Desarrollo de soluciones Cloud
Proyecto Entrega 3

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

En la actualidad, la computación en la nube, las aplicaciones web requieren infraestructuras flexibles y altamente disponibles que les permitan adaptarse de manera eficiente a la variabilidad de la demanda. Los proveedores públicos de Infraestructura como Servicio (IaaS) ofrecen un conjunto de herramientas y servicios que facilitan la implementación de arquitecturas escalables, resilientes y de alto rendimiento. Sin embargo, para aprovechar plenamente estas capacidades, es necesario comprender y aplicar de forma adecuada las consideraciones técnicas que intervienen en el proceso de escalado, balanceo de carga, almacenamiento distribuido y monitoreo de recursos.

OBJETIVOS

Objetivo General

El propósito de este trabajo es comprender y aplicar las consideraciones técnicas necesarias para escalar la capa web de una aplicación en la infraestructura de un proveedor público de IaaS. Para ello, se implementarán políticas de auto escalado en los servidores web, acompañadas de un sistema de monitoreo automatizado que permita gestionar su escalabilidad de manera dinámica. Asimismo, se integrará un balanceador de carga que optimice la distribución del tráfico entre las distintas instancias del servidor, garantizando un uso eficiente de los recursos y una alta disponibilidad del servicio. Además, se configurará e implementará un sistema de almacenamiento de objetos que responda a las necesidades de escalabilidad y rendimiento de la aplicación web. Finalmente, se realizarán pruebas de capacidad y de estrés con el fin de evaluar el desempeño global de la aplicación, considerando tanto la capa web como los procesos asíncronos involucrados en su operación.

Objetivos Específicos

- Comprender las consideraciones técnicas necesarias para escalar la capa web de una aplicación dentro de la infraestructura de un proveedor público de IaaS.
- Implementar políticas de auto escalado para los servidores web de la aplicación, junto con un sistema de monitoreo automatizado que permita gestionar dinámicamente su capacidad.
- Integrar un balanceador de carga que distribuya de forma eficiente el tráfico entre las distintas instancias del servidor web.
- Configurar e implementar un sistema de almacenamiento de objetos que satisfaga los requerimientos de disponibilidad y escalabilidad de la aplicación web.
- Realizar pruebas de capacidad y de estrés para evaluar el rendimiento global de la aplicación, considerando tanto la capa web como los componentes de procesamiento asíncrono.

Modelo de Despliegue Básico en la Nube Publica de AWS

Con el fin de lograr que la aplicación web sea capaz de escalar, se deberá modificar el aplicativo desarrollado en la Entrega 2, tal que responda a una demanda de usuarios en aumento. La compañía ha identificado que se deben implementar los siguientes servicios:

- Amazon EC2: Ejecución de la capa web y la capa worker. Por decisión de negocio, se han seleccionado instancias de cómputo con 2 vCPU, 2 GiB de RAM y 30 GiB de almacenamiento.
- Amazon RDS: Almacenamiento de la información de la aplicación en una base de datos relacional. En las fases iniciales de la implementación, puede utilizarse una instancia de EC2 y sustituirla por RDS únicamente durante las pruebas de carga.
- Amazon S3: Almacenamiento de los videos originales y los procesados.
- Amazon CloudWatch: Monitoreo de las instancias y los demás servicios empleados.
- Autoscaling: Servicio para escalar la capa web en función de los usuarios concurrentes.
- Load Balancer: Distribución de la carga entre las instancias que exponen el API REST.

Para implementar el modelo propuesto y desplegar la aplicación se contemplan las siguientes actividades:

I. (30%) Acoplar un balanceador de carga en la arquitectura.
II. (20%) Definir e implementar una estrategia para que los servidores web escalen de manera automática, fijando un máximo de tres instancias. Se deberán establecer las condiciones de escalamiento mediante los servicios de monitoreo, auto-scaling y balanceo de cargas.
III. (10%) Configurar un bucket en el servicio de almacenamiento de objetos de Amazon S3 para guardar los videos pertenecientes a los usuarios, tanto los originales como los procesados.
IV. (10%) La aplicación web debe satisfacer la totalidad de los requerimientos funcionales previamente estipulados y detallados en el enunciado del proyecto. Lo anterior implica que se mantiene la instancia que soporta la capa worker y el sistema de base de datos. Valide el correcto funcionamiento de cada uno de los *endpoints definidos en la primera entrega*.

Arquitectura actual

La arquitectura actual se encuentra desplegada en la nube de **Amazon Web Services (AWS)** y está compuesta por varios componentes distribuidos. Sin embargo, no cuenta con mecanismos de **escalamiento automático**, por lo que, en caso de falla de alguna instancia, la arquitectura podría experimentar una **interrupción en el servicio**. A continuación, se describen los principales elementos que la conforman:

- **Web Server (EC2):** Atiende las solicitudes de los usuarios y gestiona tareas mediante **Redis**, exponiendo los *endpoints* a través de una **Elastic IP** ubicada en la subred pública.
- **Worker (EC2):** Procesa las tareas enviadas desde el servidor web y mantiene conexión con el sistema de archivos compartido y la base de datos.
- **NFS (EC2):** Funciona como servidor de archivos compartidos, centralizando el almacenamiento de los archivos originales y procesados.
- **Amazon RDS (PostgreSQL):** Proporciona un sistema de base de datos relacional administrado, confiable y escalable para la persistencia de los datos.

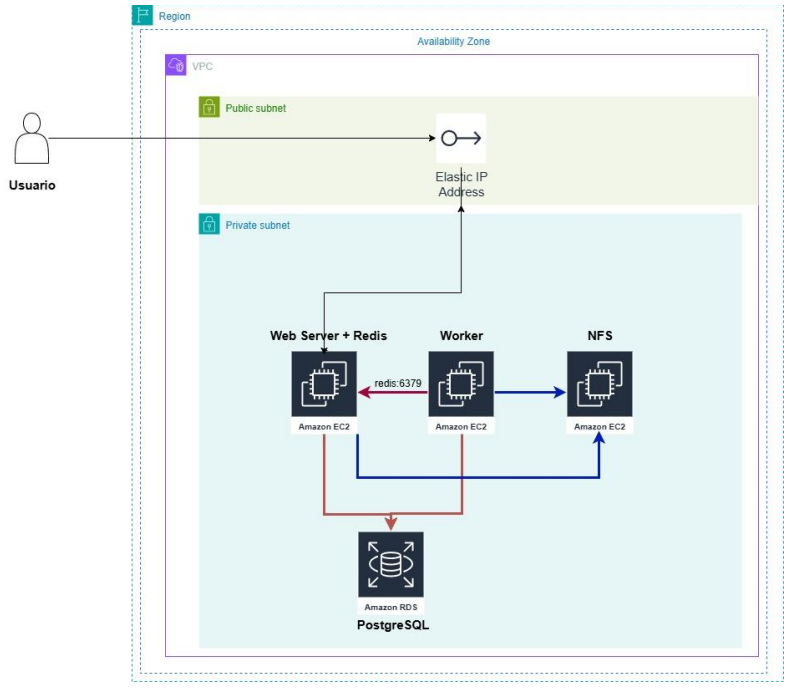

Figura 1. Arquitectura distribuida sin escalamiento automático

Arquitectura propuesta

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

Descripción general:

1. Usuario y acceso público:
   El usuario accede a la aplicación a través de Internet, ingresando por la subred pública de la VPC. El tráfico es dirigido al balanceador de carga, que distribuye las solicitudes hacia las instancias disponibles del grupo de auto escalado.
2. Capa web y Redis (Auto Scaling Group):
   En la subred privada se encuentra un grupo de Auto Scaling (ASG) que contiene las instancias Amazon EC2 encargadas de ejecutar el servidor web y el servicio de Redis.
   - El Auto Scaling Group permite aumentar o reducir dinámicamente el número de instancias EC2 según la carga de trabajo (por ejemplo, basado en el uso de CPU o la cantidad de conexiones activas).
   - Redis actúa como sistema de cola o caché, gestionando las tareas enviadas al *worker* y mejorando la velocidad de respuesta.
3. Worker (procesamiento asíncrono):
   El componente Worker, desplegado en una instancia EC2 separada, recibe las tareas desde Redis y ejecuta los procesos de backend o de procesamiento intensivo. Este diseño desacopla el procesamiento de las peticiones web, mejorando la escalabilidad y el rendimiento general del sistema.
4. Almacenamiento de objetos (Amazon S3):
   El servicio Amazon S3 reemplaza el antiguo servidor NFS, proporcionando un sistema de almacenamiento de objetos altamente disponible y escalable. Aquí se almacenan tanto los archivos originales como los resultados procesados.
5. Base de datos (Amazon RDS – PostgreSQL):
   Los datos estructurados de la aplicación se gestionan en Amazon RDS con PostgreSQL, un servicio administrado que garantiza disponibilidad, recuperación ante fallos y escalabilidad vertical.
6. Escalabilidad y resiliencia:
   - El Auto Scaling Group se encarga de ajustar automáticamente la capacidad de la capa web y Redis según la demanda.

- El balanceador de carga distribuye el tráfico de manera equitativa entre las instancias activas.
- En caso de que una instancia falle, el ASG lanza una nueva de forma automática, manteniendo la continuidad del servicio.

Beneficios principales:

- Escalabilidad automática según la demanda.
- Alta disponibilidad ante fallos de instancias.
- Separación de responsabilidades entre capa web, procesamiento y almacenamiento.
- Uso de servicios administrados que reducen la carga operativa.

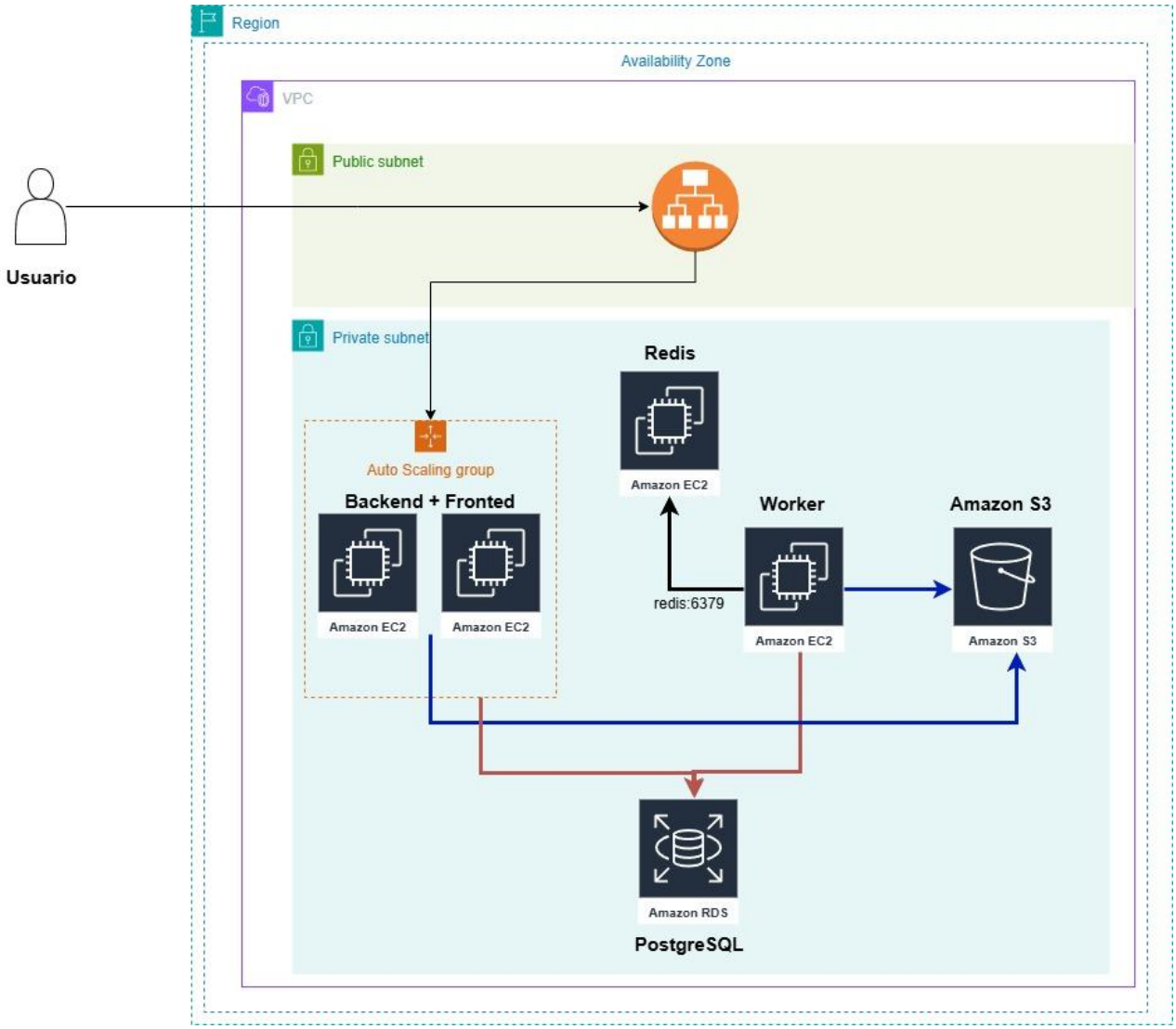

Figura 2. Arquitectura auto escalable en AWS

Configuración del servicio Amazon RDS

Para la creación de la base de datos se optó por utilizar la **AWS CLI**, debido a que la interfaz web de AWS presentaba errores durante el proceso de configuración. A continuación, se describen los pasos que se siguieron para la implementación del servicio:

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

[default]
aws_access_key_id=xxxxxxxxxxxx
aws_secret_access_key=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
aws_session_token=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Paso 2 – Creación del grupo de la subnet

```
aws rds create-db-subnet-group \
--db-subnet-group-name anb-subnet-group \
--db-subnet-group-description "Subnet group for RDS in VPC vpc-062d602ddb72afa4a" \
--subnet-ids subnet-0894c52b729e243fc subnet-086cdf64ced7c934e \
--region us-east-1
```

Paso 3 – Creación de instancia RDS

```
aws rds create-db-instance \
--db-instance-identifier anb-postgres \
--db-instance-class db.t3.micro \
--engine postgres \
--engine-version 15 \
--master-username postgres \
--master-user-password 'Password' \
--allocated-storage 20 \
--db-subnet-group-name anb-subnet-group \
--vpc-security-group-ids sg-054b228f2137c7b3e \
--backup-retention-period 0 \
--no-publicly-accessible \
--region us-east-1
```

Figura 3. Creación de instancia RDS

```
[postgres@ip-172-21-2-60 tmp]$ psql -U postgres -d proyecto_1 -f backup.sql
SET
SET
SET
SET
SET
 set_config
------------

(1 row)

SET
SET
SET
SET
CREATE EXTENSION
COMMENT
CREATE FUNCTION
ALTER FUNCTION
CREATE FUNCTION
ALTER FUNCTION
SET
SET
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
ALTER TABLE
ALTER SEQUENCE
CREATE TABLE
ALTER TABLE
CREATE MATERIALIZED VIEW
ALTER TABLE
CREATE TABLE
ALTER TABLE
CREATE SEQUENCE
```

Figura 4. Importación base de datos

Como se muestra en la figura 4, se evidencia el cargue de la data en el servicio de RDS.

Políticas de red VPC

En nuestro caso, la VPN fue creada mediante código de *CloudFormation*, con el fin de facilitar el diseño de la arquitectura de red y mantener un mejor control sobre su implementación.

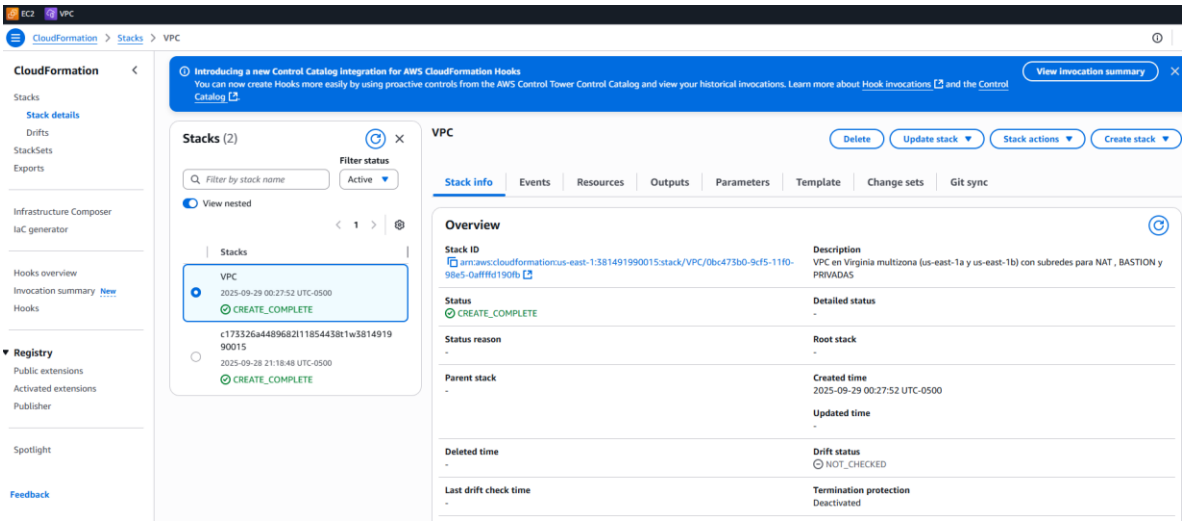

Figura 5. Importación de código para VPC

Estructura de Red

- **VPC**:
  - 172.21.0.0/16 en región **us-east-1**.
  - Etiquetada como Laboratorio.
- **Subredes**:
  - **Públicas**:
    - BastionA: 172.21.10.0/24 en **us-east-1a**.
    - NATA: 172.21.1.0/24 en **us-east-1a**.
    - NATB: 172.21.3.0/24 en **us-east-1b**.
  - **Privadas**:
    - PrivateA: 172.21.2.0/24 en **us-east-1a**.
    - PrivateB: 172.21.4.0/24 en **us-east-1b**.
- **Disponibilidad**: Arquitectura multizona (**AZ A y B**) para alta disponibilidad.

Conectividad

- **Internet Gateway (IGW)**:

  Asociado a la VPC para salida/entrada de tráfico público.

- **NAT Gateways**:

  - DEVICENATA en PublicNATA (AZ A).
  - DEVICENATB en PublicNATB (AZ B).
  - Cada subred privada enruta tráfico a Internet mediante su NAT correspondiente.

- **Tablas de ruteo**:

  - PublicRouteTable: redirige 0.0.0.0/0 hacia el IGW.
  - PrivateRouteTableA y PrivateRouteTableB: redirigen 0.0.0.0/0 hacia sus respectivos NATs.

Permite entrada desde Internet a:

- **HTTP (80)** desde 0.0.0.0/0.
- **HTTP (8080)** desde 0.0.0.0/0.
- **HTTPS (443)** desde 0.0.0.0/0.
- **SSH (22)** desde 0.0.0.0/0.
- **PostgresSQL (5432)** desde 172.21.0.0/16.
- **Redis (6379)** desde 172.21.0.0/16.
- **Puertos efímeros (1024–65535)** desde Internet.
- **Salida (Outbound)**: todo tráfico permitido hacia 0.0.0.0/0.

ACL Privada

- **Inbound**: permite todo (0–65535) desde 0.0.0.0/0.
- **Outbound**: permite todo (0–65535) hacia 0.0.0.0/0.
- Asociado a las subredes privadas A y B.

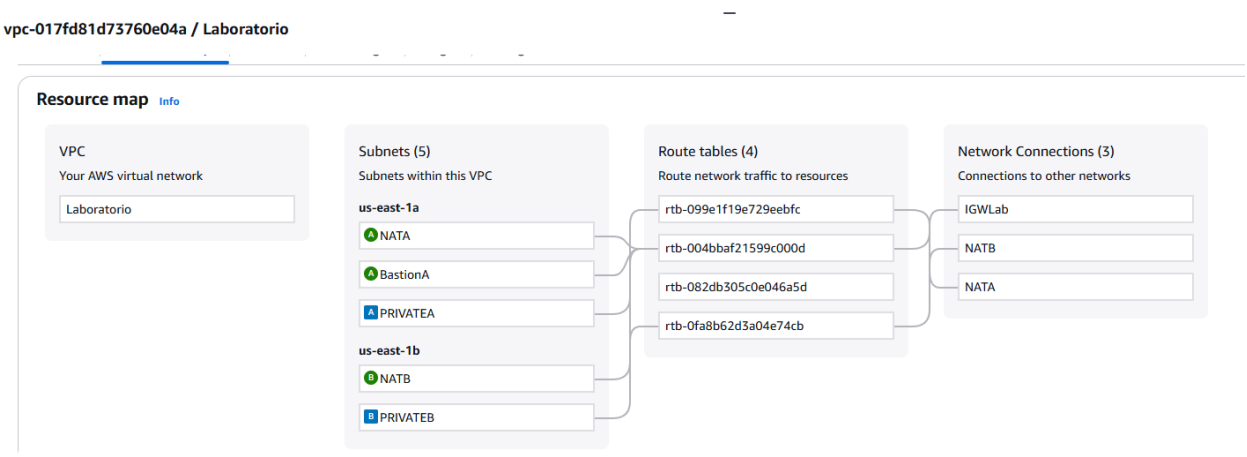

Figura 6. Diagrama de red AWS

Implementación de arquitectura propuesta

Para la implementación de la arquitectura en la nube propuesta, se crearon tres instancias principales: una instancia bastion para la conexión segura a los servidores, una instancia worker destinada al procesamiento de videos y una instancia Redis para la gestión de caché.

Además, se desarrolló una plantilla (template) que permite el auto escalado de las instancias backend y frontend, la cual se encuentra almacenada en el repositorio Git. La ejecución de esta plantilla se realiza mediante un script Shell, que despliega toda la infraestructura como código (IaC).

Propósito general

Este template **crea la capa web de una aplicación multi-AZ (alta disponibilidad)** dentro de una **VPC existente**, automatizando:

- Balanceo de carga (Load Balancer),
- Auto Scaling Group (instancias EC2 backend),
- Configuración de seguridad (Security Groups),
- Montaje de un bucket S3 con s3fs,
- Despliegue automático de una aplicación con Docker.

Componentes principales

Parámetros

Permiten personalizar la plantilla sin modificar el código:

- VpcId, Subnets, SubnetsELB: VPC y subredes donde se desplegará la capa web.

- AZs: Zonas de disponibilidad (Multi-AZ).
- KeyName: Par de claves SSH existente.
- InstanceType, InstanceCount: Tipo y número de instancias EC2.
- BucketName: Bucket S3 que se montará con s3fs.
- Credenciales temporales (AWSAccessKeyId, AWSSecretAccessKey, AWSSessionToken).
- WebServerPort: Puerto del servidor web (por defecto 80).

Recursos creados

Grupos de seguridad

- **ALBSGWebBckLab**: Permite tráfico HTTP (puerto 80) hacia el Load Balancer.
- **WebSecurityGroup**: Controla acceso a las instancias backend (HTTP/HTTPS/SSH, puertos 80, 443, 22, 8080, 5432, etc.) desde subredes específicas y el Load Balancer.

Balanceador de carga (ALB)

- **lbBckLab**: Load Balancer público (internet-facing).
- **ALBSGWebWebBckLab**: Target Group para enrutar tráfico HTTP hacia las instancias backend.
- **ALBListenerWebBckLab**: Listener en puerto 80 que reenvía tráfico al Target Group.

Capa de cómputo (EC2 Auto Scaling Group)

- **LaunchConfig**: Define cómo se lanzan las instancias EC2:
  - Usa una AMI específica (Amazon Linux).
  - Instala y configura:
    - docker, git, s3fs, postgresql, etc.
    - Monta el bucket S3 en /mnt/s3full con s3fs y enlaza carpetas locales (/app/uploads, /app/processed).
    - Clona un repositorio Git (Proyecto_1).
    - Levanta contenedores con docker compose.
- **WebServerGroup**: Auto Scaling Group que:
  - Lanza instancias en las AZs dadas.
  - Se integra con el Load Balancer.
  - Escala entre 2 y 6 instancias según la carga.

Escalamiento automático (Auto Scaling Policies)

- **WebServerScaleUpPolicy**: Aumenta 2 instancias si CPU > 65% durante 3 min.
- **WebServerScaleDownPolicy**: Reduce 1 instancia si CPU < 25% durante 10 min.

- **CPUAlarmHigh** y **CPUAlarmLow**: Alarmas de CloudWatch que activan las políticas anteriores.

WaitConditions

- **WaitHandleWeb** y **WaitConditionWeb**: Sincronizan el despliegue para asegurar que la configuración de las instancias EC2 se complete correctamente antes de continuar.

Salidas (Outputs)

Muestra los principales recursos creados:

- ARN del Load Balancer.
- Nombre del bucket montado.
- Grupo de seguridad del LB.
- Auto Scaling Group.
- Configuración de lanzamiento (Launch Configuration).

Tras el despliegue de la arquitectura, se confirma que el portal está disponible y funcionando correctamente, ofreciendo el servicio esperado.

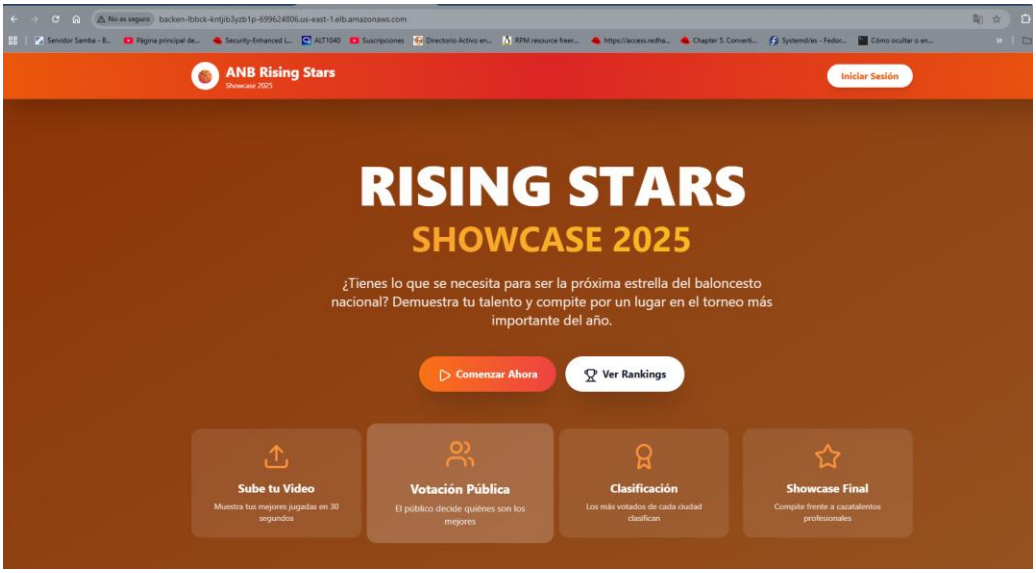

Figura 7. Portal de votación

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

8), el cual permite a los usuarios registrarse (figura 9) e iniciar sesión para acceder a las funcionalidades principales de la solución.

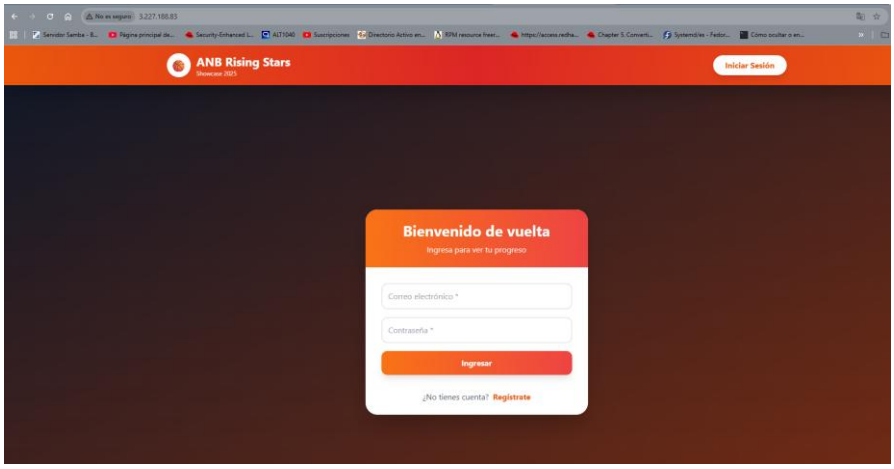

Figura 8. Portal de autenticación

Como se evidencia en la figura 8 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.

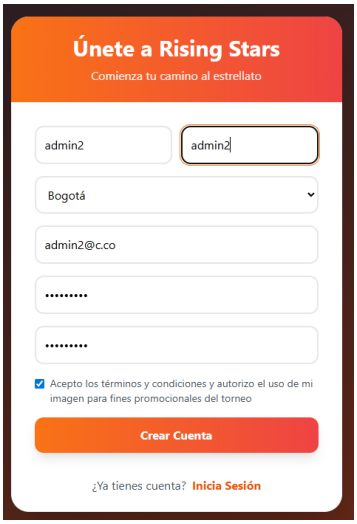

Figura 9. Registro de usuario

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 los archivos docker-compose.api.yml, garantizando la estandarización y portabilidad del servicio.

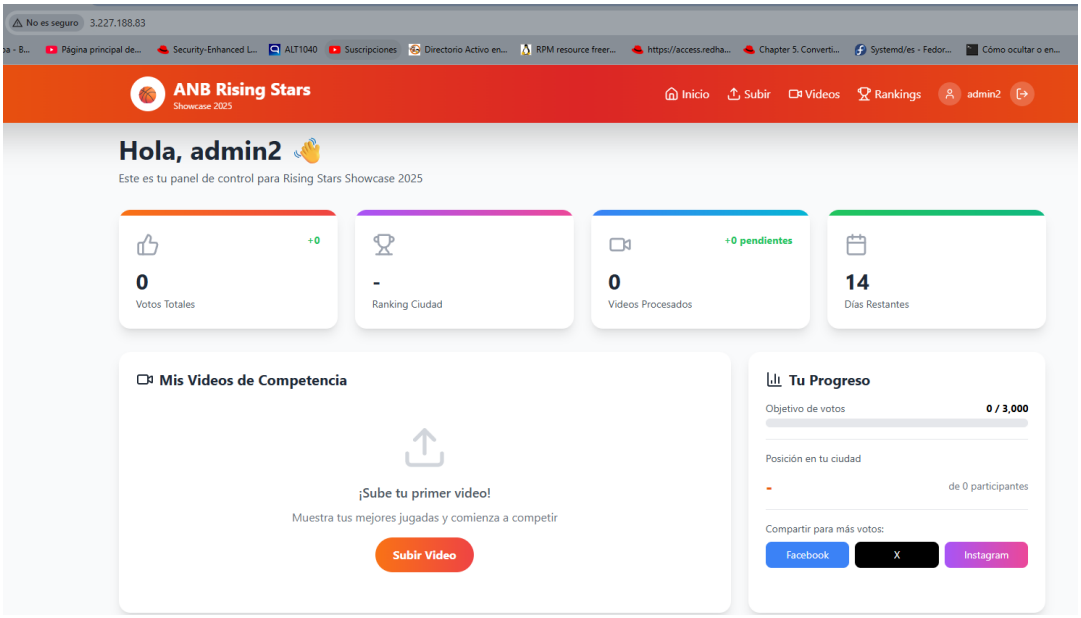

Figura 10. Panel de votación y cargue de video

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.

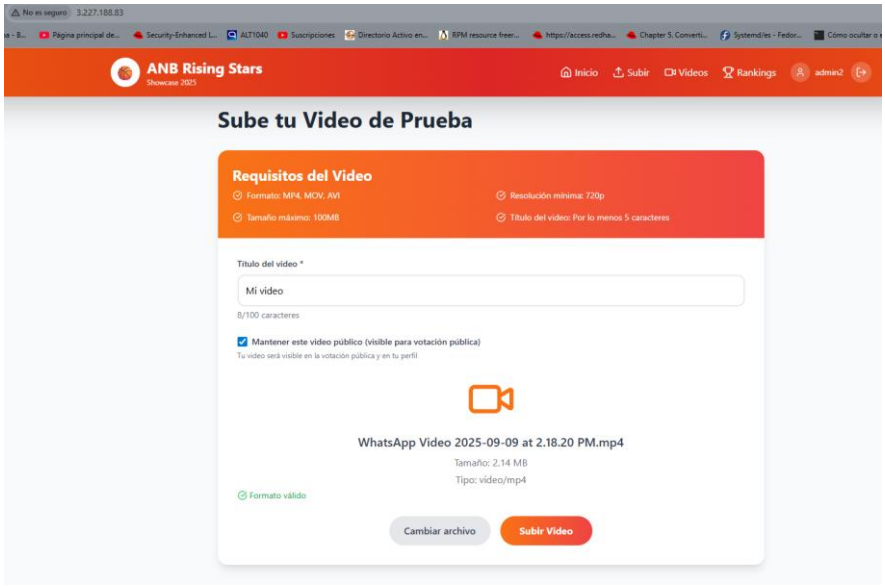

Figura 11. Panel de cargue de video

En paralelo, el **Worker**, levantado también mediante Docker con el archivo docker-compose.worker.yml, procesa en segundo plano las tareas asociadas a los videos, como el almacenamiento en el sistema de archivos compartido (NFS) y la comunicación con la base de datos.

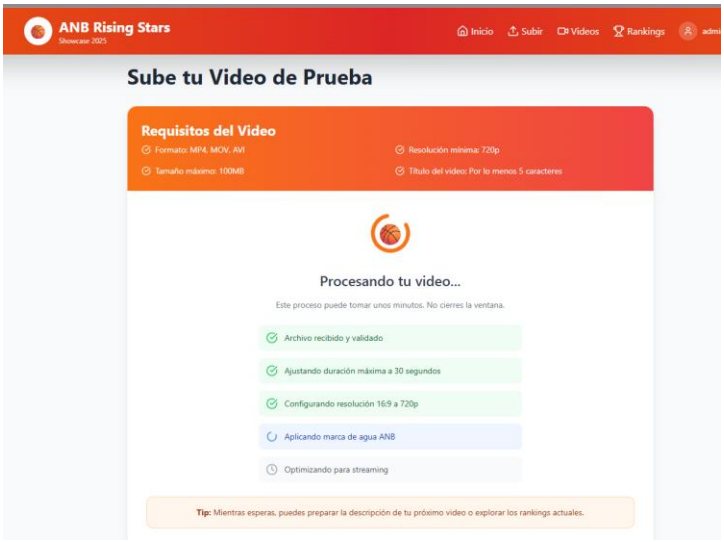

Figura 12. Procesamiento del video (web)

```
[GIN] 2025/09/29 - 19:43:25 | 200 |      1.26195ms |    186.29.32.79 | GET      "/api/public/rankings?limit=50"
[GIN] 2025/09/29 - 19:43:26 | 200 |     1.092909ms |    186.29.32.79 | GET      "/api/videos"
[GIN] 2025/09/29 - 19:43:26 | 200 |      1.07605ms |    186.29.32.79 | GET      "/api/user/votes"
[GIN] 2025/09/29 - 19:43:33 | 500 |     2.974075ms |    186.29.32.79 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 19:54:40 | 401 |       33.893µs |     172.18.0.1 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 19:55:47 | 401 |       49.654µs |     172.18.0.1 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 19:56:13 | 200 |       90.359µs |    186.29.32.79 | GET      "/swagger/favicon-32x32.png"
[GIN] 2025/09/29 - 19:58:12 | 401 |       36.216µs |     172.18.0.1 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 19:58:57 | 401 |      998.491µs |     172.18.0.1 | POST     "/api/auth/login"
[GIN] 2025/09/29 - 19:59:32 | 200 |    71.142282ms |     172.18.0.1 | POST     "/api/auth/login"
[GIN] 2025/09/29 - 20:00:17 | 500 |     3.807348ms |     172.18.0.1 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 20:03:26 | 401 |      106.096µs |   181.237.238.4 | GET      "/api/auth/profile"
[GIN] 2025/09/29 - 20:04:22 | 401 |    70.769319ms |   181.237.238.4 | POST     "/api/auth/login"
[GIN] 2025/09/29 - 20:04:28 | 401 |    70.729696ms |   181.237.238.4 | POST     "/api/auth/login"
[GIN] 2025/09/29 - 20:04:50 | 400 |     1.044479ms |   181.237.238.4 | POST     "/api/auth/signup"
[GIN] 2025/09/29 - 20:05:44 | 400 |       86.704µs |   181.237.238.4 | POST     "/api/auth/signup"
[GIN] 2025/09/29 - 20:06:46 | 201 |    73.102576ms |   181.237.238.4 | POST     "/api/auth/signup"
[GIN] 2025/09/29 - 20:06:46 | 200 |    70.294174ms |   181.237.238.4 | POST     "/api/auth/login"
[GIN] 2025/09/29 - 20:06:46 | 200 |      875.426µs |   181.237.238.4 | GET      "/api/auth/profile"
[GIN] 2025/09/29 - 20:06:46 | 200 |       885.1µs |   181.237.238.4 | GET      "/api/public/videos"
[GIN] 2025/09/29 - 20:06:47 | 200 |     1.415298ms |   181.237.238.4 | GET      "/api/public/rankings?limit=50"
[GIN] 2025/09/29 - 20:06:47 | 200 |      904.374µs |   181.237.238.4 | GET      "/api/videos"
[GIN] 2025/09/29 - 20:06:47 | 200 |      848.517µs |   181.237.238.4 | GET      "/api/user/votes"
[GIN] 2025/09/29 - 20:07:57 | 201 |  718.575591ms |   181.237.238.4 | POST     "/api/videos/upload"
[GIN] 2025/09/29 - 20:09:18 | 200 |      940.817µs |   181.237.238.4 | GET      "/api/public/videos"
[GIN] 2025/09/29 - 20:09:18 | 200 |     1.261376ms |   181.237.238.4 | GET      "/api/public/rankings?limit=50"
[GIN] 2025/09/29 - 20:09:18 | 200 |     1.142362ms |   181.237.238.4 | GET      "/api/videos"
[GIN] 2025/09/29 - 20:09:18 | 200 |     1.048211ms |   181.237.238.4 | GET      "/api/user/votes"
[GIN] 2025/09/29 - 20:09:21 | 200 |      805.971µs |   181.237.238.4 | GET      "/api/public/videos"
[GIN] 2025/09/29 - 20:09:22 | 200 |     1.314123ms |   181.237.238.4 | GET      "/api/public/rankings?limit=50"
[GIN] 2025/09/29 - 20:09:22 | 200 |     1.163057ms |   181.237.238.4 | GET      "/api/videos"
[GIN] 2025/09/29 - 20:09:22 | 200 |     1.097384ms |   181.237.238.4 | GET      "/api/user/votes"
[GIN] 2025/09/29 - 20:10:09 | 500 |     2.683015ms |    186.29.32.79 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 20:10:11 | 500 |     3.055496ms |    186.29.32.79 | POST     "/api/public/videos/2f5688b4-f1ef-4870-a79e-10b1493447727/vote"
[GIN] 2025/09/29 - 20:10:20 | 500 |     2.813771ms |   181.237.238.4 | POST     "/api/public/videos/3562007d-f8c7-40aa-b7d1-f6cc7ac4f210/vote"
[GIN] 2025/09/29 - 20:10:22 | 500 |     2.996315ms |   181.237.238.4 | POST     "/api/public/videos/3562007d-f8c7-40aa-b7d1-f6cc7ac4f210/vote"
[GIN] 2025/09/29 - 20:10:27 | 200 |     1.284409ms |   181.237.238.4 | GET      "/api/public/rankings?limit=50"
[GIN] 2025/09/29 - 20:10:28 | 200 |     1.280743ms |   181.237.238.4 | GET      "/api/public/rankings?limit=50&city=bogot%C3%A1"
[root@ip-172-21-10-97 ~]#
```

Figura 13. Procesamiento del video (API)

Figura 14. Procesamiento del video (worker)

Las figuras 13 y 14 muestran la ejecución de los contenedores Docker, lo que evidencia que la carga se realizó exitosamente tanto para el **API** como para el **Worker**, garantizando que los dos servicios críticos de la aplicación están operativos dentro de la infraestructura distribuida.

Una vez que los videos han sido cargados y procesados, los demás usuarios pueden visualizarlos dentro de la plataforma y emitir sus votos.

Gracias a esta separación de componentes, se logra que los usuarios puedan subir contenido, votar por los videos y consultar resultados en tiempo real, mientras que el procesamiento se ejecuta de manera independiente y confiable en el Worker. Con ello se confirma que la aplicación de votación de videos funciona correctamente en la arquitectura distribuida desplegada en AWS.

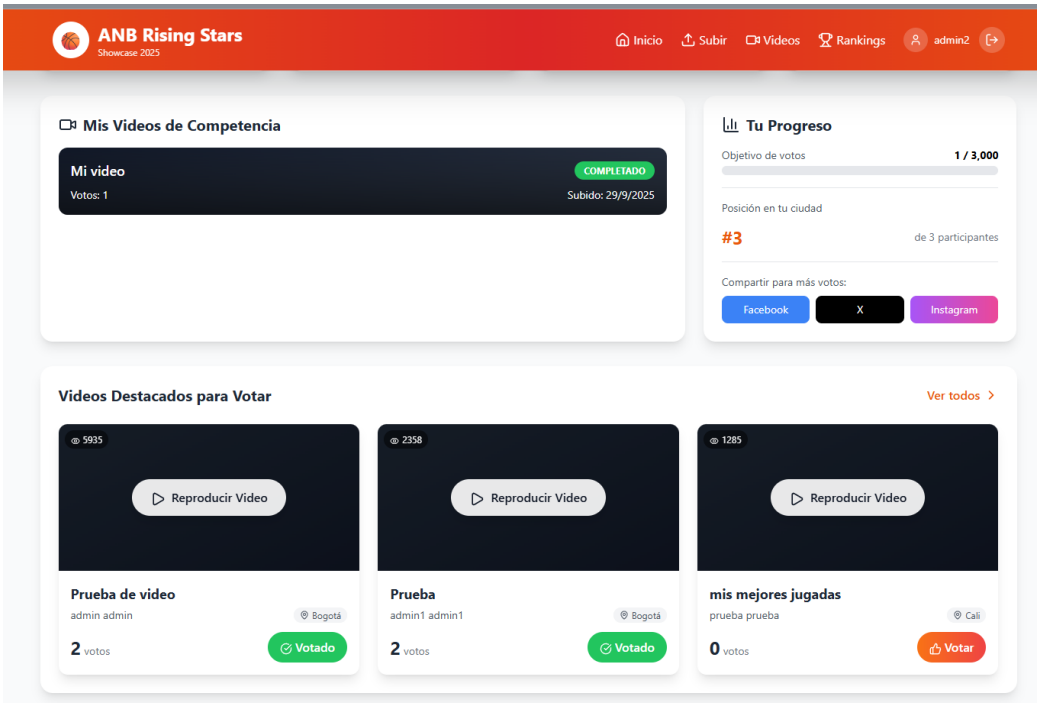

Figura 15. Confirmación de carga del video

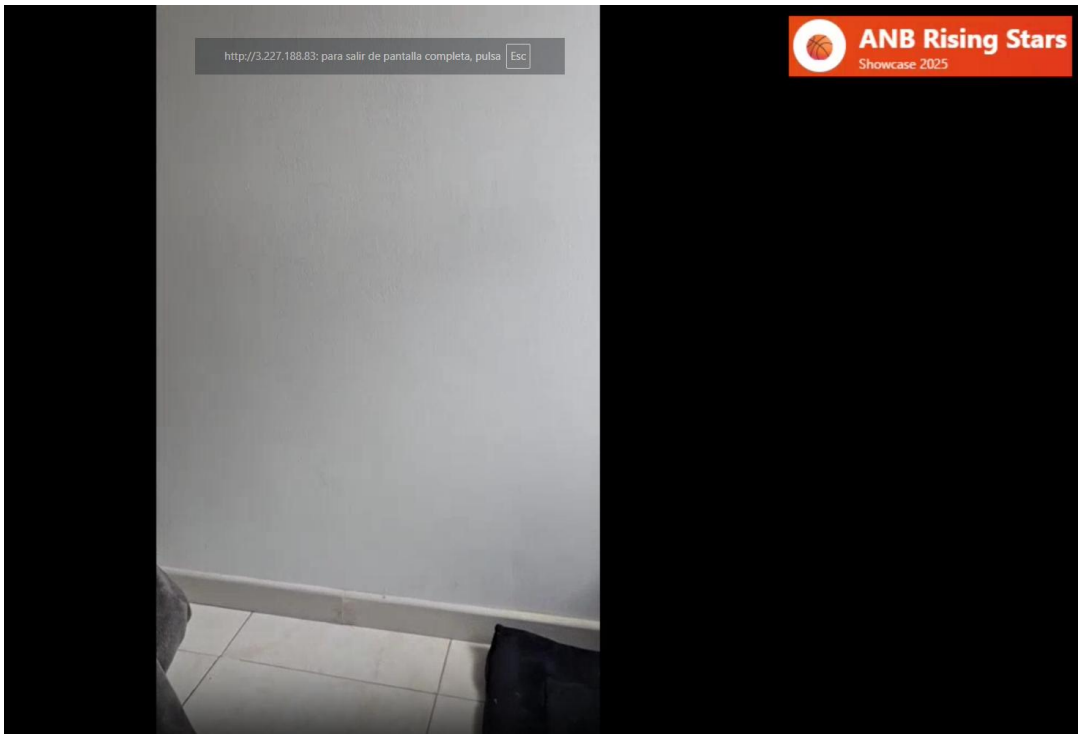

Figura 16. Prueba de reproducción del video

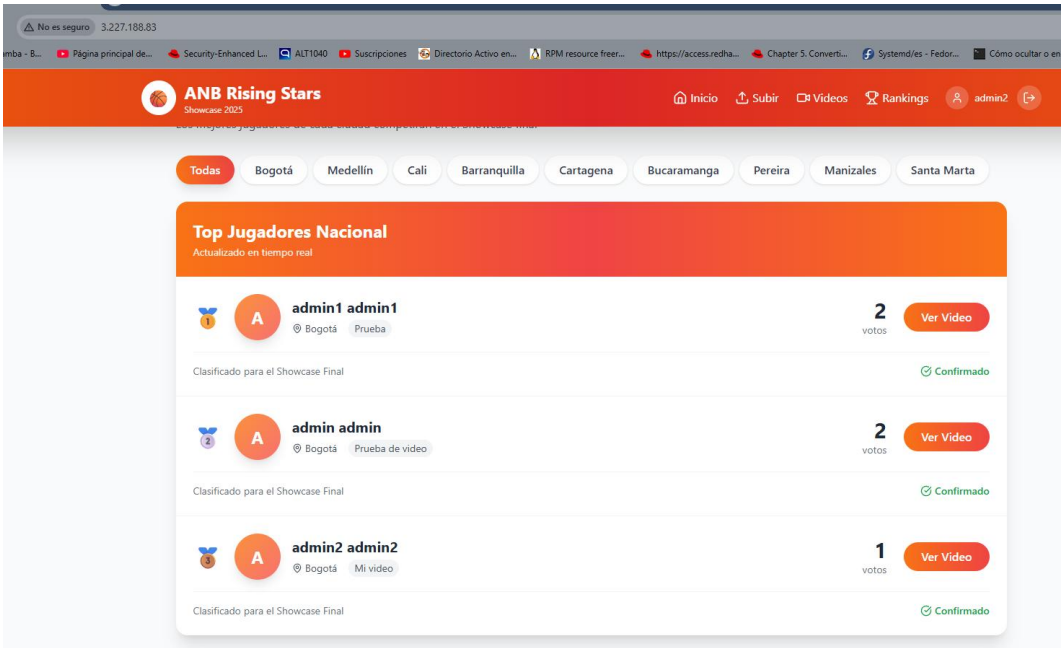

Figura 17. Ranking de videos más votados

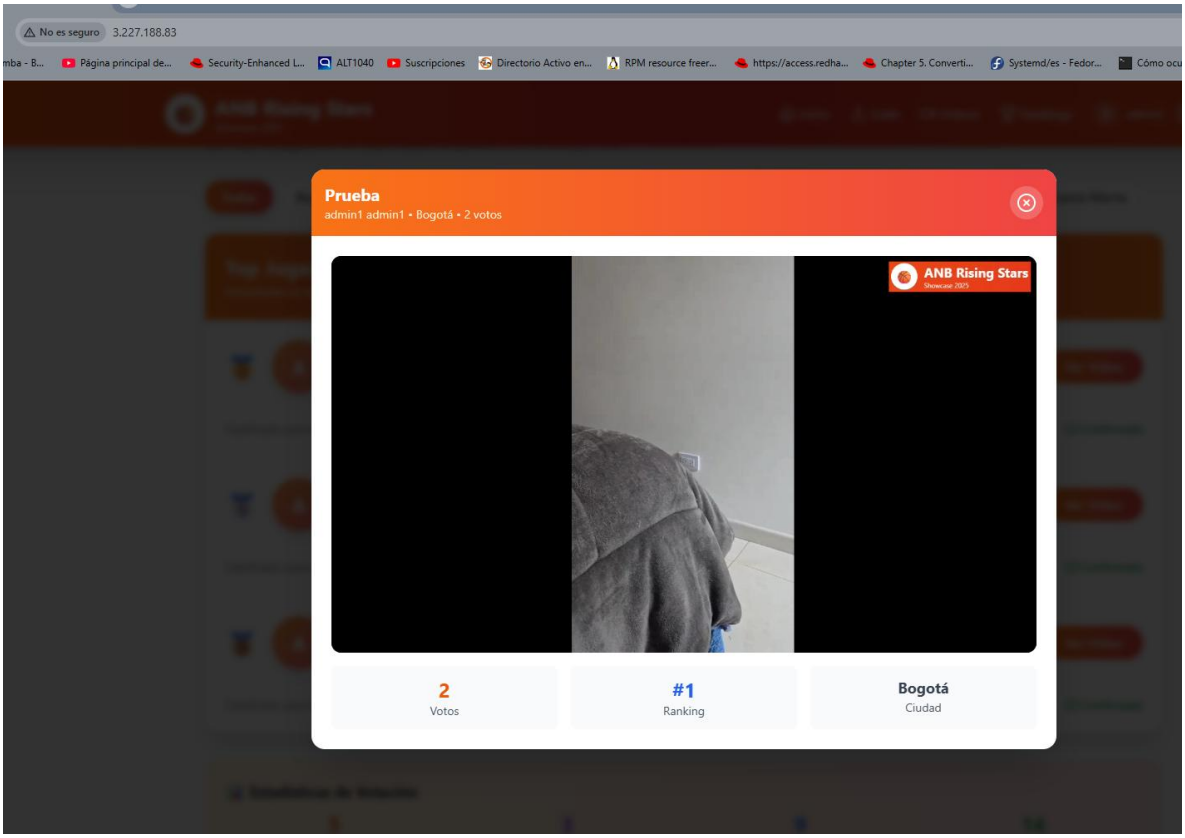

Figura 88. Visualización de videos más votados

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

CONCLUSIONES

- La implementación de la arquitectura en la nube permitió automatizar completamente el despliegue de la infraestructura, garantizando reproducibilidad, escalabilidad y facilidad de mantenimiento mediante el uso de plantillas CloudFormation y scripts Shell.
- La creación de instancias dedicadas —bastion, worker y Redis— permitió separar las responsabilidades del sistema, mejorando la seguridad, rendimiento y gestión de recursos.
- El uso de un Auto Scaling Group y un Load Balancer aseguró la alta disponibilidad y tolerancia a fallos del portal, adaptándose dinámicamente a la demanda de usuarios.
- La integración con S3 mediante s3fs facilitó el almacenamiento y manejo de archivos de forma distribuida, optimizando el flujo de trabajo del procesamiento de videos.
- Finalmente, se comprobó que el portal se encuentra funcional y disponible, cumpliendo con los objetivos de despliegue, demostrando las ventajas del enfoque Infraestructura como Código (IaC) y las buenas prácticas en arquitecturas cloud modernas.

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