



TRABAJO DE GRADO
Opción Seminario-Diplomado.

Cafetería las tres J
Seminario Amazon AWS

Corporación Universitaria Remington.
Ingeniería de sistemas

Jhonatan Pineda Carmona
Juan Pablo Ladino Betancur
Johan David Montoya Loaiza
Seminario de Grado
2024.

Tabla de Contenidos

Resumen.....	3
Marco conceptual y contextual	4
Recursos de AWS para ejecutar instancias	5
Servicios Complementarios a EC2:	5
Servicios Complementarios a EC2 con Ejemplos.....	5
Amazon Virtual Private Cloud (VPC).....	6
Elastic Load Balancing (ELB)	6
Amazon Elastic Block Store (EBS)	7
Amazon Simple Storage Service (S3).....	8
Otros Servicios Complementarios.....	8
Diagrama de flujo	9
Configuración servidor web en Amazon Linux	10
Video explicativo Primer entrega:	23
Crear y configurar el balanceador de carga.	24
Video explicativo Segunda entrega:	31
Instalación de nginx	32
Video explicativo entrega final:.....	40
Conclusiones	41
Referencias	42

Resumen

El presente trabajo tiene como objetivo implementar soluciones tecnológicas avanzadas utilizando servicios de Amazon Web Services (AWS) para optimizar la infraestructura de una aplicación web, simulando un entorno empresarial real en el contexto de una Cafetería Las Tres J. Se despliega una arquitectura que aprovecha instancias EC2 (Elastic Compute Cloud) y contenedores Docker, garantizando una plataforma con alta disponibilidad, escalabilidad y gestión eficiente del tráfico. A lo largo del documento, se detalla el proceso paso a paso para configurar y desplegar servicios en AWS.

Marco conceptual y contextual

El presente informe técnico se centra en la implementación de soluciones basadas en Amazon Web Services (AWS), específicamente en el despliegue de instancias virtuales y contenedores para garantizar una infraestructura escalable y de alta disponibilidad. Este trabajo se enmarca dentro de un seminario de grado sobre AWS, en el que se abordan conceptos fundamentales de computación en la nube, virtualización, y balanceo de carga. Los principales conceptos trabajados incluyen:

Instancias EC2 (Elastic Compute Cloud): Servidores virtuales en la nube para ejecutar aplicaciones.

Contenedores Docker: Herramientas de virtualización ligera que permiten ejecutar múltiples aplicaciones de forma aislada.

Balanceo de Carga (Elastic Load Balancer): Distribución automática del tráfico para evitar sobrecargas.

Escalado Automático (Auto Scaling): Ajuste dinámico de recursos según la demanda.

Redes Virtuales (Amazon VPC): Segmentación y aislamiento de redes en la nube.

El informe describe un ejercicio práctico de implementación en una simulación empresarial para la Cafetería Las Tres J, una empresa ficticia que busca optimizar su infraestructura tecnológica. El objetivo es asegurar que la plataforma web de la cafetería pueda manejar grandes volúmenes de usuarios sin interrupciones y con la mayor seguridad posible. Se aplican los conocimientos adquiridos en el seminario para configurar servidores, implementar balanceo de carga, y desplegar aplicaciones en contenedores, simulando una operación empresarial real.

En esta sección se detalla la implementación y configuración de servicios esenciales de AWS para el despliegue de aplicaciones. Se abordan los siguientes temas:

Recursos de AWS para ejecutar instancias

Amazon EC2: Creación y configuración de instancias.

Amazon AMI (Amazon Machine Images): Plantillas para lanzar instancias con configuraciones predefinidas.

Grupos de Seguridad: Reglas de firewall para proteger las instancias.

Pares de Claves SSH: Mecanismos de autenticación para acceso seguro.

Servicios Complementarios a EC2:

Amazon VPC: Creación de redes privadas para aislar instancias.

Elastic Load Balancing: Distribución del tráfico entre instancias.

Amazon EBS: Almacenamiento persistente para instancias.

Amazon S3: Almacenamiento de objetos para respaldos y archivos estáticos.

Se ofrece una descripción detallada de cómo estos recursos se configuran e integran para lograr una infraestructura eficiente y de alta disponibilidad, adecuada para soportar aplicaciones web en un entorno empresarial.

Servicios Complementarios a EC2 con Ejemplos

Amazon Virtual Private Cloud (VPC)

Ejemplo: Imagina que tienes una aplicación web y un servidor de base de datos. Puedes crear dos subredes dentro de tu VPC, una para la aplicación web y otra para la base de datos. Esto te permite segmentar tu red y controlar el acceso entre los diferentes componentes de tu aplicación.

Usos comunes

Aislamiento de redes: Crear redes privadas completamente separadas para diferentes equipos o aplicaciones.

Seguridad mejorada: Controlar el flujo de tráfico entre subredes y hacia internet.

Redes definidas por software: Configurar redes complejas de forma automatizada.

Elastic Load Balancing (ELB)

Ejemplo: Tienes una aplicación web muy popular que experimenta picos de tráfico. Puedes utilizar un Load Balancer para distribuir el tráfico entre múltiples instancias de tu aplicación. Esto te permite manejar un mayor volumen de solicitudes y mejorar la disponibilidad de tu aplicación.

Usos comunes

Distribución de carga: Equilibrar el tráfico entre múltiples instancia

Alta disponibilidad: Garantizar que tu aplicación esté disponible incluso si una instancia falla.

Balanceo de carga basado en el contenido: Dirigir las solicitudes a instancias específicas en función de criterios como el encabezado de host o el path de la URL.

Amazon Elastic Block Store (EBS)

Ejemplo: Necesitas almacenar datos persistentes para tu aplicación, como archivos de base de datos o imágenes de usuario. Puedes crear volúmenes EBS y adjuntarlos a tus instancias. Si una instancia falla, puedes iniciar una nueva instancia y adjuntar el volumen EBS existente para recuperar tus datos.

Usos comunes

Almacenamiento persistente para instancias: Almacenar datos que deben persistir más allá del ciclo de vida de una instancia.

Copias de seguridad: Crear instantáneas de volúmenes EBS para realizar copias de seguridad.

Snapshots: Crear imágenes de punto en el tiempo de un volumen EBS para realizar restauraciones.

Amazon Simple Storage Service (S3)

Ejemplo: Tienes una aplicación que genera grandes cantidades de archivos de registro.

Puedes utilizar S3 para almacenar estos archivos de forma segura y duradera. S3 también te permite compartir estos archivos con otros usuarios o aplicaciones.

Usos comunes

Almacenamiento de objetos: Almacenar cualquier tipo de archivo, desde imágenes y videos hasta datos de aplicaciones.

Copias de seguridad: Almacenar copias de seguridad de datos de bases de datos, sistemas operativos y aplicaciones.

Archivos estáticos: Servir archivos estáticos para sitios web, como imágenes, CSS y JavaScript.

Otros Servicios Complementarios

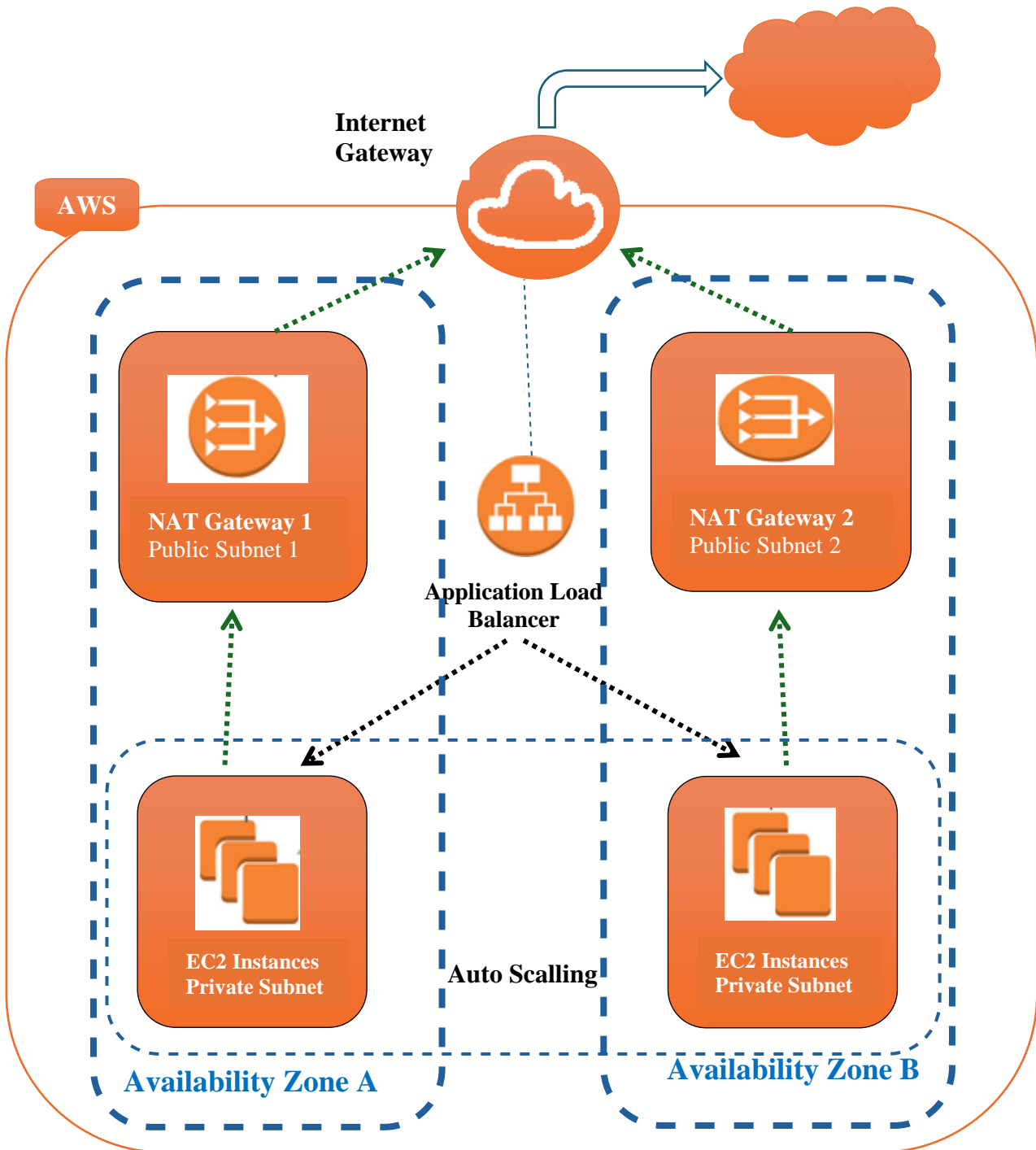
Auto Scaling: Escalar automáticamente el número de instancias en función de la demanda.

CloudWatch: Monitorear el rendimiento de tus instancias y aplicaciones.

Amazon RDS: Gestionar bases de datos relacionales.

Amazon ElastiCache: Implementar cachés in-memory para mejorar el rendimiento de tus aplicaciones.

Diagrama de flujo



Configuración servidor web en Amazon Linux

La imagen muestra el proceso de configuración inicial de una instancia EC2 en AWS. En esta configuración, se selecciona un par de claves para acceso seguro, se utiliza una VPC y subred predeterminadas, y se habilita la asignación automática de una IP pública, lo que permite que la instancia sea accesible desde Internet. Esta configuración es ideal para implementar servicios accesibles externamente, como un servidor web, y está optimizada para el nivel gratuito de AWS.

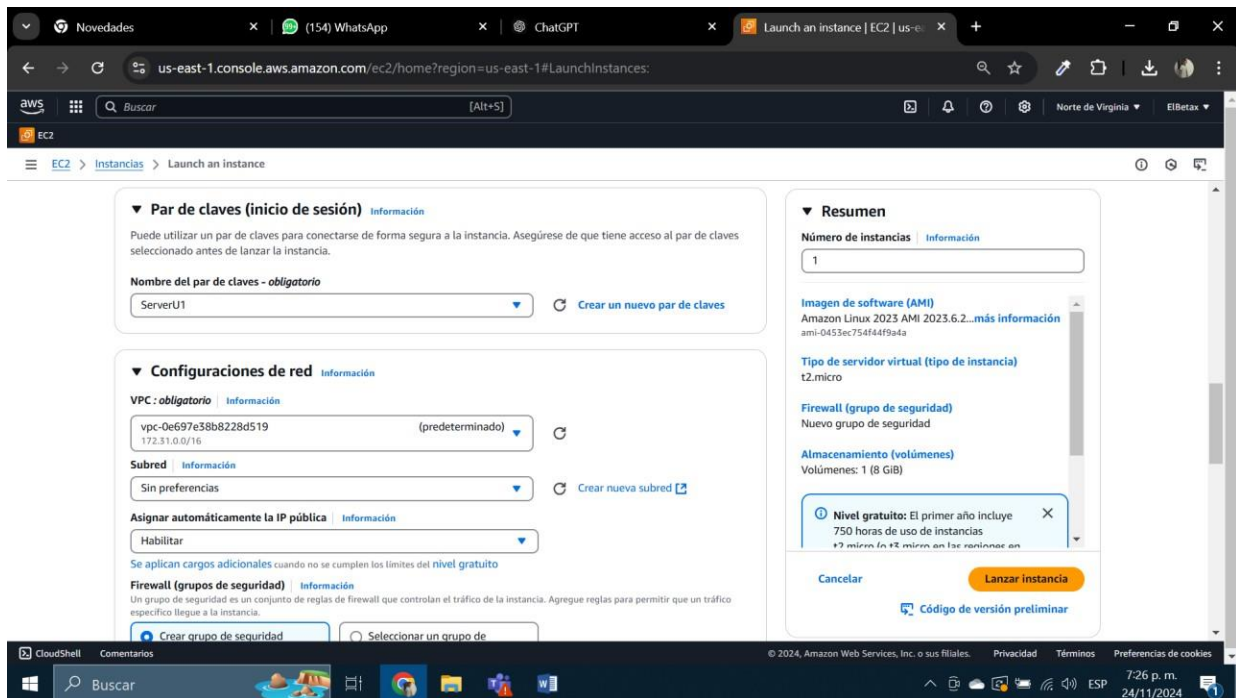


Figura 1 Se asigna subred predeterminada y se habilita la ip pública

La imagen ilustra los pasos finales en la configuración de una instancia EC2, destacando la definición de reglas de seguridad para el acceso remoto mediante SSH (puerto 22). Además, se valida el almacenamiento asignado y la configuración general antes de lanzar la instancia. Se

enfatisa la importancia de confirmar que las reglas de seguridad sean adecuadas para proteger la instancia de accesos no autorizados.

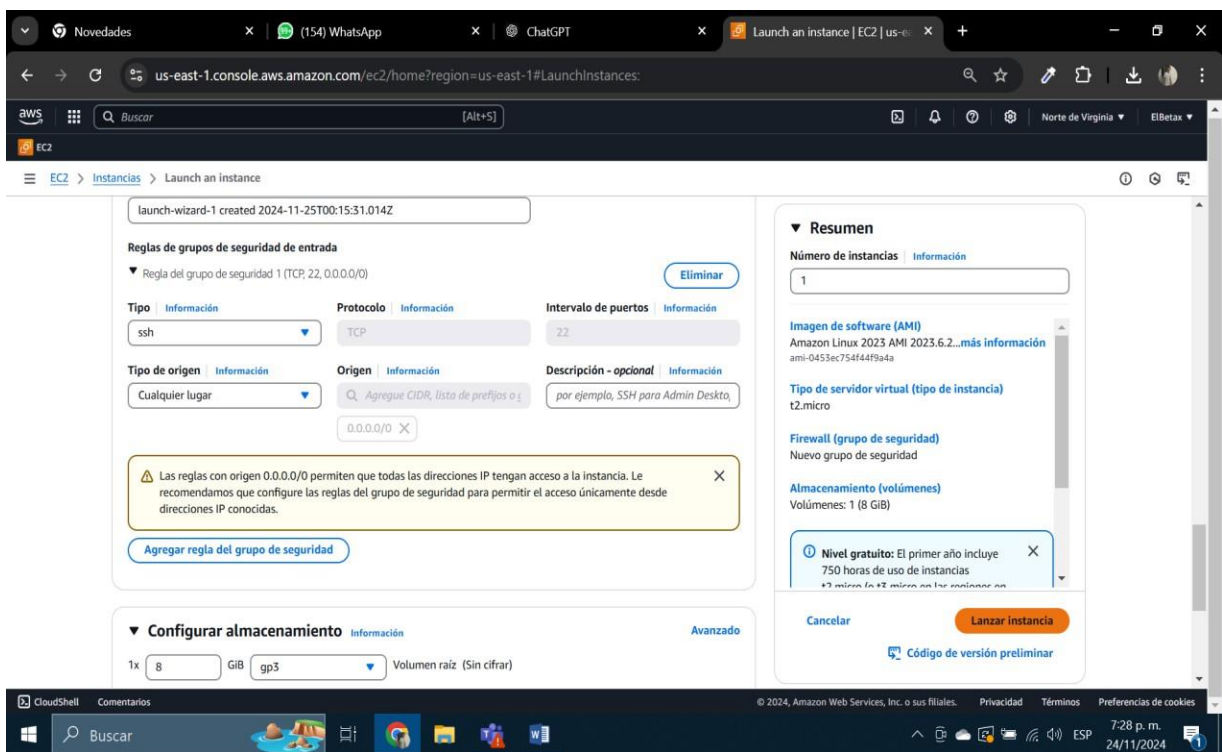


Figura 2 Se lanza la instancia después de confirmar que la configuración de seguridad esté ok

La imagen presenta los detalles completos de una instancia EC2 ya creada y en ejecución en AWS. Se destacan las direcciones IP asignadas (pública y privada), la red (VPC y subred) y el estado actual de la instancia. Esta información es clave para administrar y conectar la instancia a otros servicios o configuraciones específicas

The screenshot displays the AWS Management Console interface for an EC2 instance. The main content area is titled 'Resumen de instancia de i-0926d0d0df3490286 (ServerU1)'. The instance is currently in the 'En ejecución' state. The console provides a comprehensive overview of the instance's configuration, including its network settings, VPC, and subnets. Key details are as follows:

Categoría	Detalle
Identificación	ID de la instancia: i-0926d0d0df3490286
Direcciones IP	Dirección IPv4 pública: 54.159.232.206 dirección abierta Dirección IPv6: - Direcciones IPv4 privadas: 172.31.80.52
Estado	Estado de la instancia: En ejecución
Red	Nombre DNS de IP privada (solo IPv4): ip-172-31-80-52.ec2.internal ID de VPC: vpc-0e697e38b8228d519 ID de subred: subnet-03df374634b3a4ab0
Configuración	Tipo de instancia: t2.micro Tipo de nombre de anfitrión: Nombre de IP: ip-172-31-80-52.ec2.internal Responder al nombre DNS de recurso privado: IPv4 (A) Dirección IP asignada automáticamente: 54.159.232.206 [IP pública]
Seguridad	Rol de IAM: - IMDSv2: Required
Operación	ARN de instancia: arn:aws:ec2:us-east-1:545009857975:instance/i-0926d0d0df3490286 Operador: -
Otros servicios	DNS de IPv4 pública: ec2-54-159-232-206.compute-1.amazonaws.com dirección abierta Direcciones IP elásticas: - Hallazgo de AWS Compute Optimizer: Suscribirse a AWS Compute Optimizer para recibir recomendaciones. Nombre del grupo de Auto Scaling: - Managed: falso

Figura 3 Información de la instancia ya creada como ip privada y pública, subred, vpc, etc.

La imagen captura una sección de la consola de gestión de AWS que muestra los detalles de seguridad de una instancia EC2. Se observa que la instancia está configurada con un grupo de seguridad que permite el acceso desde cualquier dirección IP a través de cualquier puerto. Esta configuración extremadamente permisiva expone la instancia a un riesgo significativo de seguridad, ya que prácticamente cualquier dispositivo en internet puede intentar establecer una conexión.

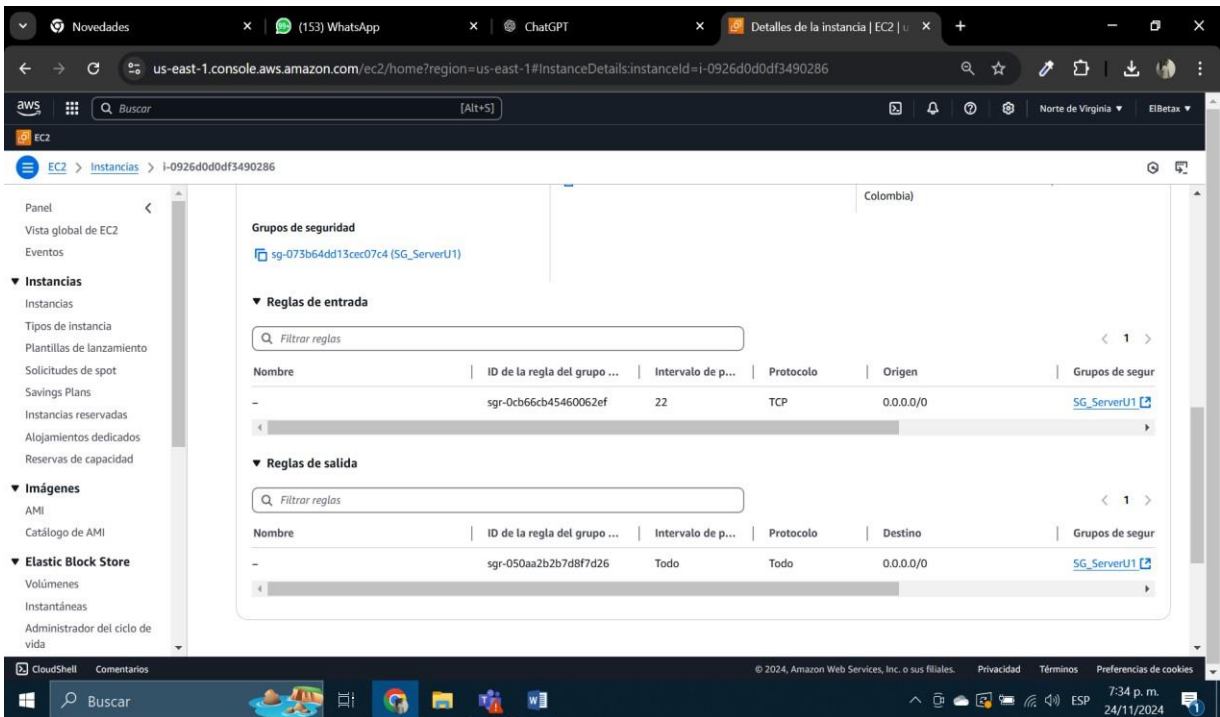


Figura 4 Regla de entrada que permite el acceso para todos

Esta imagen proporciona una vista general de las instancias EC2 creadas en un entorno específico de AWS. Permite a los administradores verificar el estado de las instancias, su configuración y realizar diversas acciones de gestión como iniciar, detener o terminar instancias.

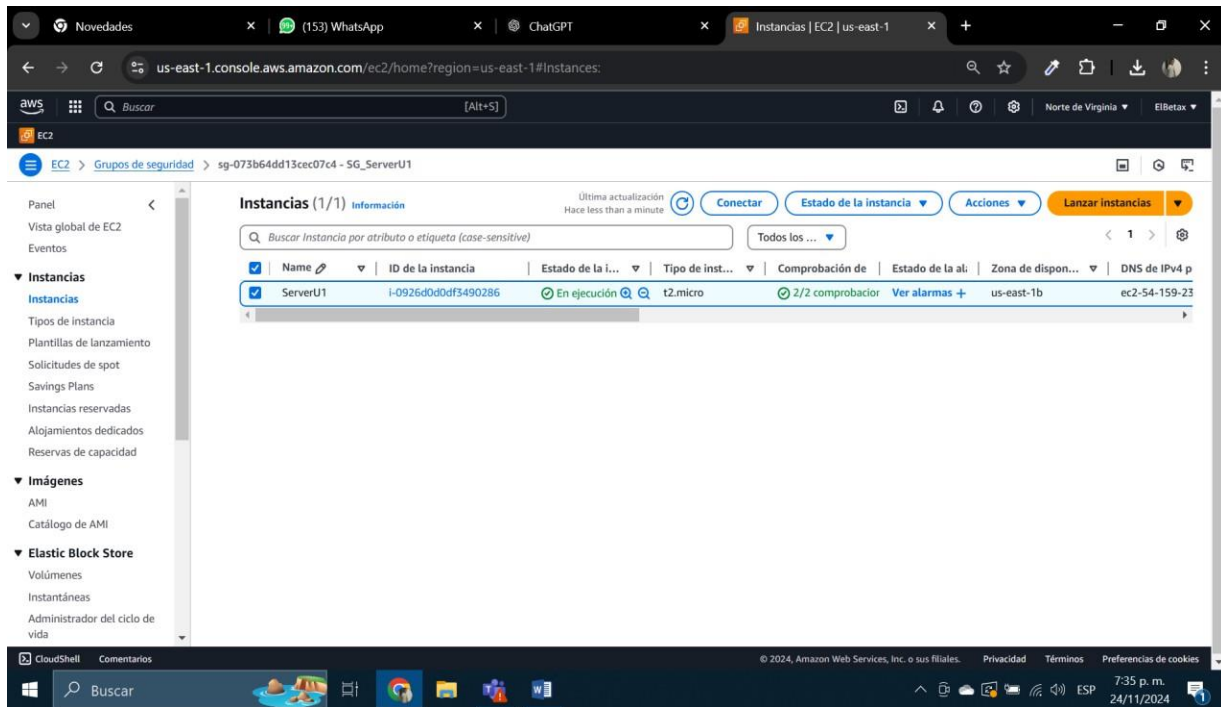


Figura 5 Confirmamos que esté 2/2 en comprobación, lo que indica que ya finalizó la creación

Esta imagen representa un paso fundamental en la gestión de una instancia EC2. Al conectar a la instancia a través de SSH, los administradores pueden ejecutar comandos directamente en la línea de comandos de la instancia, lo que les permite realizar una amplia variedad de tareas, como instalar software, configurar servicios, solucionar problemas, etc.

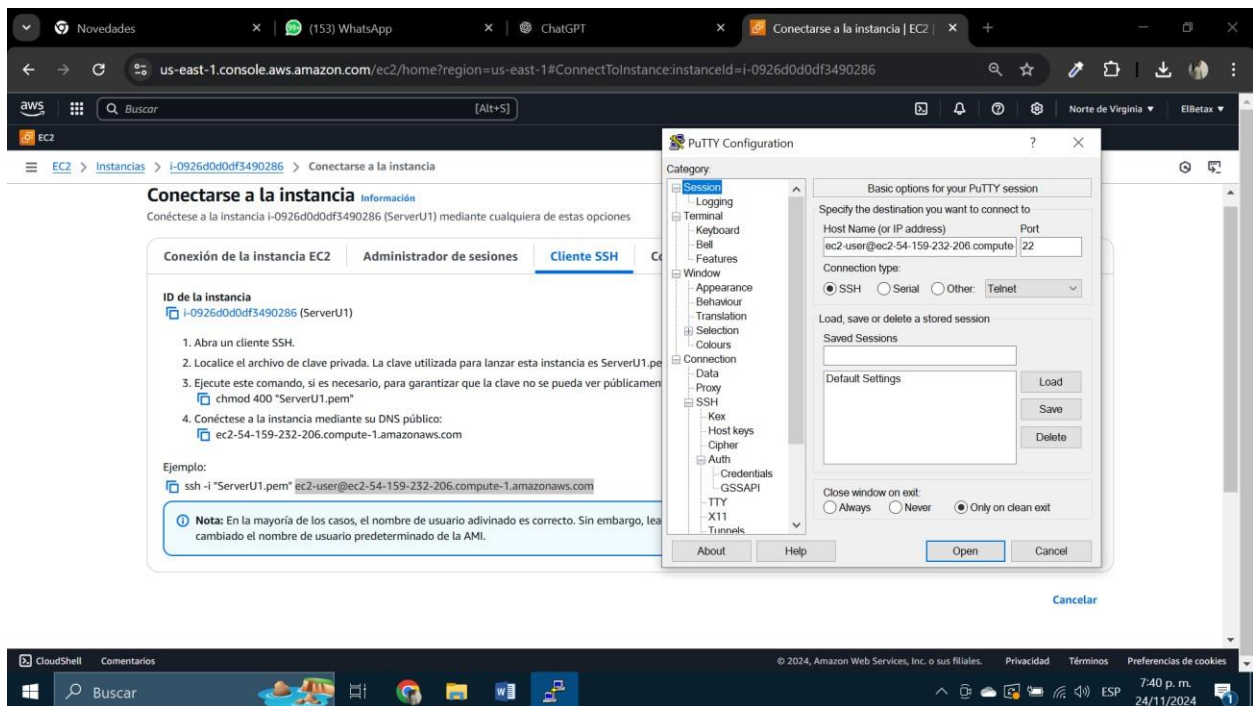


Figura 6 Se ingresa a putty y se asigna la cadena de conexión

Al configurar la autenticación basada en claves SSH, se está estableciendo un mecanismo de seguridad más robusto para acceder a la instancia EC2. Esto se debe a que las claves SSH son más difíciles de robar y crackear en comparación con las contraseñas. Además, las claves SSH permiten automatizar el proceso de conexión a la instancia.

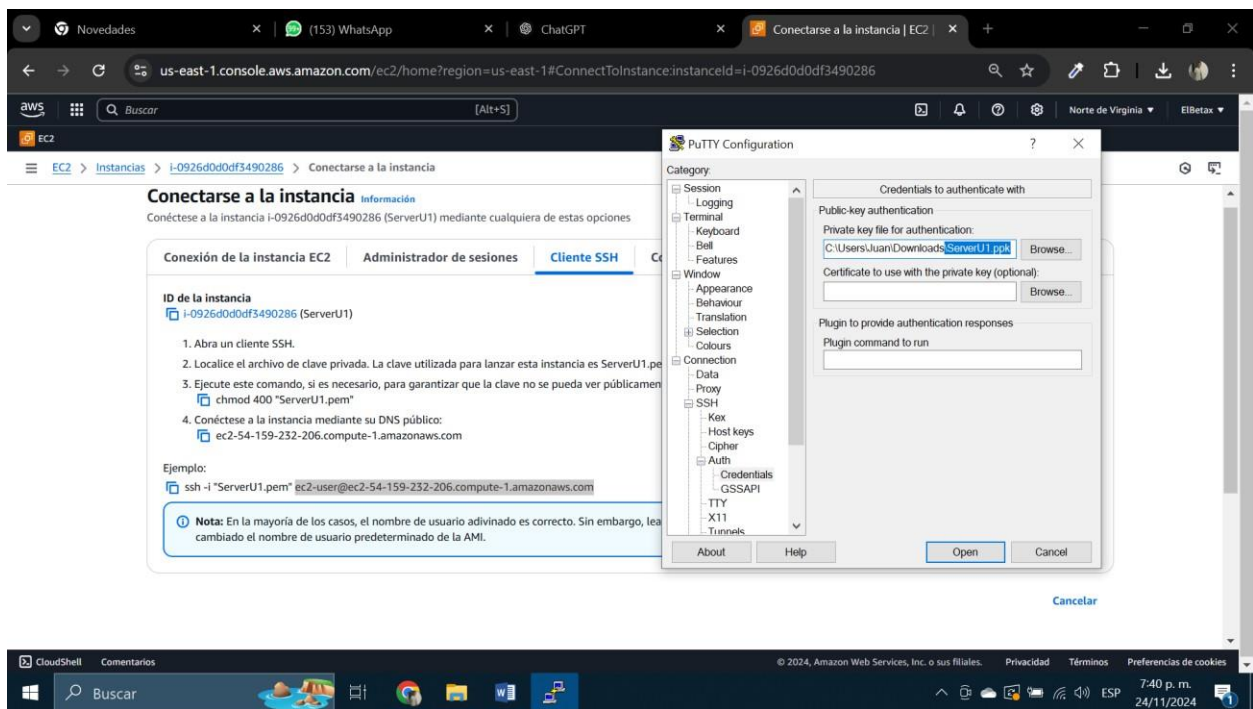
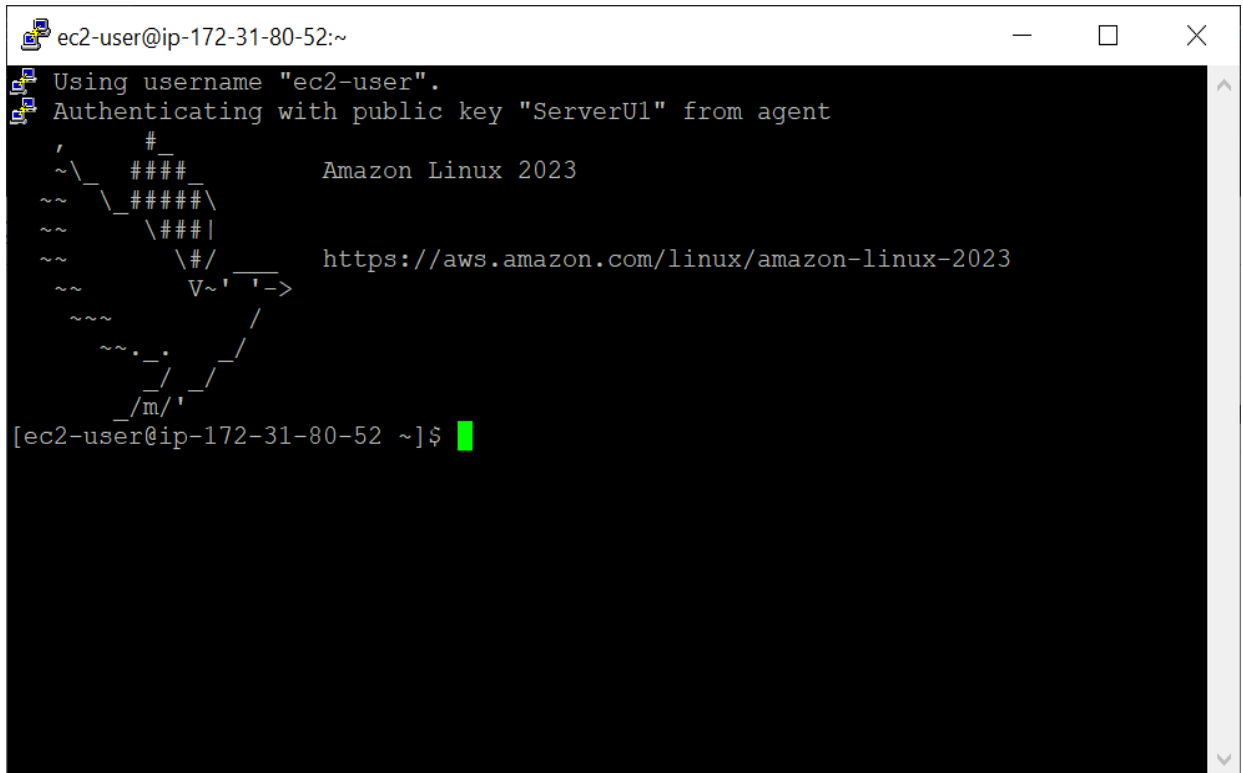


Figura 7 Se carga la cadena de seguridad que se descargó previamente.

Esta imagen representa el punto final del proceso de conexión a una instancia EC2. Al obtener acceso a la línea de comandos de la instancia, podemos realizar una amplia variedad de tareas de administración, como instalar software, configurar servicios, ejecutar scripts, etc.

A terminal window titled "ec2-user@ip-172-31-80-52:~" showing the final steps of an SSH connection. The terminal output includes: "Using username 'ec2-user'.", "Authenticating with public key 'ServerU1' from agent", and the Amazon Linux 2023 logo. The logo consists of a stylized 'A' shape formed by various symbols like '#', '~', and '|'. To the right of the logo, the text "Amazon Linux 2023" and the URL "https://aws.amazon.com/linux/amazon-linux-2023" are displayed. The terminal ends with a prompt "[ec2-user@ip-172-31-80-52 ~]\$" followed by a green cursor.

```
ec2-user@ip-172-31-80-52:~  
Using username "ec2-user".  
Authenticating with public key "ServerU1" from agent  
#  
~\#####_ Amazon Linux 2023  
~~\#####\  
~~\###|  
~~\#/  
~~V~'|'~> https://aws.amazon.com/linux/amazon-linux-2023  
~~~~  
~~~  
~~~  
~/m/'  
[ec2-user@ip-172-31-80-52 ~]$ █
```

Figura 8 Se crea acceso a la máquina virtual desde la consola.

Esta imagen nos permite visualizar de forma clara el proceso de instalación de un software en una instancia EC2. En este caso, se está instalando el servidor web Apache, lo que permitirá a la instancia servir contenido web.

```

root@ip-172-31-80-52/home/ec2-user
(12/12): mod_http2-2.0.27-1.amzn2023.0.3.x86_64.rpm 4.9 MB/s | 166 kB 00:00
-----
Total 11 MB/s | 2.3 MB 00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction:
Preparing:
Installing:
  apr-1.7.2-2.amzn2023.0.2.x86_64 1/1
Installing:
  apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64 1/12
Installing:
  apr-util-1.6.3-1.amzn2023.0.1.x86_64 2/12
Installing:
  mailcap-2.1.49-3.amzn2023.0.3.noarch 3/12
Installing:
  httpd-tools-2.4.62-1.amzn2023.x86_64 4/12
Installing:
  libbrotli-1.0.9-4.amzn2023.0.2.x86_64 5/12
Installing:
  httpd-filesystem-2.4.62-1.amzn2023.noarch 6/12
Running scriptlet: httpd-filesystem-2.4.62-1.amzn2023.noarch 7/12
Installing:
  httpd-core-2.4.62-1.amzn2023.x86_64 7/12
Installing:
  mod_http2-2.0.27-1.amzn2023.0.3.x86_64 8/12
Installing:
  mod_lua-2.4.62-1.amzn2023.x86_64 9/12
Installing:
  generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch 10/12
Installing:
  httpd-2.4.62-1.amzn2023.x86_64 11/12
Running scriptlet: httpd-2.4.62-1.amzn2023.x86_64 12/12
Verifying:
  apr-1.7.2-2.amzn2023.0.2.x86_64 1/12
Verifying:
  apr-util-1.6.3-1.amzn2023.0.1.x86_64 2/12
Verifying:
  apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64 3/12
Verifying:
  generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch 4/12
Verifying:
  httpd-2.4.62-1.amzn2023.x86_64 5/12
Verifying:
  httpd-core-2.4.62-1.amzn2023.x86_64 6/12
Verifying:
  httpd-filesystem-2.4.62-1.amzn2023.noarch 7/12
Verifying:
  httpd-tools-2.4.62-1.amzn2023.x86_64 8/12
Verifying:
  libbrotli-1.0.9-4.amzn2023.0.2.x86_64 9/12
Verifying:
  mailcap-2.1.49-3.amzn2023.0.3.noarch 10/12
Verifying:
  mod_http2-2.0.27-1.amzn2023.0.3.x86_64 11/12
Verifying:
  mod_lua-2.4.62-1.amzn2023.x86_64 12/12

Installed:
apr-1.7.2-2.amzn2023.0.2.x86_64      apr-util-1.6.3-1.amzn2023.0.1.x86_64      apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64
generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch  httpd-2.4.62-1.amzn2023.x86_64      httpd-core-2.4.62-1.amzn2023.x86_64
httpd-filesystem-2.4.62-1.amzn2023.noarch      httpd-tools-2.4.62-1.amzn2023.x86_64      libbrotli-1.0.9-4.amzn2023.0.2.x86_64
mailcap-2.1.49-3.amzn2023.0.3.noarch          mod_http2-2.0.27-1.amzn2023.0.3.x86_64      mod_lua-2.4.62-1.amzn2023.x86_64

Complete!
[root@ip-172-31-80-52 ec2-user]#

```

Figura 9 Se instala servicio apache.

Esta imagen nos muestra el paso final en la configuración de un servidor web Apache en una instancia EC2. Al verificar que el servicio Apache está en ejecución y escuchando en el puerto 80, podemos confirmar que nuestra instancia está lista para servir contenido web.

```

root@ip-172-31-80-52/home/ec2-user
Verifying      : httpd-2.4.62-1.amzn2023.x86_64          5/12
Verifying      : httpd-core-2.4.62-1.amzn2023.x86_64   6/12
Verifying      : httpd-filesystem-2.4.62-1.amzn2023.noarch 7/12
Verifying      : httpd-tools-2.4.62-1.amzn2023.x86_64   8/12
Verifying      : libbrotli-1.0.9-4.amzn2023.0.2.x86_64  9/12
Verifying      : mailcap-2.1.49-3.amzn2023.0.3.noarch   10/12
Verifying      : mod_http2-2.0.27-1.amzn2023.0.3.x86_64 11/12
Verifying      : mod_lua-2.4.62-1.amzn2023.x86_64      12/12

Installed:
apr-1.7.2-2.amzn2023.0.2.x86_64          apr-util-1.6.3-1.amzn2023.0.1.x86_64      apr-util-openssl-1.6.3-1.amzn2023.0.1.x86_64
generic-logos-httpd-18.0.0-12.amzn2023.0.3.noarch  httpd-2.4.62-1.amzn2023.x86_64          httpd-core-2.4.62-1.amzn2023.x86_64
httpd-filesystem-2.4.62-1.amzn2023.noarch  httpd-tools-2.4.62-1.amzn2023.x86_64     libbrotli-1.0.9-4.amzn2023.0.2.x86_64
mailcap-2.1.49-3.amzn2023.0.3.noarch      mod_http2-2.0.27-1.amzn2023.0.3.x86_64  mod_lua-2.4.62-1.amzn2023.x86_64

Complete!
[root@ip-172-31-80-52 ec2-user]# systemctl status httpd
o httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; preset: disabled)
   Active: inactive (dead)
     Docs: man:httpd.service(8)
[root@ip-172-31-80-52 ec2-user]# ^C
[root@ip-172-31-80-52 ec2-user]# systemctl start httpd
[root@ip-172-31-80-52 ec2-user]# systemctl status httpd
● httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; disabled; preset: disabled)
   Active: active (running) since Mon 2024-11-25 00:45:16 UTC; 3s ago
     Docs: man:httpd.service(8)
  Main PID: 26365 (httpd)
    Status: "Started, listening on: port 80"
   Tasks: 177 (limit: 1111)
  Memory: 12.9M
     CPU: 54ms
  CGroup: /system.slice/httpd.service
          └─26365 /usr/sbin/httpd -DFOREGROUND
            └─26366 /usr/sbin/httpd -DFOREGROUND
              └─26367 /usr/sbin/httpd -DFOREGROUND
                └─26368 /usr/sbin/httpd -DFOREGROUND
                  └─26369 /usr/sbin/httpd -DFOREGROUND

Nov 25 00:45:16 ip-172-31-80-52.ec2.internal systemd[1]: Starting httpd.service - The Apache HTTP Server...
Nov 25 00:45:16 ip-172-31-80-52.ec2.internal systemd[1]: Started httpd.service - The Apache HTTP Server.
Nov 25 00:45:16 ip-172-31-80-52.ec2.internal httpd[26365]: Server configured, listening on: port 80
[root@ip-172-31-80-52 ec2-user]#

```

Figura 10 Se corre servicio apache ya que se instala y queda apagado.

Esta imagen muestra un paso crucial en la configuración de la seguridad de una instancia EC2. Al crear una regla de entrada específica, se está permitiendo el acceso a la instancia desde una dirección IP determinada, lo que garantiza que solo usuarios autorizados puedan acceder a ella.

The screenshot shows the AWS Management Console interface for editing inbound security rules. The browser address bar indicates the URL: `us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#ModifyInboundSecurityGroupRules:securityGroupId=sg-073b64dd1...`. The page title is "Editar reglas de entrada" (Edit inbound rules). Below the title, there is a table of existing rules:

ID de la regla del grupo de seguridad	Tipo	Protocolo	Intervalo de puertos	Origen	Descripción: opcional	Acciones
sgr-046b2070fe32672ac	HTTP	TCP	80	Per...		Eliminar
sgr-0cb66cb45460062ef	SSH	TCP	22	Per...		Eliminar

Below the table, there is a warning message: "Las reglas cuyo origen es 0.0.0.0/0 o ::/0 permiten a todas las direcciones IP acceder a la instancia. Recomendamos configurar reglas de grupo de seguridad para permitir el acceso únicamente desde direcciones IP conocidas." (Rules whose source is 0.0.0.0/0 or ::/0 allow all IP addresses to access the instance. We recommend configuring security group rules to allow access only from known IP addresses.)

At the bottom of the page, there are buttons for "Cancelar", "Previsualizar los cambios", and "Guardar reglas".

Figura 11 Se crea nueva regla de entrada que permita acceder a la ip pública del servidor.

Esta imagen es una confirmación visual de que una determinada tarea o configuración se ha realizado correctamente, una configuración y prueba exitosa.

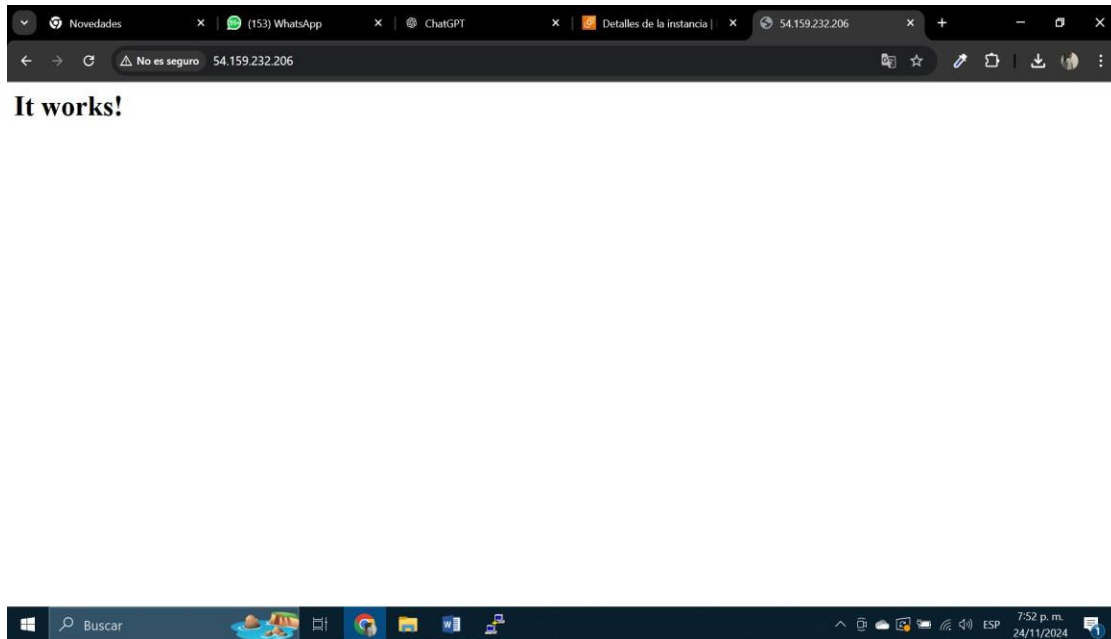
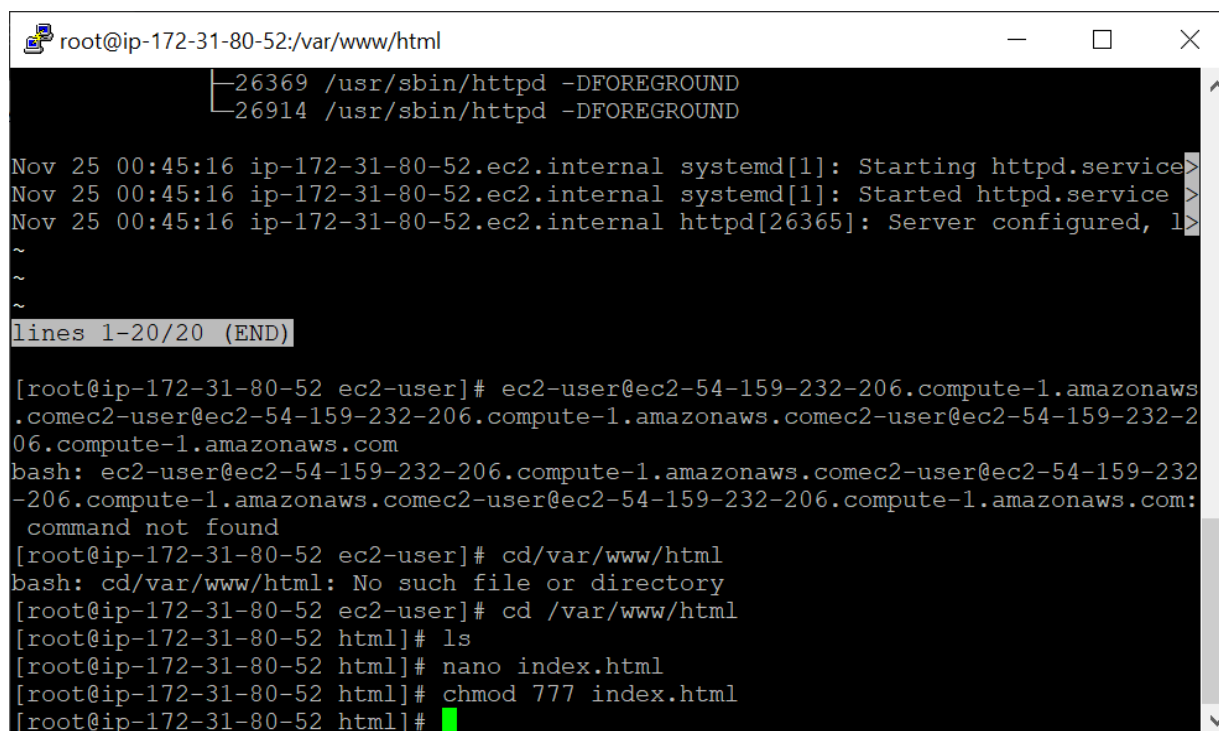


Figura 12 *Funcionamiento ok.*

Esta imagen muestra los pasos básicos para crear una página web simple en un servidor Apache. Al crear un archivo `index.html` y configurar los permisos correctamente, se puede servir contenido web desde la instancia EC2.



```
root@ip-172-31-80-52:/var/www/html
└─26369 /usr/sbin/httpd -DFOREGROUND
└─26914 /usr/sbin/httpd -DFOREGROUND
Nov 25 00:45:16 ip-172-31-80-52.ec2.internal systemd[1]: Starting httpd.service
Nov 25 00:45:16 ip-172-31-80-52.ec2.internal systemd[1]: Started httpd.service
Nov 25 00:45:16 ip-172-31-80-52.ec2.internal httpd[26365]: Server configured, 1
~
~
~
lines 1-20/20 (END)
[root@ip-172-31-80-52 ec2-user]# ec2-user@ec2-54-159-232-206.compute-1.amazonaws.com
.comec2-user@ec2-54-159-232-206.compute-1.amazonaws.com
bash: ec2-user@ec2-54-159-232-206.compute-1.amazonaws.com: command not found
[root@ip-172-31-80-52 ec2-user]# cd/var/www/html
bash: cd/var/www/html: No such file or directory
[root@ip-172-31-80-52 ec2-user]# cd /var/www/html
[root@ip-172-31-80-52 html]# ls
[root@ip-172-31-80-52 html]# nano index.html
[root@ip-172-31-80-52 html]# chmod 777 index.html
[root@ip-172-31-80-52 html]#
```

Figura 13 Se crea archivo `index.html` para editar la página

Esta imagen representa un paso básico en la configuración de un servidor web. Al crear una página HTML simple y acceder a ella a través de un navegador, se verifica que el servidor web esté funcionando correctamente y que pueda servir contenido estático.

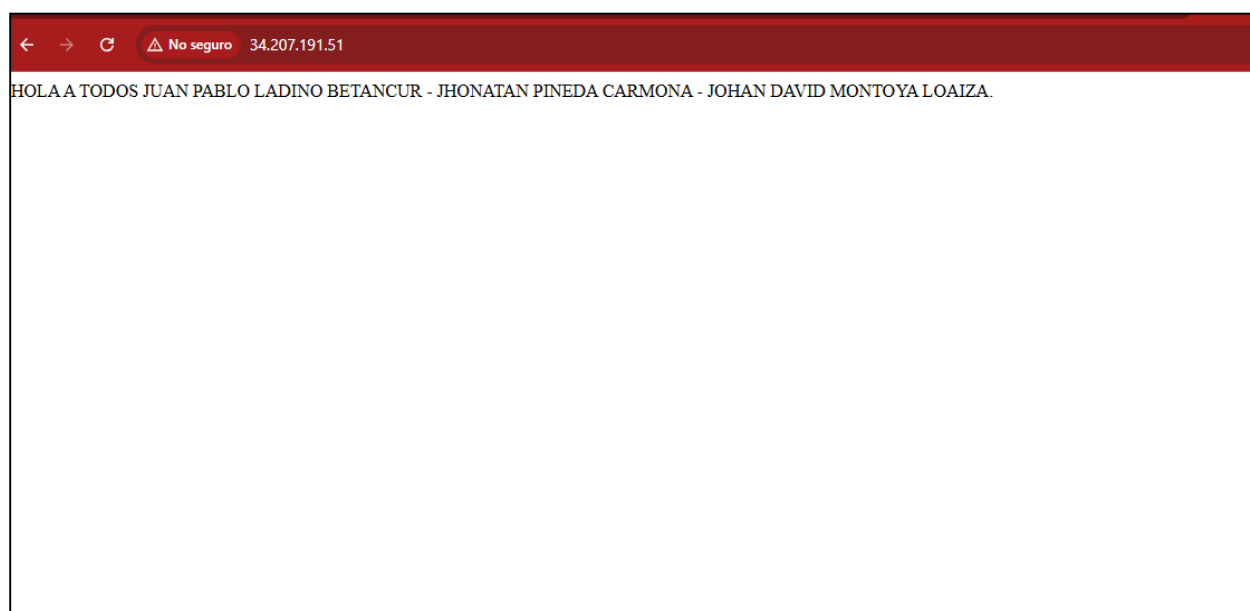


Figura 14 *Se confirma que el archivo html muestre el contenido el servidor web.*

Video explicativo Primer entrega:

[Video]. YouTube. <https://youtu.be/iQIzbyXArjE>

Crear y configurar el balanceador de carga.

La imagen captura el momento en que se está configurando un balanceador de carga de aplicaciones en AWS. El usuario está definiendo los parámetros básicos del balanceador, como su nombre, esquema de enrutamiento y tipo de dirección IP. Una vez configurado, este balanceador de carga podrá distribuir el tráfico entrante entre múltiples instancias EC2, mejorando la disponibilidad y el rendimiento de la aplicación.

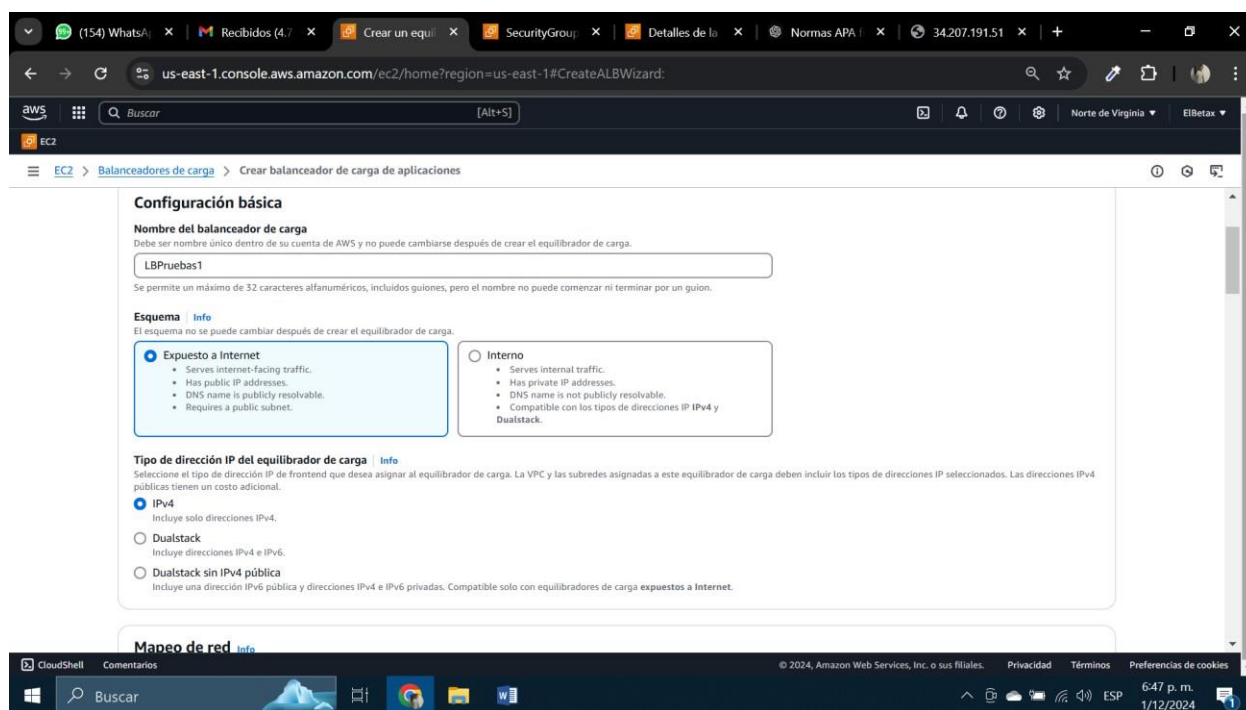


Figura 15 Configuración de un balanceador de carga en AWS

La configuración de la subred es un paso crucial en la creación de un balanceador de carga. Al seleccionar la subred adecuada, se garantiza que el balanceador pueda comunicarse con las instancias y que esté accesible desde Internet. Se debe crear en la misma VPC que la instancia, y además utilizar una subred pública.

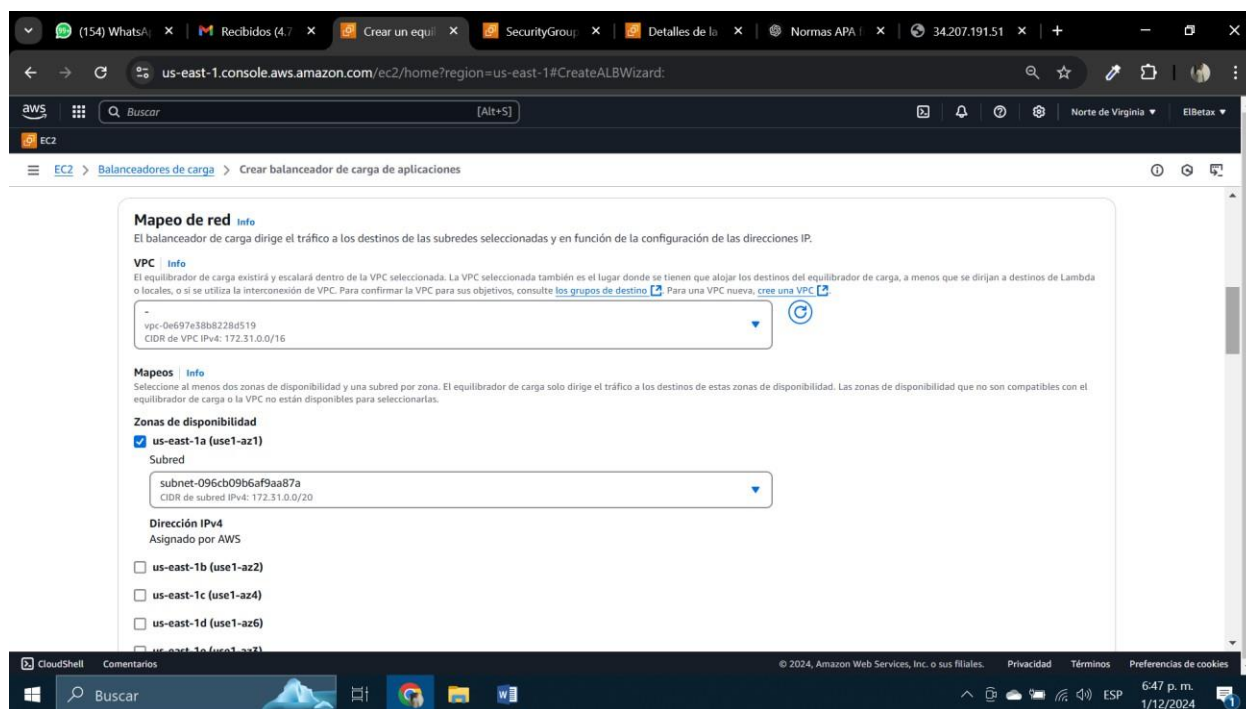


Figura 16 Configuración de la subred para el balanceador de carga

La configuración del grupo de seguridad es esencial para garantizar la seguridad del balanceador de carga. Al permitir solo el tráfico necesario y bloquear el tráfico no deseado, se reduce el riesgo de ataques y se protege la infraestructura.

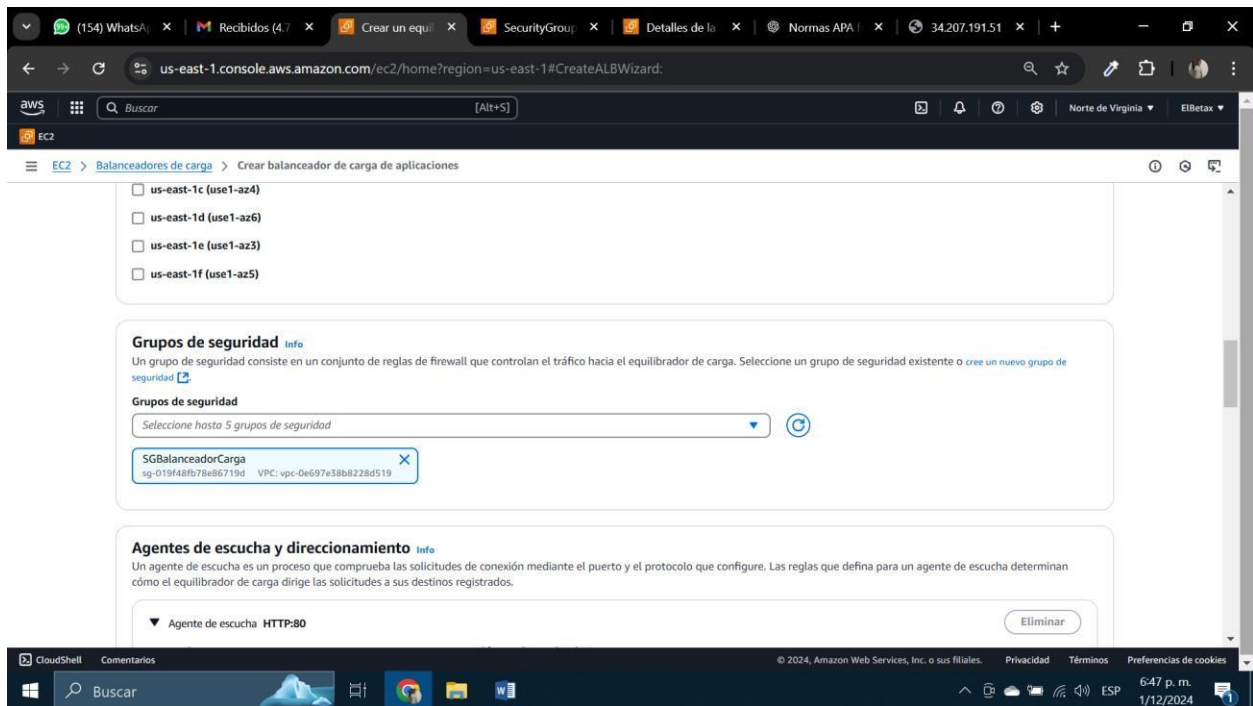


Figura 17 Configuración del grupo de seguridad para el balanceador de carga

El grupo de destino es un componente esencial de un balanceador de carga. Define los recursos a los que se dirigirá el tráfico y permite una gestión flexible y escalable de la aplicación.

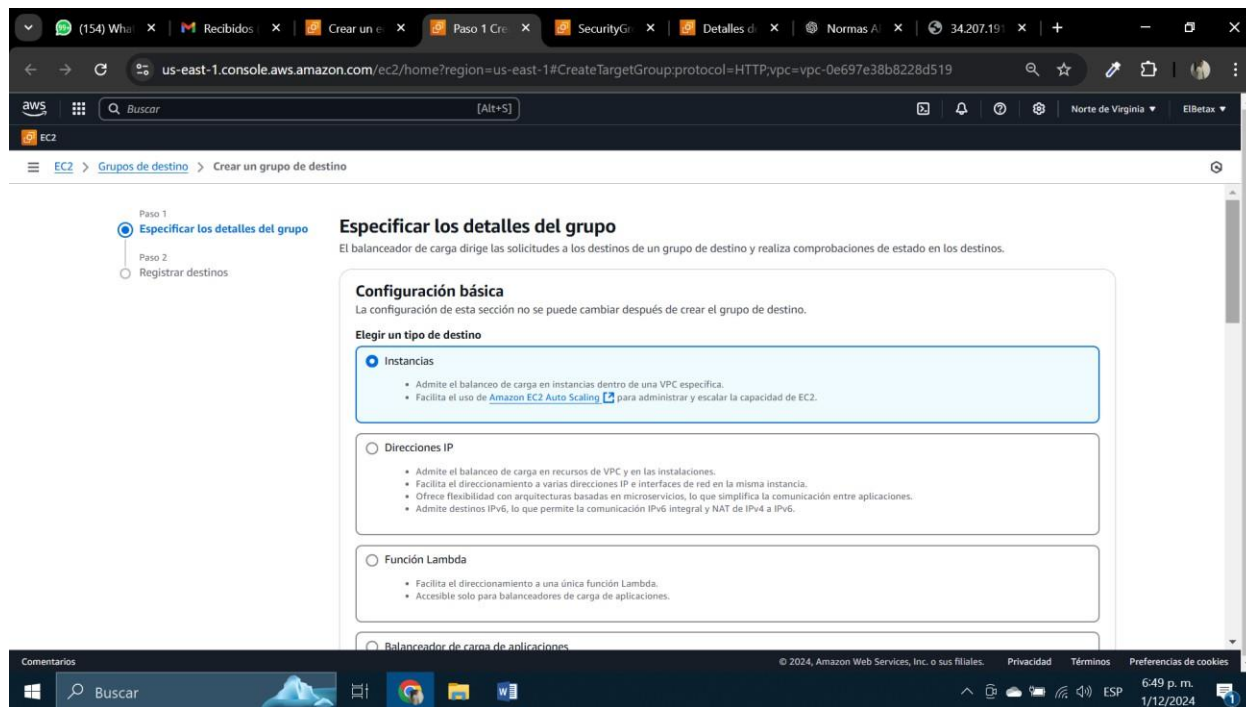


Figura 18 Configuración del grupo de destino para el balanceador de carga

La configuración del protocolo y las comprobaciones de estado es fundamental para garantizar que el balanceador de carga funcione correctamente y distribuya el tráfico de manera eficiente entre las instancias.

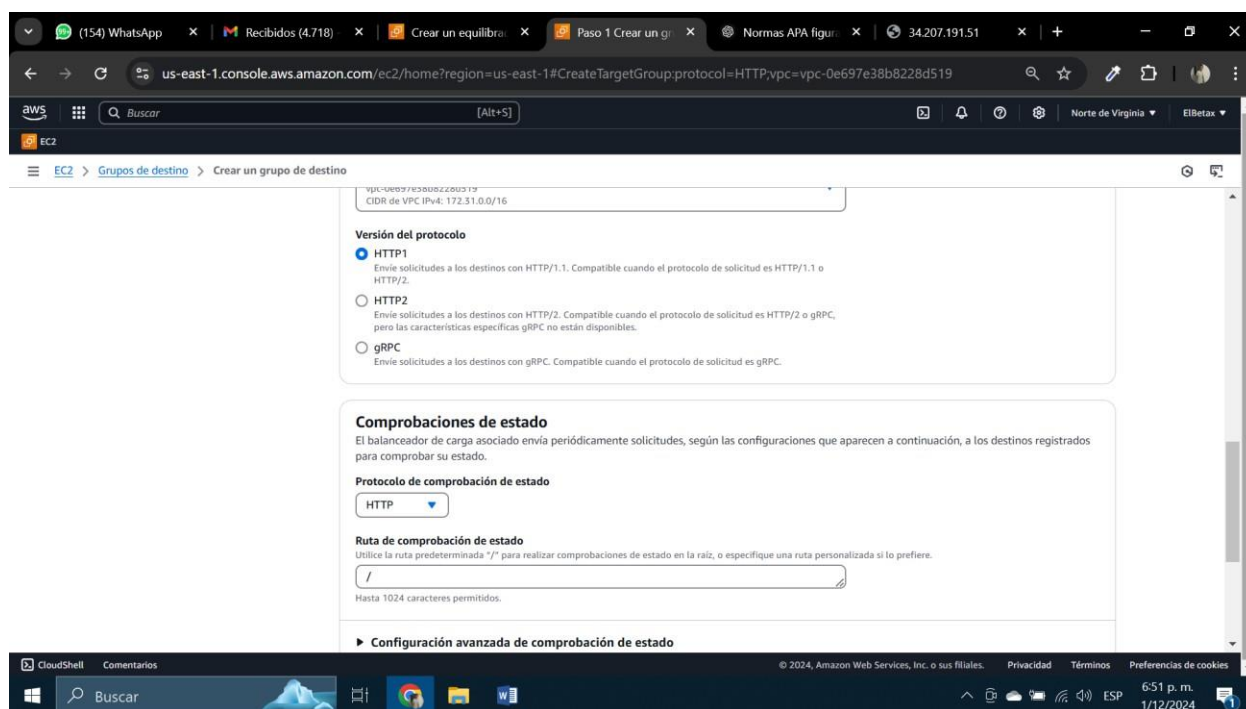


Figura 19 Configuración del protocolo y comprobaciones de estado

La captura de pantalla muestra que el grupo de destino se ha creado correctamente y que el balanceador de carga está listo para distribuir el tráfico entre las instancias especificadas.

Seleccionamos las instancias, las agregamos ambas, y el sistema nos confirma creación exitosa.

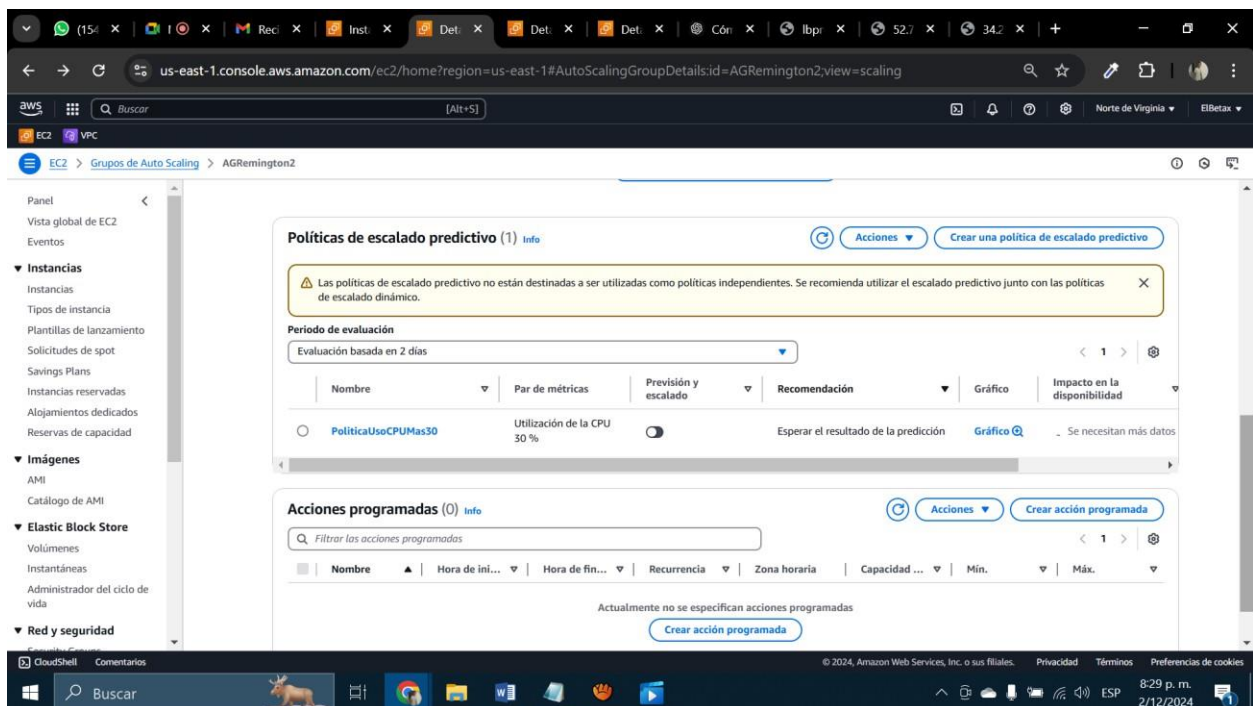
The screenshot shows the AWS Management Console interface for a newly created Target Group. At the top, a green notification banner states: "Se creó correctamente el grupo de destinos: TGReming1. La detección de anomalías se aplica automáticamente a todos los destinos registrados. El estado se puede ver en la pestaña Destinos." Below this, the "TGReming1" details are shown, including the ARN, protocol (HTTP), port (80), and VPC ID. A summary table indicates 2 total destinations, all in "En buen estado". The bottom section shows the distribution of destinations by availability zone.

Métrica	Valor
Destinos totales	2
En buen estado	0
En mal estado	0
Sin utilizar	2
Inicial	0
Vaciado	0

Distribución de destinos por zona de disponibilidad (AZ)
Seleccione los valores de esta tabla para ver los filtros correspondientes aplicados a la tabla Destinos registrados que aparece a continuación.

Figura 20 Confirmación de la creación del grupo de destino

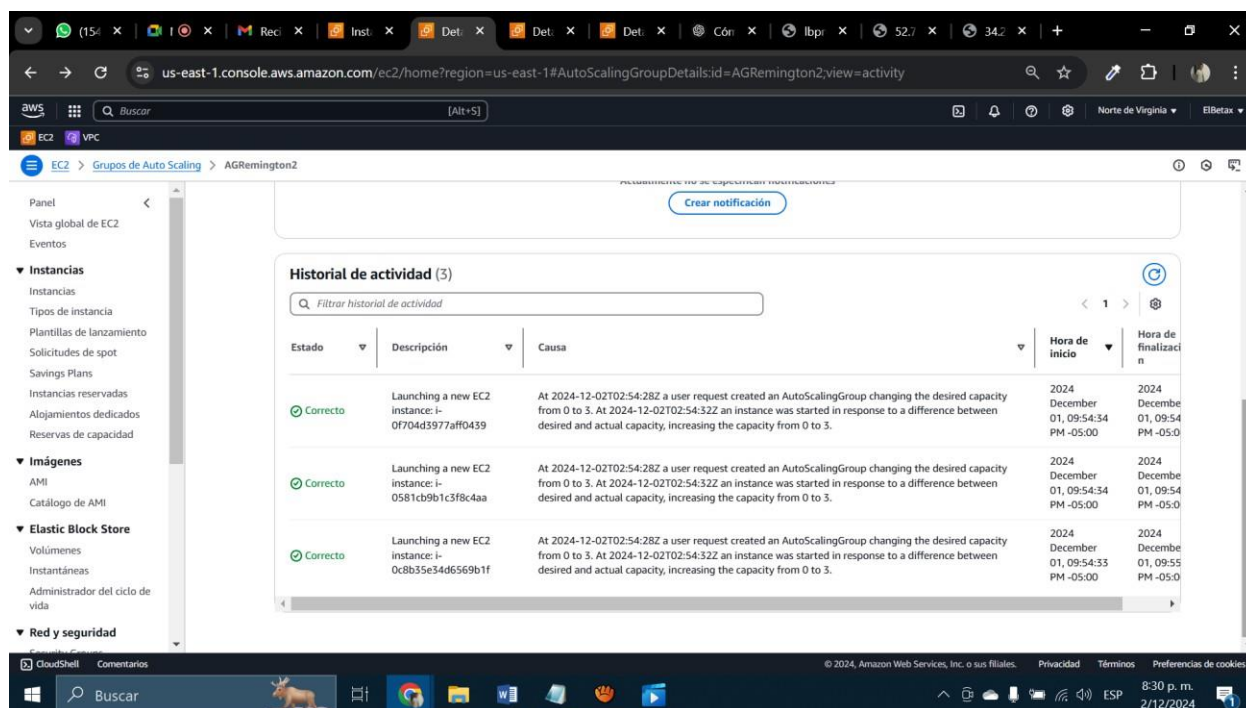
Se asigna un escalado automático del 30%. Asignar un escalado automático del 30% significa que el sistema ajustará automáticamente el número de instancias en función de la utilización de la CPU, asegurando que siempre haya suficientes recursos disponibles para atender la demanda y optimizando los costos.



The screenshot displays the AWS Management Console interface for configuring an Auto Scaling Group. The main content area is titled "Políticas de escalado predictivo (1)" and includes a warning message: "Las políticas de escalado predictivo no están destinadas a ser utilizadas como políticas independientes. Se recomienda utilizar el escalado predictivo junto con las políticas de escalado dinámico." Below this, the "Periodo de evaluación" is set to "Evaluación basada en 2 días". A table lists the policy "PoliticaUseCPUMas30" with a CPU utilization threshold of 30%. The "Acciones programadas (0)" section is currently empty, with a note: "Actualmente no se especifican acciones programadas". The left sidebar shows navigation options for EC2, including Instancias, Imágenes, Elastic Block Store, and Red y seguridad. The bottom of the screen shows the Windows taskbar with the date 2/12/2024 and time 8:29 p. m.

Figura 21 Escalado automático en AWS

En el apartado de Actividad del grupo de Auto Scaling, se puede ver el historial de creación de instancias. El historial de actividad de un grupo de Auto Scaling es una herramienta esencial para comprender el comportamiento de tu aplicación y garantizar que siempre tenga los recursos necesarios para satisfacer la demanda. Al analizar este historial, puedes identificar problemas, optimizar el rendimiento y tomar decisiones informadas sobre la configuración de tu grupo de Auto Scaling.



The screenshot displays the AWS Management Console interface for an Auto Scaling group named 'AGRemington2'. The main content area shows the 'Historial de actividad (3)' (Activity History) section, which contains a table of activity events. The table has columns for 'Estado' (Status), 'Descripción' (Description), 'Causa' (Cause), 'Hora de inicio' (Start Time), and 'Hora de finalización' (End Time). Three events are listed, all with a status of 'Correcto' (Correct).

Estado	Descripción	Causa	Hora de inicio	Hora de finalización
Correcto	Launching a new EC2 instance: i-0f704d3977aff0439	At 2024-12-02T02:54:28Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 3. At 2024-12-02T02:54:32Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 3.	2024 December 01, 09:54:34 PM -05:00	2024 December 01, 09:54 PM -05:00
Correcto	Launching a new EC2 instance: i-0581cb9b1c3f8c4aa	At 2024-12-02T02:54:28Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 3. At 2024-12-02T02:54:32Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 3.	2024 December 01, 09:54:34 PM -05:00	2024 December 01, 09:54 PM -05:00
Correcto	Launching a new EC2 instance: i-0c8b35e34d6569b1f	At 2024-12-02T02:54:28Z a user request created an AutoScalingGroup changing the desired capacity from 0 to 3. At 2024-12-02T02:54:32Z an instance was started in response to a difference between desired and actual capacity, increasing the capacity from 0 to 3.	2024 December 01, 09:54:33 PM -05:00	2024 December 01, 09:55 PM -05:00

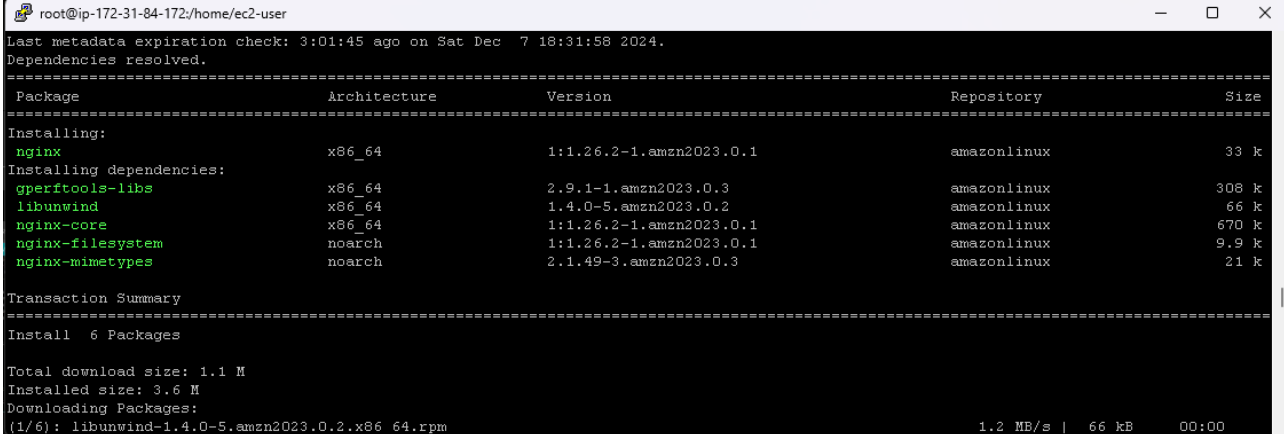
Figura 22 *Actividad del grupo de Auto Scaling*

Video explicativo Segunda entrega:

[Video]. YouTube. <https://youtu.be/ze5rz5I3XiA?si=HOuAyRaEn0FtHzQy>

Instalación de nginx

la captura de pantalla muestra el proceso de instalación de Nginx y sus dependencias en un servidor Linux. Una vez completada la instalación, el servidor estará listo para servir contenido web.

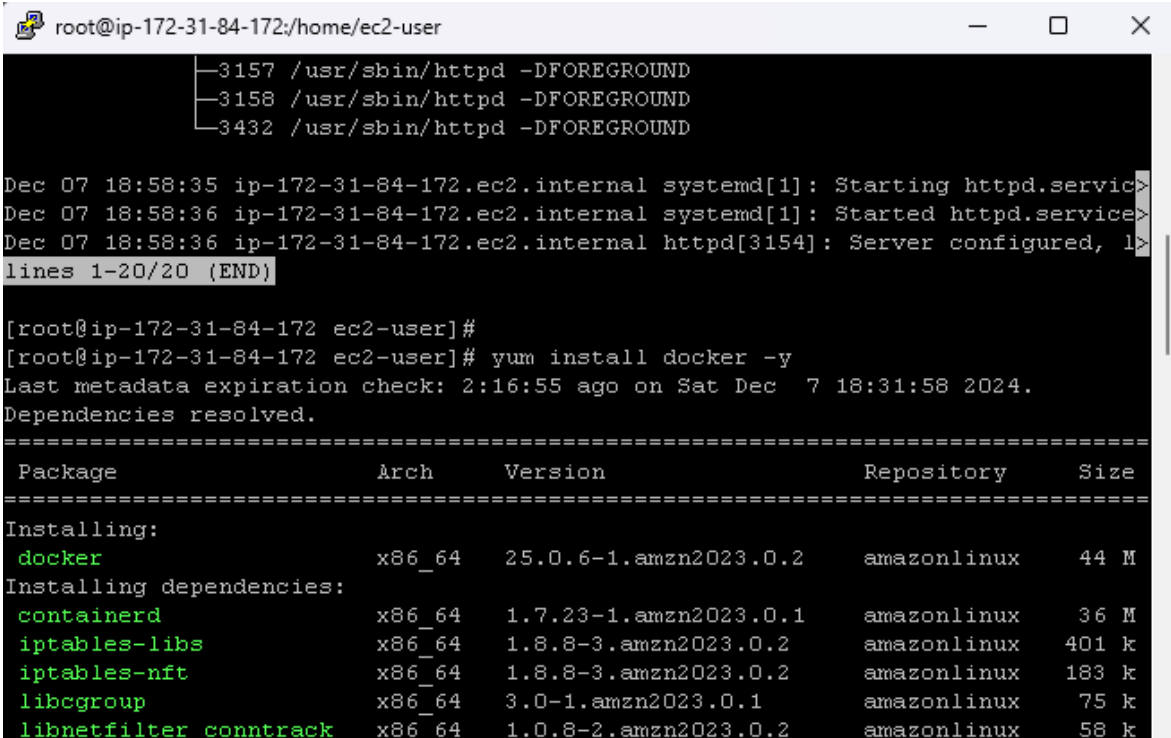


```
root@ip-172-31-84-172:/home/ec2-user
Last metadata expiration check: 3:01:45 ago on Sat Dec 7 18:31:58 2024.
Dependencies resolved.
=====
Package                Architecture          Version                Repository              Size
=====
Installing:
nginx                  x86_64                1:1.26.2-1.amzn2023.0.1  amazonlinux             33 k
Installing dependencies:
gperftools-libs       x86_64                2.9.1-1.amzn2023.0.3    amazonlinux             308 k
libunwind              x86_64                1.4.0-5.amzn2023.0.2    amazonlinux             66 k
nginx-core             x86_64                1:1.26.2-1.amzn2023.0.1  amazonlinux            670 k
nginx-filesystem      noarch                1:1.26.2-1.amzn2023.0.1  amazonlinux            9.9 k
nginx-mimetypes        noarch                2.1.49-3.amzn2023.0.3    amazonlinux            21 k
=====
Transaction Summary
=====
Install 6 Packages

Total download size: 1.1 M
Installed size: 3.6 M
Downloading Packages:
(1/6): libunwind-1.4.0-5.amzn2023.0.2.x86_64.rpm 1.2 MB/s | 66 kB 00:00
```

Figura 23 *Instalación de nginx*

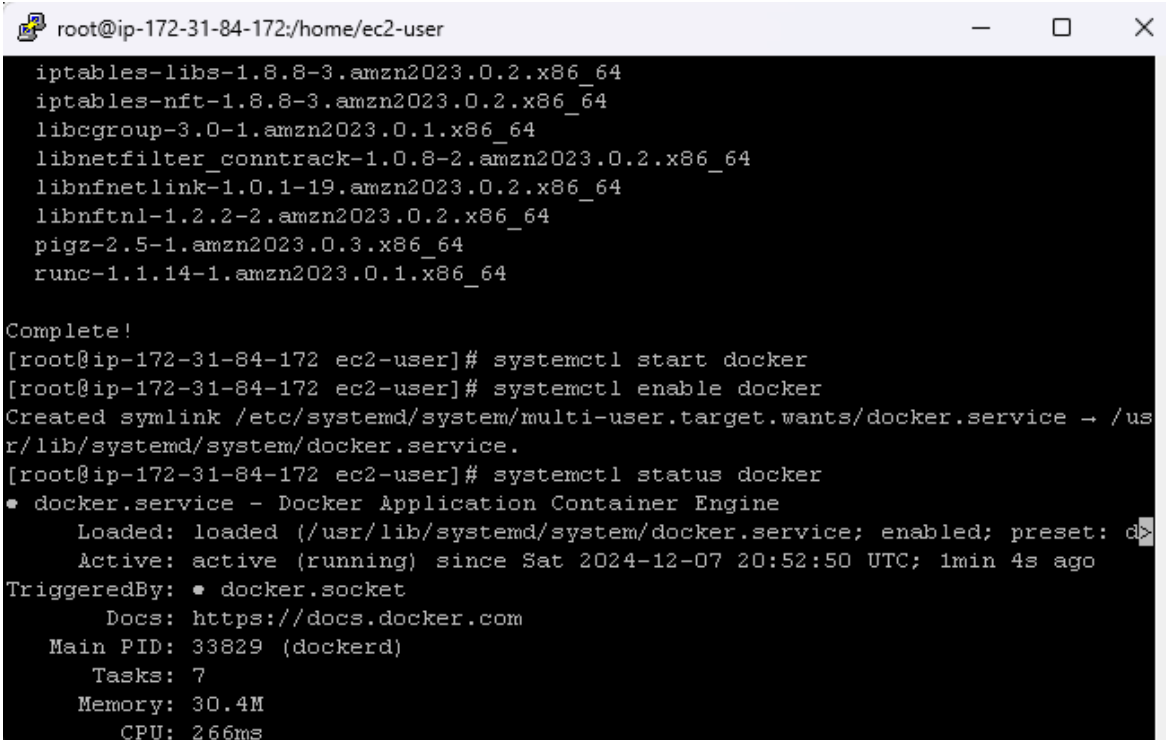
La imagen captura el momento justo antes y durante la instalación de Docker en un servidor donde ya estaba funcionando un servicio HTTP. Esta acción amplía las capacidades del servidor al permitir ejecutar múltiples aplicaciones de forma aislada y eficiente dentro de contenedores Docker.



```
root@ip-172-31-84-172:/home/ec2-user
├─3157 /usr/sbin/httpd -DFOREGROUND
├─3158 /usr/sbin/httpd -DFOREGROUND
└─3432 /usr/sbin/httpd -DFOREGROUND
Dec 07 18:58:35 ip-172-31-84-172.ec2.internal systemd[1]: Starting httpd.service>
Dec 07 18:58:36 ip-172-31-84-172.ec2.internal systemd[1]: Started httpd.service>
Dec 07 18:58:36 ip-172-31-84-172.ec2.internal httpd[3154]: Server configured, l>
lines 1-20/20 (END)
[root@ip-172-31-84-172 ec2-user]#
[root@ip-172-31-84-172 ec2-user]# yum install docker -y
Last metadata expiration check: 2:16:55 ago on Sat Dec 7 18:31:58 2024.
Dependencies resolved.
=====
Package                Arch      Version                               Repository      Size
=====
Installing:
docker                 x86_64    25.0.6-1.amzn2023.0.2                amazonlinux    44 M
Installing dependencies:
containerd             x86_64    1.7.23-1.amzn2023.0.1                amazonlinux    36 M
iptables-libs         x86_64    1.8.8-3.amzn2023.0.2                amazonlinux    401 k
iptables-nft          x86_64    1.8.8-3.amzn2023.0.2                amazonlinux    183 k
libcgroup              x86_64    3.0-1.amzn2023.0.1                  amazonlinux    75 k
libnetfilter_contrack x86_64    1.0.8-2.amzn2023.0.2                amazonlinux    58 k
=====
```

Figura 24 Instalación de Docker en la instancia

La imagen muestra el proceso de instalación de Docker y su configuración para que se inicie automáticamente al arrancar el sistema.

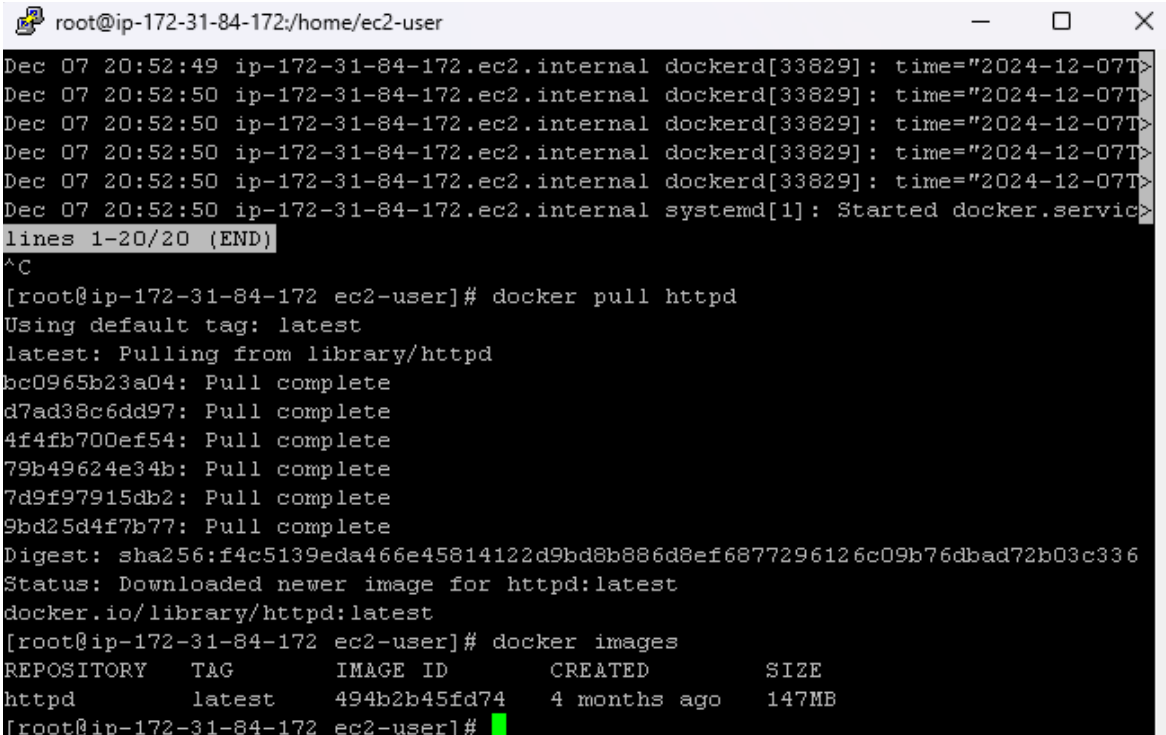


```
root@ip-172-31-84-172:/home/ec2-user
iptables-libs-1.8.8-3.amzn2023.0.2.x86_64
iptables-nft-1.8.8-3.amzn2023.0.2.x86_64
libcgroup-3.0-1.amzn2023.0.1.x86_64
libnetfilter_conntrack-1.0.8-2.amzn2023.0.2.x86_64
libnftnl-1.0.1-19.amzn2023.0.2.x86_64
libnftnl-1.2.2-2.amzn2023.0.2.x86_64
pigz-2.5-1.amzn2023.0.3.x86_64
runc-1.1.14-1.amzn2023.0.1.x86_64

Complete!
[root@ip-172-31-84-172 ec2-user]# systemctl start docker
[root@ip-172-31-84-172 ec2-user]# systemctl enable docker
Created symlink /etc/systemd/system/multi-user.target.wants/docker.service → /usr/lib/systemd/system/docker.service.
[root@ip-172-31-84-172 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; preset: disabled)
   Active: active (running) since Sat 2024-12-07 20:52:50 UTC; 1min 4s ago
 TriggeredBy: ● docker.socket
   Docs: https://docs.docker.com
  Main PID: 33829 (dockerd)
    Tasks: 7
  Memory: 30.4M
    CPU: 266ms
```

Figura 25 Se inicia y se configura para que inicie el servicio al arrancar la instancia

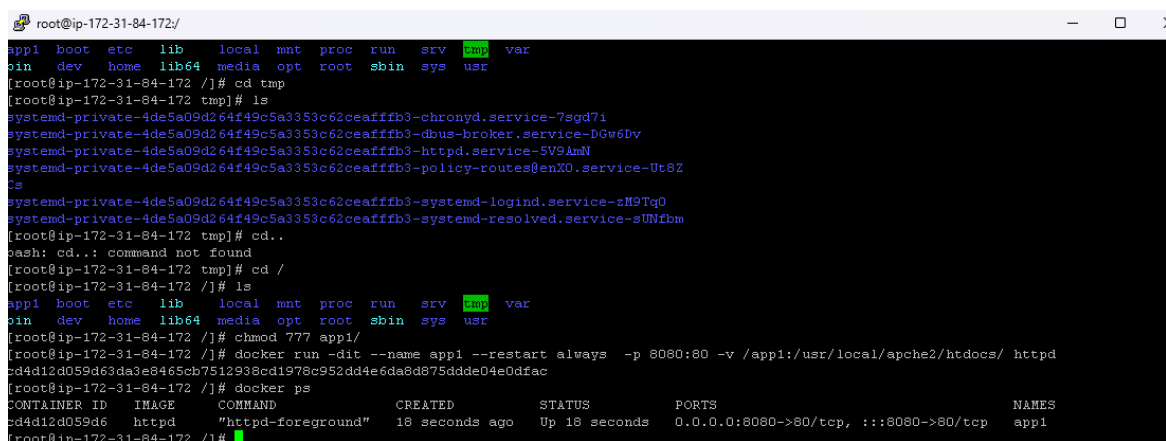
Se ha comprobado que Docker está funcionando correctamente en el sistema y se ha descargado una plantilla preconstruida (la imagen de httpd) que se puede utilizar para crear un servidor web.

A terminal window titled 'root@ip-172-31-84-172:/home/ec2-user' showing system logs and Docker commands. The logs show 'systemd[1]: Started docker.service'. The user then runs 'docker pull httpd', which outputs the pull progress for the 'latest' tag, including a list of layers and a digest. Finally, the user runs 'docker images', which displays a table of local images.

```
root@ip-172-31-84-172:/home/ec2-user
Dec 07 20:52:49 ip-172-31-84-172.ec2.internal dockerd[33829]: time="2024-12-07T20:52:49.123456789Z" level=info msg="Starting daemon"
Dec 07 20:52:50 ip-172-31-84-172.ec2.internal dockerd[33829]: time="2024-12-07T20:52:50.123456789Z" level=info msg="Starting daemon"
Dec 07 20:52:50 ip-172-31-84-172.ec2.internal dockerd[33829]: time="2024-12-07T20:52:50.123456789Z" level=info msg="Starting daemon"
Dec 07 20:52:50 ip-172-31-84-172.ec2.internal dockerd[33829]: time="2024-12-07T20:52:50.123456789Z" level=info msg="Starting daemon"
Dec 07 20:52:50 ip-172-31-84-172.ec2.internal dockerd[33829]: time="2024-12-07T20:52:50.123456789Z" level=info msg="Starting daemon"
Dec 07 20:52:50 ip-172-31-84-172.ec2.internal systemd[1]: Started docker.service.
lines 1-20/20 (END)
^C
[root@ip-172-31-84-172 ec2-user]# docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
bc0965b23a04: Pull complete
d7ad38c6dd97: Pull complete
4f4fb700ef54: Pull complete
79b49624e34b: Pull complete
7d9f97915db2: Pull complete
9bd25d4f7b77: Pull complete
Digest: sha256:f4c5139eda466e45814122d9bd8b886d8ef6877296126c09b76dbad72b03c336
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
[root@ip-172-31-84-172 ec2-user]# docker images
REPOSITORY   TAG       IMAGE ID      CREATED        SIZE
httpd        latest   494b2b45fd74  4 months ago  147MB
[root@ip-172-31-84-172 ec2-user]#
```

Figura 26 Se descarga httpd

La imagen muestra el proceso de creación de un contenedor Docker para ejecutar un servidor web Apache. Se configura el contenedor para que se ejecute en segundo plano, se le asigna un nombre, se mapean los puertos y se monta un volumen para compartir archivos entre el host y el contenedor. Este proceso se repite para crear contenedores adicionales con nombres "app2" y "app3", con configuraciones similares.



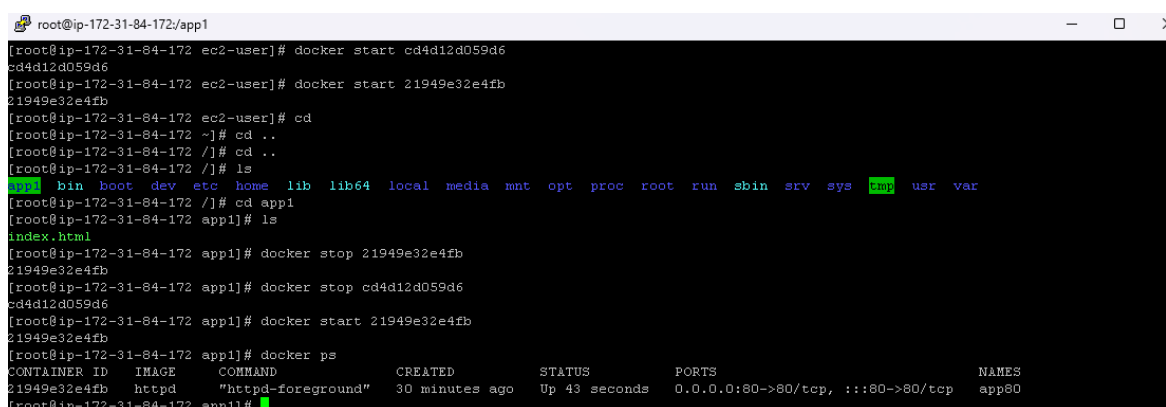
```

root@ip-172-31-84-172:/
app1 boot etc lib local mnt proc run srv tmp var
bin dev home lib64 media opt root sbin sys usr
[root@ip-172-31-84-172 /]# cd tmp
[root@ip-172-31-84-172 tmp]# ls
systemd-private-4de5a09d264f49c5a3353c62ceafffb3-chronyd.service-7sgd71
systemd-private-4de5a09d264f49c5a3353c62ceafffb3-dbus-broker.service-DGw6bv
systemd-private-4de5a09d264f49c5a3353c62ceafffb3-httpd.service-5V9AmN
systemd-private-4de5a09d264f49c5a3353c62ceafffb3-policy-routes@enX0.service-Ut8Z
ls
systemd-private-4de5a09d264f49c5a3353c62ceafffb3-systemd-logind.service-zM9Tq0
systemd-private-4de5a09d264f49c5a3353c62ceafffb3-systemd-resolved.service-SUNf5m
[root@ip-172-31-84-172 tmp]# cd..
bash: cd.: command not found
[root@ip-172-31-84-172 tmp]# cd /
[root@ip-172-31-84-172 /]# ls
app1 boot etc lib local mnt proc run srv tmp var
bin dev home lib64 media opt root sbin sys usr
[root@ip-172-31-84-172 /]# chmod 777 app1/
[root@ip-172-31-84-172 /]# docker run -dit --name app1 --restart always -p 8080:80 -v /app1:/usr/local/apache2/htdocs/ httpd
cd4d12d059d63da3e8465cb7512938cd1978c952dd4e6da8d875ddde04e0dfac
[root@ip-172-31-84-172 /]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
cd4d12d059d6        httpd               "httpd-foreground"      18 seconds ago     Up 18 seconds      0.0.0.0:8080->80/tcp, :::8080->80/tcp  app1

```

Figura 27 Creación y configuración de la carpeta *app1* y contenedor del *httpd*, esto mismo se realiza para *app2* y *app3*.

La imagen muestra la ejecución de un comando Docker para iniciar un contenedor en modo desacoplado (en segundo plano). El contenedor se basa en la imagen *my-image* y expone el puerto 80 del contenedor al puerto 80 del host. Esto significa que se está iniciando un servicio (un servidor web) que estará escuchando en el puerto 80 de la máquina host.



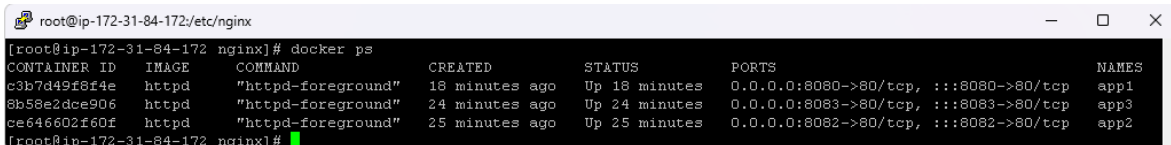
```

root@ip-172-31-84-172:/app1
[root@ip-172-31-84-172 ec2-user]# docker start cd4d12d059d6
cd4d12d059d6
[root@ip-172-31-84-172 ec2-user]# docker start 21949e32e4fb
21949e32e4fb
[root@ip-172-31-84-172 ec2-user]# cd
[root@ip-172-31-84-172 ~]# cd ..
[root@ip-172-31-84-172 /]# cd ..
[root@ip-172-31-84-172 /]# ls
tmp bin boot dev etc home lib lib64 local media mnt opt proc root run sbin srv sys tmp usr var
[root@ip-172-31-84-172 /]# cd app1
[root@ip-172-31-84-172 app1]# ls
index.html
[root@ip-172-31-84-172 app1]# docker stop 21949e32e4fb
21949e32e4fb
[root@ip-172-31-84-172 app1]# docker stop cd4d12d059d6
cd4d12d059d6
[root@ip-172-31-84-172 app1]# docker start 21949e32e4fb
21949e32e4fb
[root@ip-172-31-84-172 app1]# docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS                               NAMES
21949e32e4fb        httpd               "httpd-foreground"      30 minutes ago     Up 43 seconds      0.0.0.0:80->80/tcp, :::80->80/tcp  app80

```

Figura 28 Contenedor creado y funcional

La imagen muestra que se han creado y están en ejecución tres contenedores Docker, cada uno de los cuales actúa como un servidor web independiente. Estos contenedores se han configurado para servir contenido estático desde un archivo `index.html` ubicado en una carpeta específica dentro del contenedor. La configuración de los puertos permite acceder a cada servidor web a través de una dirección IP y puerto diferentes.



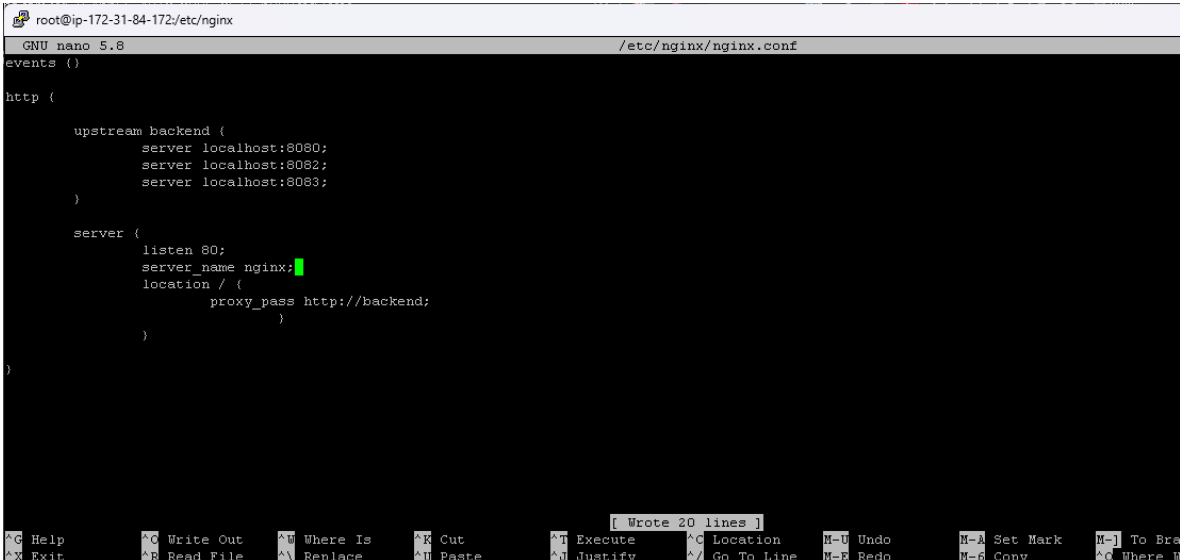
```

root@ip-172-31-84-172/etc/nginx# docker ps
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS        PORTS                               NAMES
c3b7d49f8f4e   httpd    "httpd-foreground"      18 minutes ago Up 18 minutes 0.0.0.0:8080->80/tcp, :::8080->80/tcp   app1
8b58e2dce906   httpd    "httpd-foreground"      24 minutes ago Up 24 minutes 0.0.0.0:8083->80/tcp, :::8083->80/tcp   app3
ce646602f60f   httpd    "httpd-foreground"      25 minutes ago Up 25 minutes 0.0.0.0:8082->80/tcp, :::8082->80/tcp   app2

```

Figura 29 Se crean los tres contenedores en el archivo `index.html` con su respectiva ruta

La configuración de Nginx que se muestra en la imagen ha sido diseñada para proporcionar un equilibrio de carga y alta disponibilidad al distribuir el tráfico entre múltiples servidores. Los cambios realizados permiten una gestión más flexible y automatizada del entorno, adaptándose a las dinámicas de un entorno de contenedores.



```

GNU nano 5.8 /etc/nginx/nginx.conf
events {}

http {
    upstream backend {
        server localhost:8080;
        server localhost:8082;
        server localhost:8083;
    }

    server {
        listen 80;
        server_name nginx;
        location / {
            proxy_pass http://backend;
        }
    }
}

```

Figura 30 Configuración de `nginx.conf` para que haga automáticamente los cambios entre los contenedores.

La imagen muestra el resultado final de la configuración de Nginx: la capacidad de servir múltiples aplicaciones desde una única dirección IP. Esto se logra gracias al equilibrio de carga y la configuración de múltiples ubicaciones en el archivo de configuración de Nginx. Se realizan las 3 pruebas.



Figura 31 *Prueba App1*

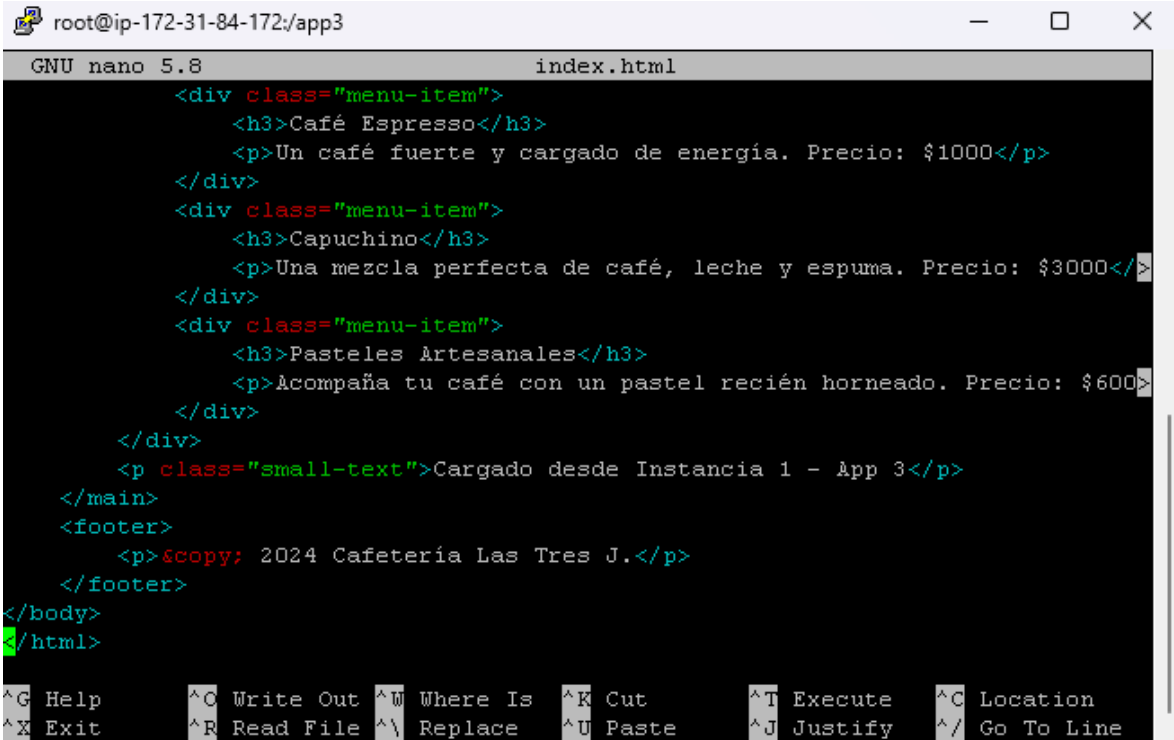


Figura 32 *Prueba App2*



Figura 33 *Prueba App3*

El código HTML muestra una forma sencilla pero efectiva de personalizar el contenido servido por diferentes instancias de una aplicación, lo que facilita la gestión y el monitoreo de un entorno de microservicios. Ya con los contenedores funcionales se realiza modificación del archivo `index.html` para así realizar una simulación de una página de negocio; dentro de la página se dejó un texto que indica desde que instancia se está cargando la página y también desde que contenedor.



```
root@ip-172-31-84-172:/app3
GNU nano 5.8 index.html
<div class="menu-item">
  <h3>Café Espresso</h3>
  <p>Un café fuerte y cargado de energía. Precio: $1000</p>
</div>
<div class="menu-item">
  <h3>Capuchino</h3>
  <p>Una mezcla perfecta de café, leche y espuma. Precio: $3000</p>
</div>
<div class="menu-item">
  <h3>Pasteles Artesanales</h3>
  <p>Acompaña tu café con un pastel recién horneado. Precio: $600</p>
</div>
</div>
<p class="small-text">Cargado desde Instancia 1 - App 3</p>
</main>
<footer>
  <p>©copy; 2024 Cafeteria Las Tres J.</p>
</footer>
</body>
</html>
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

Figura 34 Código HTML para visualizar la página web organizada

La imagen ilustra de manera clara cómo la combinación de Nginx y contenedores permite crear aplicaciones web escalables, flexibles y personalizables.

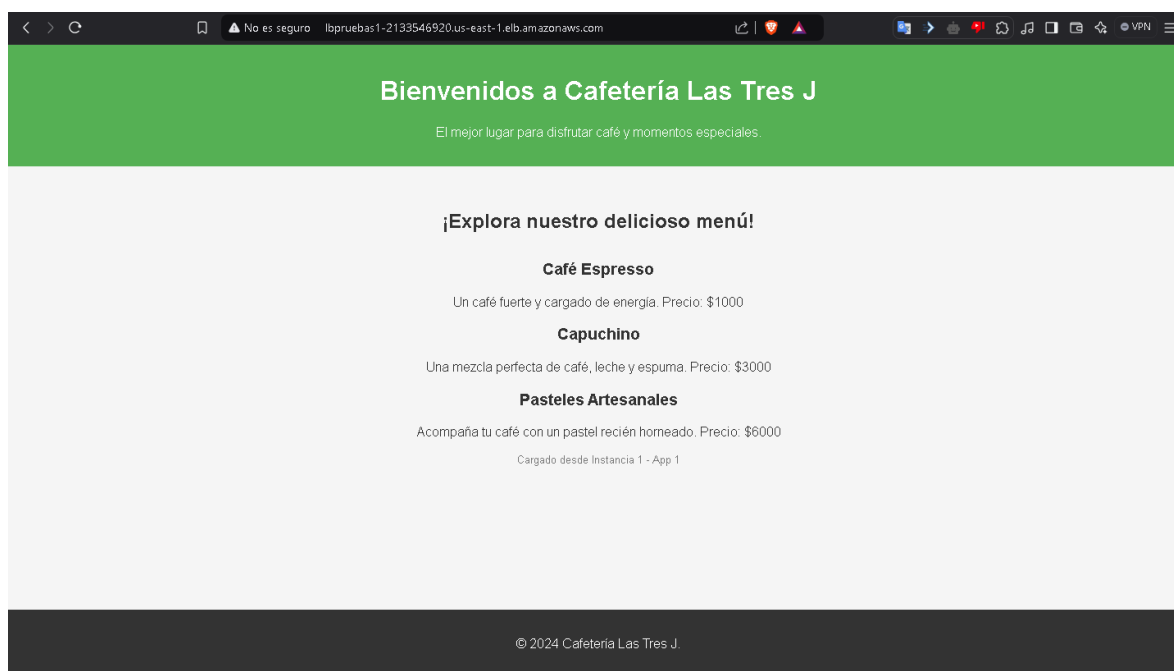


Figura 35: Resultado de la página web cargando e indicando desde instancia 1- App 1.

Nota: Cabe aclarar que toda la configuración se realizó en las dos instancias.

Video explicativo entrega final:

[Video]. YouTube. <https://youtu.be/gABNNISsbco?si=AnJOHCFU-SVHasoD>

Conclusiones

El trabajo realizado sobre la implementación de servicios de Amazon Web Services (AWS) tiene como objetivo principal demostrar cómo las tecnologías de computación en la nube pueden optimizar la infraestructura tecnológica de una empresa, asegurando una operación continua y eficiente. Se concluye que AWS es una herramienta fundamental para implementar servicios de alta disponibilidad y redundancia, garantizando que, si una instancia falla, otras instancias pueden continuar operando sin interrupciones. De este modo, se asegura que la aplicación siga funcionando y los servicios se mantengan activos, evitando caídas que puedan afectar a los usuarios.

La combinación de instancias EC2, balanceadores de carga (ELB) y políticas de Auto Scaling permite distribuir el tráfico entre múltiples servidores, adaptándose automáticamente a los picos de demanda y reemplazando instancias dañadas de manera transparente. La implementación de contenedores Docker y el balanceo de carga con Nginx facilita además el despliegue y mantenimiento de múltiples aplicaciones independientes dentro de una misma infraestructura, mejorando la eficiencia y flexibilidad operativa.

Esta solución es especialmente útil para empresas que necesitan garantizar la continuidad del servicio y reducir el riesgo de fallos en sus sistemas. Además, elimina la dependencia de una sola infraestructura física, reduciendo costos operativos y permitiendo escalar recursos de manera dinámica según las necesidades del negocio.

En conclusión, el trabajo sirve como una guía práctica para implementar servicios en la nube que aseguran una infraestructura redundante, escalable y de alta disponibilidad, lo que permite a las empresas, independientemente de su tamaño, ofrecer servicios digitales robustos y confiables sin interrupciones.

Referencias

OpenAI. (2024). *ChatGPT: Un modelo de lenguaje de inteligencia artificial*.
<https://openai.com/chatgpt>

Google. (2024). *Gemini: Una plataforma para modelos de IA y generativas*.
<https://gemini.google.com/app?hl=es>

Amazon Web Services, Inc. (2024). *Servicios de computación en la nube y soluciones*.
<https://aws.amazon.com>