TRABAJO DE GRADO
Opción Seminario-Diplomado.

Implementación y Validación de Servicios de Alta Disponibilidad y Contenerización en AWS

Corporación Universitaria Remington.
Facultad de ingeniería
Ingeniería de Sistemas

Cristian Camilo Marin Florez.
Tutor: Juan Pablo Berrio López.
Opción de Trabajo de grado Seminario-Diplomado.
2025.

1. Tabla de Contenidos

2. Resumen

Este trabajo de grado presenta la experiencia y resultados obtenidos durante el seminario de AWS, donde se llevaron a cabo dos talleres prácticos. En el primer taller se implementó una solución de balanceo de carga con Auto Scaling en instancias Linux de AWS, utilizando un Elastic Load Balancer y un Auto Scaling Group que asegura la disponibilidad continua de un sitio web básico. En el segundo taller se desplegaron tres aplicaciones web en contenedores Docker en una instancia EC2, utilizando NGINX como proxy inverso para redireccionar el tráfico a cada contenedor. La práctica permitió comprender la alta disponibilidad, escalabilidad y contenerización en un entorno real de AWS.

Palabras clave
AWS, Auto Scaling, Elastic Load Balancer, Docker, NGINX

3. Marco conceptual y contextual

En este informe se presentan los resultados obtenidos durante el seminario de AWS, donde se realizaron dos talleres prácticos. El objetivo principal fue aplicar y validar conceptos fundamentales de la computación en la nube, tales como la alta disponibilidad, la escalabilidad y la contenerización, que son esenciales para el desarrollo y la operación de aplicaciones en entornos dinámicos y de alta demanda.

3.1. Contexto del Seminario:

El seminario de AWS organizado por la universidad permitió experimentar de primera mano la implementación de soluciones de alta disponibilidad y contenerización en un entorno real de nube. Los talleres prácticos se enfocaron en:

- La implementación de un balanceador de carga con Auto Scaling, asegurando la continuidad del servicio ante fallas y variaciones en la demanda.
- El despliegue de aplicaciones web en contenedores Docker, orquestados mediante NGINX para la gestión del tráfico de red.

3.2. Contexto Organizacional:

Aunque el ejercicio se desarrolló en un ambiente de laboratorio, la metodología y herramientas empleadas tienen una aplicación directa en entornos empresariales. Las organizaciones que adoptan estas tecnologías pueden:

- Mejorar la flexibilidad de sus aplicaciones, garantizando la continuidad operativa.
- Adaptarse a cambios repentinos en la demanda sin incurrir en costos excesivos de infraestructura.
- Optimizar la gestión de recursos a través de políticas automatizadas de escalado y balanceo de carga.

3.3. Computación en la nube.

La computación en la nube es un modelo de prestación de servicios tecnológicos a través de Internet, que permite acceder a recursos como servidores, almacenamiento, bases de datos, redes, software y análisis, sin necesidad de gestionarlos físicamente. Este modelo proporciona escalabilidad, alta disponibilidad, seguridad y pago por uso, lo que lo convierte en una alternativa eficiente frente a las infraestructuras tradicionales.

La virtualización y la computación en la nube no son conceptos recientes, aunque en la actualidad sean mucho más notables. Originalmente, la virtualización surgió de la necesidad de optimizar el uso de recursos físicos, permitiendo que grandes y costosos servidores se dividieran en múltiples entornos virtuales. Con el crecimiento de internet y el avance de las tecnologías de red, la idea de ofrecer servicios a distancia evolucionó, estableciéndose como la base de la computación en la nube.

3.3.1. Línea de tiempo

1960: IBM desarrolla el concepto de virtualización en sus servidores mediante proyectos como CP-40 y CP-67, permitiendo dividir el hardware en varias "máquinas virtuales" para maximizar el uso de los costosos equipos.

1970: Surge el hipervisor o VMM (Virtual Machine Monitor), con el sistema VM/370 de IBM, que inicia las bases de la virtualización moderna al permitir gestionar múltiples entornos virtuales en un solo servidor físico.

1980: Se populariza el uso de servidores físicos dedicados en grandes centros de datos. Aunque la "nube" tal como la conocemos aún no existía, la virtualización se aplicaba en entornos especializados para mejorar la eficiencia de recursos.

1991: Se lanza Linux, un sistema operativo de código abierto que, con el tiempo, fundamental para la computación en la nube por su flexibilidad y robustez.

1998: Se funda VMware, una empresa que cambia el panorama de la virtualización al llevar esta tecnología a servidores x86. Esto hace posible que empresas de todos los tamaños puedan aprovechar la virtualización sin depender de costosos servidores.

1999: Salesforce lanza su plataforma CRM 100% online, marcando uno de los primeros usos comerciales del modelo SaaS (Software as a Service).

2002: Amazon Web Services (AWS) introduce servicios básicos de infraestructura en la nube, ofreciendo capacidades de almacenamiento y procesamiento a pequeña escala.

2006: AWS lanza oficialmente Amazon Elastic Compute Cloud (EC2), un servicio que permite "alquilar" máquinas virtuales bajo demanda. Con este lanzamiento se consolida el modelo IaaS (Infraestructura como Servicio), abriendo nuevas posibilidades en el despliegue de aplicaciones.

2008: Google lanza Google App Engine, una plataforma que permite desplegar aplicaciones web sin la necesidad de gestionar servidores físicos, fortaleciendo el concepto de PaaS (Plataforma como Servicio).

2010: Microsoft lanza Azure, su servicio de nube pública que se posiciona como un competidor directo de AWS y Google Cloud. Además, en este periodo emergen iniciativas de código abierto como OpenStack, que impulsan el modelo de nube híbrida.

2013: Docker revoluciona la virtualización al popularizar el uso de contenedores, una tecnología que permite empaquetar aplicaciones y sus dependencias de forma liviana y eficiente, facilitando el despliegue en cualquier entorno.

2014: Se lanza Kubernetes por Google, una herramienta de orquestación de contenedores que se convierte en estándar para gestionar aplicaciones distribuidas a gran escala. En el

mismo año, AWS introduce Lambda, iniciando la computación serverless, donde se paga solo por el tiempo de ejecución de código.

2020 en adelante: La computación en la nube se consolida y evoluciona hacia arquitecturas híbridas y multi-cloud. Las grandes y pequeñas empresas integran tecnologías avanzadas como inteligencia artificial, Edge computing y bases de datos distribuidas, lo que permite soluciones altamente escalables y de alto rendimiento, adaptadas a las necesidades actuales.

3.4. Alta Disponibilidad.

La alta disponibilidad se refiere a la capacidad de un sistema para continuar operando sin interrupciones, incluso ante fallas parciales.

3.5. Escalabilidad.

La escalabilidad es la funcionalidad de un sistema para adaptarse a cambios en la demanda, aumentando o disminuyendo sus recursos de forma automática.

3.6. Contenerización.

La contenerización consiste en encapsular aplicaciones y sus dependencias en contenedores aislados, lo que facilita su portabilidad y gestión. Docker y Podman son tecnologías usadas para esta tarea.

3.7. AWS.

Amazon Web Services (AWS) es una plataforma integral de servicios en la nube que ofrece infraestructura, herramientas y soluciones para desplegar, gestionar y escalar aplicaciones y servicios. Abarca diversas categorías como servidores, almacenamiento, bases de datos y redes, entre otras.

3.8. Instancia.

Una instancia es un servidor virtual que se ejecuta sobre la infraestructura de AWS. Cada instancia se configura con recursos virtuales (CPU, memoria, almacenamiento y red)

3.9. Elastic Load Balancer:

Distribuye automáticamente el tráfico de red o de aplicaciones entre múltiples instancias de servidores, lo que permite mejorar la disponibilidad y tolerancia a fallos de las aplicaciones. Actúa como un punto de entrada único que reparte la carga de manera eficiente, asegurando que ninguna instancia se sature y que los usuarios siempre puedan acceder al servicio, incluso si alguna instancia falla o se encuentra inactiva.

3.10. Auto Scaling:

Permite gestionar de forma automática la cantidad de instancias EC2 en función de la demanda del sistema.

3.11. NGINX:

Es un servidor web ligero y rápido que también funciona como **proxy inverso** y **balanceador de carga**. Se usa para dirigir el tráfico a diferentes aplicaciones, mejorar el rendimiento y gestionar múltiples conexiones eficientemente.

3.12. AMI:

Una **AMI** (Amazon Machine Image) es una plantilla preconfigurada que contiene el sistema operativo, aplicaciones y configuraciones necesarias para lanzar una instancia EC2 en AWS.

3.13. Target Group:

En AWS es un conjunto de instancias u objetivos (como instancias EC2 o contenedores) al que un **Load Balancer** dirige el tráfico. Permite definir reglas de enrutamiento y monitorear la salud de los objetivos para asegurar una distribución eficiente del tráfico.

4. Desarrollo e implementación del aprendizaje

En este apartado se detalla la información de la implementación práctica de los conceptos aprendidos en el seminario de AWS, mediante la realización de dos talleres que van desde la configuración de balanceadores de carga y escalabilidad automática hasta el despliegue de aplicaciones en contenedores Docker con NGINX.

4.1. Taller 1: Balanceador de Carga y Auto Scaling

Este taller tuvo como objetivo la creación de una infraestructura altamente disponible y escalable sobre Amazon Web Services (AWS), utilizando instancias EC2, un balanceador de carga y un grupo de Auto Scaling. Se buscó implementar una solución capaz de adaptarse dinámicamente a cambios en la demanda, garantizar continuidad del servicio ante fallos y automatizar la administración de recursos.

4.1.1. Preparación de la Imagen Base (AMI)

El primer paso fue la creación de una imagen personalizada (AMI) que sirviera como base para las instancias EC2. Para ello, se lanzó una instancia de Amazon Linux en la que se realizaron las siguientes configuraciones:

- Instalación del servidor Apache.
- Configuración de un sitio web de prueba.
- Verificación de que el sitio web funcionara correctamente.
- Creación de una AMI a partir de esta instancia (linux-webmaster), para ser utilizada posteriormente en la plantilla de lanzamiento.

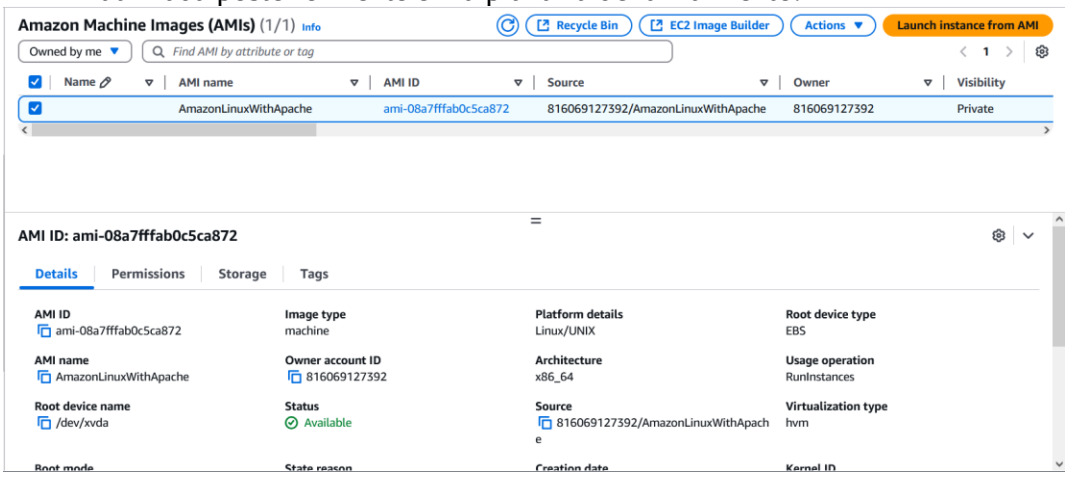

Ilustración 1. AMI personalizada con apache

4.1.2. Creación del Target Group

Se configuró el **Target Group**, donde se agruparon las 2 instancias para recibir tráfico desde el balanceador de carga. El Target Group fue configurado con los siguientes parámetros:

- Protocolo: HTTP
- Puerto: 80
- Ruta de chequeo de salud: /
- Intervalo de chequeo: 30 segundos

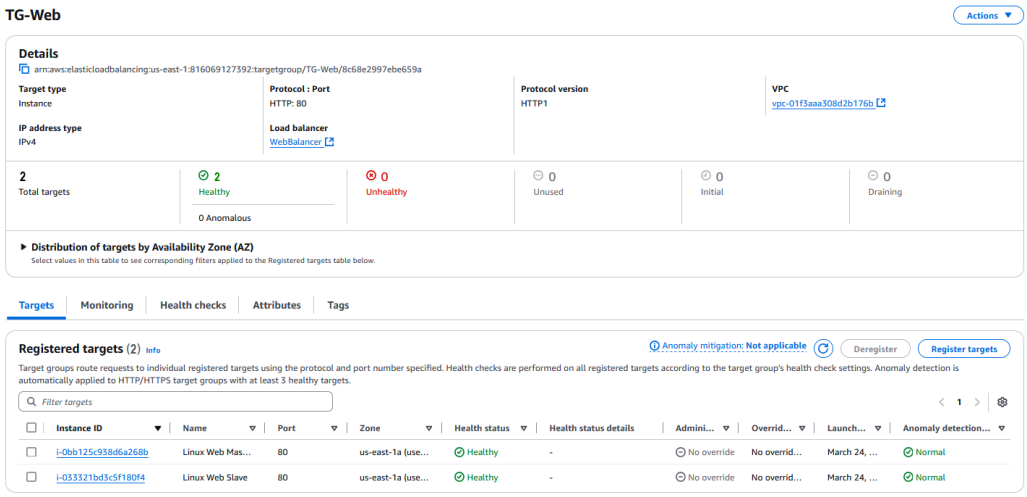

Ilustración 2. Target Group con las instancias con apache

4.1.3. Configuración de la Plantilla de Lanzamiento

Se creó una plantilla de lanzamiento que utiliza la AMI personalizada previamente generada, el mismo tipo de instancia y configuración de seguridad. Esta plantilla define los parámetros para las instancias que se crearán automáticamente,

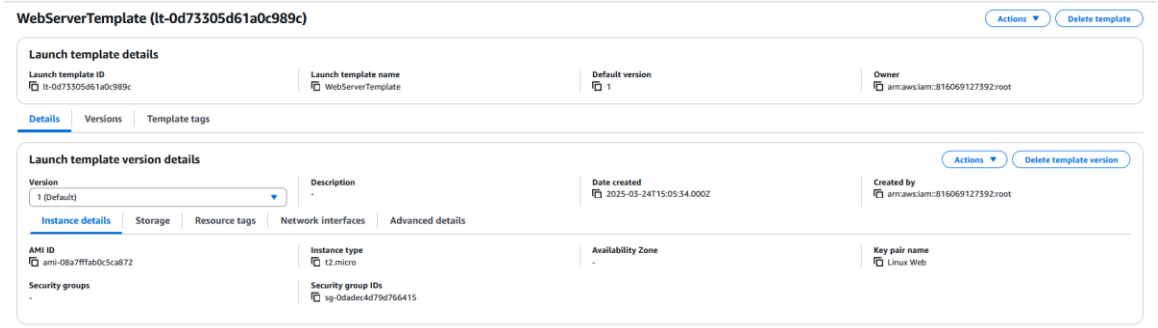

Ilustración 3. Plantilla para generar nuevas instancias con apache

4.1.4. Configuración del Auto Scaling Group

Una vez disponible el Target Group y la plantilla de lanzamiento, se configuró el Auto Scaling Group, encargado de mantener la cantidad adecuada de instancias EC2 en función de la demanda. Sus parámetros fueron:

- Capacidad mínima: 2 instancias
- Capacidad máxima: 5 instancias
- Asociación con el Target Group (para balanceo de carga)
- Política de escalado: generar una nueva instancia cuando el uso de CPU supere el 30%

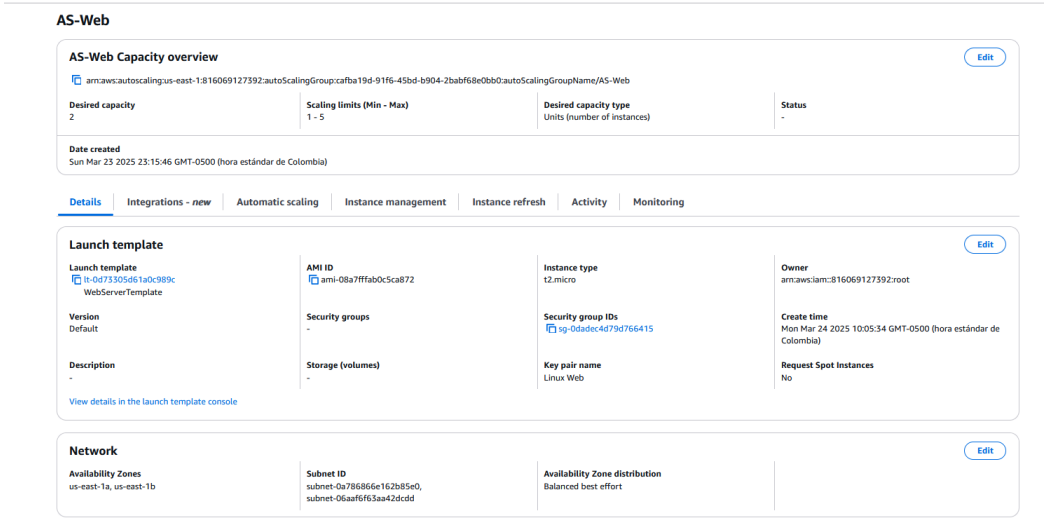

Ilustración 4. Resumen de configuración del Auto Scaling

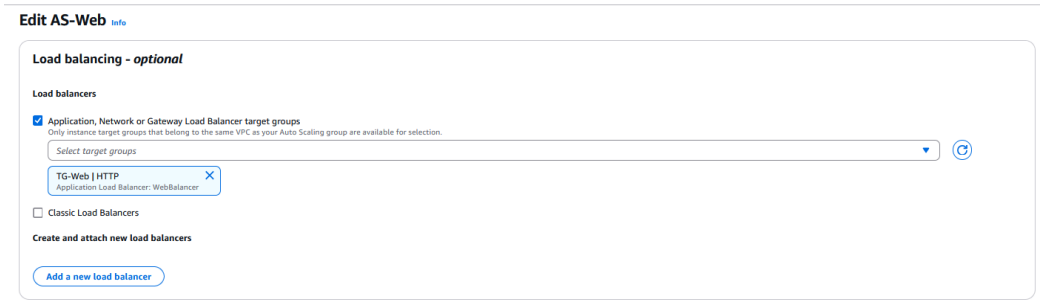

Ilustración 5. Relación del Auto Scaling con el Target Group.

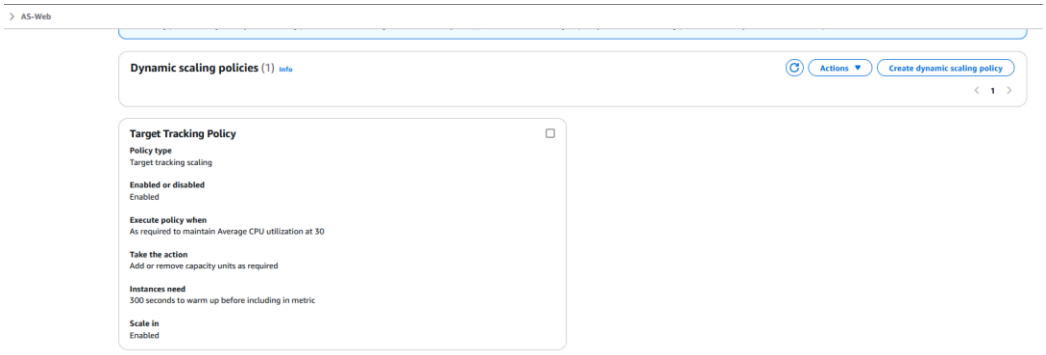

Ilustración 6. Política de Auto Scaling

4.1.5. Configuración del Balanceador de Carga.

Se implementó un balanceador de carga tipo **Application Load Balancer** que actúa como punto de entrada al sistema. Este se configuró para escuchar peticiones HTTP en el puerto 80, redirigir tráfico al Target Group configurado anteriormente y monitorear constantemente el estado de las instancias para asegurar que solo las saludables reciban tráfico.

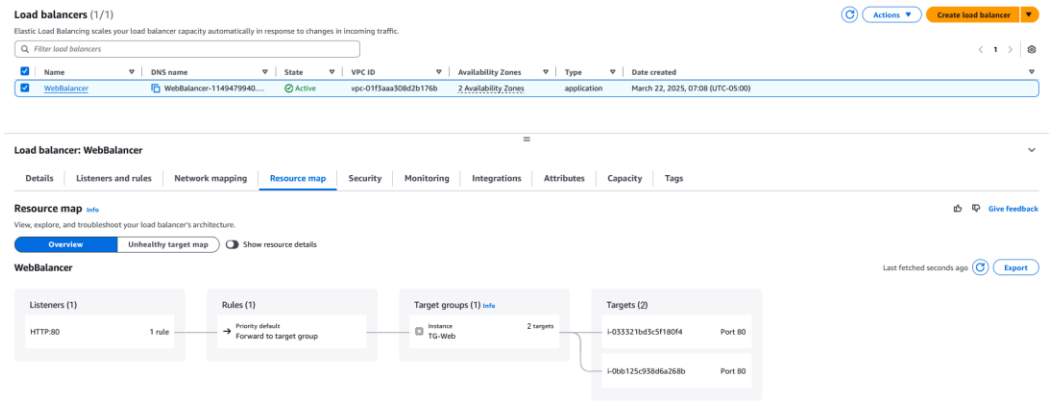

Ilustración 7. Estructura del Load Balancer

4.1.6. Validación del Despliegue

4.1.6.1. Estado Inicial

Inicialmente, el Auto Scaling Group lanzó dos instancias:

- La primera fue la instancia configurada manualmente (linux-webmaster).
- La segunda fue creada automáticamente a partir de su AMI.

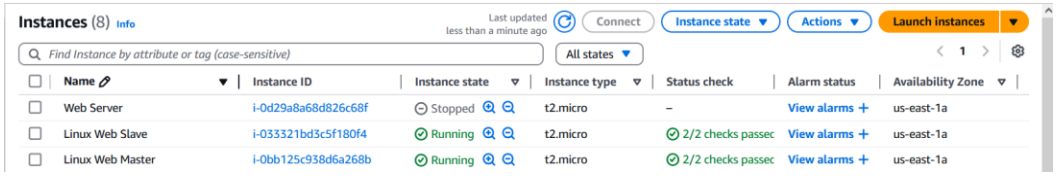

Ilustración 8. Instancias principales

Se comprobó el acceso al sitio mediante la URL del Load Balancer, validando la correcta distribución de tráfico. Las pruebas se realizaron en navegadores en modo normal e incógnito para descartar caché local.

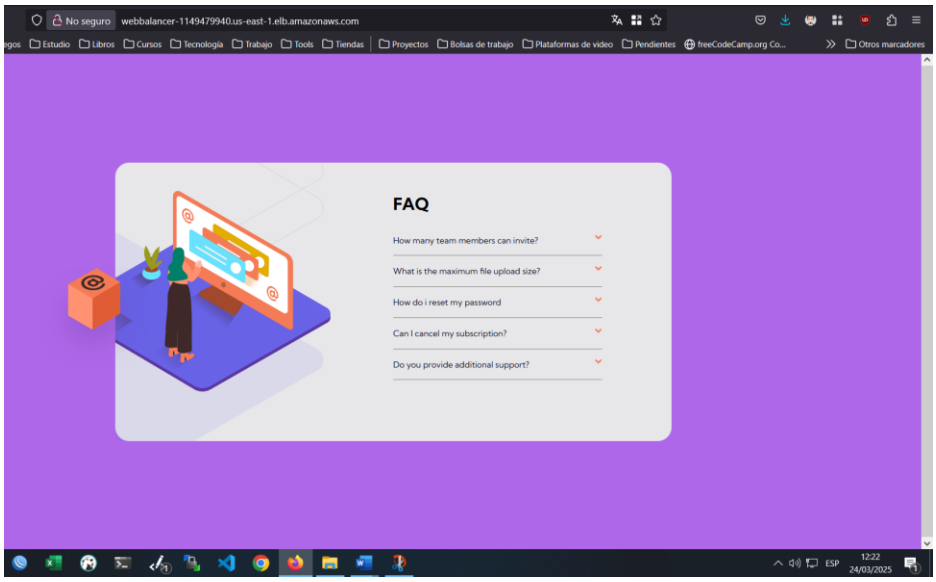

Ilustración 9. Prueba de acceso al sitio de las instancias.

4.1.6.2. Simulación de Carga y Escalado Automático

Para verificar la política de escalado, se simuló una alta carga en las instancias existentes. El sistema reaccionó creando dos nuevas instancias automáticamente, alcanzando un total de cuatro instancias activas. Estas se integraron al Target Group y comenzaron a servir tráfico.

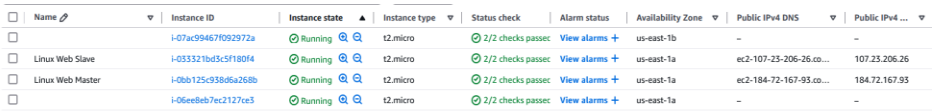

Ilustración 10. Resumen de instancias, aparecen las creadas por el Auto Scaling

4.1.6.3. Prueba de Alta Disponibilidad

Se detuvieron manualmente las dos instancias iniciales. A pesar de ello, el sitio web continuó funcionando correctamente, ya que las instancias generadas por el Auto Scaling

estaban activas y correctamente configuradas. Esta prueba demostró la capacidad de la arquitectura para mantener la disponibilidad del servicio ante fallos.

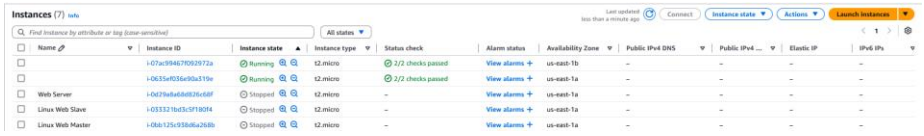

Ilustración 11. Resumen de instancias con las principales detenidas.

4.2. Taller 2: Aplicaciones en Contenedores con NGINX como Proxy Inverso

En este segundo trabajo del seminario de Amazon Web Services (AWS), se implementó una solución basada en contenedores Docker dentro de una instancia EC2. El objetivo principal fue desplegar tres aplicaciones web independientes utilizando NGINX como proxy inverso para enrutar correctamente las solicitudes HTTP hacia cada contenedor, garantizando una arquitectura flexible y escalable.

4.2.1. Entorno de Trabajo

Se utilizó una instancia EC2 de Amazon Linux. En ella se realizaron las siguientes configuraciones:

- Instalación del servidor web **NGINX**.
- Instalación y configuración del servicio **Docker**.
- Habilitación de los puertos necesarios en las reglas del grupo de seguridad.

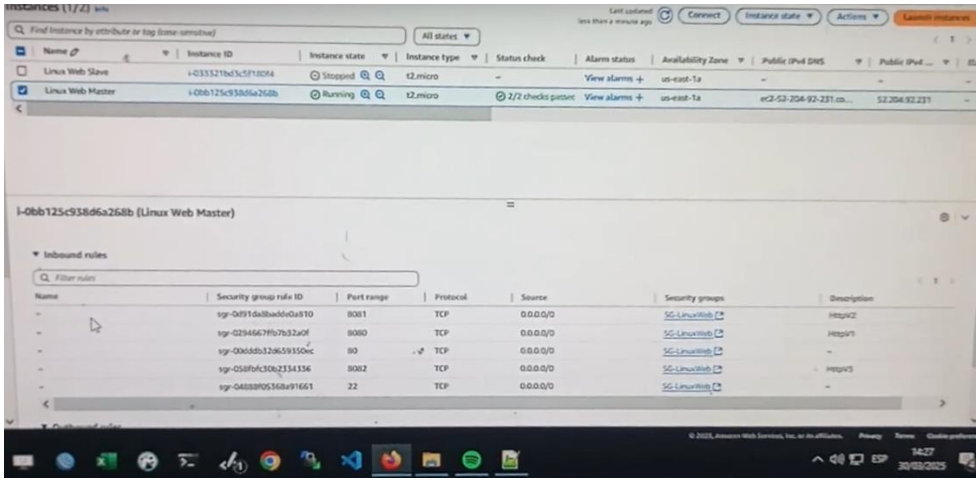

Ilustración 12. Puertos abiertos de la instancia.

4.2.2. Despliegue de Aplicaciones Web

4.2.2.1. Aplicaciones Utilizadas

Desde un repositorio personal se descargaron tres aplicaciones web para ser desplegadas en contenedores:

- **Control del Tiempo**
- **Concéntrese**
- **Piedra, Papel o Tijera**

Cada aplicación fue ubicada en una carpeta específica y se le asignaron los permisos adecuados para ser utilizada por Docker.

4.2.2.2. Contenedores Docker

Se crearon tres contenedores Docker, uno para cada aplicación, con las siguientes características:

Tabla 1. Relación de puertos y contenedores.

Aplicación	Puerto del contendor	Puerto de la instancia
Control del tiempo	80	8080
Concéntrese	80	8081
Piedra, Papel o Tijera	80	8082

Todos los contenedores se ejecutan con el parámetro --restart always, lo que garantiza su persistencia ante reinicios o fallos del sistema.

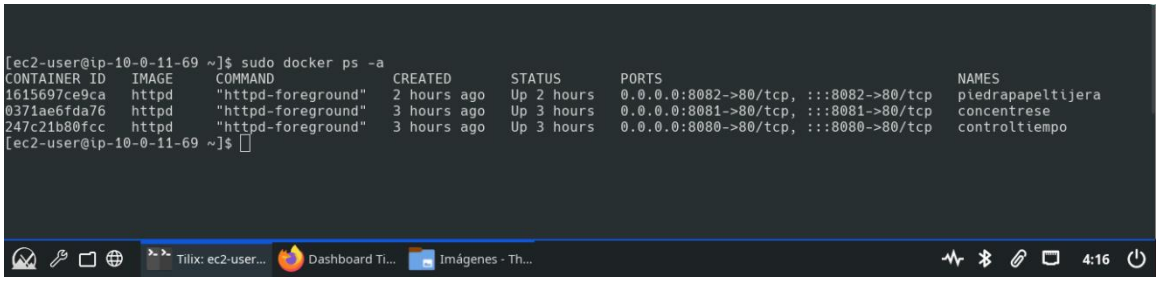

Ilustración 12. Estado de los contenedores con los puertos mapeados

4.2.3. Configuración del Servidor NGINX

Se configuró NGINX como **proxy inverso**, permitiendo que el tráfico HTTP recibido en el puerto 80 sea redirigido al contenedor correspondiente, dependiendo del nombre del servidor (server name).

4.2.3.1. Estructura del Archivo de Configuración

Se definieron tres bloques ‘server’ en el archivo de configuración de NGINX, cada uno escuchando en el puerto 80 y respondiendo a un dominio distinto:

- controltiempo.dev → redirecciona al puerto 8080
- concentrese.dev → redirecciona al puerto 8081
- piedrapapeltijera.dev → redirecciona al puerto 8082

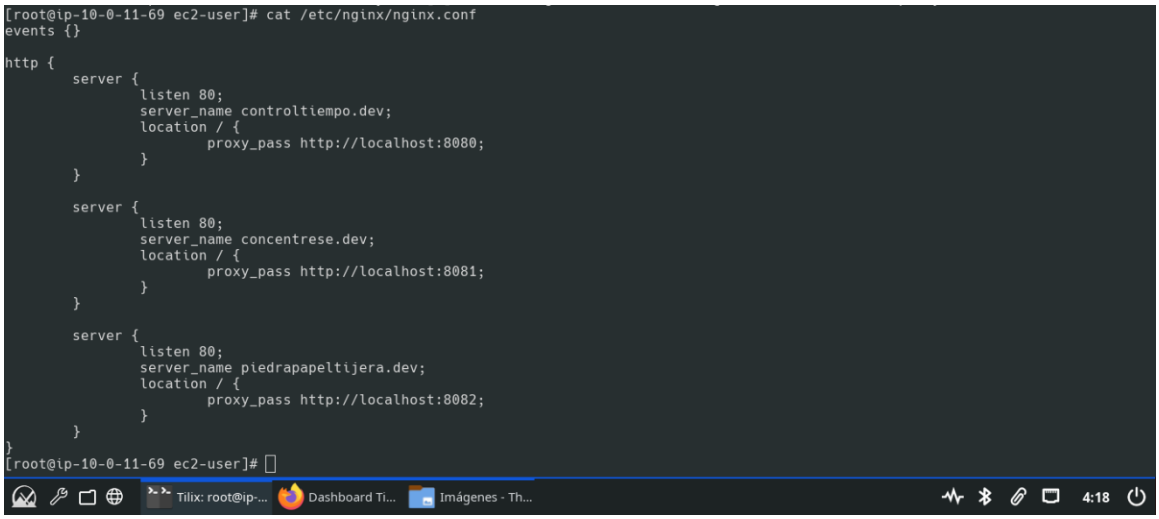

Ilustración 13. Código de configuración de NGINX

4.2.4. Configuración del Archivo Hosts

Para permitir el acceso desde el navegador mediante dominios personalizados, se editaron las entradas del archivo hosts en el sistema operativo local, asociando los dominios con la IP pública de la instancia EC2.

4.2.4.1. Entradas añadidas:

52.204.92.231 controltiempo.dev
52.204.92.231 concentrese.dev
52.204.92.231 piedrapapeltijera.dev

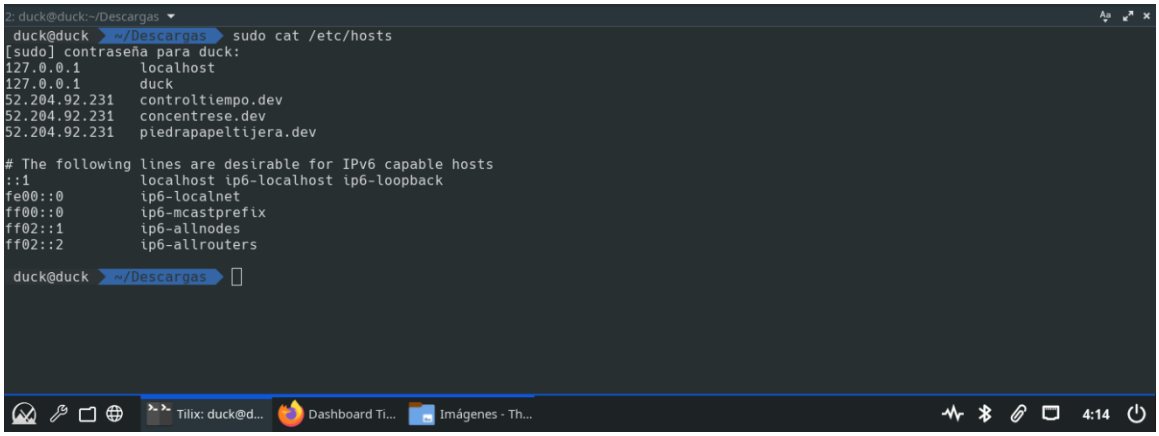

Ilustración 14. Archivo hosts local

4.2.5. Validación de la Solución

Se realizaron pruebas accediendo a los siguientes enlaces desde el navegador:

- http://controltiempo.dev
- http://concentrese.dev
- http://piedrapapeltijera.dev

Todas las aplicaciones se cargaron correctamente, confirmando que el proxy inverso estaba funcionando según lo esperado y que cada contenedor servía su respectiva aplicación.

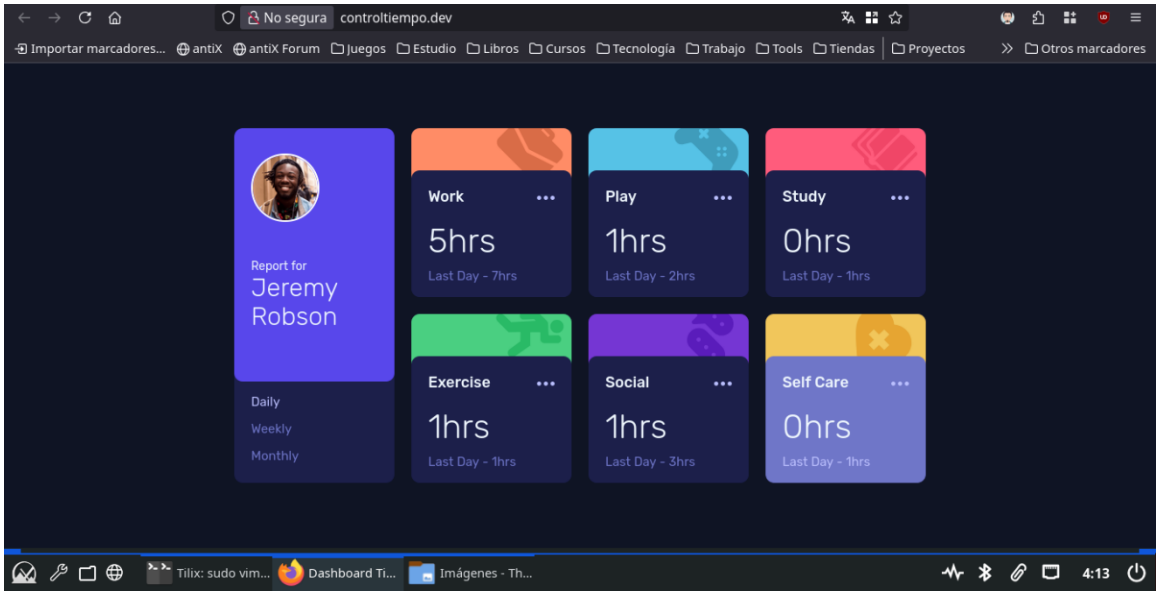

Ilustración 15. Imagen de accediendo a http://controltiempo.dev

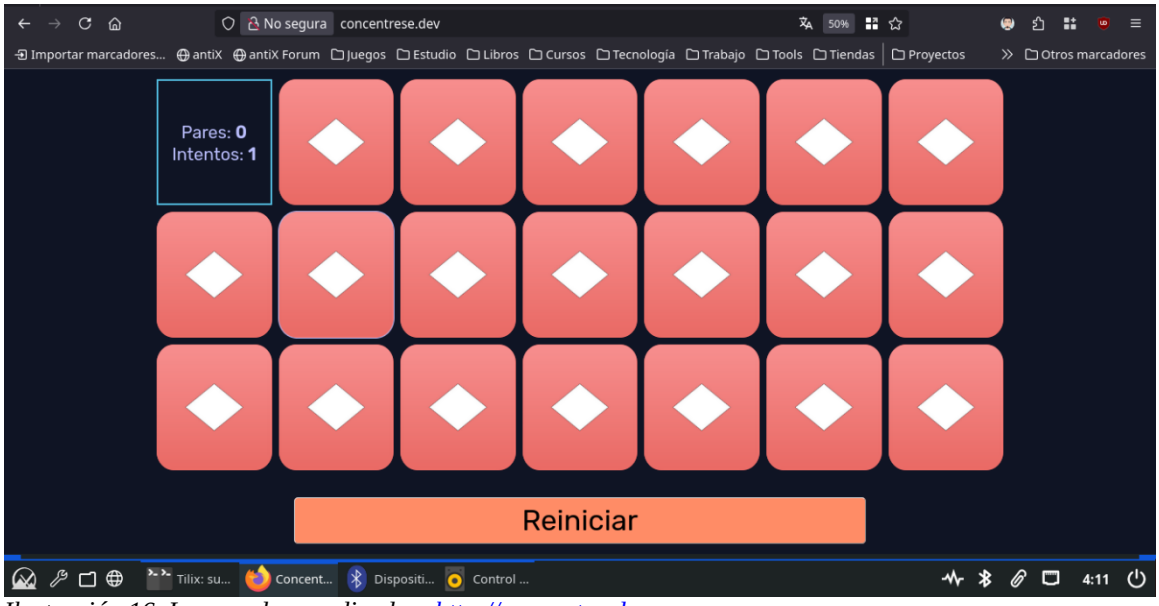

Ilustración 16. Imagen de accediendo a http://concentre.dev

Ilustración 17. Imagen de accediendo a http://piedrapapeltijera.dev

5. Conclusiones

Durante el seminario de AWS pude experimentar de primera mano cómo la computación en la nube transforma la forma de diseñar y manejar infraestructuras, permitiendo automatizar y optimizar el despliegue de servicios y aplicaciones de forma rápida y sencilla.

Quedé sorprendido al ver que, gracias a herramientas como el Auto Scaling y el Balanceador de Carga, el sistema puede ajustar automáticamente sus recursos según la demanda, reduciendo la necesidad de intervención manual y evitando interrupciones en el servicio.

También aprendí la importancia de crear imágenes personalizadas para asegurar que todas las instancias se configuren de la misma manera, lo que garantiza que los despliegues sean consistentes y confiables.

El uso de contenedores para alojar diferentes aplicaciones demostró lo flexible y práctico que es separar cada servicio, facilitando el mantenimiento y la transferencia entre distintos entornos.

Además, la configuración de NGINX como proxy inverso permitió dirigir el tráfico de manera eficiente a cada aplicación, mejorando la experiencia del usuario y optimizando la distribución de carga.

Esta experiencia reafirma mi interés en la computación en la nube pues entendí que es fundamental para desarrollar soluciones robustas y escalables, permitiendo a las organizaciones innovar y competir de manera más efectiva en un mercado cada vez más exigente.

6. Referencias

Amazon Web Services. (s.f.). *Título de la página o sección consultada*. AWS. Recuperado el 6 de abril de 2025, de https://docs.aws.amazon.com/

Docker Hub Container Image Library | App Containerization. (s. f.). https://hub.docker.com/

nginx documentation. (s. f.). https://nginx.org/en/docs/

«Home». (2025, 27 febrero). Docker Documentation. https://docs.docker.com/