

Desarrollo de soluciones Cloud
Proyecto Entrega 2

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	3
Modelo de Despliegue Básico en la Nube Publica de AWS.....	4
Despliegue de arquitectura.....	4
Arquitectura actual	5
Arquitectura propuesta.....	5
Configuración del servicio Amazon RDS.....	7
Políticas de red VPC	11
Implementación de arquitectura propuesta	13
Funcionamiento de la aplicación	17
CONCLUSIONES	24
BIBLIOGRAFÍA	25

INTRODUCCIÓN

En la actualidad, las aplicaciones web monolíticas presentan limitaciones para escalar, mantener y desplegarse en entornos de alta disponibilidad. La migración a la nube pública permite aprovechar los servicios de infraestructura como servicio (IaaS), proporcionando mayor flexibilidad, escalabilidad y seguridad.

En este contexto, el presente trabajo corresponde a la segunda fase del proyecto de migración de una aplicación web desarrollada en Go. En esta entrega, los componentes críticos de la aplicación como servidor web, worker, sistema de archivos y base de datos, se distribuyen en instancias independientes dentro de Amazon Web Services (AWS), garantizando su correcta operación, integración y evaluación de desempeño mediante pruebas de carga. En esta fase no se implementarán mecanismos de autoscaling ni se utilizará almacenamiento en buckets de Amazon S3.

OBJETIVOS

Objetivo General

Migrar y desplegar de manera distribuida una aplicación web desarrollada en Go en la nube pública AWS, garantizando su funcionamiento, rendimiento y seguridad, mediante la configuración de instancias EC2 para los componentes críticos y el uso de Amazon RDS para la gestión de la base de datos.

Objetivos Específicos

- Analizar la arquitectura monolítica existente y definir una estrategia de distribución de componentes en la nube.
- Configurar instancias de Amazon EC2 para alojar el servidor web, el worker y el sistema de archivos NFS, asegurando la correcta comunicación entre ellos.
- Implementar y parametrizar Amazon RDS como sistema de gestión de la base de datos, garantizando persistencia y disponibilidad de los datos.
- Validar el funcionamiento de la aplicación mediante pruebas de estrés y carga, documentando el rendimiento y proponiendo mejoras para escenarios de alta concurrencia.
- Documentar la arquitectura distribuida, los resultados de las pruebas de carga y las recomendaciones para futuras mejoras y escalabilidad.

Modelo de Despliegue Básico en la Nube Publica de AWS

La compañía planea mover su aplicación a un proveedor público de **IaaS**, específicamente AWS. En esta primera migración, se ha decidido aprovechar los siguientes servicios:

- **Amazon EC2:** Ejecución del servidor web, el worker y el almacenamiento de tipo NFS. 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. Se recomienda establecer el entorno de desarrollo como ajuste predeterminado para minimizar costos. En las fases iniciales de la implementación, se puede utilizar una instancia de Amazon EC2 con un contenedor de base de datos y sustituirla por Amazon RDS únicamente durante las pruebas de estrés.

Despliegue de arquitectura

El despliegue de la arquitectura deberá componerse de tres instancias de cómputo (máquinas virtuales) separadas, cada una dedicada a un componente específico: el componente web, el worker y el sistema de almacenamiento de archivos. Para ello, se requiere:

- a. Configurar todas las dependencias (servidor de aplicaciones, librerías, etc.) y herramientas necesarias (reglas de firewall, llaves de acceso, etc.) para exponer la aplicación web en una instancia denominada **Web Server**.
- b. Configurar el componente **Worker** que procesa los archivos en una segunda instancia de Amazon EC2.
- c. Configurar un sistema de archivos de red (**NFS**) en la tercera instancia de Amazon EC2. La instancia se denominará File Server, y tanto el componente web como el worker deberán almacenar y acceder a todos los archivos originales y procesados en este servidor. No utilice el servicio Amazon EFS.

Arquitectura actual

La arquitectura existente representa un enfoque monolítico containerizado, donde el servidor de aplicaciones y la base de datos se ejecutan conjuntamente en Docker dentro de un mismo entorno servidor.

- **Ventajas:**

- Fácil de desarrollar y desplegar inicialmente.
- Comunicación simple entre componentes (localhost).
- Menor complejidad operativa.

- **Desventajas:**

- Punto único de falla.
- Escalabilidad limitada.
- Acoplamiento fuerte entre componentes.
- Dificultad para actualizar componentes individualmente.

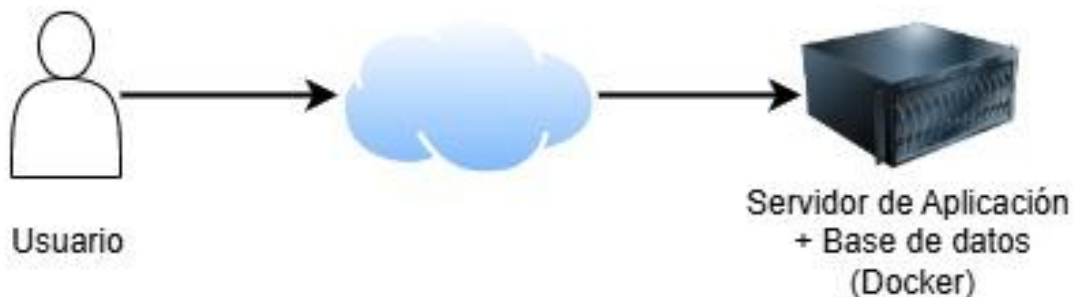


Figura 1. Arquitectura Monolítica Inicial

Arquitectura propuesta

La arquitectura propuesta representa la migración de una arquitectura monolítica hacia una arquitectura distribuida en la nube de AWS. En la nueva arquitectura, los componentes críticos se desacoplaron en instancias independientes de la siguiente manera:

- ✓ **Web Server (EC2):** Atiende solicitudes de los usuarios y gestiona tareas con Redis, exponiendo los endpoints mediante un Elastic IP en la subred pública.
- ✓ **Worker (EC2):** Procesa las tareas enviadas desde el servidor web, conectándose al sistema de archivos y a la base de datos.

- ✓ **NFS (EC2):** Actúa como servidor de archivos compartidos, centralizando el almacenamiento de los archivos originales y procesados.
- ✓ **Amazon RDS (PostgreSQL):** Administra de forma confiable y escalable la persistencia de los datos.

El acceso del usuario se realizará mediante una dirección IP elástica en la subred pública, mientras que los demás servicios se encuentran aislados en una subred privada, lo que mejora la seguridad y la gestión de la infraestructura.

Características de servidor recomendadas por el negocio:

- ✓ **CPU:** 2 vCPUs.
- ✓ **Memoria RAM:** 2 GiB.
- ✓ **Almacenamiento:** 30 GiB (Volumen EBS).

Nombre	vCPU	Memoria (GiB)	Rendimiento base/CPU virtual	Créditos de CPU obtenidos por hora	Ancho de banda de ráfaga de la red (Gbps)	Ancho de banda de ráfaga de EBS (Mbps)	Precio de la instancia bajo demanda por hora*	Por hora real en una instancia reservada por 1 año*	Por hora real en una instancia reservada por 3 años*
t3.nano	2	0,5	5 %	6	5	hasta 2085	0,0052 USD	0,003 USD	0,002 USD
t3.micro	2	1,0	10 %	12	5	hasta 2085	0,0104 USD	0,006 USD	0,005 USD
t3.small	2	2,0	20 %	24	5	hasta 2085	0,0209 USD	0,012 USD	0,008 USD
t3.medium	2	4,0	20 %	24	5	hasta 2085	0,0418 USD	0,025 USD	0,017 USD
t3.large	2	8,0	30 %	36	5	Hasta 2780	0,0835 USD	0,05 USD	0,036 USD
t3.xlarge	4	16,0	40 %	96	5	Hasta 2780	0,1670 USD	0,099 USD	0,067 USD
t3.2xlarge	8	32,0	40 %	192	5	Hasta 2780	0,3341 USD	0,199 USD	0,133 USD

Figura 2. Listado de servidores familia T3

Según validación en AWS el servidor que se ajusta a la solicitud es una **t3.small** teniendo en cuenta que la carga de CPU no será intensiva, puesto que la familia de servidores T3 no ofrecen un rendimiento constante de CPU. Para eso deberíamos pensar en pasarnos a una instancia de familia **M5, C5 o M6G**.

Nota: la instancia de la familia T3 puede cambiar si la carga de trabajo no es soportada por la instancia t3.smalll

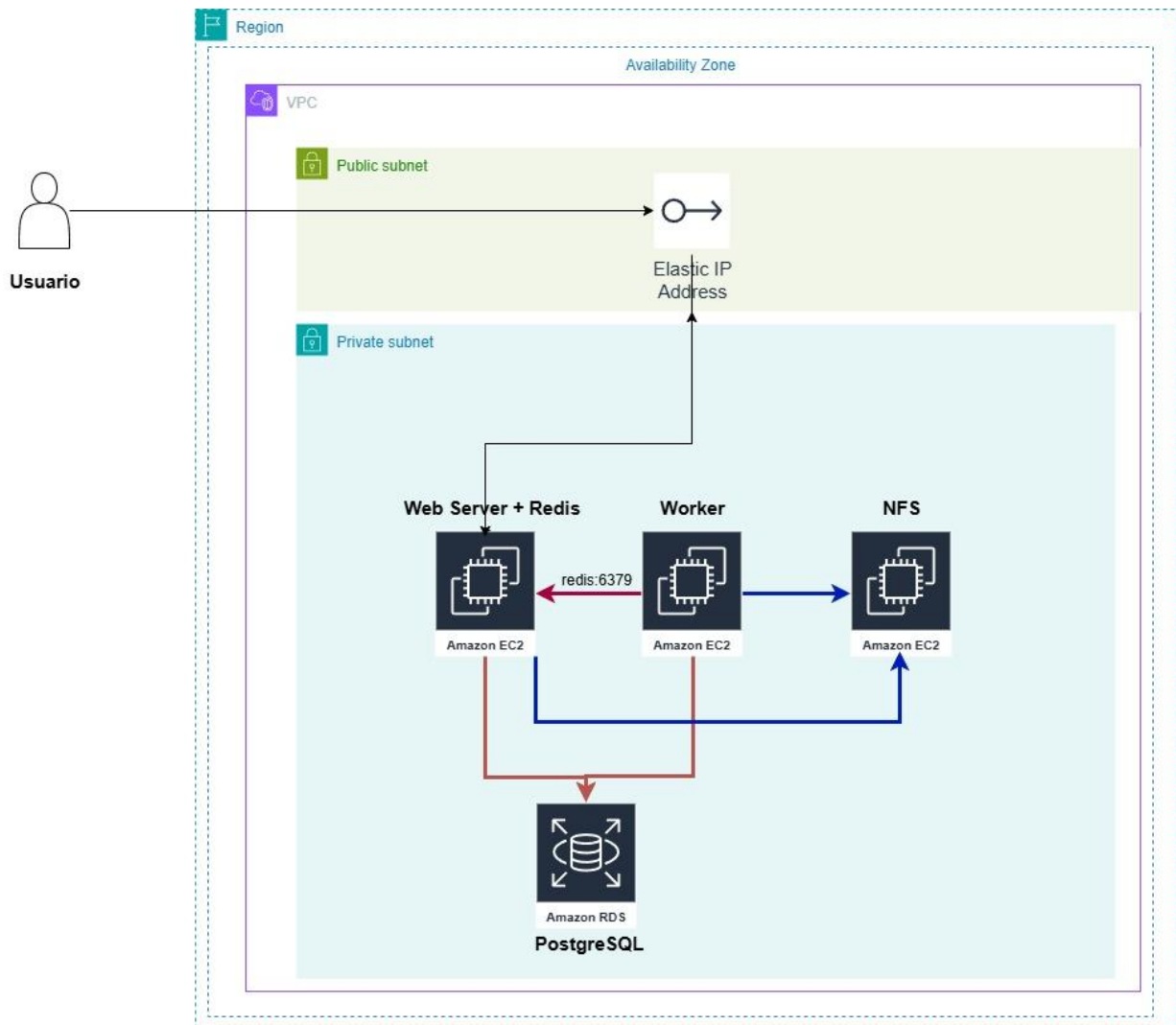


Figura 3. Arquitectura Distribuida en AWS

Configuración del servicio Amazon RDS

Para administrar la base de datos de la aplicación web. No es necesario definir ningún mecanismo de replicación ni alta disponibilidad.

EC2 Aurora and RDS > Create database

Create database [Info](#)


Choose a database creation method


☒ **Standard create**
You set all of the configuration options, including ones for availability, security, backups, and maintenance.


☐ **Easy create**
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.


Engine options


Engine type [Info](#)


☐ Aurora (MySQL Compatible) 


☐ Aurora (PostgreSQL Compatible) 


☐ MySQL 

☒ PostgreSQL 

☐ MariaDB 

☐ Oracle 

☐ Microsoft SQL Server 

☐ IBM Db2 

Engine version [Info](#)

Figura 4. Creación de instancia RDS

Templates

Choose a sample template to meet your use case.

☐ **Production**
Use defaults for high availability and fast, consistent performance.

☒ **Dev/Test**
This instance is intended for development use outside of a production environment.

☐ **Sandbox**
To develop new applications, test existing applications, or gain hands-on experience with Amazon RDS.

Availability and durability

Deployment options [Info](#)

Choose the deployment option that provides the availability and durability needed for your use case. AWS is committed to a certain level of uptime depending on the deployment option you choose. Learn more in the [Amazon RDS service level agreement \(SLA\)](#).

☐ **Multi-AZ DB cluster deployment (3 instances)**
Creates a primary DB instance with two readable standbys in separate Availability Zones. This setup provides:

- 99.99% uptime
- Redundancy across Availability Zones
- Increased read capacity
- Reduced write latency

☐ **Multi-AZ DB instance deployment (2 instances)**
Creates a primary DB instance with a non-readable standby instance in a separate Availability Zone. This setup provides:

- 99.99% uptime
- Redundancy across Availability Zones

☒ **Single-AZ DB instance deployment (1 instance)**
Creates a single DB instance without standby instances. This setup provides:

- 99.9% uptime
- No data redundancy

Diagram 1: Multi-AZ DB cluster deployment (3 instances)

Write/read endpoint (AZ 1): Primary instance + SSD

Reader endpoints (AZ 2, AZ 3): Readable standby + SSD

Diagram 2: Multi-AZ DB instance deployment (2 instances)

Write/read endpoint (AZ 1): Primary instance

Standby (no endpoint) (AZ 2): Standby

Diagram 3: Single-AZ DB instance deployment (1 instance)

Write/read endpoint (AZ 1): Primary instance

Figura 5. Configuración de instancia RDS

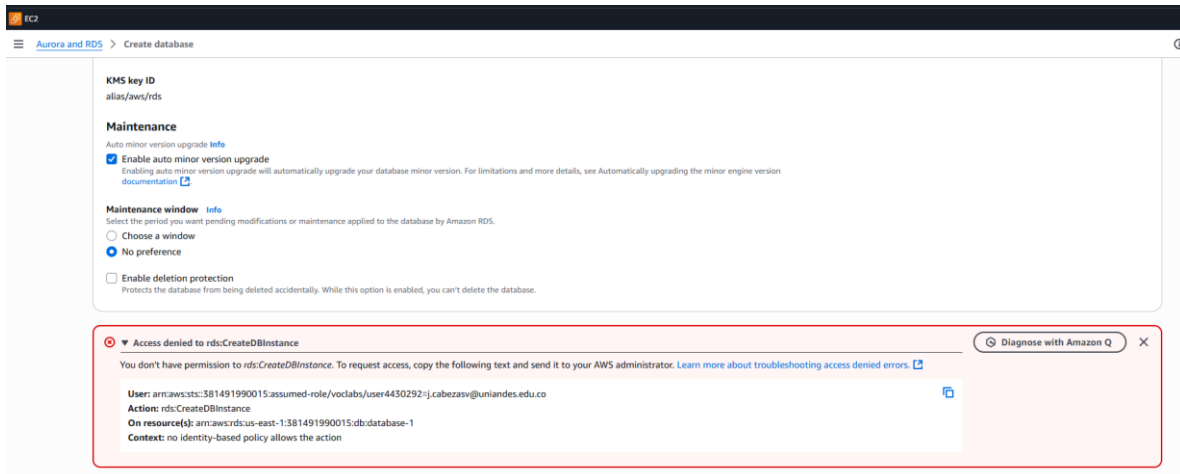


Figura 6. Arquitectura Distribuida en AWS

Según la validación realizada dentro de **AWS Academy**, no fue posible crear una instancia de Amazon RDS, ya que las cuentas asignadas a los estudiantes cuentan con permisos limitados. Específicamente, se bloquea la acción `rds:CreateDBInstance`, la cual es indispensable para aprovisionar una base de datos gestionada en RDS.

Esto se debe a que AWS Academy restringe el uso de ciertos servicios que pueden generar costos elevados o un consumo excesivo de recursos. Por esta razón:

- Se permite el uso de servicios básicos como EC2, S3, VPC o IAM con permisos acotados.
- Servicios de mayor costo como RDS, Redshift u otros administrados permanecen bloqueados para los estudiantes.

Ante esta limitación, como grupo optamos por implementar una instancia EC2 para simular el funcionamiento de una base de datos en RDS. De esta forma, instalamos manualmente el motor de base de datos en el servidor EC2, lo que nos permitió:

- Reproducir las funciones esenciales de un RDS (almacenamiento, consultas, gestión de usuarios).
- Tener control sobre la configuración del motor de base de datos.
- Continuar con las prácticas de laboratorio sin depender de un servicio restringido.

```
[postgres@ip-172-21-2-60 tmp]$ psql -U postgres -d proyecto_1 -f backup.sql
SET
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 7. Importación base de datos

Como se muestra en la figura 7, la base de datos fue ejecutada en una instancia EC2, lo que permitió simular un entorno de base de datos y realizar de manera exitosa las operaciones de restauración y configuración.

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.

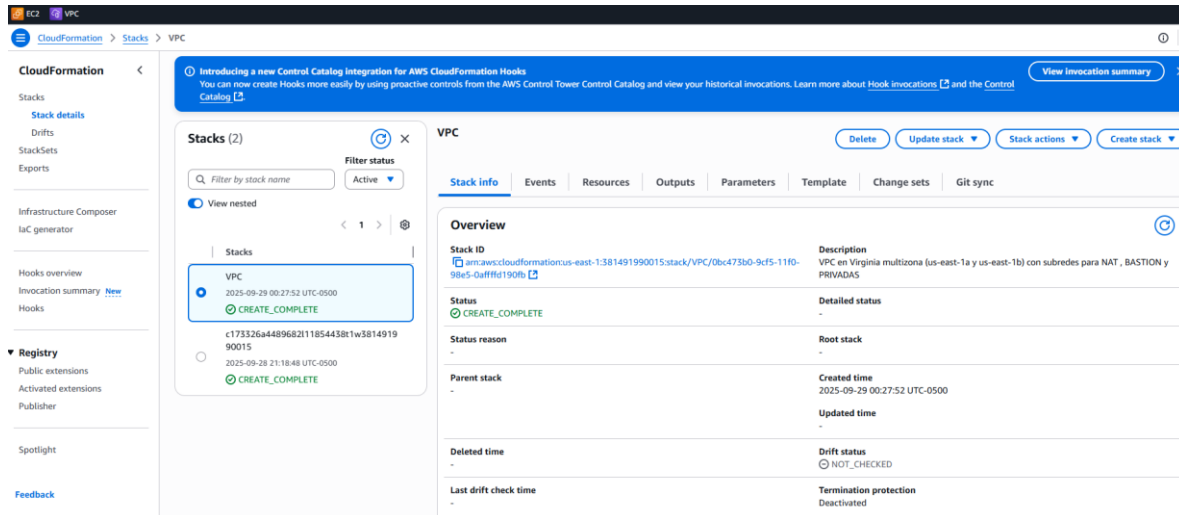


Figura 8. Importación de código para VPC

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

vpc-017fd81d73760e04a / Laboratorio

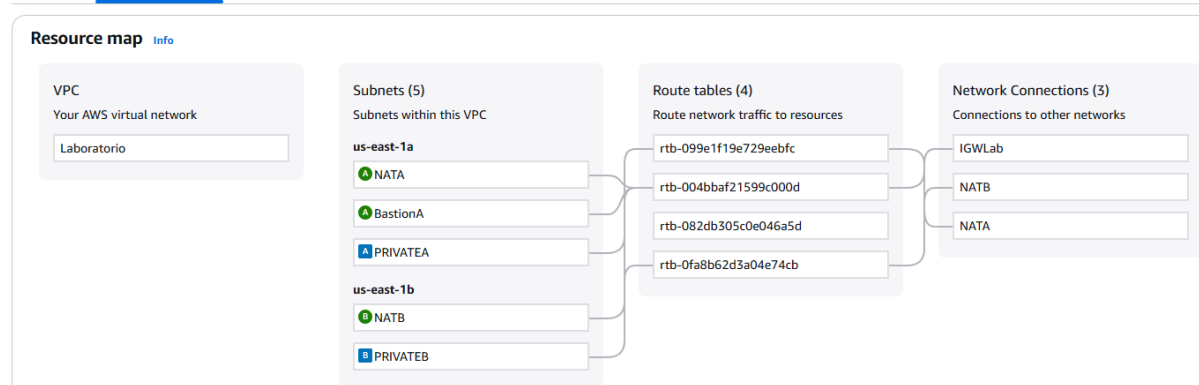


Figura 9. Diagrama de red AWS

Implementación de arquitectura propuesta

Para la implementación de la arquitectura en la nube propuesta, se crearon cinco instancias EC2, como se muestra en la figura 10, las cuales fueron distribuidas conforme a la arquitectura definida en la figura 2.

Instances (5)

Name	Instance ID	Instance state	Instance type
WebServer	i-0324a915997536907	Running	t3.micro
RDS - PostgreSQL	i-03d780ebb8a7700d1	Running	t3.micro
NFS	i-046af5999bee2ac2d	Running	t3.micro
Bastion	i-03dc3326f0a3cdfc3	Running	t3.micro
Worker	i-0fda4a00d7c522d21	Running	t3.micro

Figura 10. Despliegue de infraestructura en AWS

Para garantizar el acceso público al sitio web, se creó una dirección **Elastic IP**, la cual fue asociada a la instancia **WebServer**. De esta manera, se asegura que la aplicación cuente con una dirección IP estática y persistente, independientemente de reinicios o cambios en la infraestructura, lo que facilita tanto la disponibilidad como la estabilidad del acceso desde Internet.

Elastic IP addresses (1/3) Info

Find elastic IP addresses by attribute or tag

Name	Allocated IPv4 address	Type	Allocation ID	Reverse DNS record	Associated instance ID	Private IP address
-	3.215.14.127	Public IP	eipalloc-0a740f4dcd24bbcef	-	-	172.21.1.13
WebServer	3.227.188.83	Public IP	eipalloc-07f673b2041f0ce31	-	i-0903aae53430621ed	172.21.10.97
-	34.195.119.218	Public IP	eipalloc-07e44dc89a0ee4dac	-	-	172.21.3.212

Figura 10. Asignación de IP elástica

Para el resto de los servidores se generó una **AMI (Amazon Machine Image)** con el fin de facilitar su creación. Esto permitió evitar la repetición de instalaciones y configuraciones ya realizadas previamente, optimizando así el despliegue y asegurando la consistencia entre las instancias.

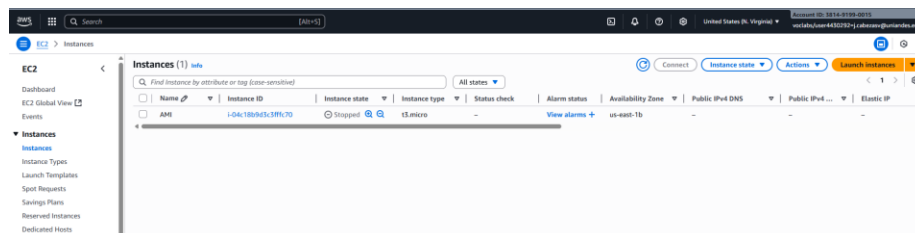


Figura 11. Validación de AMI

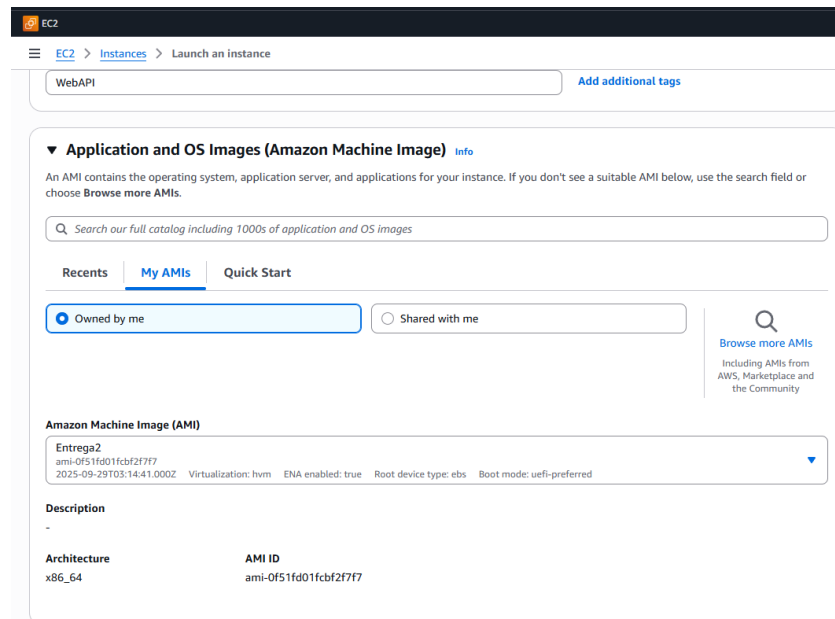


Figura 12. Despliegue de instancia a través de AMI

Para el tema del NFS En el archivo de configuración `/etc/fstab` se añadieron entradas para montar automáticamente recursos compartidos mediante **NFS (Network File System)**. En este caso, se configuró el servidor 172.21.2.5 para exportar el directorio `/srv/nfs/files`, el cual se montó en las rutas locales `/mnt/nfs` y `/app`. Con esta configuración, cada vez que el sistema se reinicie, los volúmenes NFS quedarán disponibles de forma persistente, facilitando el acceso centralizado a archivos compartidos entre servidores dentro de la arquitectura propuesta.

```
[root@ip-172-21-10-97 sites]# cat /etc/fstab
#
UUID=elf0d2ec-93af-4d46-8039-6fd7a938a4de / xfs defaults,noatime 1 1
UUID=E147-FCAD /boot/efi vfat defaults,noatime,uid=0,gid=0,umask=0077,shortname=winnt,x-systemd.automount 0 2
172.21.2.5:/srv/nfs/files /mnt/nfs nfs defaults 0 0
172.21.2.5:/srv/nfs/files /app nfs defaults 0 0
```

Figura 13. Configuración de NFS

Por el lado de base de datos (RDS) se configuro el usuario `usrpostgres` como **usuario de aplicación** con autenticación por contraseña, aislado del `admin postgres`. Permitiendo así que solo se conecte a la base `proyecto_1`, mejorando la seguridad al reducir los riesgos de exponer la cuenta principal de administración.

```
# TYPE DATABASE USER ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all peer
local all usrpostgres md5
# IPv4 local connections:
host all all 127.0.0.1/32 ident
host all all 172.21.0.0/16 md5
# IPv6 local connections:
host all all ::1/128 ident
# Allow replication connections from localhost, by a user with the
# replication privilege.
local replication all peer
host replication all 127.0.0.1/32 ident
host replication all ::1/128 ident
[root@ip-172-21-2-60 data]#
```

Figura 14. Conexión base de datos

Para hacerlo posible se editó el archivo `pg_hba.conf` para crear un usuario independiente del administrador, llamado `usrpostgres`, que se autentica siempre con contraseña (md5) y tiene acceso únicamente a la base de datos `proyecto_1`. De esta manera, el usuario administrador (`postgres`) queda reservado para tareas críticas, mientras que `usrpostgres` se utiliza en la aplicación. Con esta separación de privilegios se logra mayor seguridad, ya que, si la aplicación fuera comprometida, el acceso estaría limitado solo a esa base de datos y no a todo el servidor PostgreSQL.

Una vez disponibles los servidores y con la conectividad establecida entre ellos, se procedió al despliegue de los contenedores Docker correspondientes a los roles de Worker y WebServer. Para ello, se utilizaron archivos de definición `docker-compose`, lo

- Despliegue del WebServer:
 - `docker compose -f docker-compose.api.yml up`
- Despliegue del Worker:
 - `docker compose -f docker-compose.worker.yml up`

[illegible]

Figura 15. Cargue de Docker

[illegible]

Figura 16. Cargue de Docker

16

aplicación era accesible desde Internet y que la configuración de red, seguridad y de aplicación se encontraba correctamente implementada.

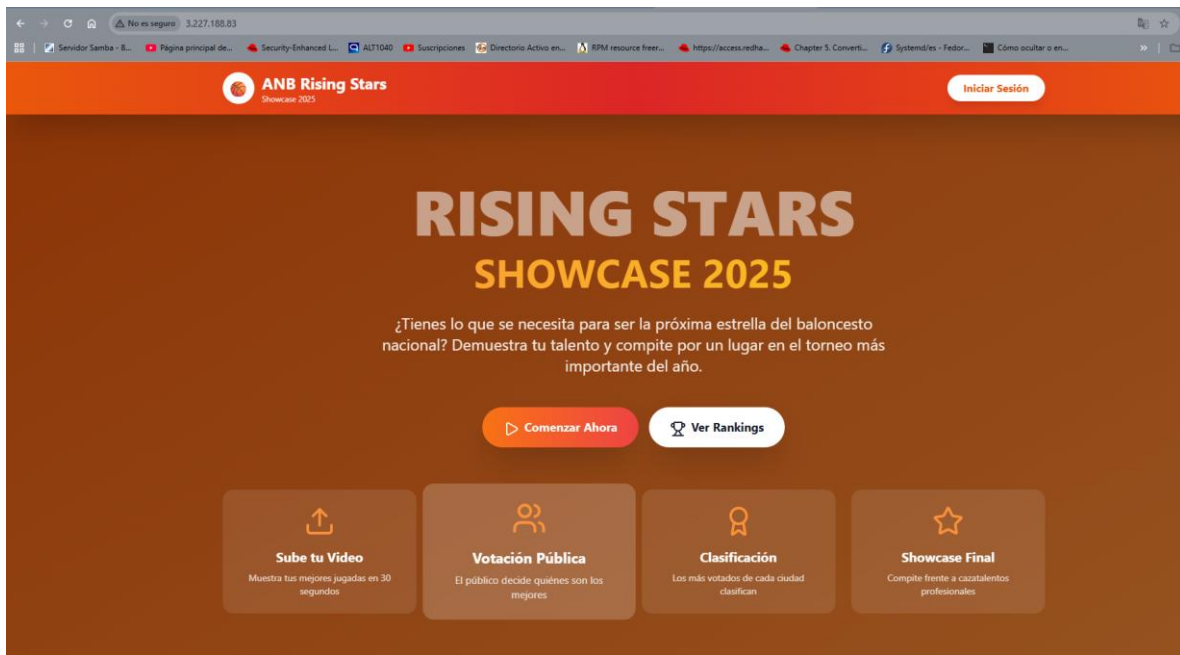


Figura 17. Portal de votación

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 18), el cual permite a los usuarios registrarse (figura 19) e iniciar sesión para acceder a las funcionalidades principales de la solución.

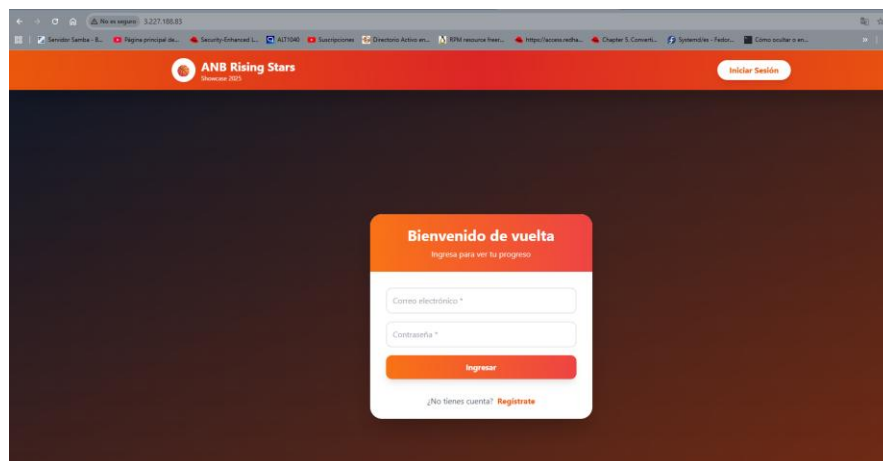
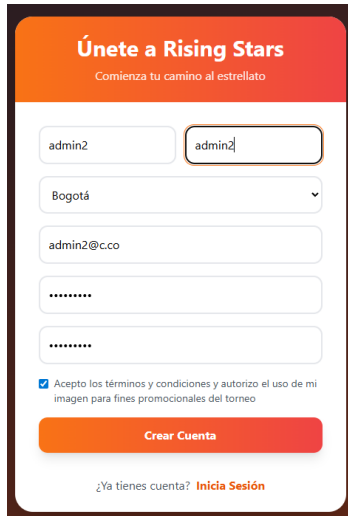


Figura 18. Portal de autenticación

Como se evidencia en la figura 18 el panel de autenticación cuenta con un apartado de registro, esto con el fin de que los usuarios puedan registrarse para iniciar su proceso de votaciones.



Únete a Rising Stars
Comienza tu camino al estrellato

admin2 admin2

Bogotá

admin2@c.co

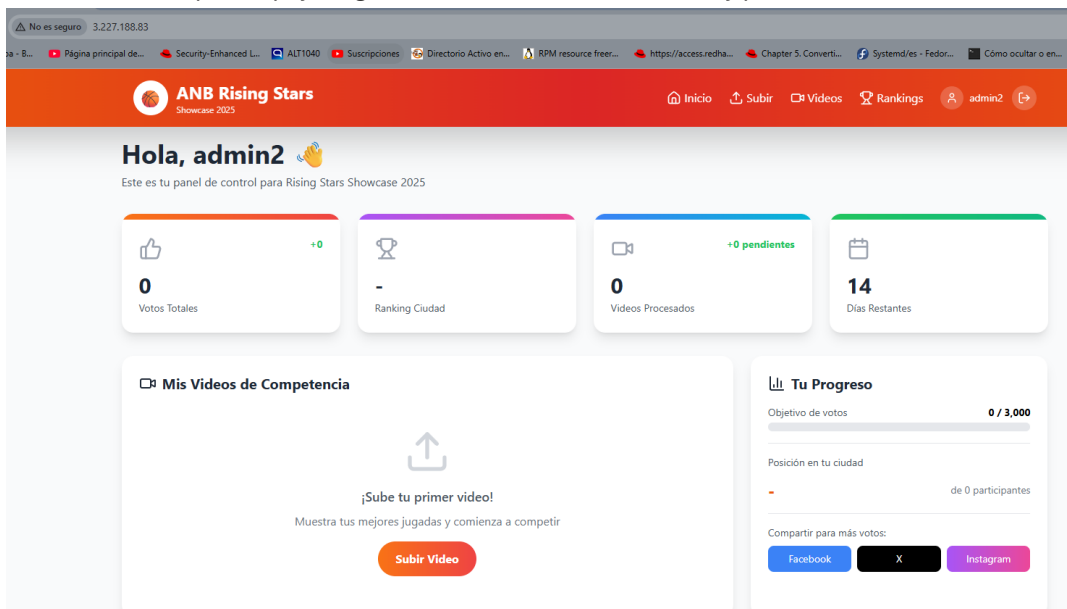
☒ Acepto los términos y condiciones y autorizo el uso de mi imagen para fines promocionales del torneo

Crear Cuenta

¿Ya tienes cuenta? [Inicia Sesión](#)

Figura 19. 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.



ANB Rising Stars
Showcase 2025

Hola, admin2 🙌
Este es tu panel de control para Rising Stars Showcase 2025

0 Votos Totales

- Ranking Ciudad

0 Videos Procesados

14 Días Restantes

Mis Videos de Competencia

¡Sube tu primer video!
Muestra tus mejores jugadas y comienza a competir

Subir Video

Tu Progreso

Objetivo de votos: 0 / 3,000

Posición en tu ciudad: - de 0 participantes

Compartir para más votos:

Facebook X Instagram

Figura 20. 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.

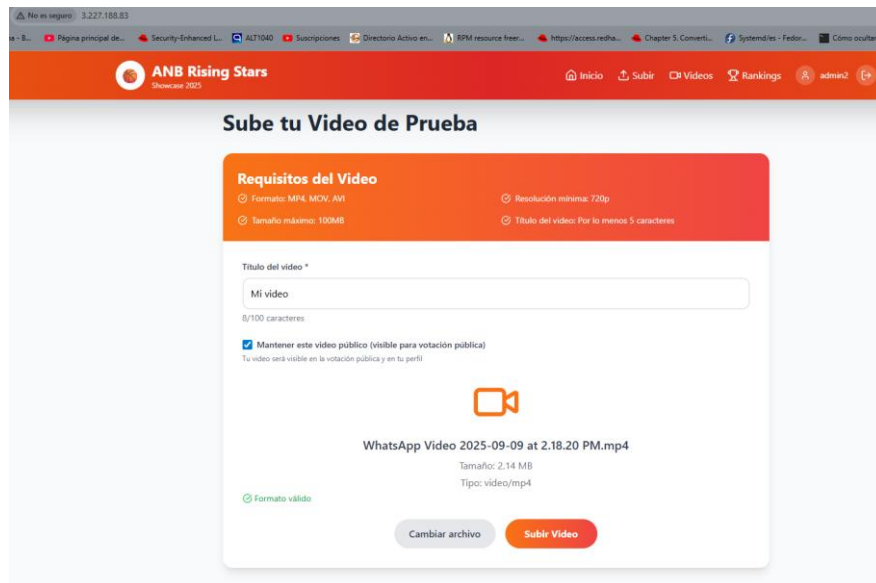


Figura 21. Panel de cargue de video

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.

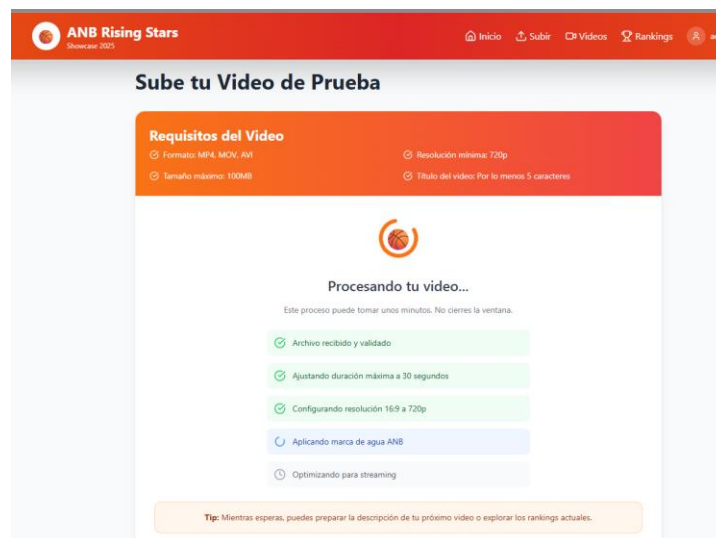


Figura 22. Procesamiento del video (web)

[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-10b149344727/vote"
[GIN]	2025/09/29	- 19:54:40	401	33.893µs	172.18.0.1	POST	"/api/public/videos/2f5688b4-f1ef-4870-a79e-10b149344727/vote"
[GIN]	2025/09/29	- 19:55:47	401	49.654µs	172.18.0.1	POST	"/api/public/videos/2f5688b4-f1ef-4870-a79e-10b149344727/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-10b149344727/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-10b149344727/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	1.0444479ms	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-10b149344727/vote"
[GIN]	2025/09/29	- 20:10:11	500	3.055496ms	186.29.32.79	POST	"/api/public/videos/2f5688b4-f1ef-4870-a79e-10b149344727/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 23. Procesamiento del video (API)

```
[ec2-user@ip-172-21-2-34 ~]$ sudo su -
Last login: Tue Sep 30 00:35:26 UTC 2025 on pts/1
[root@ip-172-21-2-34 ~]# vim http://3.227.188.83/^C
[root@ip-172-21-2-34 ~]# docker ps
CONTAINER ID   IMAGE          COMMAND                  CREATED        STATUS        PORTS        NAMES
11aca2047be1   sites-worker   "./worker"              32 minutes ago Up 32 minutes          anb_worker
[root@ip-172-21-2-34 ~]# docker logs anb_worker
2025/09/29 19:36:13 No .env file found
2025/09/29 19:36:13 Successfully connected to database
2025/09/29 19:36:13 Starting in worker mode...
2025/09/29 19:36:13 video processor: iniciando server
asynq: pid=1 2025/09/30 00:36:13.797157 INFO: Starting processing
asynq: pid=1 2025/09/30 00:36:13.797184 INFO: Send signal TSTP to stop processing new tasks
asynq: pid=1 2025/09/30 00:36:13.797189 INFO: Send signal TERM or INT to terminate the process
2025/09/29 19:36:13 Processing video: 3562007d-f8c7-40aa-b7d1-f6cc7ac4f210
2025/09/29 19:36:14 Removing audio from video 3562007d-f8c7-40aa-b7d1-f6cc7ac4f210
2025/09/29 19:36:14 Converting video 3562007d-f8c7-40aa-b7d1-f6cc7ac4f210 to 720p with watermark
asynq: pid=1 2025/09/30 00:36:16.469244 INFO: Stopping processor
2025/09/29 19:36:16 Processed video: 3562007d-f8c7-40aa-b7d1-f6cc7ac4f210
asynq: pid=1 2025/09/30 00:36:17.016580 INFO: Processor stopped
asynq: pid=1 2025/09/30 00:36:17.016794 INFO: Starting graceful shutdown
asynq: pid=1 2025/09/30 00:36:17.016909 INFO: Waiting for all workers to finish...
asynq: pid=1 2025/09/30 00:36:17.017026 INFO: ALL workers have finished
asynq: pid=1 2025/09/30 00:36:17.018953 INFO: Exiting
2025/09/29 19:36:23 No .env file found
2025/09/29 19:36:23 Successfully connected to database
2025/09/29 19:36:23 Starting in worker mode...
2025/09/29 19:36:23 video processor: iniciando server
asynq: pid=1 2025/09/30 00:36:23.495244 INFO: Starting processing
asynq: pid=1 2025/09/30 00:36:23.495354 INFO: Send signal TSTP to stop processing new tasks
asynq: pid=1 2025/09/30 00:36:23.495361 INFO: Send signal TERM or INT to terminate the process
2025/09/29 19:37:02 Processing video: 2f5688b4-f1ef-4870-a79e-10b149344727
2025/09/29 19:37:02 Trimming video 2f5688b4-f1ef-4870-a79e-10b149344727 from 63.03 seconds to 30 seconds
2025/09/29 19:37:03 Removing audio from video 2f5688b4-f1ef-4870-a79e-10b149344727
2025/09/29 19:37:03 Converting video 2f5688b4-f1ef-4870-a79e-10b149344727 to 720p with watermark
2025/09/29 19:37:18 Processed video: 2f5688b4-f1ef-4870-a79e-10b149344727
2025/09/29 20:07:57 Processing video: de05683b-0374-47e3-829a-c4a8d2d4f2c6
2025/09/29 20:07:57 Removing audio from video de05683b-0374-47e3-829a-c4a8d2d4f2c6
2025/09/29 20:07:57 Converting video de05683b-0374-47e3-829a-c4a8d2d4f2c6 to 720p with watermark
2025/09/29 20:07:59 Processed video: de05683b-0374-47e3-829a-c4a8d2d4f2c6
[root@ip-172-21-2-34 ~]#
```

Figura 24. Procesamiento del video (worker)

Las figuras 23 y 24 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.

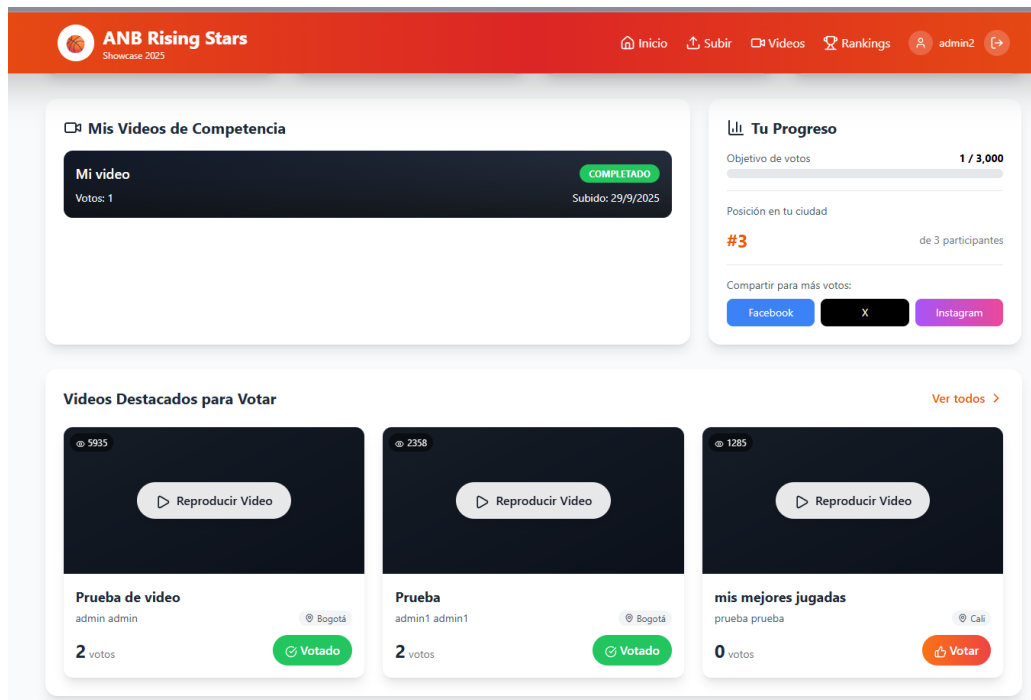


Figura 25. Confirmación de carga del video

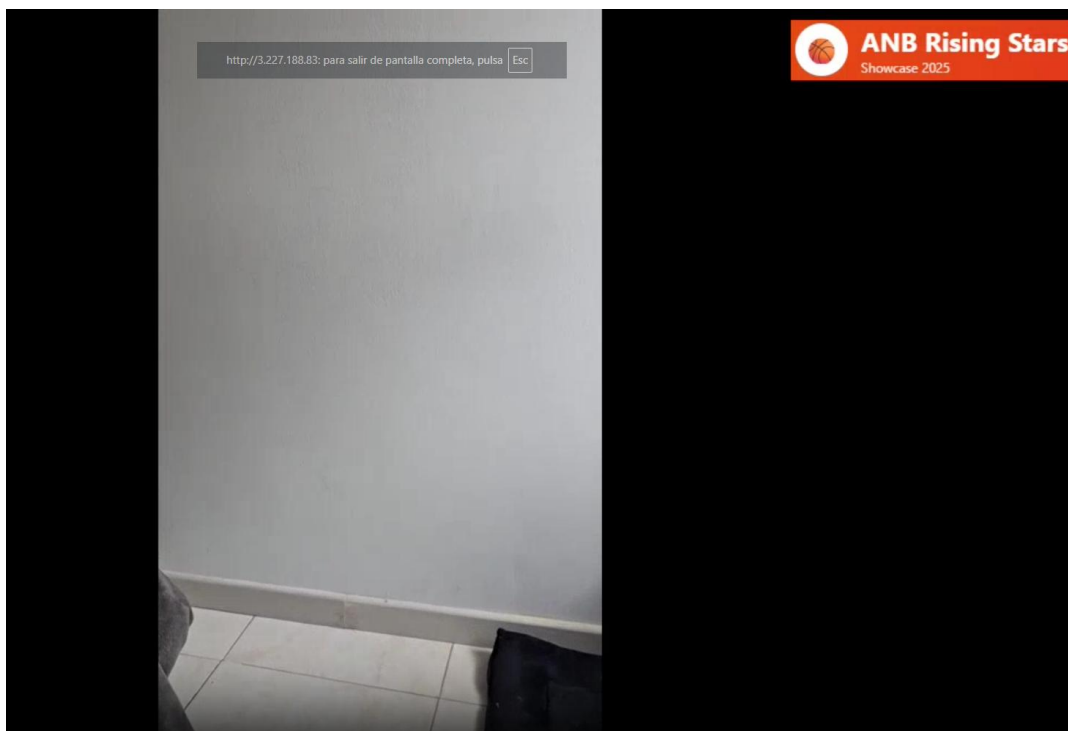


Figura 26. Prueba de reproducción del video

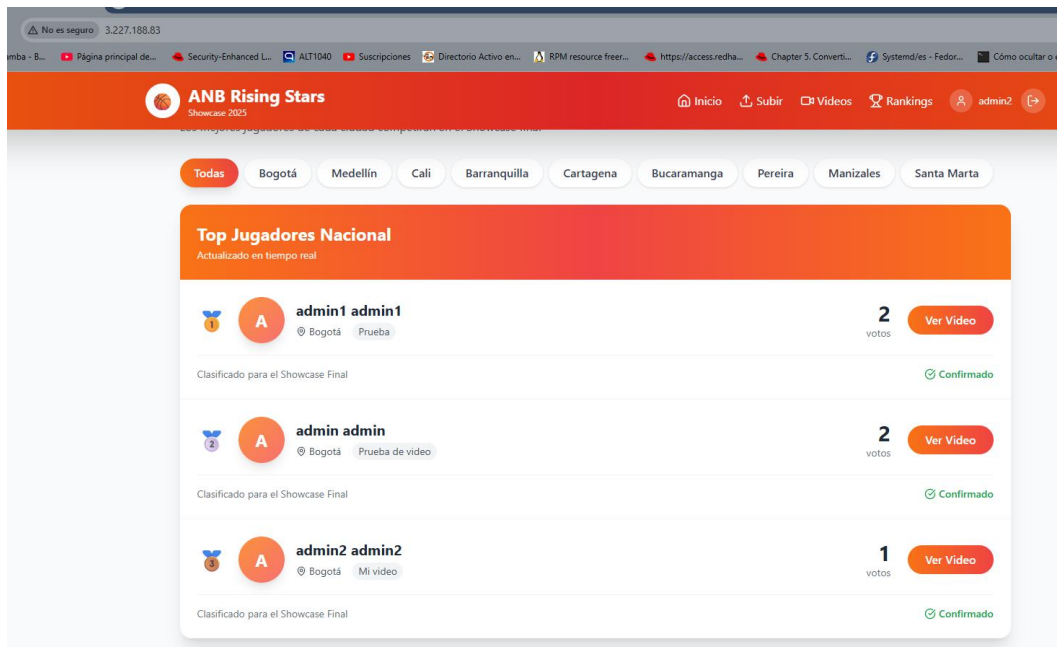


Figura 27. Ranking de videos más votados

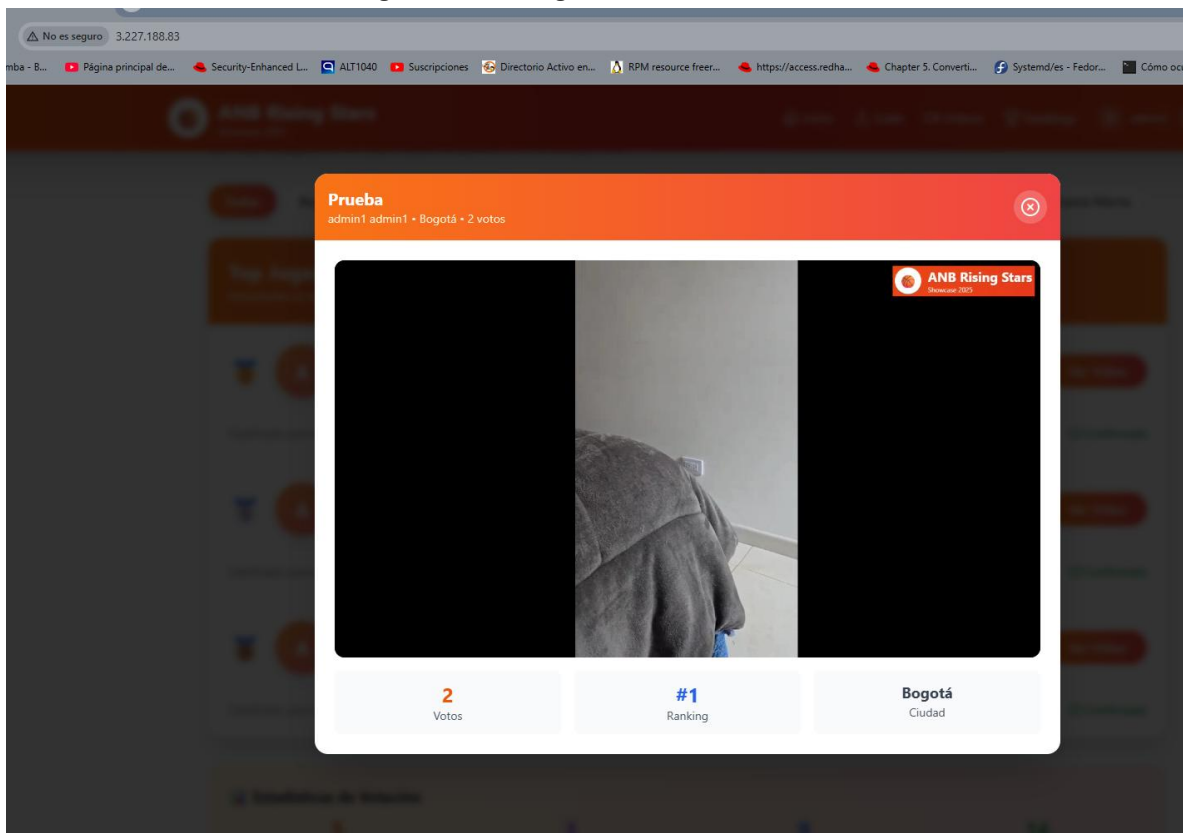


Figura 28. Visualización de videos más votados

Repositorio: [GitHub - development-cloud-solutions/Proyecto_1](https://github.com/development-cloud-solutions/Proyecto_1)

CONCLUSIONES

- La migración de la aplicación desarrollada en Go, originalmente desplegada en un único contenedor Docker bajo una arquitectura monolítica, hacia un modelo distribuido en AWS permitió desacoplar de manera exitosa sus componentes principales, favoreciendo la escalabilidad del sistema y una gestión más eficiente de los recursos.
- Las pruebas de estrés realizadas validaron el comportamiento bajo carga y identificaron oportunidades de optimización, como el ajuste de configuraciones de instancias o la incorporación de servicios como Amazon EFS en futuras versiones.
- Este proyecto no solo cumplió con los objetivos de despliegue en la nube, sino que también sentó las bases para una evolución hacia arquitecturas más elásticas y tolerantes a fallos.
- Finalmente, la documentación y los diagramas de arquitectura contribuyen a comprender cómo se encuentra distribuida la solución, facilitando el entendimiento de la operación de la aplicación y permitiendo identificar de manera más sencilla futuros problemas o mejoras que deban implementarse y validarse.

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>