TRABAJO DE GRADO
Opción Seminario-Diplomado.

Infraestructura Escalable y Virtualización en AWS: De la Teoría a la Práctica

Corporación Universitaria Remington.
Facultad de ingeniería
Ingeniería de sistemas

Juan Sebastián Rodríguez Avendaño
Juan Pablo Berrio Lopez

Seminario-Diplomado.
2025

Dedicatoria

A Dios, por darme la fortaleza, la sabiduría y la perseverancia necesarias para culminar este trabajo. Sin su guía, nada de esto habría sido posible.

Tabla de Contenidos

Resumen

En este proyecto se discute de forma general y práctica, algunos de los servicios en la nube con el proveedor Amazon Web Services (AWS). Por un lado, se desarrolla la historia que detalla dónde comenzó la virtualización y cómo esta tecnología evolucionó hasta lo que hoy conocemos como computación en la nube. También se detalla una línea del tiempo para comprender los logros e hitos claves desde la década de 1960 hasta la actualidad, centrándose en algunos de los proveedores más importantes desde el inicio de la virtualización y su impacto en el desarrollo de infraestructura de software. Al final de esta primera sección se comparan los nombres de los servicios de los 3 proveedores de servicios en la nube más grandes de la actualidad como lo son Amazon Web Services, Google Cloud y Microsoft Azure, destacando, así como los proveedores comercializan los mismos tipos de servicios en áreas como bases de datos, contenedores, máquinas virtuales, manejo de datos, entre otros, pero cada proveedor nombrándolos de una forma específica

También de forma práctica se comparten dos ejemplos de implementaciones de infraestructura en AWS simulando recursos de un entorno real o de producción. En el primer ejercicio se demuestra cómo construir una arquitectura escalable basada en instancias EC2, implementando a la par un balanceador de carga y escalado automático cuando la carga de la CPU aumenta según ciertas políticas.

Por otro lado, en el segundo ejercicio, se trató de desarrollar un entorno para desplegar múltiples contenedores Docker en una única instancia de máquina virtual (EC2), con el fin de que cada despliegue sirviera a una dirección web o DNS diferente, y que al final a través de un despliegue usando funciones de NGINX, se pudiera implementar un proxy inverso dependiendo del dominio solicitado.

Es así como desde el desarrollo teórico practico, se busca demostrar la implementación e soluciones reales, organizadas y funcionales al usar herramientas de proveedores en la nube como AWS.

Palabras clave

Virtualización, AWS, Contenedores Docker, Escalabilidad, Proxy inverso.

Marco conceptual y contextual

1.1 Marco contextual

En este documento se busca la comprensión y desarrollo del seminario de Amazon Web Services (AWS). Se puede decir que en los últimos años, el desarrollo de software y la nube han estado relacionados ya que todo profesional u organización necesita implementar una serie de soluciones de infraestructura con el fin de poder automatizar y virtualizar ciertas soluciones de los proyectos de desarrollo de software.

De esta forma, contextualizando este proyecto, se busca dar una breve explicación de las soluciones disponibles en la nube, no sin antes dar un contexto de su inicio, evolución y transformación, además de incluir a los proveedores principales y realizar un proyecto para realzar la teoría. Finalmente, en este proyecto se refleja cómo las ideas del seminario podrían implementarse en soluciones reales dentro de un entorno profesional.

1.2 Definición de conceptos:

1.1 Virtualización

Te permite convertir un solo servidor físico en varios “computadores virtuales”, cada uno con su propio sistema operativo. Así aprovechas mejor el hardware y reduces costos.

1.2 Computación en la Nube

En vez de comprar y mantener servidores, “rentas” los recursos que necesites por internet. Proveedores como AWS te dan servicios listos para usar (bases de datos, almacenamiento, etc.) y tú solo pagas por lo que consumes.

1.3 EC2 (Elastic Compute Cloud)

Es el servicio de AWS que te deja crear servidores virtuales en la nube. Tienes total control: eliges el sistema operativo, las aplicaciones, las configuraciones, etc.

1.4 Auto Scaling Group (ASG)

Es como un sistema “automático” que enciende más servidores cuando hay mucha demanda y los apaga cuando baja la carga. Así optimizas recursos y ahorras dinero.

1.5 Load Balancer (ALB)

Reparte las solicitudes entre varios servidores para evitar sobrecargas. Piensa en él como un semáforo inteligente que dirige el tráfico de la forma más eficiente.

1.6 VPC (Virtual Private Cloud)

Es tu propia “red privada” en AWS. Allí controlas subredes, reglas de acceso y rutas, manteniendo un ambiente seguro y bien organizado.

1.7 Docker

Es una herramienta que “empaqueta” tus aplicaciones junto con todo lo que necesitan para funcionar. Te facilita que corran igual en cualquier lugar, sin importar el entorno.

1.8 Contenedores Docker

Son mini “cajas” o entornos aislados que incluyen solo lo esencial para tu aplicación. Ocupan menos recursos que una máquina virtual y arrancan súper rápido.

1.9 NGINX como Proxy Inverso

NGINX recibe las peticiones y las redirige al contenedor o servicio correcto según el dominio o la ruta. Te permite administrar varios servicios en un mismo servidor.

1.10 Proxy Inverso

Hace de “intermediario” entre el cliente y tu servicio. Te ayuda a tener muchos servicios detrás de una sola IP, mejorar la seguridad y organizar mejor tu tráfico.

1.11 Grupos de Seguridad

En AWS funcionan como un firewall. Decides qué puertos y protocolos pueden entrar o salir de tus servidores (por ejemplo, SSH, HTTP, etc.).

1.12 Plantilla de Lanzamiento

Guarda una configuración predefinida (SO, scripts, seguridad) para crear nuevas instancias EC2 con solo unos clics, reduciendo errores y ahorrando tiempo.

1.13 Prueba de Esfuerzo en TI

Simulas mucha carga en tu sistema para ver cómo responde. Sirve para comprobar si tu Auto Scaling realmente añade (o elimina) servidores cuando se necesita.

1.14 Subred Pública y Subred Privada

Las públicas tienen acceso a internet, las privadas no. Esto ayuda a proteger recursos sensibles (por ejemplo, bases de datos) y optimizar la seguridad.

1.15 DNS

Traduce nombres de dominio (como “ejemplo.com”) a direcciones IP. Es la “guía telefónica” de internet, sin la cual tendríamos que recordar puros números.

1.16 Archivos host

Es un archivo local en tu computadora para “simular” un DNS. Puedes apuntar nombres de dominio a direcciones IP específicas sin afectar a los demás usuarios de internet.

Desarrollo e implementación del aprendizaje

1.1 ¿Cuáles fueron los orígenes de la virtualización y la computación en la nube?

La virtualización en sí misma combina conceptos que se remontan a la década de 1960, cuando IBM estaba trabajando en una tecnología para dividir un mainframe en varias máquinas virtuales. Dicho de otra manera, permitía a diferentes usuarios trabajar en una misma máquina física en paralelo. Esto aumentó enormemente la utilización del hardware, lo cual era muy costoso en esos días. Y a pesar de ser muy limitada y costosa en aquel entonces, fue el primer paso hacia algo mucho más grande en el futuro.

El tiempo pasó, y estas nuevas soluciones realmente comenzaron a aparecer con más frecuencia alrededor del cambio de siglo, cuando la virtualización de servidores de alta gama comenzó a trasladarse a modelos más bajos. Un actor principal fue VMware, que permitió que un servidor físico ejecutara más de un sistema operativo. Esto significaba que para muchas empresas tenía sentido ahorrar dinero en los costos de operación de los centros de datos, así como en sus facturas de electricidad.

A medida que las velocidades y la fiabilidad de Internet mejoraron, esta idea comenzó a popularizarse, y el concepto de virtualización fue la base de la computación en la nube. En efecto, una empresa ya no tenía que organizar y operar su propio servidor informático, podía alquilar espacio con potencia de procesamiento, reduciendo costos en general y eliminando la necesidad de que la empresa adquiriera hardware.

Después de que el Profesor introdujo este concepto, las ramas del árbol comenzaron a crecer a partir de esto; algunas organizaciones crearon sus propios negocios en línea siguiendo estas ideas. Cuando Amazon abrió públicamente AWS con servicios como EC2 (cómputo) y S3 (almacenamiento) en 2006, fue un hito de la computación en la nube. Luego vinieron otros actores, como Microsoft y Google, que comenzaron a entrar en este mercado.

Es así como se puede concluir que la virtualización ayudó en la evolución de la computación en la nube basada en la utilización óptima de recursos. Lo que comenzó como un medio para compartir un mainframe evolucionó hasta convertirse en la base de la computación moderna. Hoy en día, hay pocos servicios digitales que las personas utilizan que no dependen de alguna manera de la nube, cuya evolución aún está siendo escrita mientras redefine el hardware de TI.

1.2 Línea de tiempo: Los hitos clave que han llevado a la nube tal como la conocemos hoy en día

1. **Cronología: Los Principales Hitos que Han Formado la Nube Hoy en Día**

- **Años 1960 – IBM funda la virtualización**: IBM desarrolló un sistema que particionaba sus enormes mainframes, permitiendo que múltiples usuarios los accedieran al mismo tiempo, lo que ahora conocemos como compartir tiempo. La palabra "virtualización" aún no había sido inventada, pero la idea de multiplexar recursos a nivel de hardware ya había comenzado a ser considerada. Fue un paso significativo hacia la nube que conocemos ahora.

- **Años 1970 – Sistemas Operativos Más Avanzados:** En esta década se hicieron importantes avances en el diseño de sistemas operativos capaces de gestionar numerosos usuarios y proceso. En este año se hicieron visibles conceptos clave como máquinas virtuales, hipervisores y particionamiento de recursos. Esto se convertiría en la base para la futura virtualización de equipos, aunque en ese tiempo aún era muy costoso implementar estos recursos en la mayoría de las grandes empresas y universidades.

- **Años 1980 – Mayor Conectividad y Redes:** Aunque todavía no se usaba el término nube, esta época trajo muchos avances en redes informáticas, incluyendo ARPANET y más tarde TCP/IP. Esto permitió que las computadoras interactuaran entre sí a gran escala.

- **Años 1990 – VMware:** El nacimiento de la virtualización moderna: Los años 1990 sentaron las bases de la virtualización de servidores tal como la conocemos. VMware tenía decenas de títulos de ingeniería y una fecha de nacimiento en 1998 ya haciendo muchas cosas, incluyendo permitir que múltiples sistemas operativos operaran en un solo servidor físico. Transformó el uso de los servidores, eliminando la necesidad de miles de dólares y metros cuadrados de costoso hardware.

- **2002 – Permitiendo a AWS Desarrollar Su Capacidad para Dejar Su Huella:** Aunque la mayoría de nosotros conocimos AWS en 2006, Amazon había comenzado a construir infraestructura en 2002. Estaban tratando de resolver sus propios problemas de escalabilidad y terminaron creando, en cierto sentido, un servicio que más tarde proporcionaron a otros. Ese fue el verdadero comienzo de la nube como un servicio.

- **2006 – El Lanzamiento Oficial de Amazon Web Services (AWS):** Este año, el ecosistema de AWS introdujo una nueva magnitud a la historia: EC2 (para máquinas virtuales) y S3 (para almacenamiento de archivos). Ya no era necesario poseer tu propio hardware; podías "alquilar" recursos y solo pagar por lo que consumieras. Esto fue un cambio radical para el desarrollo de software y atrajo a muchas nuevas empresas.

- **2010 – Introducción de Microsoft Azure, Mencionado como un Competidor:** Microsoft introdujo su nube, Azure, a este género. Al principio, no tenía la amplitud y escala que tenía AWS, pero tenía algo muy importante: la interactividad con todos los productos de Microsoft. Esto resultó atractivo para muchas grandes corporaciones que ya trabajaban con Windows, SQL Server o Active Directory.

- **2011 – Google Cloud Platform Entra para Competir con GCP:** Google quería una parte del mercado, por lo que lanzó Google Cloud Platform. Este proveedor comenzó proveyendo recursos relacionados con el almacenamiento y el análisis de datos y luego ampliaron su cobertura. Su contribución principal fue el sistema de Kubernetes, que Google lanzó en 2014 y desde entonces se ha convertido en el estándar para ejecutar contenedores en la nube.

- **2014 – Kubernetes y la Explosión de los Contenedores:** Docker ya existía, pero Kubernetes permitió que la contenedorización floreciera. Esto permite a las empresas automatizar la gestión de contenedores, un aspecto importante de las aplicaciones contemporáneas. La mayoría de las empresas hicieron la transición a arquitecturas más elásticas y escalables con Kubernetes, lo que facilitó aún más la vida de los desarrolladores en la nube.

- **Desde 2016 – Servicios Sin Servidor:** AWS Lambda y Azure Functions dominan. Finalmente, se introdujeron las funciones de Google Cloud. Tal idea te permite ejecutar tu aplicación sin preocuparte por las granjas de servidores y otros servicios de back-end. Por ejemplo, durante la década pasada o más, los tres proveedores, Mahout, SparkML y MLlib, entrenaron toneladas de modelos de aprendizaje profundo y machine learning en sus respectivas nubes, lo que permitía a cualquiera usar sus modelos preentrenados al instante sin necesariamente necesitar tener recursos de hardware para hacerlos funcionar, e incluso entrenar otros modelos.

- **2020 en Adelante – Uso de Múltiples Nubes:** Muchas organizaciones están utilizando múltiples nubes simultáneamente o mezclando la nube con su hardware privado. También hemos visto como diversos sectores adoptan esta ola de tecnología. Desde la educación hasta la salud, el transporte (¡compartición de viajes!) hasta el

entretenimiento, y todo lo que hay en medio. Hoy en día, la nube es uno de los instrumentos fundamentales de la sociedad contemporánea.

1.3. Comparación de servicios entre AWS, Azure y GCP

1.3.1 Cómputo bajo demanda (máquinas virtuales): permite ejecutar servidores virtuales.

- AWS: EC2 (Elastic Compute Cloud)
- Azure: Virtual Machines
- GCP: Compute Engine

1.3.2 Contenedores administrados: ejecución de contenedores sin administrar VMs.

- AWS: ECS / Fargate
- Azure: Container Instances
- GCP: Cloud Run

1.3.4 Orquestación de contenedores: gestión de clústeres de Kubernetes.

- AWS: EKS (Elastic Kubernetes Service)
- Azure: AKS (Azure Kubernetes Service)
- GCP: GKE (Google Kubernetes Engine)

1.3.5 Almacenamiento de objetos: guardar archivos de manera escalable.

- AWS: S3
- Azure: Blob Storage
- GCP: Cloud Storage

1.3.6 Almacenamiento en bloque: almacenamiento para VMs.

- AWS: EBS (Elastic Block Store)
- Azure: Disk Storage
- GCP: Persistent Disk

1.3.7 Base de datos relacional: bases de datos como MySQL, PostgreSQL.

- AWS: RDS
- Azure: SQL Database
- GCP: Cloud SQL

1.3.8 Base de datos NoSQL (clave-valor/documentos):

- AWS: DynamoDB

- Azure: Cosmos DB
- GCP: Firestore / Datastore

1.3.9 Base de datos en memoria: para caché o sesiones.

- AWS: ElastiCache (Redis/Memcached)
- Azure: Azure Cache for Redis
- GCP: Memorystore

1.3.10 Servidor de archivos compartidos (NFS):

- AWS: EFS (Elastic File System)
- Azure: Azure Files
- GCP: Filestore

1.3.11 Funciones serverless: ejecutar código sin servidores.

- AWS: Lambda
- Azure: Azure Functions
- GCP: Cloud Functions

1.3.12 Mensajería (colas):

- AWS: SQS (Simple Queue Service)
- Azure: Azure Queue Storage / Service Bus
- GCP: Pub/Sub

1.3.13 Notificaciones push y pub/sub:

- AWS: SNS (Simple Notification Service)
- Azure: Event Grid / Notification Hubs
- GCP: Pub/Sub

1.3.14 Balanceador de carga:

- AWS: ELB (Elastic Load Balancer)
- Azure: Load Balancer / Application Gateway
- GCP: Cloud Load Balancing

1.3.15 DNS administrado:

- AWS: Route 53
- Azure: Azure DNS

- GCP: Cloud DNS

1.3.16 CDN (entrega de contenido):

- AWS: CloudFront
- Azure: Azure CDN
- GCP: Cloud CDN

1.3.17 Autenticación y gestión de identidades:

- AWS: Cognito / IAM
- Azure: Azure Active Directory
- GCP: Identity Platform / IAM

1.3.18 Monitoreo y logs:

- AWS: CloudWatch
- Azure: Monitor / Log Analytics
- GCP: Operations Suite (antes Stackdriver)

1.3.19 DevOps / CI-CD:

- AWS: CodePipeline / CodeBuild / CodeDeploy
- Azure: Azure DevOps / GitHub Actions
- GCP: Cloud Build / Cloud Deploy

1.3.20 Data warehouse / análisis de grandes datos:

- AWS: Redshift
- Azure: Synapse Analytics
- GCP: BigQuery

1.3.21 ETL y procesamiento de datos:

- AWS: Glue / Data Pipeline
- Azure: Data Factory
- GCP: Dataflow

1.3.22 Machine Learning / IA:

- AWS: SageMaker
- Azure: Azure Machine Learning

- GCP: Vertex AI

1.3.23 Almacenamiento de respaldo y recuperación:

- AWS: Backup / Glacier
- Azure: Azure Backup / Recovery Services Vault
- GCP: Backup and DR

1.3.24 Seguridad / Administración de llaves (KMS):

- AWS: KMS (Key Management Service)
- Azure: Key Vault
- GCP: Cloud KMS

2. Implementación de un balanceador de carga con autoscaling:

En las instrucciones se tiene contemplado lo siguiente para este proyecto:

- Las instancias deben ser Linux, debe tener una aplicación web básica que cargue automáticamente al crearse la instancia. Se debe mostrar cómo funciona el autoscaling al terminar instancias manualmente.

- Se debe crear una política de autoscaling al tener un aumento de cpu de un umbral definidio. Ej: 50%

2.1 Propósito de los proyectos a desarrollar:

2.1.1 Desarrollo de infraestructura en Amazon Web Services (AWS) que será escalable, incluyendo las siguientes características:

- Amazon EC2 junto con Auto Scaling Group (ASG)
- Balanceador de carga (ALB)
- VPC personalizada incluyendo subredes públicas
- Grupos de seguridad personalizados
- Plantilla de lanzamiento con la aplicación web instalada
- Página HTML personalizada que contiene detalles de la instancia
- Realización de una prueba de escalado automático mediante el incremento del uso de CPU.

2.1.2 Implementar la ejecución de múltiples contenedores Docker y redirigir el tráfico usando proxy reverso con las siguientes características:

- 3 contenedores cada uno con un html distinto
- 1 contenedor con NGINX que actuará como proxy reverso
- Redirección basada en dominios

2.2 Desarrollo de proyecto 1:

2.2.1 Creación de VPC y configuración de red:

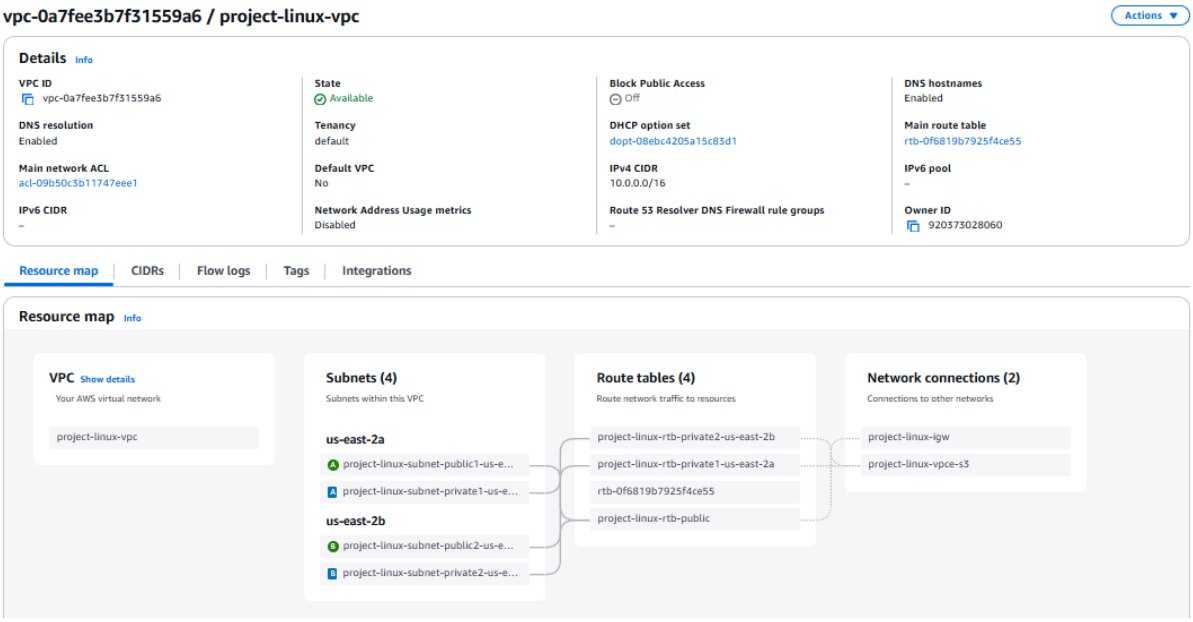
- Nombre: project-vpc
- CIDR IPV4: 10.0.0.0/16
- Sin bloque IPV6
- 2 zonas de disponibilidad (AZ): us-east-2a, us-east-2b
- Creación de subred: Se crearon automáticamente 2 públicas, 2 privadas
- Puerta de Enlace a Internet: Configurada automáticamente

En este primer punto se creó una VPC personalizada con el nombre de “de project-linux-vpc”. A la misma se le configuró un bloque de CIDR con un valor de “10.0.0.0/16” el cual nos permite trabajar en una base bastante grande de direcciones IP para poder ser implementadas en subredes futuras. Esta configuración también incluye una resolución DNS y nombres DNS debidamente habilitados para que las instancias que se están creando puedan tener comunicación entre si mediante el nombre que se estableció dentro de la red. También se eligió un “tenancy” predeterminado y no se habilitó el IPv6.

Continuando con el desarrollo, dentro de la VPC, se establecieron y configuraron un total de 4 subredes, las cuales se crearon de forma automática de la siguiente manera: dos públicas (project-linux-subnet-public1-us-east-2a y project-linux-subnet-public2-us-east-2b) y dos privadas (project-linux-subnet-private1-us-east-2a y project-linux-subnet-private2-us-east-2b).

Anexo:

- **En la imagen se detalla la configuración final de la red VPC:**



2.2.2 Creación de Grupos de Seguridad

- Configuración:
  - Nombre: sgv3-linux-web
  - VPC: project-vpc
  - Reglas de entrada:
    - HTTP – TCP: puerto 80; desde 0.0.0.0/0
    - SSH – TCP: puerto 22; desde tu IP o 0.0.0.0/0

Se creó un Grupo de Seguridad llamado sgv3-linux-web, asociado con la VPC project-linux-vpc. Este grupo permite el tráfico necesario para aplicaciones web y acceso a la consola. Tiene dos reglas de entrada:

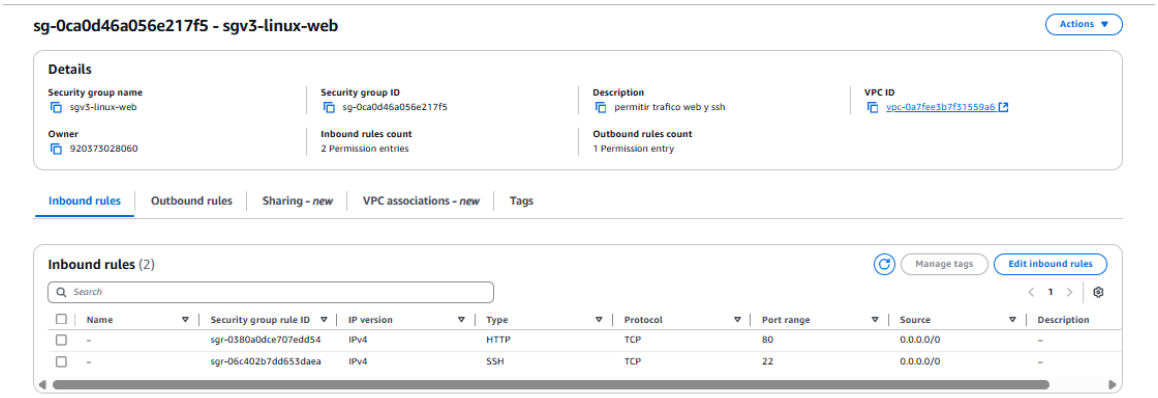
- HTTP (puerto 80): para poder acceder a este desde cualquier dirección IP (0.0.0.0/0)

- SSH (puerto 22): Con el fin de permitir conexiones SSh desde cualquier IP pública.

De esta manera se permite el tráfico de entrada y salida de peticiones o recursos a través de esta configuración. Este security group, también fue usado en servicios que se usaran luego como el grupo de Auto scaling para poder tener una conectividad uniforme.

Anexos:

- **En esta imagen se detalla la configuración de la creación del grupo de seguridad:**



2.2.3 Creación del Launch Template (plantilla de lanzamiento)

- **Configuración:**
    - Nombre: template-linux-web-v3
    - AMI: Amazon Linux 2
    - Instancia: t2.micro
    - Par de claves: test-apache-v2.pem
    - Subred: no incluida en plantilla
    - Security Group: sgv3-linux-web

- **User data script lunch:** Se configure este script para instalar de forma automática el apache y crear una página HTML personalizada:

Anexo:

- En esta imagen se evidencia la plantilla de inicio de instancias:

```
#!/bin/bash
sudo yum update -y
sudo yum install -y httpd
sudo systemctl start httpd
sudo systemctl enable httpd

PRIVATE_IP=$(curl http://169.254.169.254/latest/meta-data/local-ipv4)
AZ=$(curl http://169.254.169.254/latest/meta-data/placement/availability-zone)
HOSTNAME=$(hostname)
DATE=$(date)

cat <<EOF > /var/www/html/index.html
<!DOCTYPE html>
<html>
<head><title>Instancia EC2</title>
<style>
body { font-family: sans-serif; background: #f0f0f0; padding: 2rem; }
.card { background: white; padding: 2rem; border-radius: 10px; box-shadow: 0 0 10px rgba(0,0,0,0.1); }
h1 { color: #007acc; }
</style>
</head>
<body><div class="card">
<h1>Hola desde EC2</h1>
<p><strong>Hostname:</strong> $HOSTNAME</p>
<p><strong>IP privada:</strong> $PRIVATE_IP</p>
<p><strong>AZ:</strong> $AZ</p>
<p><strong>Fecha:</strong> $DATE</p>
</div></body>
</html>
EOF
```

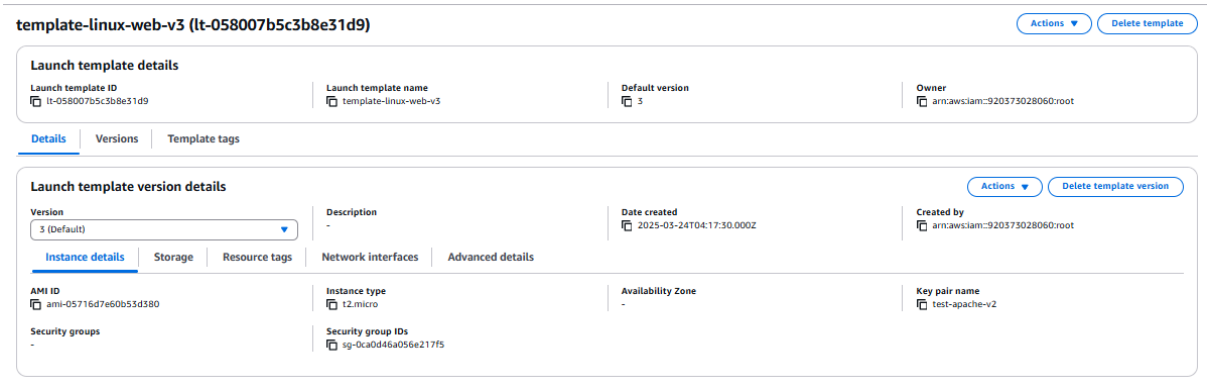
En este punto se implementó una plantilla para poder hacer lanzamientos automáticos llamada “template-linux-web-v3”, que ayudará a definir la configuración inicial de cada instancia dentro del grupo de Auto Scaling..

Se utilizó también la **AMI oficial de Amazon Linux 2** (ami-05716d76c0b53d380) y con esta, un tipo de instancia t2.micro, la cual es elegible para el la capa gratuita que se está usando. Esta plantilla permitirá crear nuevas instancias de forma automática con los mismos parámetros.

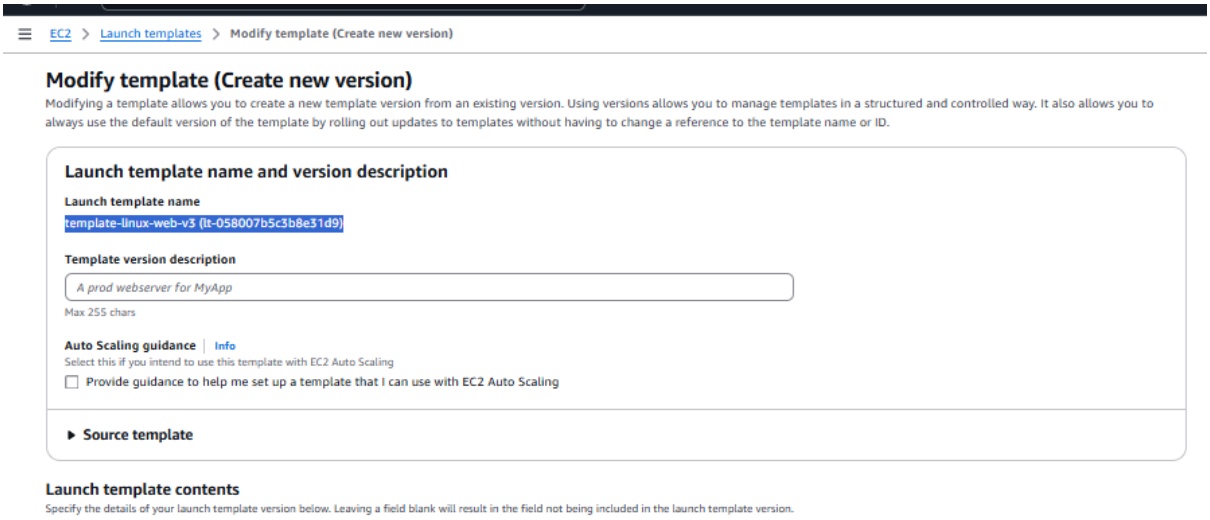
También se asociaron unas claves llamadas test-apache-v2 (para permitir la conectividad SSH si es necesario) y se aplicó el Grupo de Seguridad previamente definido llamado sgv3-linux-web (permitiendo el tráfico web y de gestión).

Anexos:

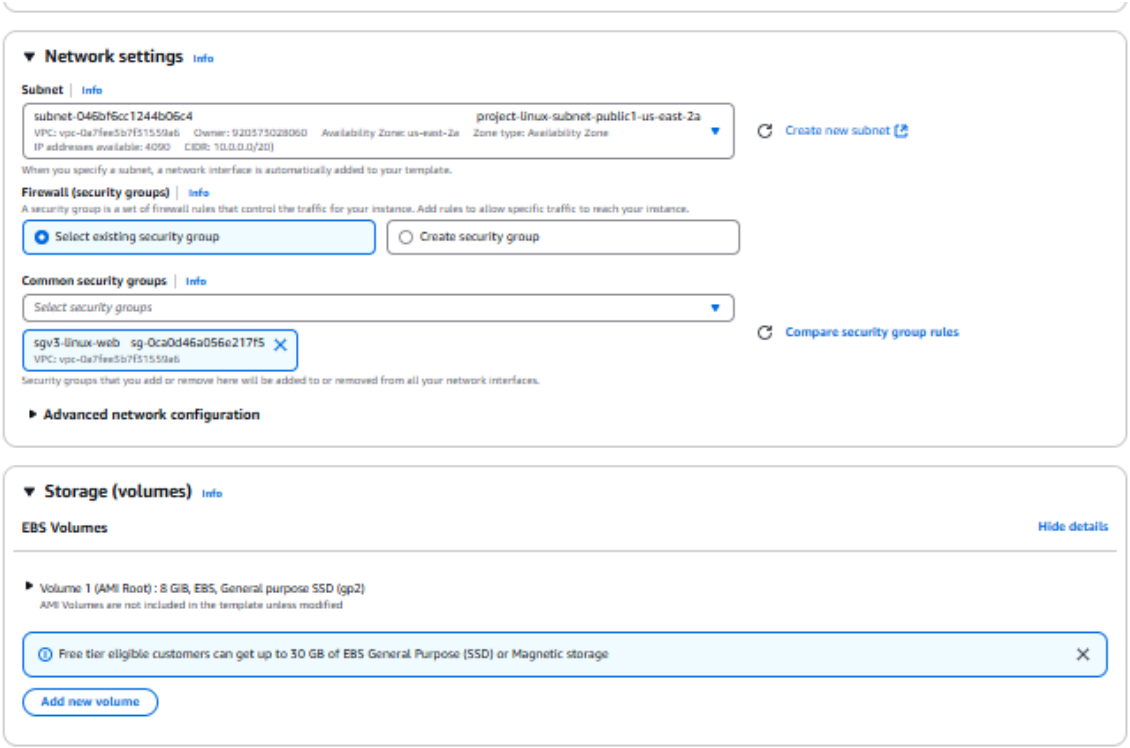
- **La siguiente imagen es un template del dashboard**



- **La siguiente imagen es la configuración del launch template:**



- **En la siguiente imagen se demuestra la configuración de las redes y la capacidad de almacenamiento**



- **En la siguiente imagen se especifica la imagen a usar y las especificaciones de la instancia**

Launch template contents

Specify the details of your launch template version below. Leaving a field blank will result in the field not being included in the launch template version.

▼ **Application and OS Images (Amazon Machine Image)** Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

AMI from catalog | Recents | Quick Start

Name

Verified provider | Free tier eligible

Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type

Description

Amazon Linux 2 comes with five years support. It provides Linux kernel 5.10 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is now under maintenance only mode and has been removed from this wizard.

Browse more AMIs

Including AMIs from AWS, Marketplace and the Community

Amazon Linux 2 Kernel 5.10 AMI 2.0.20250305.0 x86_64 HVM gp2

Image ID

ami-05716d7e60b53d380

Username ⓘ

ec2-user

Published	Architecture	Virtualization	Root device type	ENA Enabled
2025-03-05T02:45:28.000Z	x86_64	hvm	ebs	Yes

▼ **Instance type** Info | Get advice

Advanced

Instance type

t2.micro — Free tier eligible

Family: t2 1 vCPU 1 GiB Memory Current generation: true On-Demand Ubuntu Pro base pricing: 0.0154 USD per Hour
On-Demand Linux base pricing: 0.0116 USD per Hour On-Demand SUSE base pricing: 0.0116 USD per Hour
On-Demand Windows base pricing: 0.0162 USD per Hour On-Demand RHEL base pricing: 0.026 USD per Hour

All generations

Compare instance types

Additional costs apply for AMIs with pre-installed software

▼ **Key pair (login)** Info

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name

test-apache-v2 — Template value

Create new key pair

-En la siguiente imagen se especifica la configuración de la plantilla de lanzamiento de instancia con los requisitos que se desearon ajustar.

2.2.4 Creación del Target Group

- **Configuración:**
  - Nombre: tg-linux-web-v3
  - Tipo: Instance
  - Protocolo: HTTP
  - Puerto: 80
  - VPC: project-vpc

En este paso se creó un nuevo grupo de destino llamado “tg-linux-web-v3”, el cual hace referencia a los target groups. Este permitirá el enrutamiento de tráfico HTTP a través del puerto configurado en el 80 hacia las instancias de EC2 dentro del mismo VPC de nombre “project-linux-vpc”.

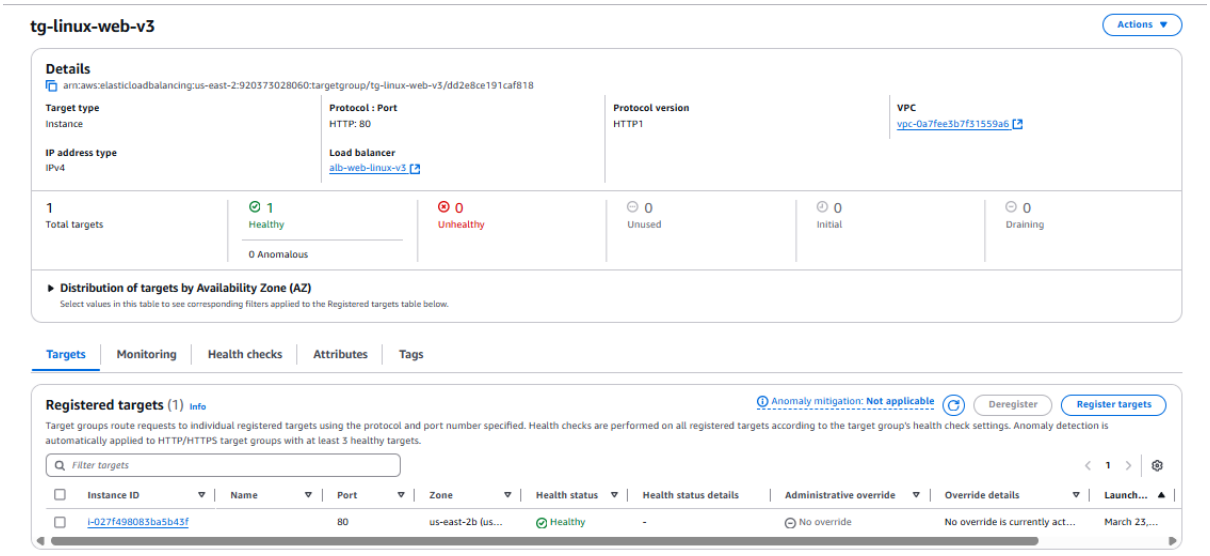
Se enlaza este grupo con el “Application Load Balancer” de nombre “alb-web-linux-v3”. El cual ayudará a distribuir el tráfico de forma más eficiente entre todas las instancias que estén disponibles.

Así el tipo de destino que fue finalmente seleccionado, fue instance, para el cual se usó el protocolo HTTP y se definió una verificación del estado o mejor llamado “health check”. Todo estos e realizó en el puerto 80 para asegurar que las instancias van a poder recibir el tráfico.

En el siguiente ejemplo se puede ver una instancia saludable registrada en la zona: “us-east-2b”, a cual indica que el balanceador va a poder enrutar el tráfico.

Anexos:

- **En la siguiente imagen se demuestra la creación del target group.**



2.2.5 Creación del Application Load Balancer

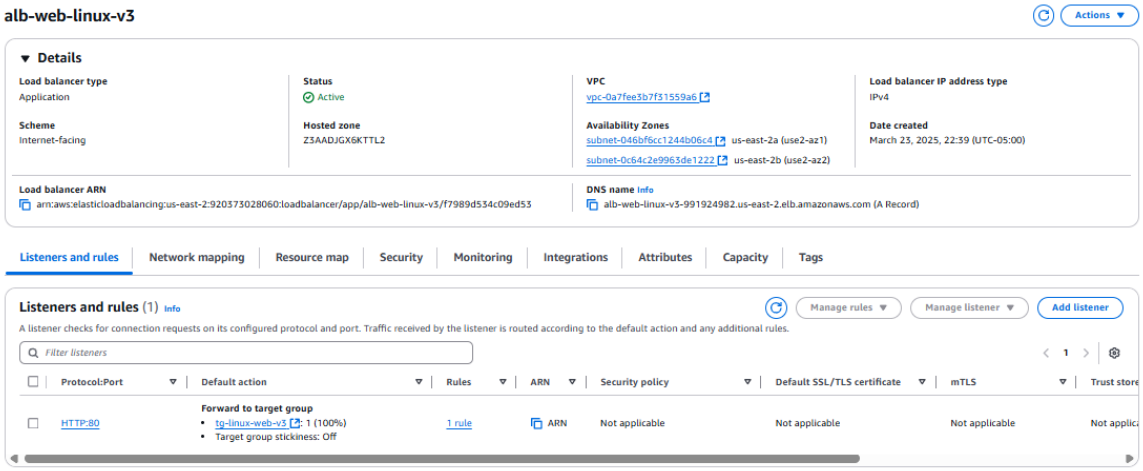
- **Configuración:**
  - Nombre: alb-web-linux-v3
  - Tipo: Application Load Balancer (ALB)
  - Público (Internet-facing)
  - IP type: IPv4
  - Subredes: 2 públicas
  - Listener: HTTP en puerto 80
  - Redirección al Target Group tg-linux-web-v3
  - Security Group asociado: sgv3-linux-web

En este paso se va a manejar el balanceador de carga que tiene por nombre: “alb-web-linux-v3”, el cual se configuró como un ALB de tipo público con alcance en dos zonas de disponibilidad públicas llamadas: “us-east-2ª” y “us-east-2b”. Esto garantiza la tolerancia a fallos y puede distribuir de mejor manera el tráfico entrante a las mismas. También se asoció directamente a la VPC “project-linux-vpc”.

Este ALB se configuró con un listener en el puerto 80 del protocolo HTTP que enrutará las solicitudes al Target Group: “tg-linux-web-v3”. Con esta configuración se balanceará la carga de forma automática en las instancias que estén con el estado de saludables.

Anexos:

- En esta imagen se puede evidenciar la configuración del balanceador de carga:



2.2.6 Creación del Auto Scaling Group

Configuración:

- Nombre: asg-web-linux-v3
- Launch Template: template-linux-web-v3 (última versión con IP pública habilitada)
- Subredes: 2 públicas
- Balanceador de carga: alb-web-linux-v3
- Target Group: tg-linux-web-v3

Capacidad:

- Deseada: 3
- Mínima: 1
- Máxima: 3

Política de escalado:

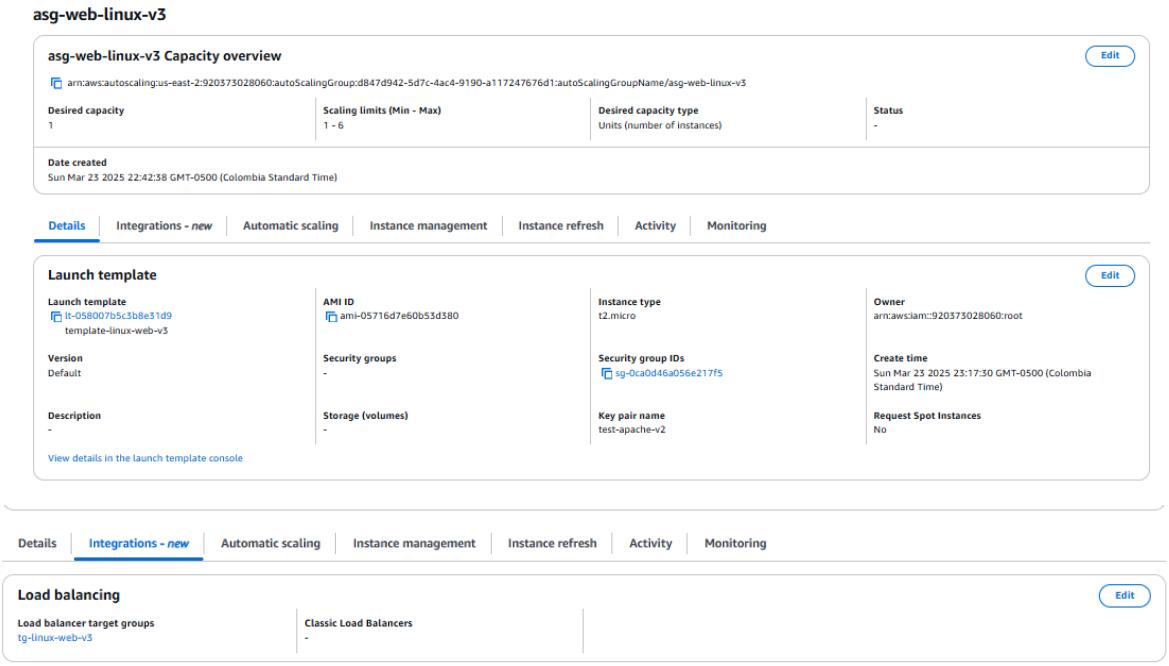
- Target tracking policy basada en CPU:
  - Escala hacia arriba si la CPU promedio > 30%
  - Reduce instancias si baja del umbral

En este punto, se configuró un Auto Scaling Group (ASG) con el nombre de “asg-web-linux-v3”, con el cual se estaría automatizando la creación, eliminación y reemplazo de las instancias según la carga de trabajo que tenga cada una. Este grupo está usando de base el servicio de launch template “template-linux-web-v3”.

Para esto, inicialmente se definió que la capacidad deseada será de 1 instancia y el máximo de 6 instancias.

Anexos:

- **En la siguiente imagen vemos la configuración de el auto scaling group**



2.2.7 Prueba de escalado automático

Proceso realizado:

1. Conexión vía SSH a una instancia
2. Instalación de herramienta stress:
    - sudo amazon-linux-extras install epel -y

    - sudo yum install stress -y

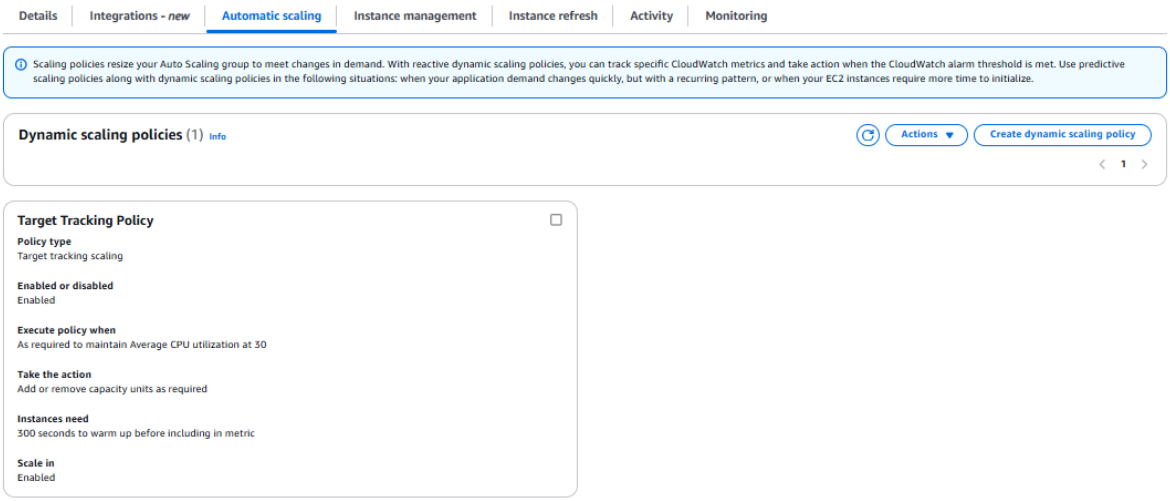
3. Simulación de carga:
    - stress --cpu 2 --timeout 300
4. Resultado:
    - CPU aumentó
    - Auto Scaling lanzó automáticamente una nueva instancia
    - Se pudo verificar por la pestaña **Activity** del ASG

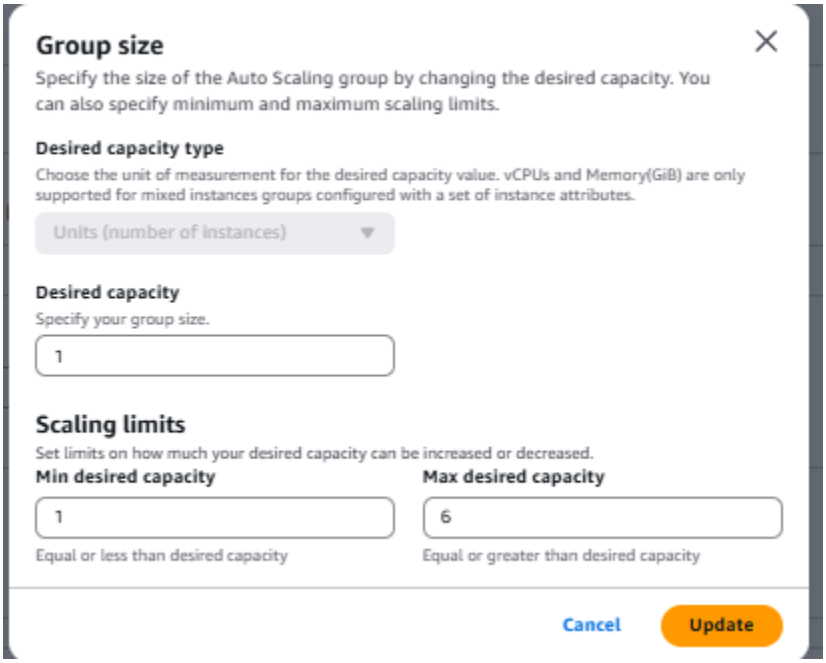
En esta prueba, se configuró en el Auto Scaling Group con nombre “asg-web-linux-v3” una política de escalado que se basa en las métricas específicas de la CPU. Esta política activa una nueva instancia cuando el promedio de uso de CPU es mayor que 60% durante un periodo específico.

Con esta política el sistema puede responder más rápido y actuar con el Auto Scaling con el fin de evitar sobrecargas a las instancias en un momento específico. Para poder simular este proceso se usaron las llaves para conexión SSH y se usó la herramienta stress en la instancia principal.

Anexos

- **En estas imágenes se pueden ver las políticas para hacer efectiva esta prueba:**





2.3 Implementar la ejecución de múltiples contenedores Docker y redirigir el tráfico usando proxy reverso

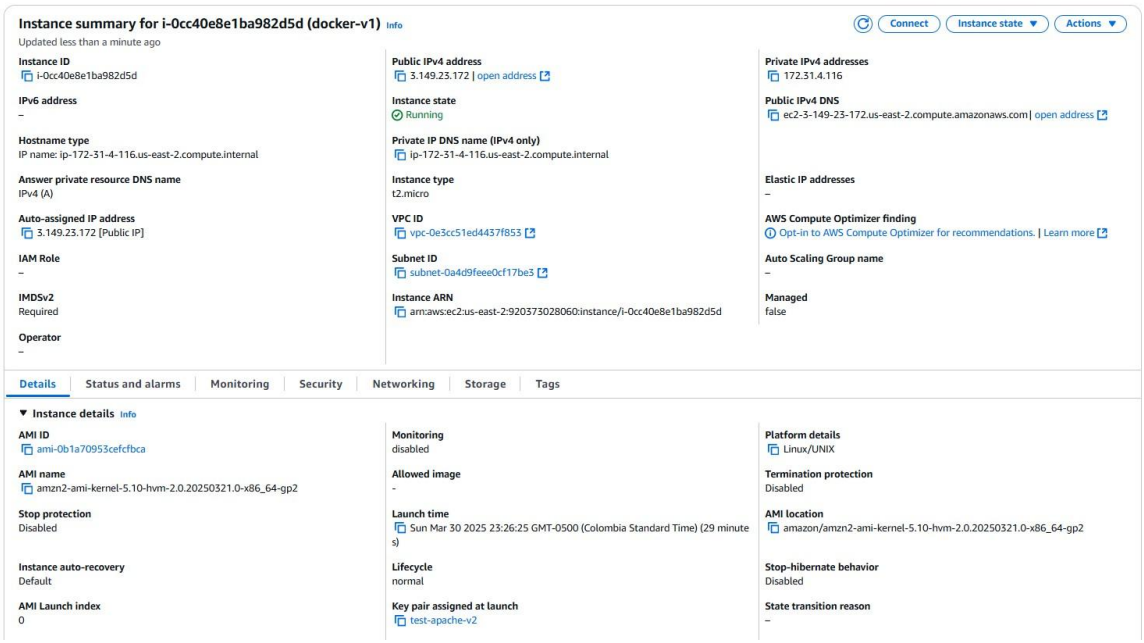
En este proyecto se configurará una instancia EC2 en AWS para ejecutar múltiples contenedores Docker (cada uno con su propia página web) y redirigir el tráfico usando un contenedor NGINX como proxy inverso, según el dominio ingresado (tienda1.com, exito.com, d1.com).

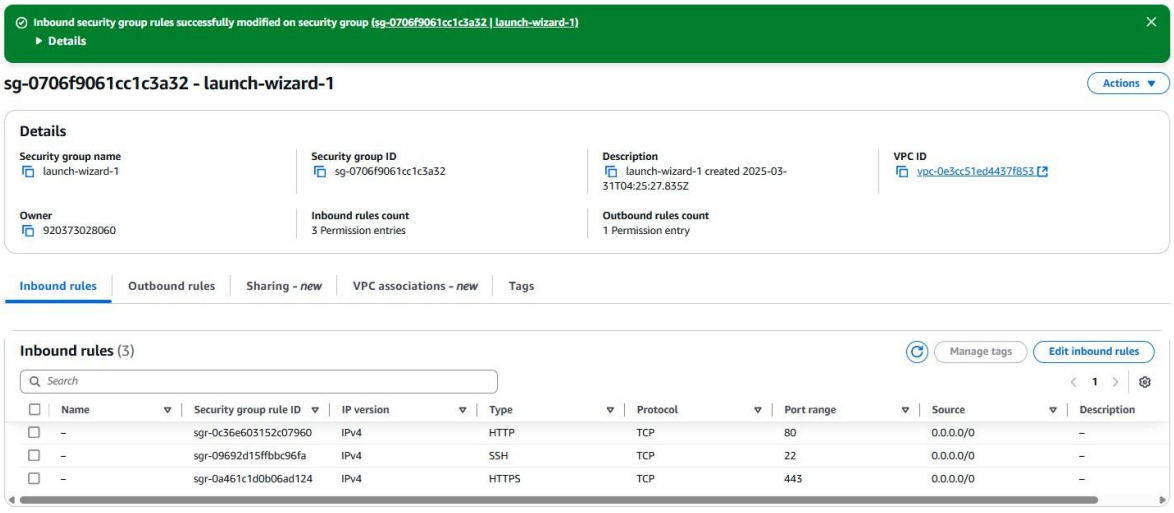
2.2.1 Crear la instancia EC2 en AWS

- Se ingresó a la consola de Amazon EC2.
- Se lanzó una nueva instancia utilizando Amazon Linux 2.
- Se seleccionó el tipo t2.micro (gratis en el tier gratuito).
- Se configuró un **Security Group** para permitir:
    - Conexiones SSH (puerto 22)
    - Conexiones HTTP (puerto 80)
- Se asignó una IP pública automática.

Anexos:

- **En la siguiente imagen se ve la configuración y el estado de la instancia creada:**





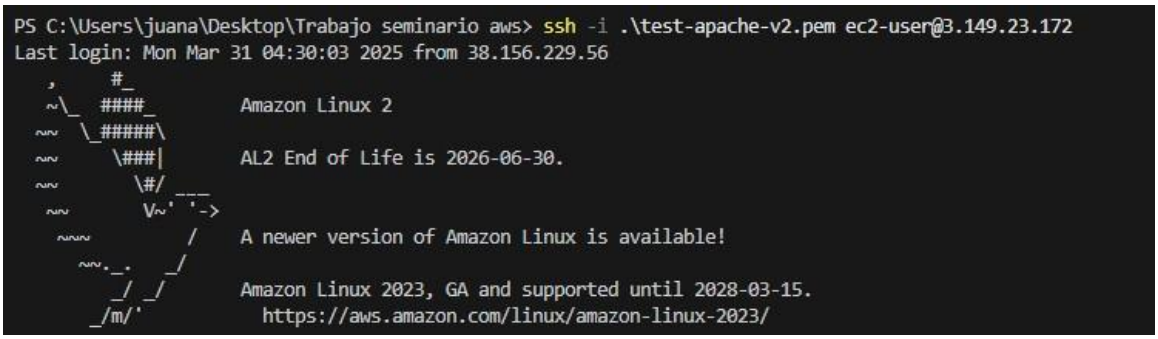
2.2.2 Conectarse a la instancia EC2 desde Windows con PowerShell

- Se ubicó el archivo .pem descargado al lanzar la instancia.
- Luego se ejecutó el comando SSH para conectarse:

ssh -i .\test-apache-v2.pem ec2-user@[IP_PUBLICA]

Anexos:

- Instancia inicializada



2.2.3 Instalar Docker en la instancia

- Primero se ejecutan los siguientes comandos:

“ sudo yum update -y
sudo amazon-linux-extras
install docker -y sudo
service docker start
sudo usermod -a -G docker ec2-user”

- Al final, se cerró sesión y se volvió a ingresar para aplicar permisos del grupo docker.

2.2.4 Crear estructura como vamos a correr Docker en la instancia:

- Se creó una carpeta de trabajo:

mkdir my-dockers cd my-dockers

-Se crearon tres subcarpetas, una para cada contenedor:

mkdir docker1 docker2 docker3

Anexos:

-En esta imagen se puede ver la creación de las carpetas:

```
[ec2-user@ip-172-31-4-116 my-dockers]$ mkdir docker1 docker2 docker3
[ec2-user@ip-172-31-4-116 my-dockers]$ ls
docker1  docker2  docker3
```

2.2.5 En cada carpeta se creó un index.html con un mensaje único:

echo "<h1>Hola desde Docker 1</h1>" > docker1/index.html

echo "<h1>Hola desde Docker 2</h1>" > docker2/index.html echo "<h1>Hola desde Docker 3</h1>" > docker3/index.html

Anexos:

- **En esta imagen se puede ver la creación de los index.html**

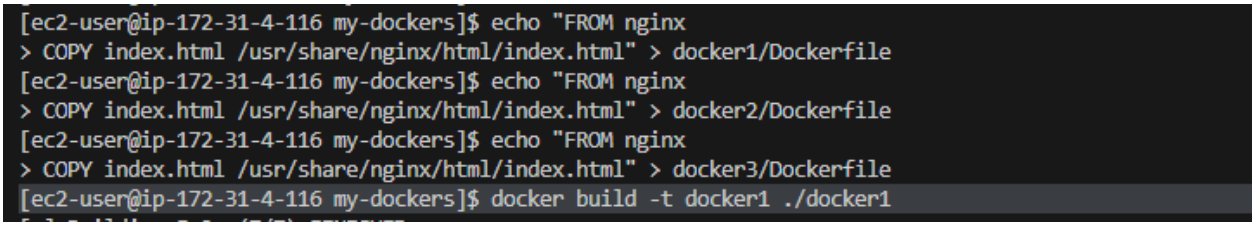
```
[ec2-user@ip-172-31-4-116 my-dockers]$ echo "<h1>Hola desde Docker 1</h1>" > docker1/index.html
[ec2-user@ip-172-31-4-116 my-dockers]$ echo "<h1>Hola desde Docker 2</h1>" > docker2/index.html
[ec2-user@ip-172-31-4-116 my-dockers]$ echo "<h1>Hola desde Docker 3</h1>" > docker3/index.html
```

2.2.6 Luego se creó un Dockerfile para cada contenedor que copia ese HTML dentro de NGINX:

```
echo "FROM nginx\nCOPY index.html
/usr/share/nginx/html/index.html" > docker1/Dockerfile

echo "FROM nginx\nCOPY index.html
/usr/share/nginx/html/index.html" > docker2/Dockerfile

echo "FROM nginx\nCOPY index.html
/usr/share/nginx/html/index.html" > docker3/Dockerfile
```

Anexos:

- En esta imagen se pueden ver la creación de los Docker files:



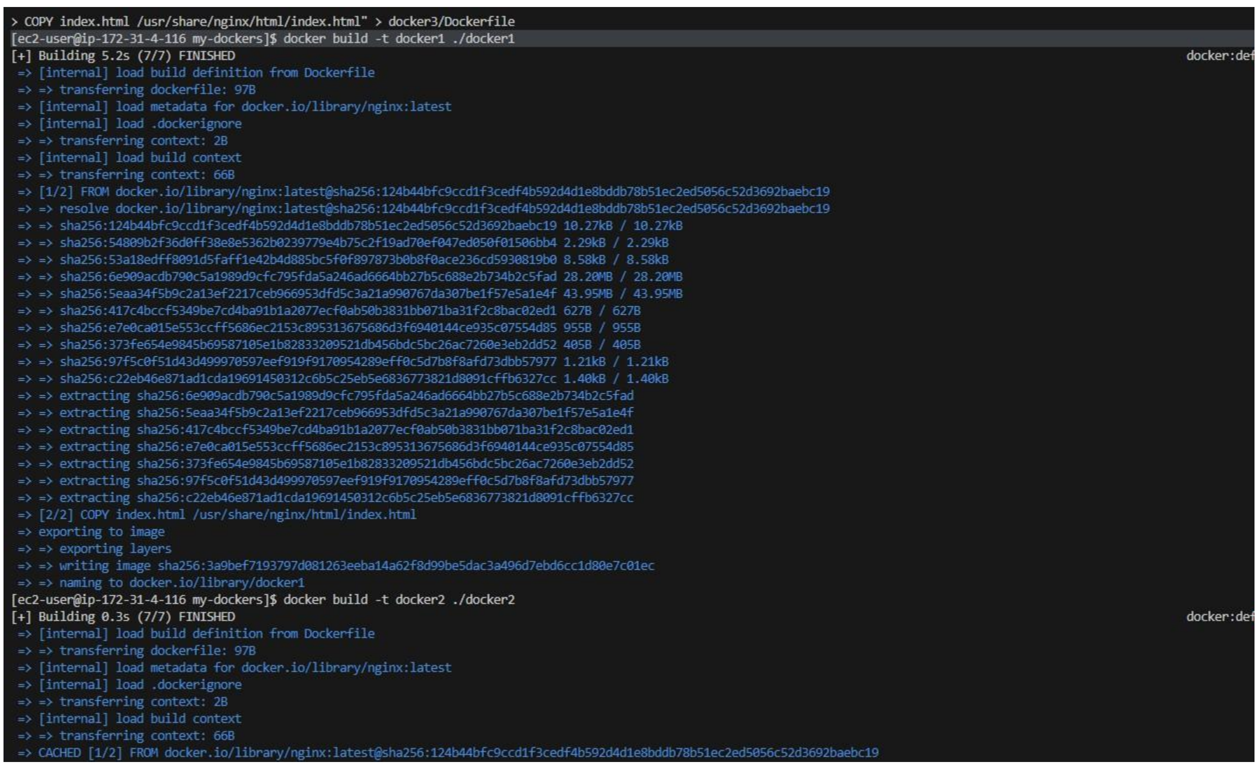
2.2.7 Se construyeron las imágenes:

docker build -t docker1 ./docker1

docker build -t docker2 ./docker2

docker build -t docker3 ./docker3

Anexo:



2.2.8 Finalmente se ejecutaron los contenedores, cada uno en un puerto distinto:

docker run -d --name docker1 -p 8081:80 docker1

docker run -d --name docker2 -p 8082:80

docker2 docker run -d --name docker3 -p

8083:80 docker3

Anexo:

```
[ec2-user@ip-172-31-4-116 my-dockers]$ docker run -d --name docker1 -p 8081:80 docker1
b64422673cc75415f5f1f3db76065bd52bdf9bb8758e8b7abe580ef5ce8a5e5a
[ec2-user@ip-172-31-4-116 my-dockers]$ docker run -d --name docker2 -p 8082:80 docker2
acb16b181056aefb43f2a284d2dc94577d94329ca175216bf2c8dd76f405bd62
[ec2-user@ip-172-31-4-116 my-dockers]$ docker run -d --name docker3 -p 8083:80 docker3
24f7ac0e3117f5fbbb9f4459ff2cdf0645053b01493acdda66d9430226807a0e
```

2.2.9 Crear contenedor de NGINX como Reverse Proxy

- Se creó una carpeta para el proxy:

  mkdir reverse-proxy CC cd reverse-proxy

- Se creó un archivo default.conf con esta configuración (usando 172.17.0.1 para redirigir al host desde el contenedor):

```
server {
listen 80;

server_name tienda1.com; location / {
proxy_pass http://172.17.0.1:8081;
}
}


server {
listen 80;
server_name exito.com; location / {
proxy_pass http://172.17.0.1:8082;
}
}


server {
listen 80; server_name d1.com; location / {
proxy_pass http://172.17.0.1:8083;
}
}
```

2.2.10 Se creó un Dockerfile para usar esta configuración en una imagen de NGINX: FROM nginx

COPY default.conf /etc/nginx/conf.d/default.conf

2.2.11 Se construyó y ejecutó el contenedor y se solucionaro errores:

docker build -t nginx-proxy .

docker run -d --name nginx-proxy -p 80:80 nginx-proxy

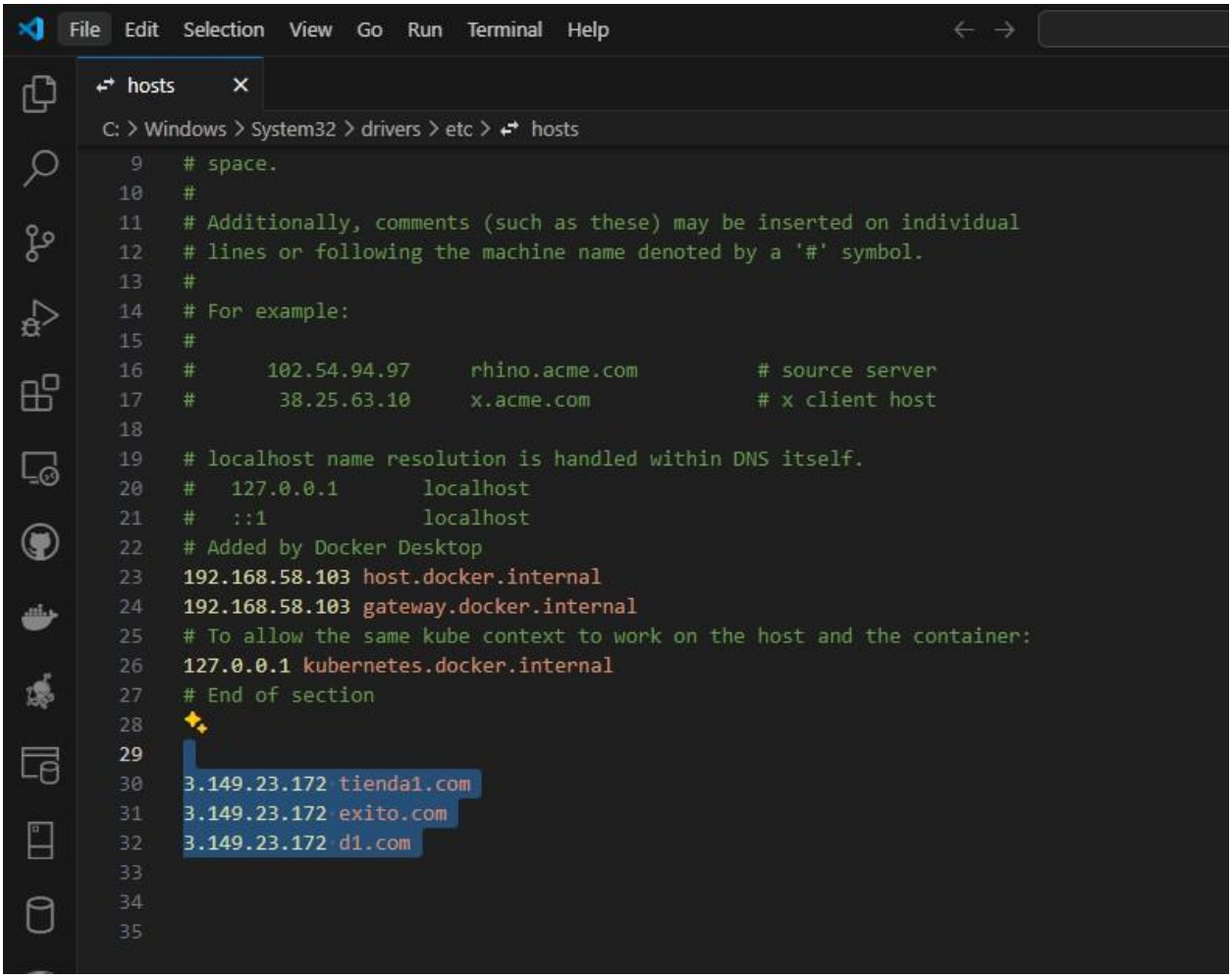
Anexo:

```
[ec2-user@ip-172-31-4-116 reverse-proxy]$ docker logs nginx-proxy
/docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
/docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
/docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
/docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
/docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
/docker-entrypoint.sh: Configuration complete; ready for start up
2025/03/31 04:47:06 [emerg] 1#1: host not found in upstream "host.docker.internal" in /etc/nginx/conf.d/default.conf:6
nginx: [emerg] host not found in upstream "host.docker.internal" in /etc/nginx/conf.d/default.conf:6
[ec2-user@ip-172-31-4-116 reverse-proxy]$ ls
default.conf  Dockerfile
[ec2-user@ip-172-31-4-116 reverse-proxy]$ ls
default.conf  Dockerfile
[ec2-user@ip-172-31-4-116 reverse-proxy]$ cat default.conf
server {
    listen 80;
    server_name tienda1.com;

    location / {
        proxy_pass http://host.docker.internal:8081;
    }
}

server {
    listen 80;
    server_name exito.com;

    location / {
        proxy_pass http://host.docker.internal:8082;
    }
}

server {
    listen 80;
    server_name d1.com;

    location / {
        proxy_pass http://host.docker.internal:8083;
    }
}
[ec2-user@ip-172-31-4-116 reverse-proxy]$ nano default.conf
[ec2-user@ip-172-31-4-116 reverse-proxy]$ docker rm -f nginx-proxy
nginx-proxy
```

2.2.12 Configurar archivo hosts en Windows

- Se editó el archivo:

C:\Windows\System32\drivers\etc\hosts



- Se agregaron las siguientes entradas (reemplazando con la IP pública de la instancia):

3.149.23.172 tienda1.com

3.149.23.172 exito.com

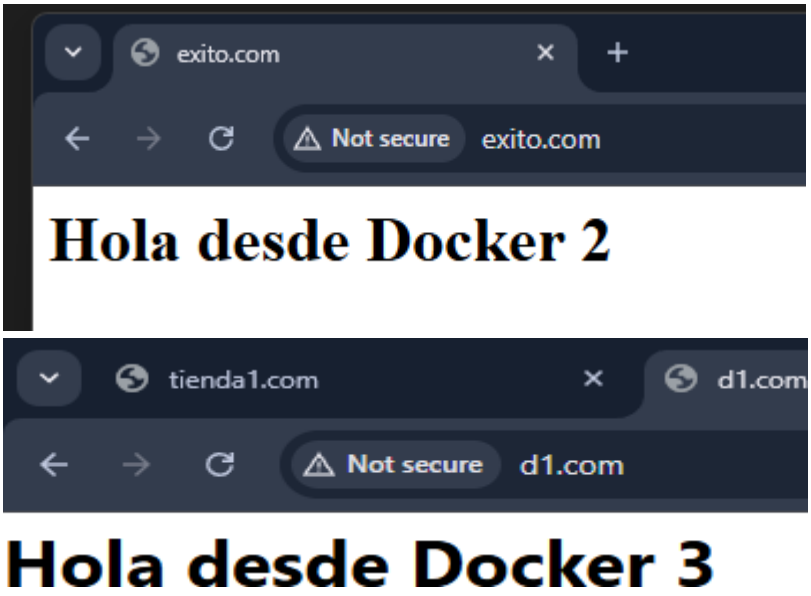
3.149.23.172 d1.com

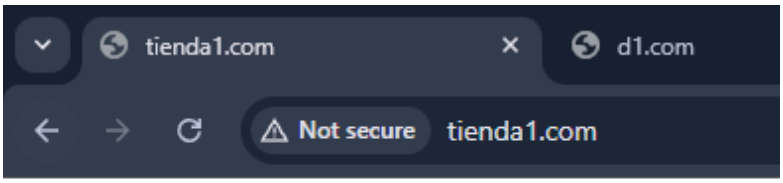
- Esto hace que Windows resuelva esos dominios apuntando directamente a la instancia EC2.

2.2.13 Pruebas de acceso

- Se abrieron los navegadores y se ingresaron las URLs:
- http://tienda1.com ➞ muestra "Hola desde Docker 1"
- http://exito.com ➞ muestra "Hola desde Docker 2"
- http://d1.com ➞ muestra "Hola desde Docker 3"
- Cada dominio redirigió correctamente al contenedor correspondiente.

Anexos:





Hola desde Docker 1

Conclusiones

Conclusión general

Este trabajo permitió comprender de forma integral cómo la virtualización y la computación en la nube han transformado el desarrollo de soluciones tecnológicas modernas. A través de la historia y evolución de estas tecnologías, se evidenció su impacto en la eficiencia del uso de recursos y su papel como base de la infraestructura actual. La implementación práctica con Amazon Web Services no solo reforzó los conceptos teóricos, sino que también demostró la capacidad de diseñar soluciones escalables y funcionales. En conjunto, esta experiencia sirvió como un puente entre el aprendizaje académico y la aplicación profesional.

Conclusiones específicas

- La virtualización fue el punto de partida para optimizar el uso de hardware, y hoy representa la base técnica sobre la cual se construyen los servicios de computación en la nube.
- AWS demostró ser una plataforma robusta que permite crear infraestructuras completas de manera flexible y automatizada, ajustándose a demandas reales de tráfico y procesamiento.
- La configuración de un Auto Scaling Group con balanceador de carga permitió validar que es posible garantizar disponibilidad y eficiencia ante cargas variables de trabajo.
- La implementación de contenedores Docker y su gestión mediante NGINX como proxy inverso mostró una solución eficaz para administrar múltiples servicios web en un solo servidor.
- El desarrollo práctico facilitó la comprensión profunda de conceptos clave como VPC, grupos de seguridad, plantillas de lanzamiento y políticas de escalado, aplicándolos en escenarios reales.

Referencias

Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing* (NIST Special Publication 800-145). National Institute of Standards and Technology. Recuperado de https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

Amazon Web Services (AWS). (2006). *Amazon Elastic Compute Cloud (EC2)*. Amazon. Recuperado de https://aws.amazon.com/ec2/

Google Cloud. (n.d.). *Google Cloud Platform (GCP)*. Google. Recuperado de https://cloud.google.com/

Microsoft Azure. (n.d.). *Microsoft Azure Cloud Services*. Microsoft. Recuperado de https://azure.microsoft.com/