Informe:
Práctica 4: Redes y Conectividad Avanzada y Contenedores

Jordi Blasco Lozano
Infraestructuras y Servicios Cloud
Universidad de Alicante

29 de octubre de 2025

Resumen

En esta práctica se ha diseñado y desplegado una arquitectura cloud para un sistema de recomendaciones, empleando segmentación de red en una VPC, balanceo de carga, despliegue canario y transición a contenedores. Se han creado subredes públicas y privadas distribuidas en dos zonas de disponibilidad, configurado reglas de control de acceso (SGs y NACLs), y se implementó un Application Load Balancer para distribuir tráfico entre versiones V1 y V2 del modelo.

Índice

1 Configuración de la VPC

1.1 Segmentación y subredes

Para el despliegue del sistema de recomendaciones se requiere una estructura específica en la VPC. La segmentación se realiza mediante bloques CIDR dedicados para cada subred, con máscara /24. El enunciado exige utilizar `10.0.10.0/24` para la subred pública principal y `10.0.20.0/24` para la privada principal. Al emplear dos zonas de disponibilidad, se crean cuatro subredes:

- Pública 1: `10.0.10.0/24`
- Pública 2: `10.0.30.0/24`
- Privada 1: `10.0.20.0/24`
- Privada 2: `10.0.40.0/24`

Cada subred permite hasta 256 direcciones IP individuales gracias a la máscara /24.

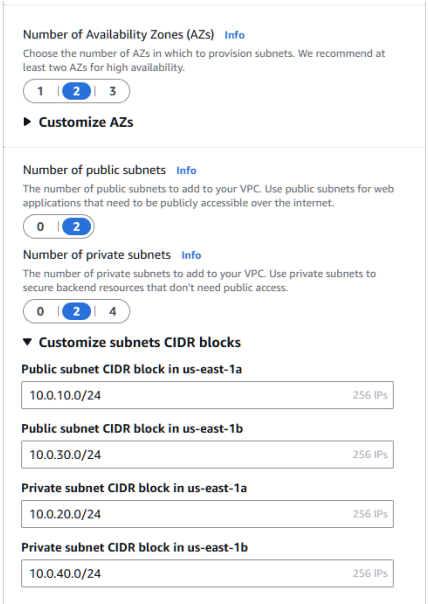


Figura 1: CIDR subredes

1.2 Creación simplificada de la VPC

Para facilitar el proceso y evitar configuraciones manuales, AWS proporciona el botón de ”VPC and more”. Seleccionando esta opción durante la creación, es posible definir directamente los bloques CIDR personalizados para cada subred desde el menú desplegable, ajustando el tamaño y las direcciones antes de crear la VPC.

1.3 NAT Gateway y Endpoint S3 automatizados

Adicionalmente, desde el mismo ”VPC and more”se permite añadir el NAT Gateway para las subredes privadas y el endpoint de S3. Esto implica que tanto el NAT Gateway como el endpoint de S3 quedarán automáticamente conectados a las tablas de rutas necesarias de cada subred, sin necesidad de modificar manualmente las tablas de ruta después de la creación de la VPC. Cuando salgamos de la pestaña de configuración obtendremos el siguiente esquema de nuestra VPC, listo y funcional.

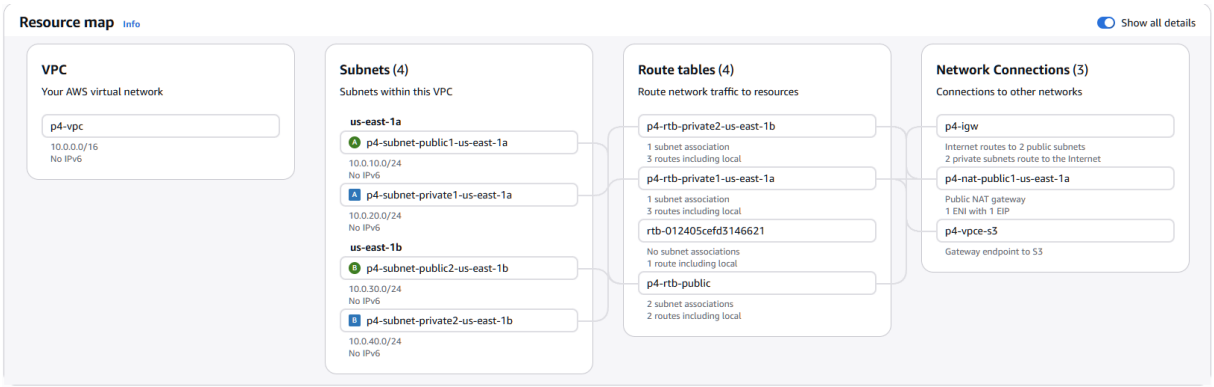


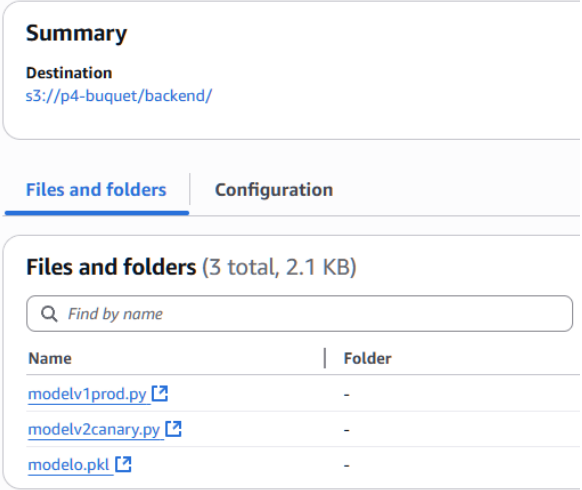
Figura 2: Esquema VPC

2 Configuración del S3

Para descargar los archivos y los modelos nos tendremos que conectar a un S3 que almacene estos datos. Esto lo haremos mediante el S3 endpoint que conectamos en el paso anterior.

Subiremos los archivos siguiendo la estructura del user data proporcionado, de forma que tengamos estos tres archivos (el modelo.pkl simplemente servirá como archivo de prueba sin valor, no se ejecutará, pero sí descargará):

- `/backend/model_v1_prod.py`
- `/backend/model_v2_canary.py`
- `/backend/modelo.pkl`



2.1 Política del bucket

Como en la práctica anterior, en esta también tenemos que permitir el acceso al bucket usando la política, de forma que permita accesos de descarga únicamente si utilizamos nuestro endpoint específico. Para esto definiremos el siguiente json:

Bloque 1: política-s3.json

```
{
    "Version": "2012-10-17",
    "Statement":
    [
        {
            "Sid": "AllowBackendFolderViaVPCE",
            "Effect": "Allow",
            "Principal": "*",
            "Action": ["s3:GetObject"],
            "Resource": "arn:aws:s3:::p4-buquet/backend/*",
            "Condition":
            {
                "StringEquals":
                {
                    "aws:SourceVpce": "vpce-0aa42de173d26c739"
                }
            }
        }

    ]
}
```

3 Configuración de la seguridad

La seguridad de esta arquitectura está estructurada en varias capas y cada componente (subred pública, subred privada, bastión, backend, ALB) tiene reglas específicas tanto en NACLs como en Security Groups, diseñadas para minimizar la superficie de ataque y garantizar que solo el tráfico necesario fluya entre los recursos.

3.1 NACL pública

Para las subredes públicas (10.0.10.0/24 y 10.0.30.0/24), las NACLs permiten la entrada de peticiones HTTP (puerto 80) desde cualquier origen para el ALB, y entrada SSH (puerto 22) únicamente desde mi IP pública, facilitando la gestión y el acceso seguro. Además, se permite la entrada de puertos efímeros provenientes de las subredes privadas, para soportar las respuestas a las conexiones originadas en las instancias backend. En cuanto a la salida, la NACL pública autoriza puertos efímeros hacia cualquier destino y la salida HTTP (puerto 80) hacia Internet para el ALB. Todo tráfico no contemplado se deniega, permitiendo sólo los flujos estrictamente necesarios.

Nº regla	Tráfico	Protocolo	Puerto	Origen	Acción
100	HTTP	TCP	80	0.0.0.0/0	Allow
110	SSH	TCP	22	Mi IP pública	Allow
120	efímero	TCP	1024-65535	10.0.20.0/24 y 10.0.40.0/24	Allow
*	All	All	All	All	Deny

Cuadro 1: NACL Pública - Reglas de entrada (pública-1 y pública-2)

Nº regla	Tráfico	Protocolo	Puerto	Destino	Acción
100	efímero	TCP	1024-65535	0.0.0.0/0	Allow
110	HTTP Resp.	TCP	80	0.0.0.0/0	Allow
*	All	All	All	All	Deny

Cuadro 2: NACL Pública - Reglas de salida (pública-1 y pública-2)

3.2 NACL privada

Las NACLs de las subredes privadas (10.0.20.0/24 y 10.0.40.0/24) son aún más restrictivas. Permiten la entrada solamente desde las subredes públicas por el puerto 8000 (tráfico backend API), SSH desde la IP privada del bastión para gestión, y entrada a través del endpoint de S3 para descargas. En cuanto a la salida, estas NACLs permiten puertos efímeros únicamente hacia las subredes públicas, para que las respuestas de las aplicaciones backend puedan volver correctamente al ALB, y salida por el puerto 443 hacia el endpoint S3 para acceso seguro a los datos almacenados. Todo lo demás queda explícitamente bloqueado.

Nº regla	Tráfico	Protocolo	Puerto	Origen	Acción
100	Backend In	TCP	8000	10.0.10.0/24 y 10.0.30.0/24	Allow
110	SSH	TCP	22	IP privada Bastión	Allow
120	S3 Endpoint	TCP	443	0.0.0.0/0	Allow
*	All	All	All	All	Deny

Cuadro 3: NACL Privada - Reglas de entrada (privada-1 y privada-2)

Nº regla	Tráfico	Protocolo	Puerto	Destino	Acción
100	efímero	TCP	1024-65535	10.0.10.0/24 y 10.0.30.0/24	Allow
110	S3 Endpoint	TCP	443	0.0.0.0/0	Allow
*	All	All	All	All	Deny

Cuadro 4: NACL Privada - Reglas de salida (privada-1 y privada-2)

3.3 Segurity grups

A nivel de Security Group, cada recurso se configura con reglas mínimas y específicas. El ALB solamente acepta conexiones HTTP desde cualquier origen y, voy a poner también que acepte SSH desde mi IP por si necesitara conectarme. Las instancias backend restringen su entrada al puerto 8000 proveniente del SG del ALB y SSH desde el SG del bastión, evitando cualquier acceso no autorizado. Finalmente, las reglas de salida permiten el flujo de datos hacia el ALB, Internet (para descargas), y hacia el endpoint S3. El bastión está configurado para aceptar conexiones SSH únicamente desde mi IP, y permite que sus conexiones salientes alcancen las instancias backend.

Tráfico	Protocolo	Puerto	Origen	Acción
HTTP	TCP	80	0.0.0.0/0	Allow
SSH	TCP	22	Mi IP pública	Allow

Cuadro 5: SG ALB - Reglas de entrada

Tráfico	Protocolo	Puerto	Destino	Acción
HTTP/Backend	TCP	8000	SG-Backend	Allow
efímero	TCP	1024-65535	0.0.0.0/0	Allow

Cuadro 6: SG ALB - Reglas de salida

Tráfico	Protocolo	Puerto	Origen	Acción
API/Backend	TCP	8000	SG-ALB	Allow
SSH	TCP	22	SG-Bastión	Allow

Cuadro 7: SG Backend - Reglas de entrada

Tráfico	Protocolo	Puerto	Destino	Acción
efímero	TCP	1024-65535	0.0.0.0/0	Allow
S3 Endpoint	TCP	443	0.0.0.0/0	Allow

Cuadro 8: SG Backend - Reglas de salida

Tráfico	Protocolo	Puerto	Origen	Acción
SSH	TCP	22	Mi IP pública	Allow

Cuadro 9: SG Bastion - Reglas de entrada

Tráfico	Protocolo	Puerto	Destino	Acción
SSH	TCP	22	SG-Backend	Allow
efímero	TCP	1024-65535	0.0.0.0/0	Allow

Cuadro 10: SG Bastion - Reglas de salida

4 Instancias

Después de haber configurado todo el entorno debemos de lanzar las instancias. Para ahorrar tiempo instalaré en una instancia de la subred publica todas las dependencias, tales como: python awscli, etc. Esto lo hare para poder crear una AMI de esta instancia con todas las dependencias instaladas. Posteriormente creare una instancia privada a partir de la AMI, a esta instancia me conectaré mediante el bastión y probare que todos los comandos del user data me funcionen correctamente. Una vez que me asegure de que el user data funciona podremos configurar el balanceador de carga.

4.1 Creación de la AMI

Una vez tengamos todo instalado creamos la AMI. He tenido que cambiar un poco los comandos, sobre todo el de awscli porque no me dejaba instalarlo de la forma en la que estaba en la práctica (no encontraba el paquete). He instalado únicamente lo siguiente en la AMI:

Bloque 2: dependencias.sh

```
sudo apt-get update -y
sudo apt-get install -y python3-pip python3-venv git nodejs npm unzip curl
sudo apt install python3-flask
pip3 install joblib
cd /tmp
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
cd -
```

Bloque 3: salida terminal instancia base

```
ubuntu@ip-10-0-10-133:/tmp$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
ubuntu@ip-10-0-10-133:/tmp$ cd -
/home/ubuntu
ubuntu@ip-10-0-10-133:~$ aws --version
aws-cli/2.31.23 Python/3.13.7 Linux/6.14.0-1011-aws exe/x86_64.ubuntu.24
ubuntu@ip-10-0-10-133:~$
```

Después instalar las dependencias cerramos la instancia y creamos la AMI a partir de la instancia anterior.

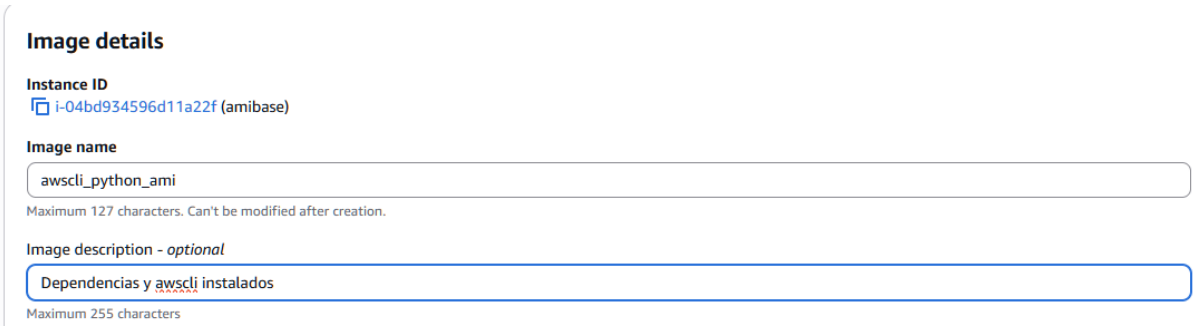


Figura 3: Creación AMI

4.2 Comprobación User Data

A partir de esta AMI podremos crear las instancias privadas y posteriormente descargar los archivos. Descargaremos los archivos de forma manual para comprobar los comandos del user data, así si nos da error, los solucionaremos antes de tener que volver a lanzar otra instancia. Para conectarnos nos bastará con conectarnos a una instancia intermedia, la cual llamamos bastión, a esta le pasaremos nuestra clave .pem mediante otra terminal usando:
`scp -i ./v2.pem ./v2.pem ubuntu@ec2-23-20-27-162.compute-1.amazonaws.com:/home/ubuntu/.` Al hacer ls en nuestra terminal del bastión ya nos saldrá la clave y podremos conectarnos a nuestra instancia privada.

Bloque 4: salida terminal bastión

```
ubuntu@ip-10-0-10-22:~$ ls
v2.pem
ubuntu@ip-10-0-10-22:~$
```

Ahora desde el bastión nos conectamos a la instancia privada y vamos probando las credenciales y comandos de descarga del user data. Después de probar bastantes veces me di cuenta que los comandos que se indican en la práctica para ingresar las claves de awscli, ingresaban las claves en `/home/ubuntu/.aws/credentials`, y no en `/root/.aws/credentials`, por lo que cambié las credenciales de lugar y sí que me funcionó la descarga del s3. Dejo a continuación la prueba que hice:

Bloque 5: salida terminal instancia privada

```
ubuntu@ip-10-0-20-189:~/app$ sudo HOME=/home/ubuntu aws s3 cp s3://p4-buquet/backend
 /modelv1prod.py .
fatal error: Unable to locate credentials
ubuntu@ip-10-0-20-189:~/app$ cat /home/ubuntu/.aws/credentials
cat: /home/ubuntu/.aws/credentials: No such file or directory
ubuntu@ip-10-0-20-189:~/app$ sudo cat /root/.aws/credentials
[default]
aws_access_key_id = ASIA2HPTD7N2H2PSGE5C
aws_secret_access_key = 2Qc1gPNpKoHJYZegKopCGZMQx3Xer9ehBS7KvNaf
ubuntu@ip-10-0-20-189:~/app$ sudo cp /root/.aws/credentials /home/ubuntu/.aws/
 credentials
cp: cannot create regular file '/home/ubuntu/.aws/credentials': No such file or
 directory
ubuntu@ip-10-0-20-189:~/app$ sudo mkdir -p /home/ubuntu/.aws
ubuntu@ip-10-0-20-189:~/app$ sudo cp /root/.aws/credentials /home/ubuntu/.aws/
 credentials
ubuntu@ip-10-0-20-189:~/app$ sudo chown ubuntu:ubuntu /home/ubuntu/.aws/credentials
ubuntu@ip-10-0-20-189:~/app$ sudo chmod 600 /home/ubuntu/.aws/credentials
ubuntu@ip-10-0-20-189:~/app$ sudo HOME=/home/ubuntu aws s3 cp s3://p4-buquet/backend
 /modelv1prod.py .
download: s3://p4-buquet/backend/modelv1prod.py to ./modelv1prod.py
ubuntu@ip-10-0-20-189:~/app$ ls
modelv1prod.py
ubuntu@ip-10-0-20-189:~/app$
```

Después de comprobar lo que fallaba, simplemente introduje el traslado de las credenciales de lugar en el user data justo antes de las descargas del s3. Finalmente lancé una instancia para probar que todo iba bien con este user data actualizado. La prueba del user data salió como esperaba y ya podemos comenzar con el paso siguiente.

Bloque 6: logs instancia

```
[   23.650346] cloud-init[915]: Completed 673 Bytes/673 Bytes (6.4 KiB/s) with 1
 file(s) remaining
download: s3://p4-buquet/backend/modelv1prod.py to ./modelv1prod.py
[   24.701384] cloud-init[915]: Completed 896 Bytes/896 Bytes (7.7 KiB/s) with 1
 file(s) remaining
download: s3://p4-buquet/backend/modelv2canary.py to ./modelv2canary.py
[   25.763120] cloud-init[915]: Completed 561 Bytes/561 Bytes (7.0 KiB/s) with 1
 file(s) remaining
download: s3://p4-buquet/backend/modelo.pkl to ./modelo.pkl
[   25.949240] cloud-init[915]: total 12K
[   25.949367] cloud-init[915]: -rw-r--r-- 1 ubuntu ubuntu 561 Oct 28 12:15 modelo.
 pkl
[   25.949582] cloud-init[915]: -rw-r--r-- 1 ubuntu ubuntu 673 Oct 28 12:15
 modelv1prod.py
[   25.949784] cloud-init[915]: -rw-r--r-- 1 ubuntu ubuntu 896 Oct 28 12:15
 modelv2canary.py
```

Los endpoinds también se están ejecutando adecuadamente

Bloque 7: consola endpoints

```
ubuntu@ip-10-0-20-16:~$ curl -X POST http://127.0.0.1:8000/api/v1/recommendation -H
 "Content-Type: application/json" -d '{"user_id": "test"}'
{"event_log":"User test purchase processed.","latency_ms":0,"model_id":"V1 - Stable"
 ,"recommendation":["itemX_oldmodel","itemY_oldmodel"]}
ubuntu@ip-10-0-20-16:~$
```

5 Configuración del Balanceador de Carga (ALB)

5.1 Target grups

Para esta parte debemos de tener corriendo 3 instancias con la versión de python_v_1 y 1 instancia con la versión de python_v_2. Esto lo haremos para crear los dos target groups, el de la V1 y el de la V2. Procederemos a crear 4 instancias privadas a partir de nuestra AMI y de nuestro user data, una vez las tengamos ejecutando nos conectamos una a una y vamos ejecutando el archivo python que corresponda a cada una, a las V1 `modelv1prod.py` y a la V2 `modelv2canary.py`, lo hacemos con 'nohup' para poder cerrar la terminal y que sigan funcionando en segundo plano. Debemos de tener estas instancias corriendo escuchando en el puerto correspondiente para poder configurar el ALB con estos target groups.

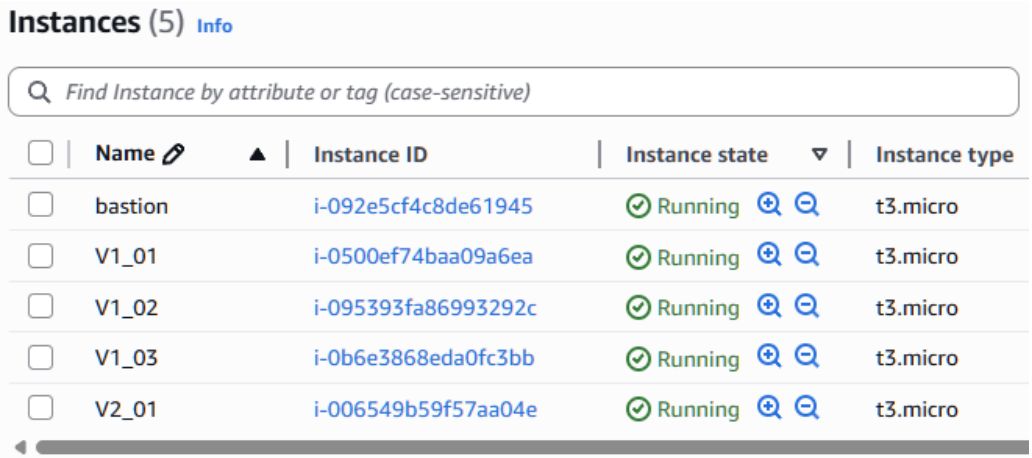


Figura 4: Instancias

Configuramos los dos targets al puerto 8000 con las instancias que hemos creado, de esta forma:

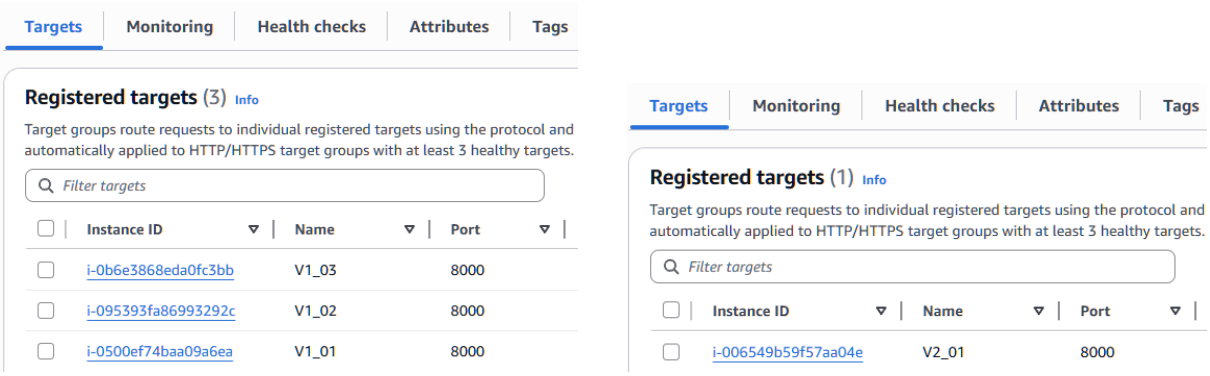


Figura 5: TG-V1 y TG-V2

5.2 ALB y pruebas

Cuando tengamos los grupos creamos la ALB de forma pública usando nuestra vpc de la práctica, el SG ALB y las subredes públicas. El listener lo ponemos en el puerto 80 como indica el enunciado y ajustamos los pesos de 90 % para V1 y 10 % para V2

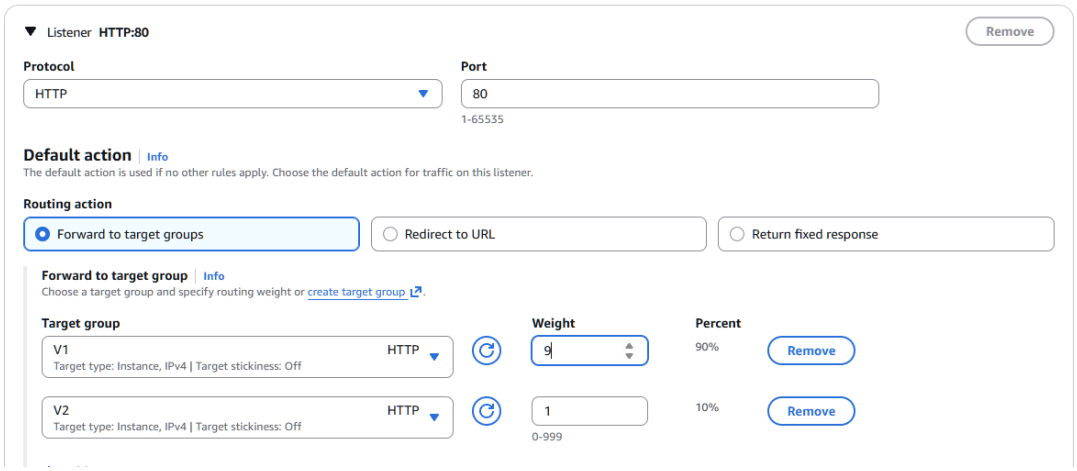


Figura 6: ALB

Al tener el ALB corriendo podemos probar nuestros endpoinds desde el nombre de dns que nos proporciona AWS. Comprobamos como una de cada diez veces aproximadamente nos sale que se está usando el modelo V2.

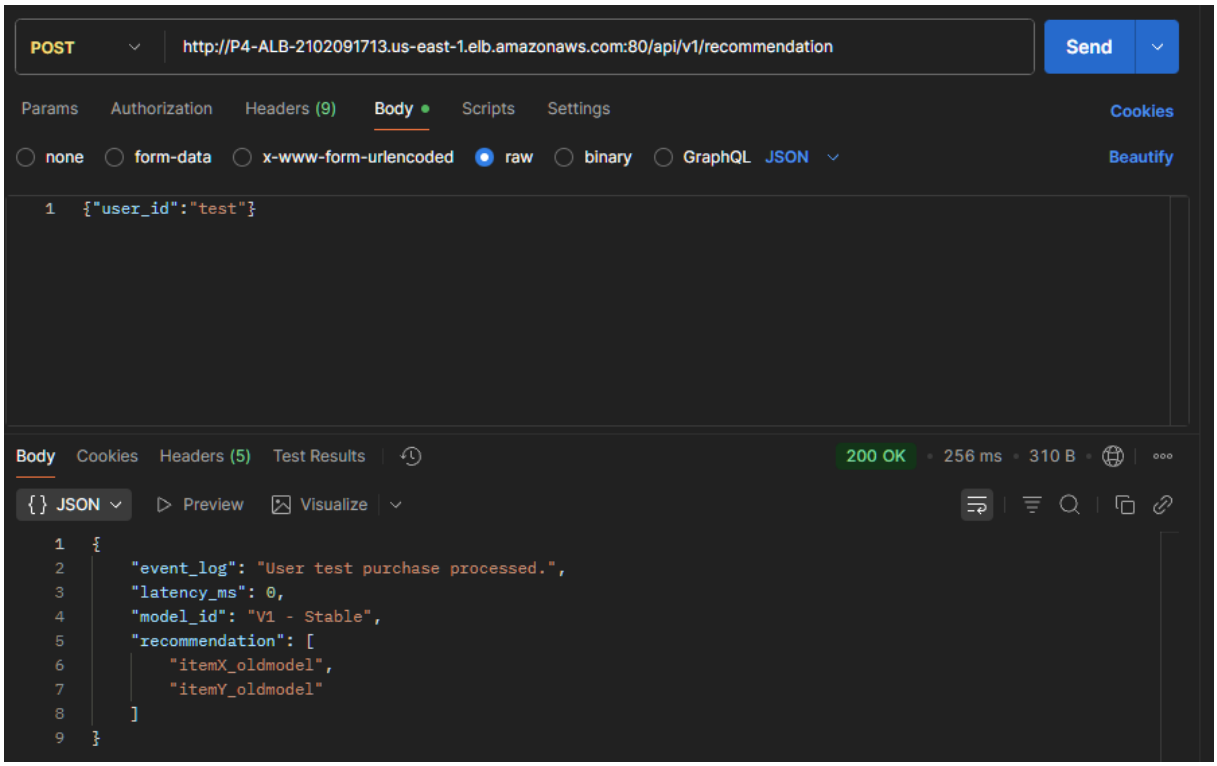


Figura 7: prueba V1

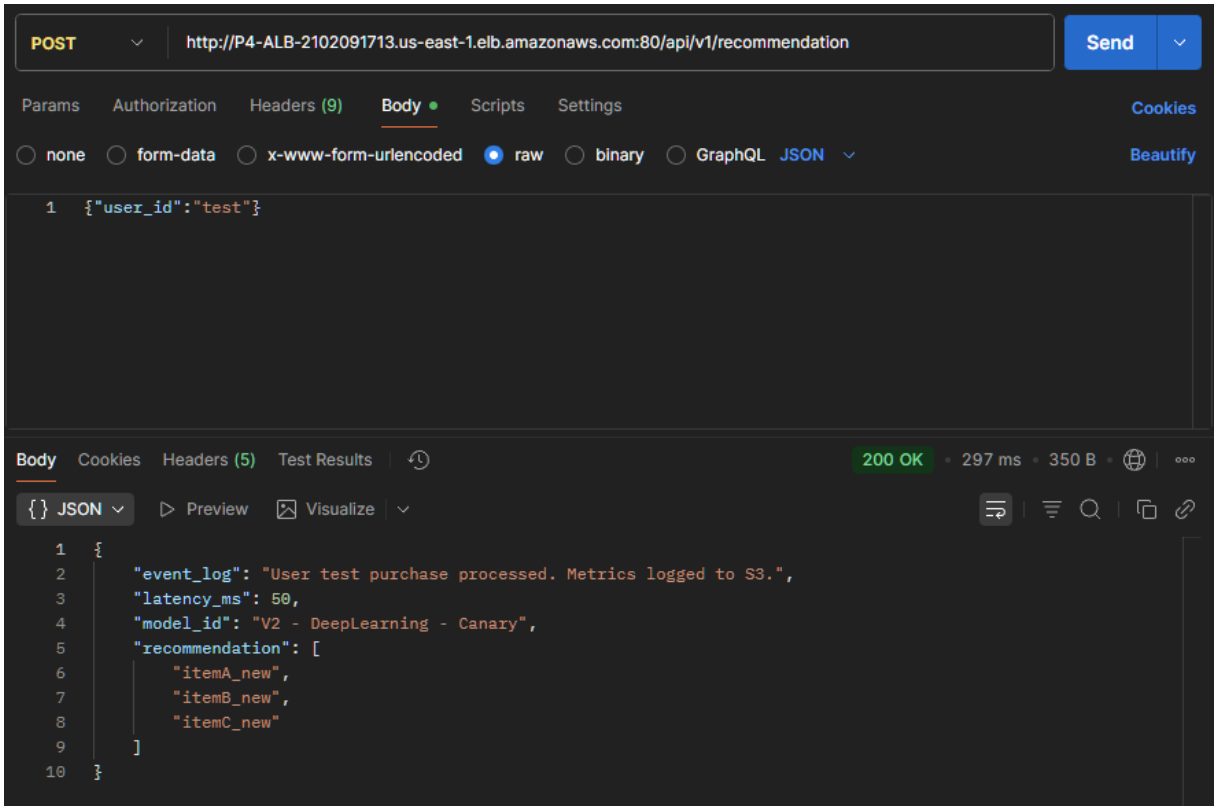


Figura 8: prueba V2

6 Transicion a contenedores

Para este apartado ejecutaremos nuestra aplicación canary dentro de un contenedor docker. Lanzamos una instancia, esta vez puede ser sin la AMI ya que debemos de instalar por nuestra cuenta el docker únicamente. Instalaremos el docker, descargaremos nuestro modelv2canary.py desde el s3, o para mayor control, y así darnos cuenta de que proviene de la version con docker, podemos copiar y pegar una version del .py modificada que contenga el nombre de modelo como V2-docker. Cuando tengamos el archivo .py dentro de la instancia crearemos un dockerfile que copiara el archivo .py en el contenedor y lo ejecutará. Finalmente construiremos la imagen a partir del dockerfile y la lanzaremos mapeando el puerto 80 de la instancia con el puerto 8000 del contenedor.

Bloque 8: Consola isntancia docker

```
ubuntu@ip-10-0-20-71:~$ nano modelv2canary.py
(copypaste)
ubuntu@ip-10-0-20-71:~$ nano Ddockerfile
FROM python:3.10-slim
WORKDIR /app
COPY modelv2canary.py .
RUN pip install flask
EXPOSE 8000
CMD ["python", "modelv2canary.py"]
ubuntu@ip-10-0-20-71:~$ sudo docker build -t modelv2canaryimage .
...
...
Successfully built 5865039f3e08
Successfully tagged modelv2canaryimage:latest
ubuntu@ip-10-0-20-71:~$ sudo docker run -d -p 80:8000 modelv2canaryimage
02af51a5fb4c5d18145061323f93872ed9311caa210b0e1d2db38dc9e0158140
ubuntu@ip-10-0-20-71:~$ sudo docker ps
CONTAINER ID   IMAGE                COMMAND                  CREATED          STATUS
           PORTS                                   NAMES
02af51a5fb4c   modelv2canaryimage   "python …modelv2canar"   55 seconds ago   Up 54
 seconds   0.0.0.0:80->8000/tcp, [::]:80->8000/tcp   gifted_chatterjee
ubuntu@ip-10-0-20-71:~$
```

6.1 Nueva regla SG Backend

Como estamos conectando el puerto 80 en vez de el 8000, debemos de cambiar la configuración del SG Backend para que permita la entrada del puerto 80 desde el SG ALB. También al crear el target group seleccionamos el puerto 80 en vez del 8000 y conectamos la instancia docker.

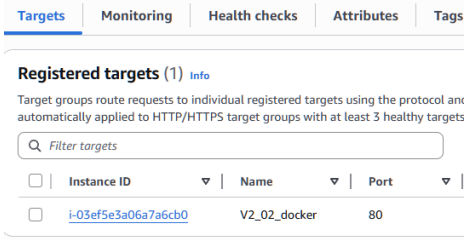


Figura 9: TG V2 docker

6.2 Editar listener

Cuando tengamos el TG editamos el listener añadiendo otro grupo. No se si debía incluir TG V2 docker nuevo o sustituirlo por el anterior TG V2, pero optaré por incluir el nuevo TG de forma que tengamos el 80 % de trafico para V1, el 10 % para V2 normal y el ultimo 10 % para el V2 con docker. Al incluirlo lo probamos con el postman.

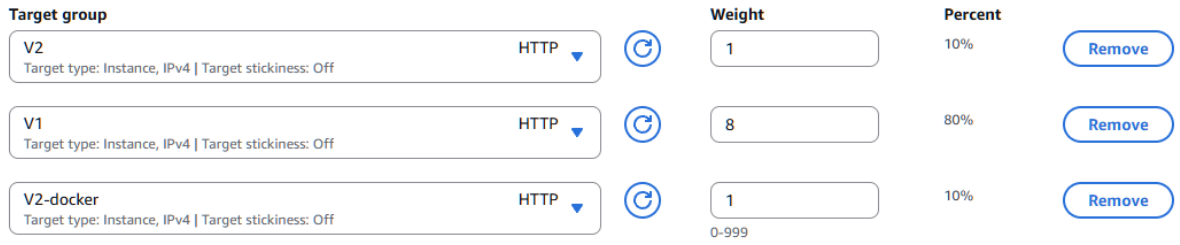


Figura 10: ALB final

6.3 Pruebas endpoint del ALB

Y como podemos observar en la última captura, nos redirige de forma correcta al endpoint del contenedor.

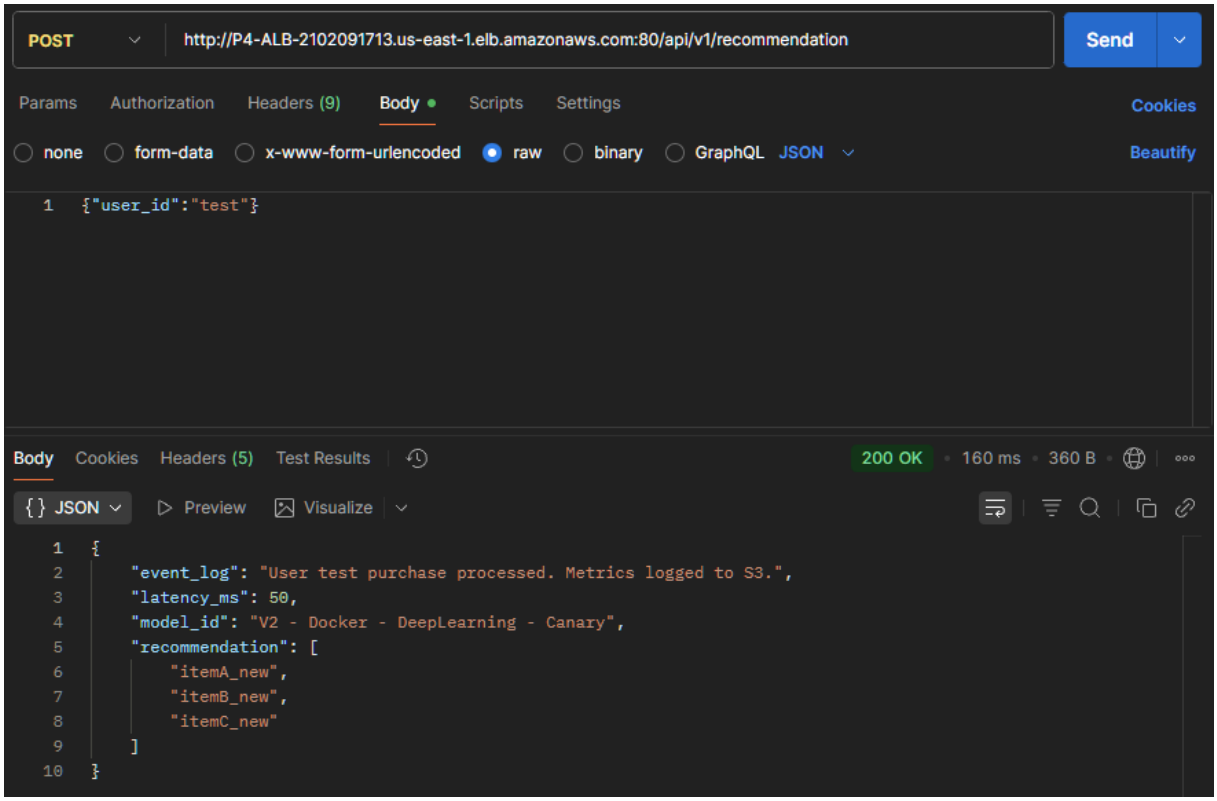


Figura 11: post docker

7 Preguntas de reflexión

7.1 Seguridad y NACLs

Aunque el SG-BACKEND ya filtra el tráfico no deseado al puerto 8000, la NACL añade una capa adicional de defensa a nivel de red (network layer) para todos los recursos de la subred, independiente de los SG de cada instancia. Así, mitiga riesgos ante errores en configuración de SGs y protege ante tráfico malicioso de otras fuentes dentro de la VPC. Un atacante no podría interceptar la respuesta de la API usando solo la NACL, ya que una NACL únicamente permite o deniega paquetes; no tienen capacidad de leer, modificar ni reenviar tráfico como lo haría un proxy. La confidencialidad y protección de datos requieren cifrado (HTTPS) además del filtrado de red.

7.2 WAF vs. NACL

Un ataque de Inyección de Código SQL sería detenido por el WAF (Web Application Firewall) y no por la NACL. El motivo es que el WAF inspecciona y filtra el contenido de las cargas útiles HTTP, detectando patrones peligrosos en la capa de aplicación. En cambio, la NACL solo filtra por IP, puerto y protocolo, sin analizar el contenido, por lo que nunca detectaría ataques específicos de aplicaciones.

7.3 VPC Endpoints

El VPC Endpoint permite que el tráfico de logging del Modelo V2 (por ejemplo, guardar logs en S3) viaje directamente entre la VPC y el servicio de AWS sin atravesar el Internet Gateway (IGW). Esto significa que los datos nunca salen a Internet, manteniendo toda la transferencia dentro de la red privada de AWS y, por tanto, aumentando la seguridad y reduciendo la latencia.

7.4 Redes de Contenedores

El Balanceador de Carga debe dirigir el tráfico a la IP privada de la VM Host Docker y al puerto 80 (el puerto mapeado externamente). Docker se encarga de reenviar ese tráfico interno al puerto 8000 del proceso Flask en el contenedor. El componente concreto de red de Docker responsable de esta traducción de puertos se llama Docker bridge network + Port Mapping (NAT); es decir, Docker redirige automáticamente las peticiones del puerto publicado en el host al puerto interno del contenedor.

7.5 Rollback Rápido

En un canary deployment, el empleo de contenedores Docker permite una reversión a la versión V1 mucho más rápida y segura que si el V2 estuviera en una VM tradicional. Con Docker, basta detener y eliminar el contenedor V2 para cortar completamente el tráfico nuevo, sin eliminar la VM ni reconfigurar el sistema operativo. Esto reduce a segundos el tiempo de rollback y minimiza los riesgos de "código residual", facilitando además automatización y consistencia en los cambios.