas se puede asignar una contraseña fija; sin embargo, en producción debe generarse dinámicamente y almacenarse de forma segura.

Generar contraseña segura para RDS (solo para entornos de desarrollo)

```
DB_PASSWORD="MySecurePassword123!"
```

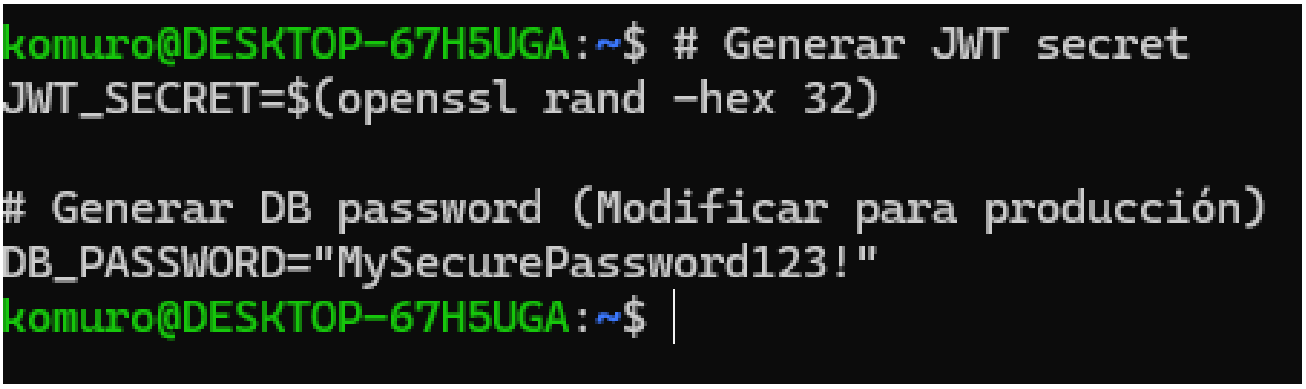

Figura 5. Generación de claves

Paso 5 – Instalación de Terraform

- Actualizar paquetes e instalar dependencias:

```
sudo apt update && sudo apt install -y gnupg software-properties-common curl
```

- Agregar la clave GPG de HashiCorp:

```
curl -fsSL https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor
-o /usr/share/keyrings/hashicorp-archive-keyring.gpg
```

- Añadir el repositorio de HashiCorp:

```
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/hashicorp-archive-keyring.gpg]
https://apt.releases.hashicorp.com $(lsb_release -cs) main" | sudo tee
/etc/apt/sources.list.d/hashicorp.list
```

- Actualizar e instalar Terraform:

```
sudo apt update && sudo apt install terraform
```

- Verificar instalación:

```
terraform version
```

Paso 6 – Ejecución del script de despliegue (Terraform)

Una vez configuradas las credenciales y variables de entorno necesarias, se procede a realizar el despliegue automatizado de la aplicación sobre la infraestructura de AWS. Para ello, se utiliza el script deploy.sh, ubicado dentro del directorio ./aws-terraform/deploy.sh del proyecto.
Este script tiene como objetivo provisionar los recursos en AWS (instancias EC2, balanceador de carga, grupos de *autoscaling*, entre otros) y configurar la aplicación web junto con su capa *worker* para el entorno de producción.

```
./aws-terraform/deploy.sh anb-production anb-keypair "$DB_PASSWORD" "$JWT_SECRET"
```

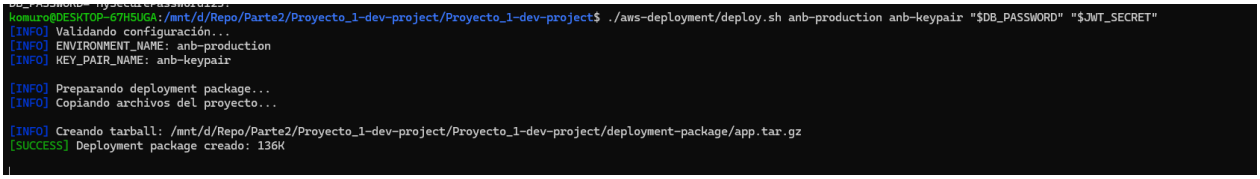

Figura 6. Deploy de infraestructura

Descripción de los parámetros:

- anb-production → nombre del entorno de despliegue. En este caso, corresponde al entorno de producción configurado en AWS.

- anb-keypair → nombre del key pair registrado en AWS EC2, utilizado para la autenticación SSH de las instancias creadas.
- "$DB_PASSWORD" → contraseña de la base de datos RDS, generada en el paso anterior. Se pasa como argumento para su uso seguro dentro del script.
- "$JWT_SECRET" → clave secreta utilizada para la generación y validación de tokens JWT en la aplicación.

Consideraciones importantes

- Asegúrese de ejecutar el comando desde la raíz del proyecto para que las rutas relativas dentro del script funcionen correctamente.
- Verifique que el *key pair* (anb-keypair) exista en la cuenta de AWS y esté correctamente configurado en la región seleccionada.
- Los valores de las variables $DB_PASSWORD y $JWT_SECRET deben haberse generado previamente y no deben registrarse en texto plano en ningún archivo del repositorio.
- Durante la ejecución, el script puede tardar varios minutos, ya que realizará el aprovisionamiento y la configuración completa de los recursos en AWS.

Paso 7 – Verificación del despliegue con terraform

Una vez ejecutado el script ./aws-terraform/deploy.sh, se inicia el proceso de provisionamiento automatizado de los recursos definidos en las plantillas de infraestructura.

Este proceso puede observarse desde en la pantalla de comando de donde se ejecuta el despliegue con terraform.

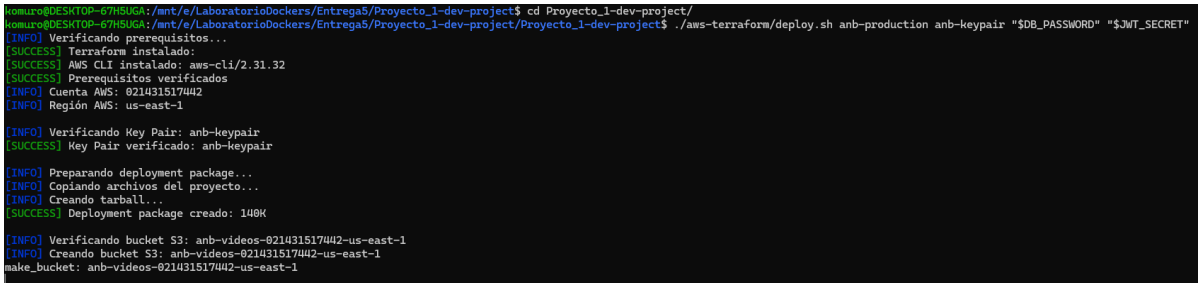

Figura 7. Deploy de infraestructura

```
[INFO] Verificando Key Pair: anb-keypair
[SUCCESS] Key Pair verificado: anb-keypair

[INFO] Preparando deployment package...
[INFO] Copiando archivos del proyecto...
[INFO] Creando tarball...
[SUCCESS] Deployment package creado: 140K

[INFO] Verificando bucket S3: anb-videos-021431517442-us-east-1
[INFO] Creando bucket S3: anb-videos-021431517442-us-east-1
make_bucket: anb-videos-021431517442-us-east-1
[SUCCESS] Bucket creado y configurado: anb-videos-021431517442-us-east-1

[INFO] Subiendo deployment package a S3...
[INFO] Destino: s3://anb-videos-021431517442-us-east-1/deployments/latest/app.tar.gz
upload: ../deployment-package/app.tar.gz to s3://anb-videos-021431517442-us-east-1/deployments/latest/app.tar.gz
[SUCCESS] Deployment package subido exitosamente

[INFO] Creando archivo terraform.tfvars...
[SUCCESS] Archivo terraform.tfvars creado

[INFO] Inicializando Terraform...
Initializing the backend...
Initializing modules...
- alb_autoscaling in modules/alb-autoscaling
- rds in modules/rds
- s3_iam in modules/s3-iam
- sqs in modules/sqs
- vpc in modules/vpc
- workers in modules/workers
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.100.0...
- Installed hashicorp/aws v5.100.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

[INFO] Validando configuración de Terraform...
Success! The configuration is valid.
```

Figura 8. Deploy de infraestructura

```
Terraform will perform the following actions:

  # aws_security_group.ec2 will be created
  + resource "aws_security_group" "ec2" {
      + arn                    = (known after apply)
      + description            = "Security group for EC2 instances (API and Workers)"
      + egress                 = [
          + {
              + cidr_blocks      = [
                  + "0.0.0.0/0",
                ]
              + description      = "Allow all outbound traffic"
              + from_port        = 0
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "-1"
              + security_groups  = []
              + self             = false
              + to_port          = 0
            },
        ]
      + id                     = (known after apply)
      + ingress                = [
          + {
              + cidr_blocks      = [
                  + "0.0.0.0/0",
                ]
              + description      = "API port for ALB health checks"
              + from_port        = 8080
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "tcp"
              + security_groups  = []
              + self             = false
              + to_port          = 8080
            },
          + {
              + cidr_blocks      = [
                  + "0.0.0.0/0",
                ]
              + description      = "HTTP from anywhere (for ALB and direct access)"
              + from_port        = 80
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "tcp"
```

Figura 9. Deploy de infraestructura

```
Changes to Outputs:
  + api_autoscaling_group_name    = "anb-production-api-asg"
  + application_url               = (known after apply)
  + db_connection_string          = (known after apply)
  + db_endpoint                   = (known after apply)
  + environment_name              = "anb-production"
  + load_balancer_dns             = (known after apply)
  + s3_bucket_arn                 = "arn:aws:s3:::anb-videos-021431517442-us-east-1"
  + s3_bucket_name                = "anb-videos-021431517442-us-east-1"
  + sqs_dlq_url                   = (known after apply)
  + sqs_queue_arn                 = (known after apply)
  + sqs_queue_name                = "anb-production-video-processing"
  + sqs_queue_url                 = (known after apply)
  + vpc_id                        = (known after apply)
  + worker_autoscaling_group_name = "anb-production-worker-asg"
  + worker_launch_template_id     = (known after apply)

Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"

========================================
[WARNING] REVISIÓN DEL PLAN
========================================

Terraform ha generado un plan de ejecución.
Revisa los cambios que se van a realizar.

¿Deseas aplicar este plan? (y/n): y

[INFO] Aplicando Terraform (esto puede tomar 10-15 minutos)...
module.s3_iam.aws_cloudwatch_log_group.api: Creating...
module.s3_iam.aws_cloudwatch_log_group.user_data: Creating...
module.vpc.aws_vpc.main: Creating...
module.sqs.aws_sqs_queue.dlq: Creating...
module.s3_iam.aws_cloudwatch_log_group.worker: Creating...
```

Figura 10. Deploy de infraestructura

Paso 7 – Ejecución de las migraciones de base de datos

Una vez completado el despliegue de la infraestructura y verificado que las instancias del grupo de Auto Scaling están saludables, se procede a ejecutar las migraciones SQL sobre la base de datos PostgreSQL desplegada en Amazon RDS.

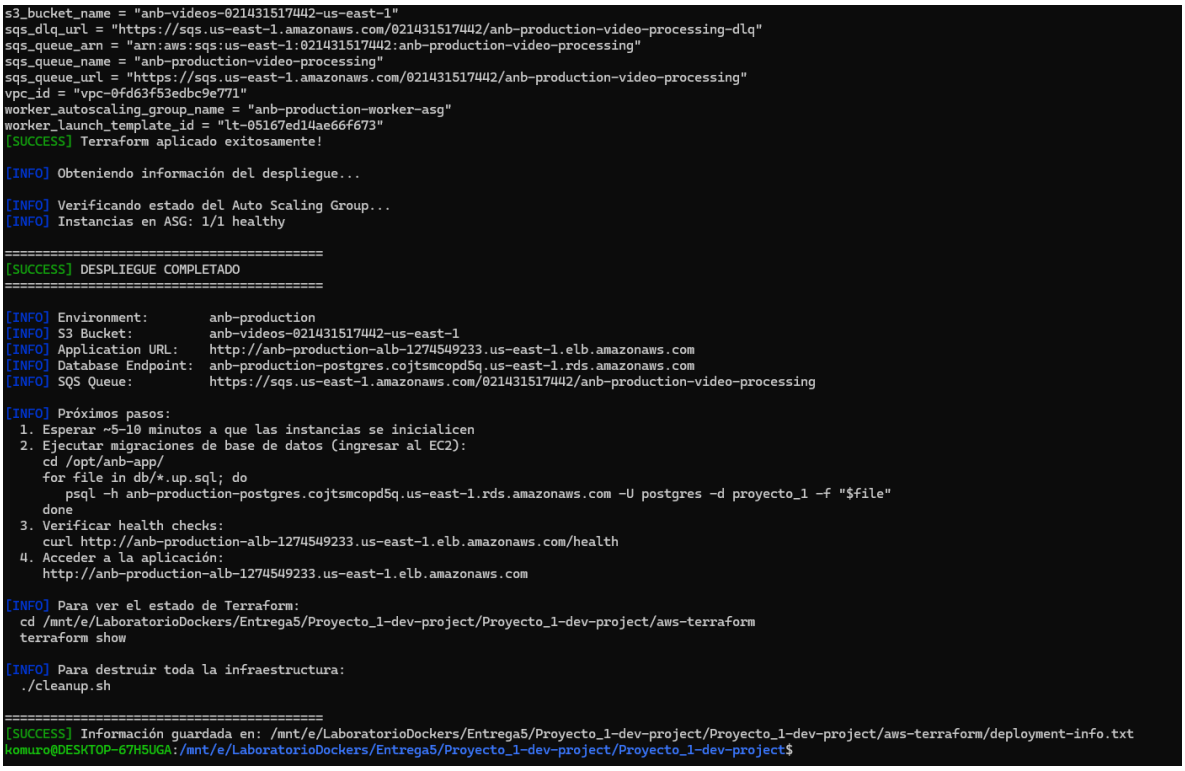

Figura 11. Verificación Terraform

Estas migraciones crean las tablas, índices y estructuras requeridas por la aplicación para su correcto funcionamiento.

I. Obtener la información de conexión

Del resultado del despliegue, el sistema muestra los siguientes datos de conexión:

- *Endpoint RDS:* anb-production-postgres.cojtsmcopd5q.us-east-1.rds.amazonaws.com
- *Base de datos:* proyecto_1
- *Usuario:* postgres

Asegúrate de tener la variable de entorno con la contraseña de la base de datos ($DB_PASSWORD) configurada antes de continuar.

II. Ejecutar las migraciones

En el servidor donde está desplegada la aplicación (o desde un cliente con acceso a la base de datos RDS), ejecuta el siguiente comando:

```
for file in /opt/anb-app/db/*.up.sql; do
    psql -h anb-production-postgres.cojtsmcopd5q.us-east-
1.rds.amazonaws.com -U postgres -d proyecto_1 -f "$file"
done
```

Este script recorre todos los archivos SQL con extensión .up.sql ubicados en el directorio /opt/anb-app/db/ y los ejecuta de manera secuencial sobre la base de datos proyecto_1.

III. Confirmar la correcta aplicación de las migraciones

Para validar que las migraciones se aplicaron correctamente, puedes conectarte al RDS usando psql:

```
psql -h anb-production-postgres.cojtsmcopd5q.us-east-
1.rds.amazonaws.com -U postgres -d proyecto_1
```

Luego, dentro de la consola de PostgreSQL, verifica la existencia de las tablas:

```
\dt
```

Deberías ver listadas las tablas definidas en tus archivos .up.sql.

```
[ec2-user@ip-10-0-1-237 ~]$ psql -h anb-production-postgres.cojtsmcopd5q.us-east-1.rds.amazonaws.com -U postgres -d proyecto_1
Password for user postgres:
psql (15.14, server 15.12)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, compression: off)
Type "help" for help.

proyecto_1=> \dt
            List of relations
 Schema |     Name      | Type  |  Owner
--------+---------------+-------+----------
 public | task_results  | table | postgres
 public | user_sessions | table | postgres
 public | users         | table | postgres
 public | videos        | table | postgres
 public | votes         | table | postgres
(5 rows)

proyecto_1=>
```

Figura 12. Verificación importación base de datos en el RDS

Comprobación del despliegue

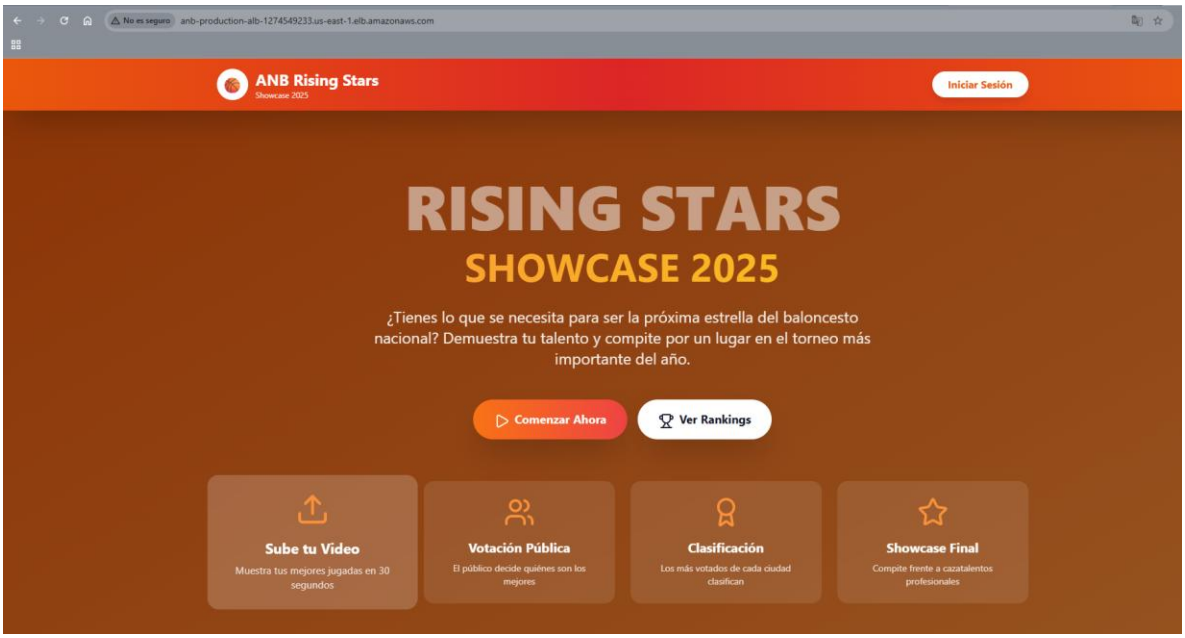

Figura 13. Portal de votación

Con esto se valida que el despliegue de la aplicación se realizó de manera exitosa. El siguiente paso consiste en verificar el funcionamiento operativo del sitio, asegurando que la comunicación con el servicio **Amazon SQS** se encuentre activa y que las tareas en la capa *worker* estén procesando correctamente los mensajes enviados desde la API web.

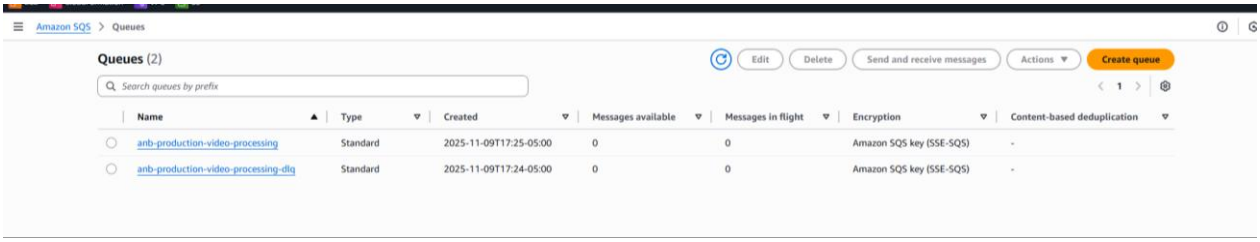

Figura 14. Amazon SQS

Funcionamiento de la aplicación

El funcionamiento de la aplicación inicia con el acceso desde el navegador web a la dirección pública asignada al **WebServer**. Allí se despliega el portal de autenticación (figura 11), el cual permite a los usuarios registrarse (figura 14) e iniciar sesión para acceder a las funcionalidades principales de la solución.

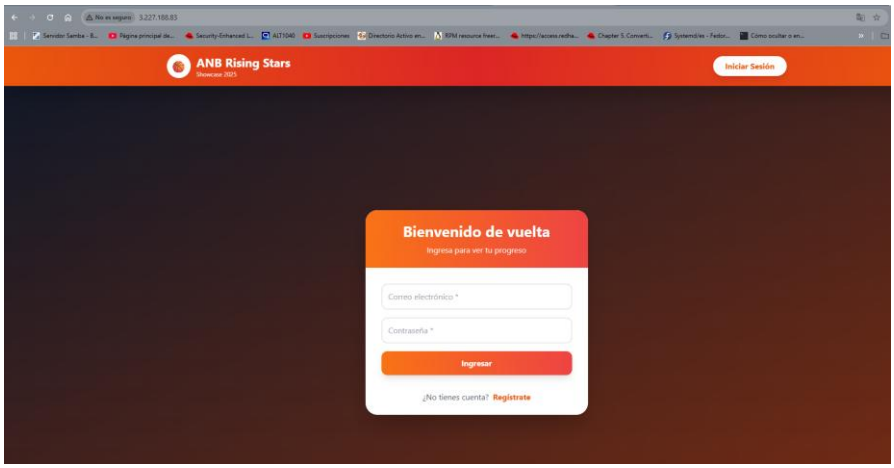

Figura 15. Portal de autenticación

Como se evidencia en la figura 15 el panel de autenticación cuenta con un apartado de registro, esto con el fin de que los usuarios puedan registrase para iniciar su proceso de votaciones.

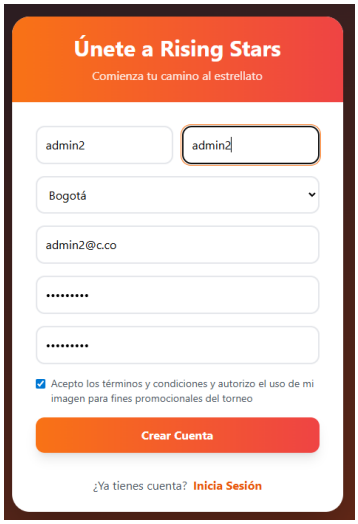

Figura 16. Registro de usuario

Una vez autenticados, los usuarios pueden interactuar con el sistema mediante la API desplegada en la instancia correspondiente. Esta API fue levantada mediante contenedores Docker definidos en el CloudFormation, garantizando la estandarización y portabilidad del servicio.

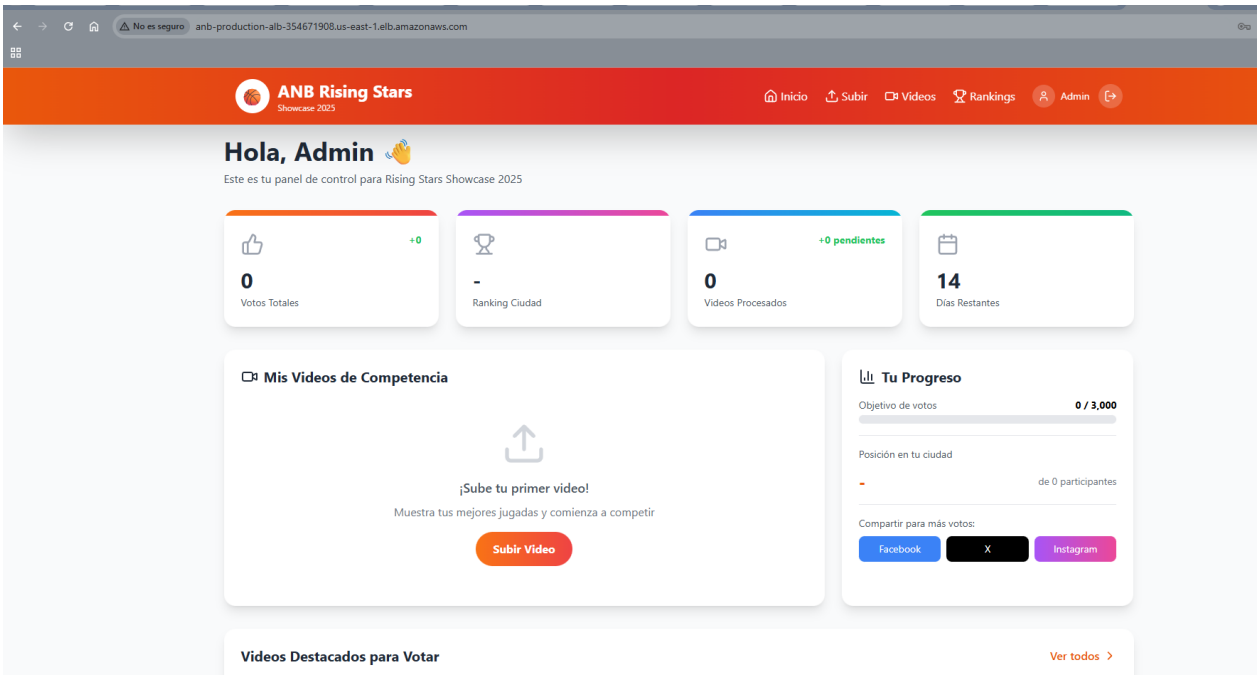

Figura 17. Panel de votación y cargue de video

Dentro de las funciones de carga de video, se ofrece la opción de mantener el video en modo público (visible para votación) o configurarlo como privado para uso interno.

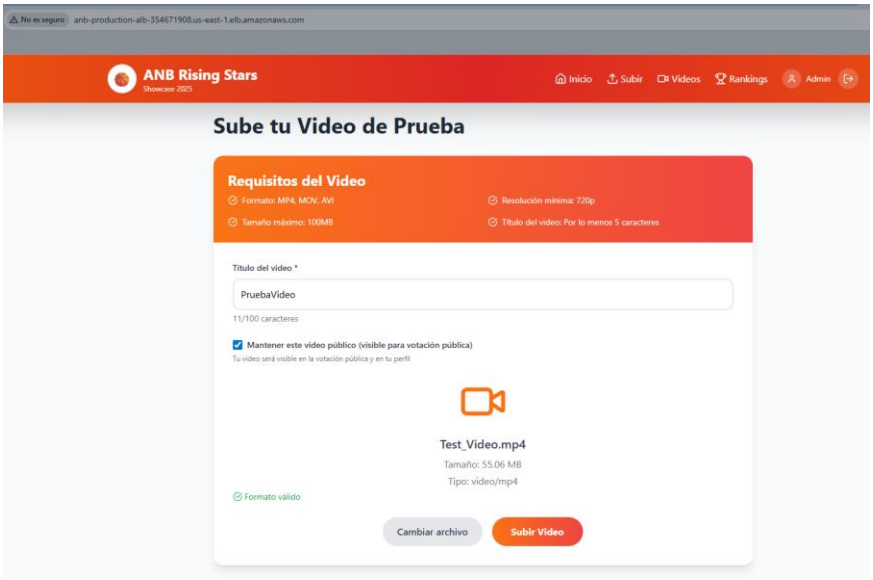

Figura 11. Panel de cargue de video

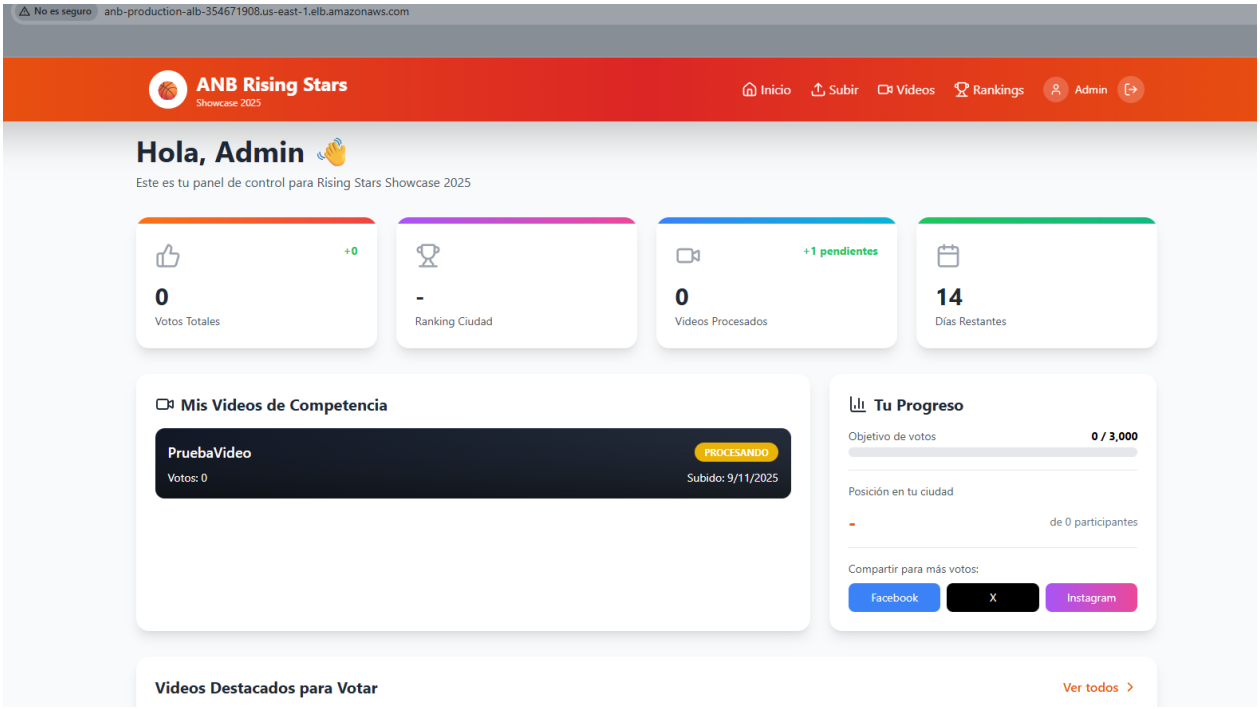

Figura 18. Procesamiento del video

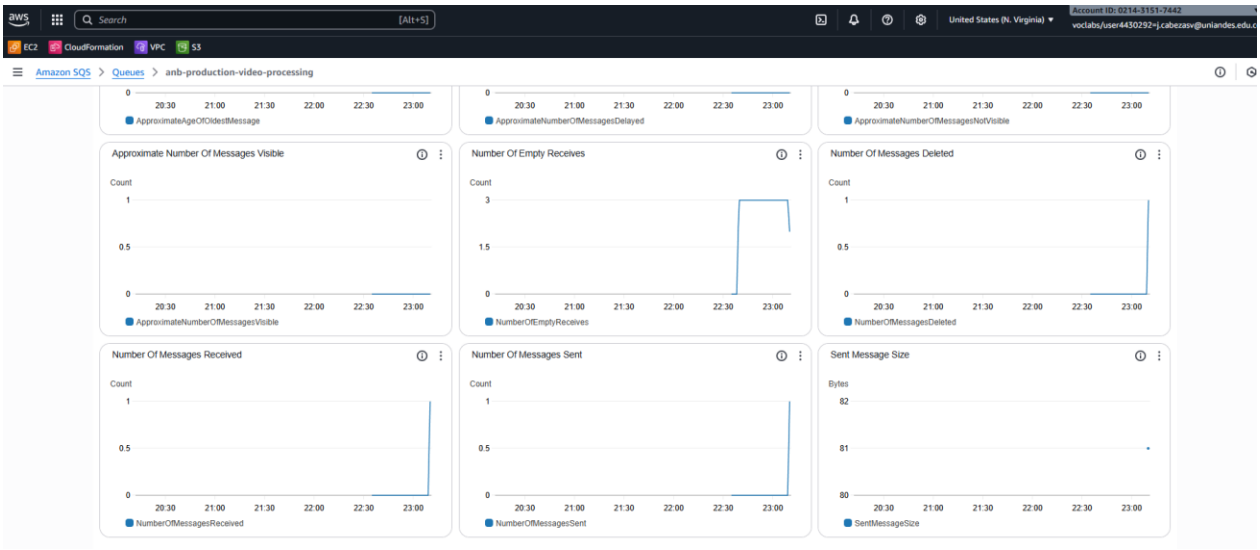

Figura 19. Verificación de procesamiento del video en Amazon SQS

La **Figura 19** presenta la ejecución del servicio **Amazon SQS**, evidenciando que la carga y el procesamiento de mensajes se realizaron exitosamente. Este resultado confirma el correcto funcionamiento del mecanismo de mensajería asíncrona, garantizando una comunicación efectiva entre la capa web y los procesos *worker* dentro de la arquitectura distribuida.

Una vez que los videos son cargados y procesados por los *workers*, los demás usuarios pueden visualizarlos en la plataforma y emitir sus votos sin interrupciones.

Gracias a esta separación de responsabilidades entre los componentes, la aplicación permite que las acciones de los usuarios —como subir contenido, votar por los videos o consultar resultados en tiempo real— se ejecuten de forma fluida, mientras que el procesamiento pesado se realiza de manera independiente y confiable en los *workers* administrados a través de **Amazon SQS**.

Con ello, se confirma que la **aplicación de votación de videos** opera correctamente sobre la arquitectura distribuida desplegada en **AWS**, demostrando una integración exitosa entre los servicios de cómputo, mensajería y procesamiento en la nube.

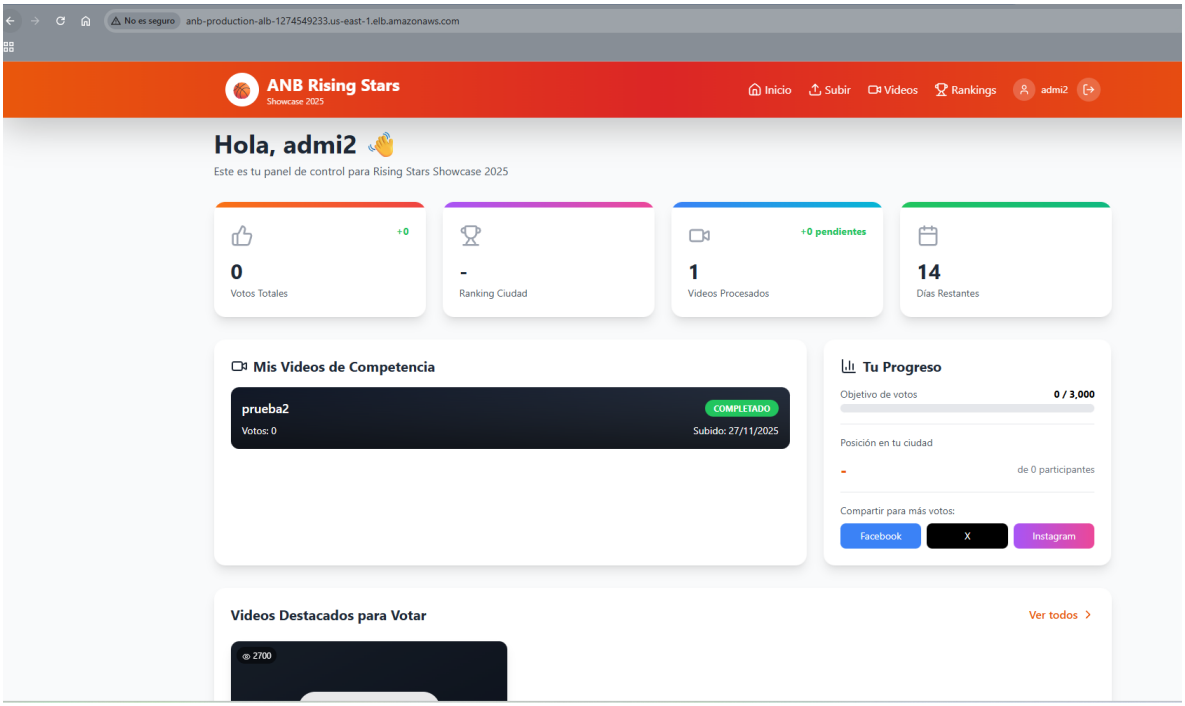

Figura 20. Confirmación de carga del video

Repositorio: [GitHub - development-cloud-solutions/Proyecto_1]

CONCLUSIONES

- El uso de **Terraform** para el despliegue de la infraestructura demostró ser una estrategia altamente efectiva para garantizar consistencia, escalabilidad y control en entornos distribuidos basados en AWS. La adopción de infraestructura como código permitió definir, versionar y reproducir entornos completos de manera confiable, reduciendo errores manuales y facilitando la colaboración dentro del equipo.
- Asimismo, Terraform proporcionó una capa de abstracción que simplificó la administración de recursos como EC2, RDS, S3, SQS/Kinesis y componentes de red, permitiendo gestionar la arquitectura de forma declarativa y auditable. Esta automatización no solo optimizó los tiempos de despliegue, sino que también contribuyó a una mayor eficiencia operativa, especialmente al manejar entornos con alto dinamismo o requerimientos de escalabilidad.
- La integración del flujo de trabajo con Terraform fortaleció las buenas prácticas de DevOps, promoviendo entornos reproducibles y facilitando la evolución del sistema ante nuevas necesidades. En conjunto, la experiencia evidenció que el despliegue automatizado mediante Terraform es una herramienta clave para garantizar infraestructuras robustas, mantenibles y alineadas con los principios modernos de ingeniería de software.

BIBLIOGRAFÍA

- Amazon Web Services Academy. (s. f.). *[Laboratorio de aprendizaje de AWS Academy]*. AWS Academy. Recuperado el 29 de septiembre de 2025, de https://awsacademy.instructure.com/courses/130819/modules/items/12511346