TRABAJO DE GRADO
Opción Seminario-Diplomado.

DE LO FÍSICO A LA NUBE CON AWS

Corporación Universitaria Remington.
Facultad de Ingenierías

Pregrado en Ingeniería de Sistemas,
Postgrado Especialización en Seguridad de la Información

Presentado Por:

Anderson David García Perez (Postgrado)
Andres Londoño Orrego (Pregrado)
Dairo Luis Barrera Jiménez (Pregrado)

Presentado a:

Juan Pablo Berrio López (Docente)

Opción de Trabajo de grado Seminario.

2025.

Tabla de Contenidos

Resumen

En este proyecto se podrán en práctica los conceptos aprendidos durante el transcurso del seminario donde trabajaremos con instancias las cuales crearemos de manera manual y posteriormente de manera automática(AMI), Adicional, aprenderemos sobre la importancia de la escalabilidad ya que es un factor determinante en el proceso de nuestras compañías donde aplicaremos autoScaling, proxy reverso, grupos de seguridad, etc. También veremos entornos de contenedores utilizando Dockers, donde se crearán pruebas con una o más páginas web a través de contenedores que ejecutan en un servidor Apache (httpd) y estos mismos se podrán administrar bajo un proxy reverso atreves de Nginx. Este proyecto se desarrollará atreves de instancia de AWS, en este proyecto daremos un manejo dinámico y escalable de la estructura web que nos presenta la computación en la nube.

Palabras claves

Computación en la nube
Virtualización con AWS
Docker
Infraestructura Computacional
Orquestación de Servicios

Marco conceptual y contextual

Para conceptualizar, las **instancias** es un modelo computacional, implementado por el que en adelante lo llamaremos “AWS” por su nombre Amazon Web Servicies, este modelo computacional en la nube hace que a sus clientes les sea llamativo dicho modelo ya que su relación costo beneficio es muy grande, por su facilidad de uso y amplio paquete de servicios en la nube, estas instancias “Maquinas o servidores” se pueden implementar de manera automática para que lleven a cabo procesos stand alone de manera independiente con muy poca intervención humana, para esto se usa el termino AMI lo cual significa que se hacen de manera automática. Esto agiliza los procesos y pone fin a los servidores de manera física en las empresas los cuales corren el riesgo de daños físicos, apagones, caída de internet, etc.
Para que dichos sistemas sean competentes se usa el termino AutoScaling lo cual significa un escalado automático el cual consiste en observar las necesidades del cliente, controlar y vigilar para poder así tomar las decisiones dado el caso, si es necesario generar un aumento en la capacidad de los servidores o incluso crear una instancia nueva la cual pueda ayudar a suplir la necesidad mejorando el rendimiento y las respuestas.

Dockerizacion es un nuevo modelo de código abierto que se utiliza para empaquetar y correr aplicaciones sin importar el tipo de maquina ya que corre en contenedores, los cuales distribuyen la carga según la necesidad del cliente, es decir, cada usuario ingresara de manera normal al aplicativo y el Docker es el encargado de saber qué tipo y para donde va la petición que el usuario le esta asignando, esto permite un entorno dinámico y fluido.

Título 1

Sub-Título 1.1.

Desarrollo e implementación del aprendizaje

Línea de tiempo

A Continuación trataremos de dar una visión clara sobre el termino línea de tiempo en la virtualización y la computación en la Nube:
Pudiésemos decir que esto se remonta a la década de los 60.
Donde Surge por primera vez el concepto en informática de time-sharing o tiempo compartido en las computadoras centrales o mainframes, computadoras de alto rendimiento utilizadas para el procesamiento y almacenaje de datos, maximizando el uso de recursos.

En los años 60. IBM fue una de las empresas más innovadoras en este aspecto de mainframes por lo cual compañías como bancos, grandes multinacionales, agencias de viaje comenzaron a utilizar este tipo de sistema novedoso para poder cumplir con el procesamiento de grandes volúmenes de datos y la ejecución de programas críticos.

En la década de los 70. IBM desarrollaría la tecnología de la virtualización de su sistema VM “Virtual Machine”, permitiendo marcar un hito en la computación ya que permitiría la ejecución de múltiples sistemas operativos en una misma máquina, algo que para esa fecha era inimaginable para le gente del común. IBM encabeza la tecnología con un revolucionario invento como lo fue la virtualización, esto se benefició gran parte del sector financiero y compañías las cuales tenían problemas por su demanda y poca eficiencia en los procesos.

En los 80, se vuelve muy común y popular utilizar el concepto de cliente-servidor, lo que conllevaría a que la virtualización perdiera relevancia debido al enfoque en servidores físicos, en esta época comenzó el auge de las redes locales LAN en lo cual IBM con sus tipos de Software ayudarían a optimizar procesos y hacer de este tipo de estructura algo novedoso y prometedor para la época facilitando el uso y una fácil adaptabilidad según el requerimiento de la empresa en la cual se implementaba.

Para los 90, Surgiría VMware (1998), que introduciría la virtualización en sistemas con arquitectura de x86 o 32 bits como se le conoce, permitiendo la creación de máquinas virtuales en hardware comunes y corriente atrás quedaría el tema de que la virtualización sólo se podía en máquinas de grandes superficies ya que con este nuevo concepto se pudiese virtualizar maquinas en equipos de hogares y oficinas, este nuevo sistema revolucionario permitió que la tecnología no fuera algo exclusivamente para las empresas, permitió que las PYMES y las personas del común pudieran tener acceso lo

cual redujo considerablemente los costos de las tecnologías y haciendo de la tecnología algo de todos y no de unos pocos.

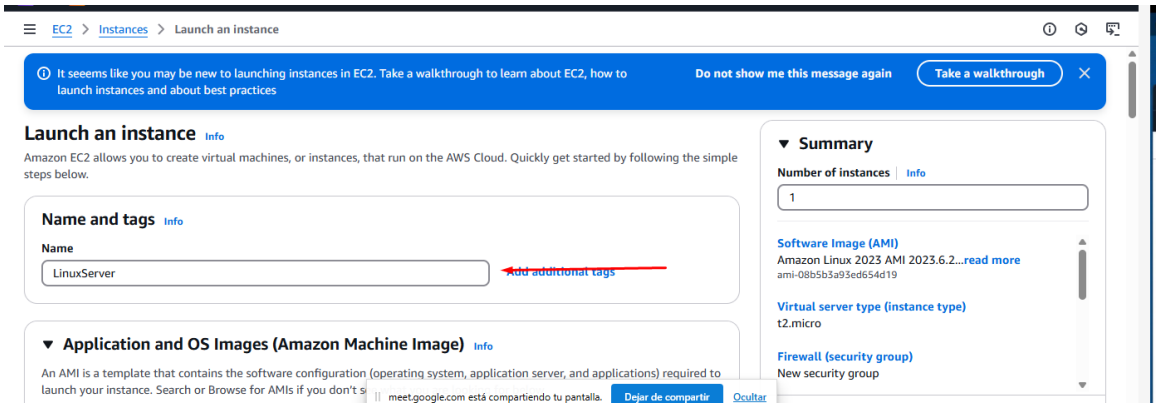
En la década del 2000, surge Amazon con su revolucionario AWS en 2006 con S3 y EC2, introduciendo el concepto de computación en la nube, Google y Microsoft no se quedarían a tras ya que inician sus servicios en la nube. Google con su Google App Engine en el 2008 y Microsoft con su ya conocido Azure en el 2010.
Para este mismo periodo se estandarizan las tecnologías de virtualización como lo son: Xen, KVM y Hyper-V, a principios de la década Netflix era una empresa pequeña la cual tenía su servicio de alquiler de DVD cada vez que su público comenzaba a aumentar se dieron cuenta que experimentar con el streaming era un reto importante pero que podría traerle una escalabilidad significativa ya que podrían manejar picos masivos, también almacenar grandes volúmenes de datos en este caso videos.

Para el año 2010, contenedorización se vuelve popular con Docker en el 2013 y Kubernetes en el 2015, optimizando el uso de la nube de maneras muy eficientes, en este mismo periodo aparecen modelos como SaaS, PaaS e IaaS, que le darían un gran impulso al crecimiento de la nube.

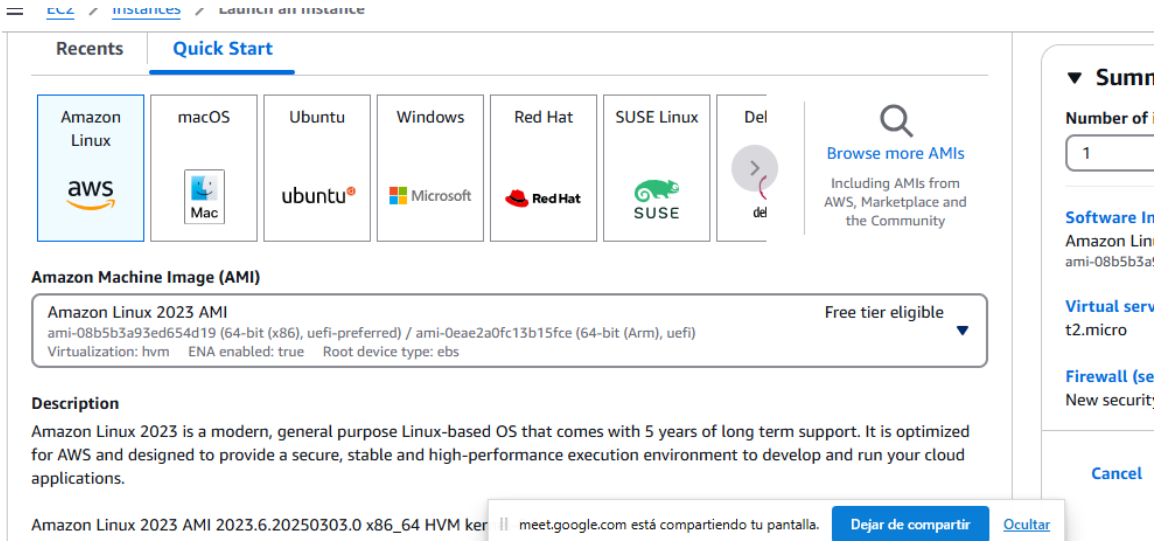
Ya para el 2020. la computación en la nube se integra con IA, Big Data y Edge Computing, alcanzando niveles de automatización y escalabilidad sin precedentes e inimaginables.

Como un aparte trataremos de crear un paso a paso para realizar la correcta configuración para realizar el **AutoScaling** utilizado para crear un balanceo automático de los recursos informáticos:

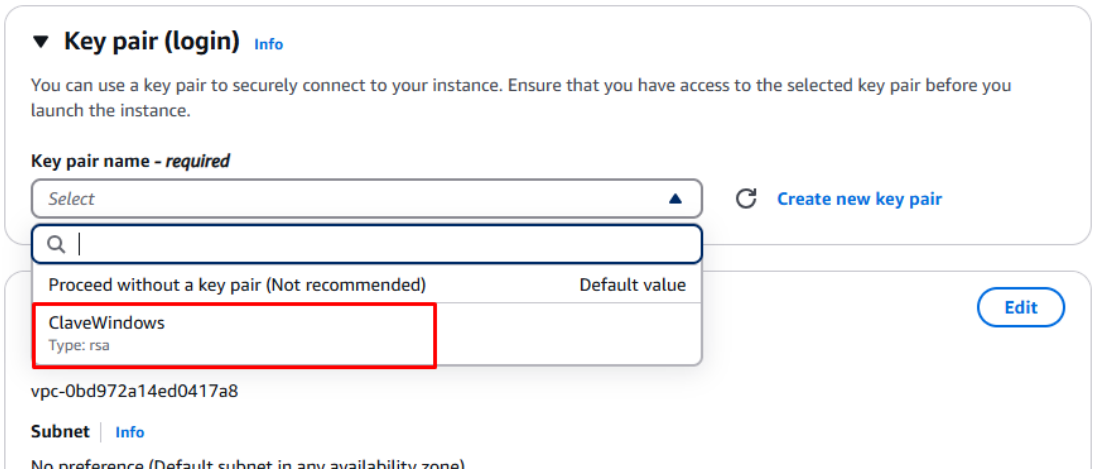
Como primer paso, crearemos una instancia para el server de Amazon Linux ya que será nuestro OS seleccionado, le asignaremos el nombre que deseemos ya que este será el indicador de nuestra instancia.



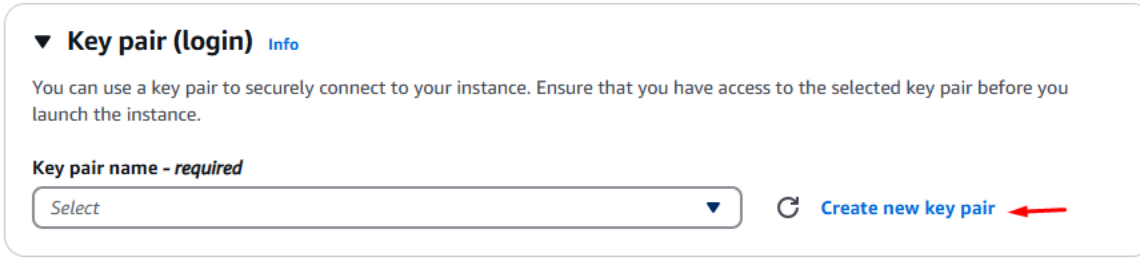
Seleccionamos el Sistema operativo, Amazon Linux suele tener más soporte ya que es creado por el mismo Amazon:



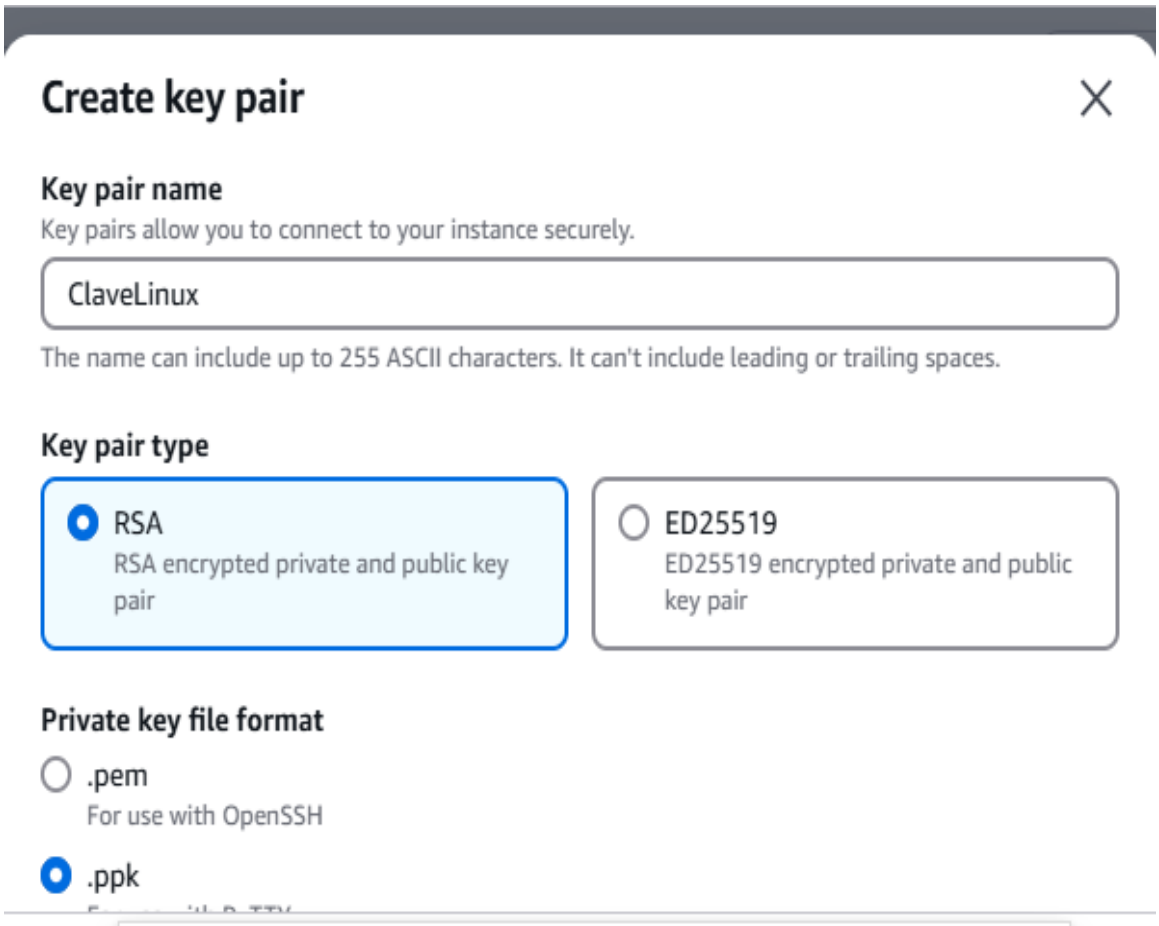
Acá visualizamos las claves que tenemos inscritas o ya realizadas.
Cabe resaltar que esta no se puede utilizar ya que fue creada para un sistema operativo Windows. Por lo cual debemos realizar una nueva .



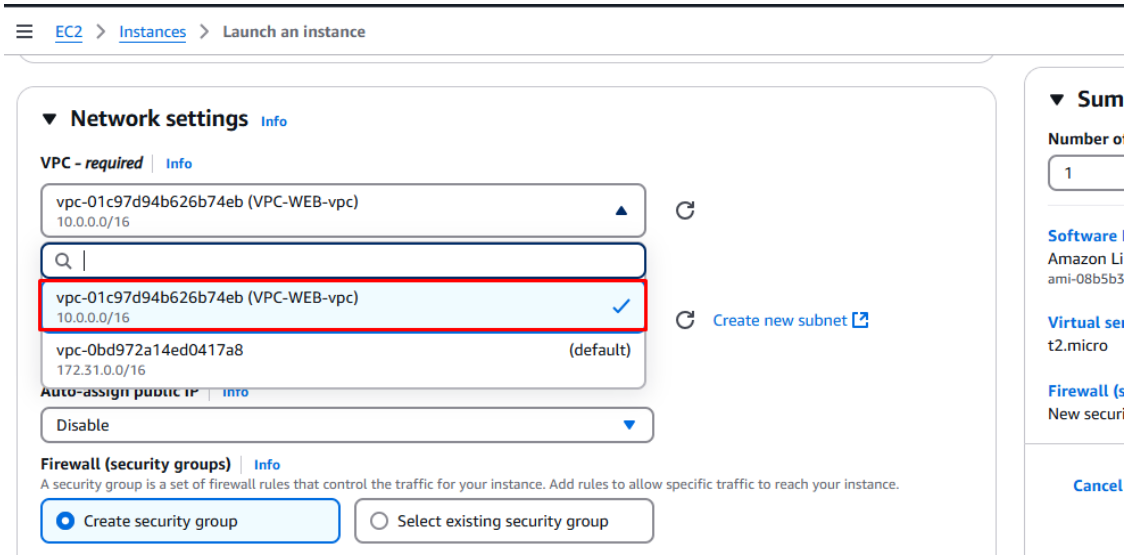
Crearemos una nueva Clave para nuestro server Linux de la siguiente manera:



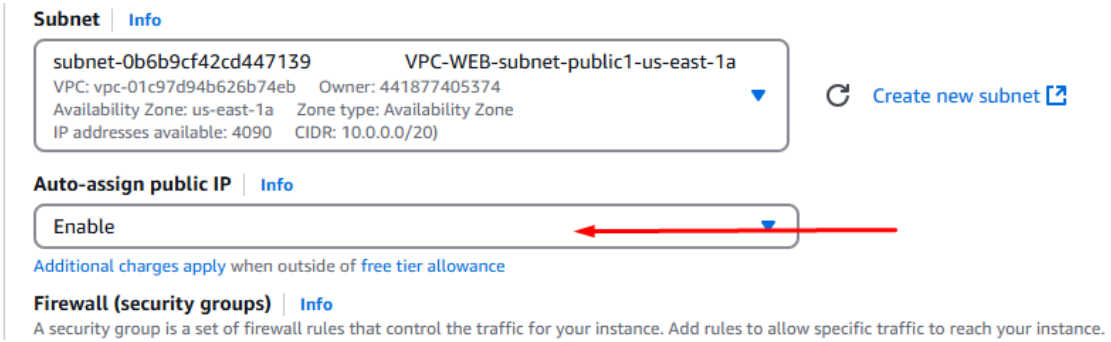
Seleccionaremos la que tenemos resaltada “RSA” este es un modelo de encriptación exclusivo para Linux:



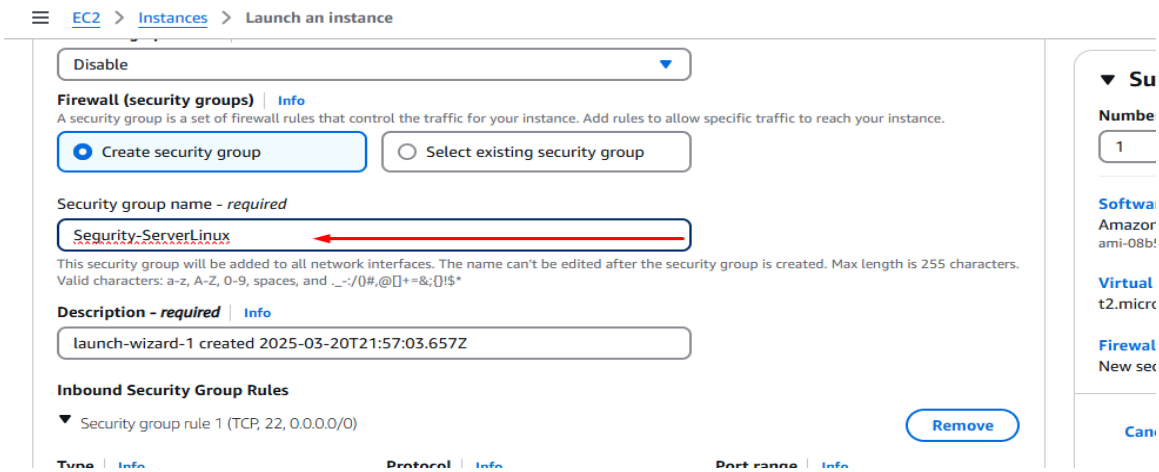
Seleccionaremos el VPC (red privada virtual) creada por nosotros mismo ya que viene una por defecto, y para este caso necesitamos uno creado por nosotros:



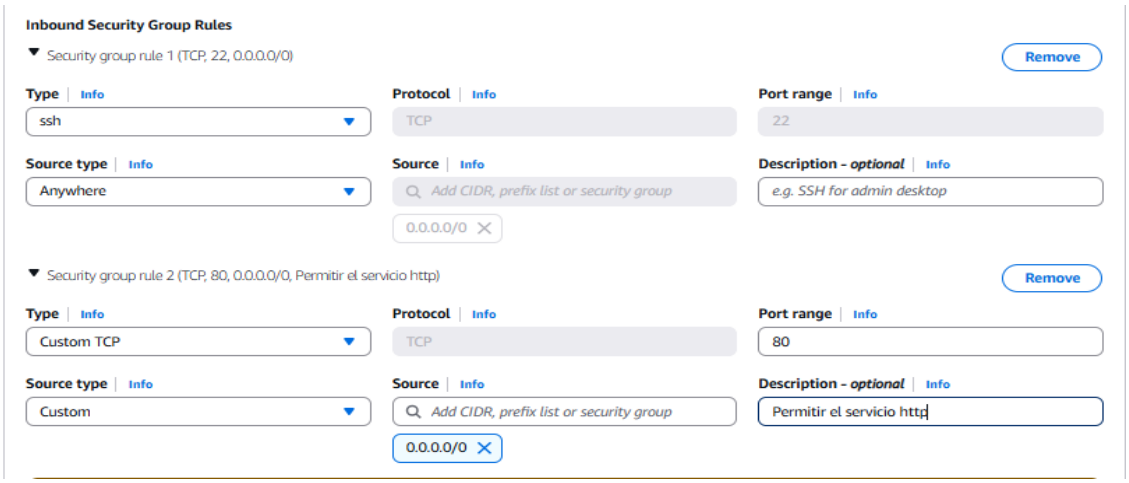
Necesitaremos una ip publica activada para que AWS no nos cree de manera automática y así poder seguir sin afectar el proceso.



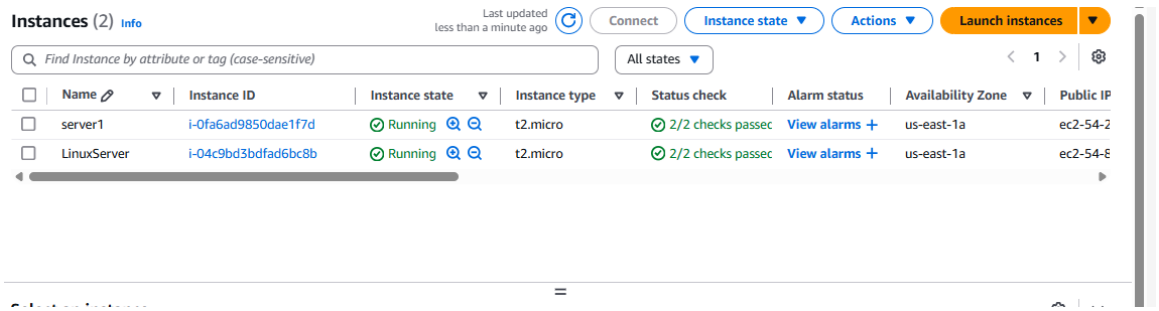
Crearemos un grupo de seguridad para nuestra instancia, le daremos un nombre como se evidencia en la siguiente imagen.



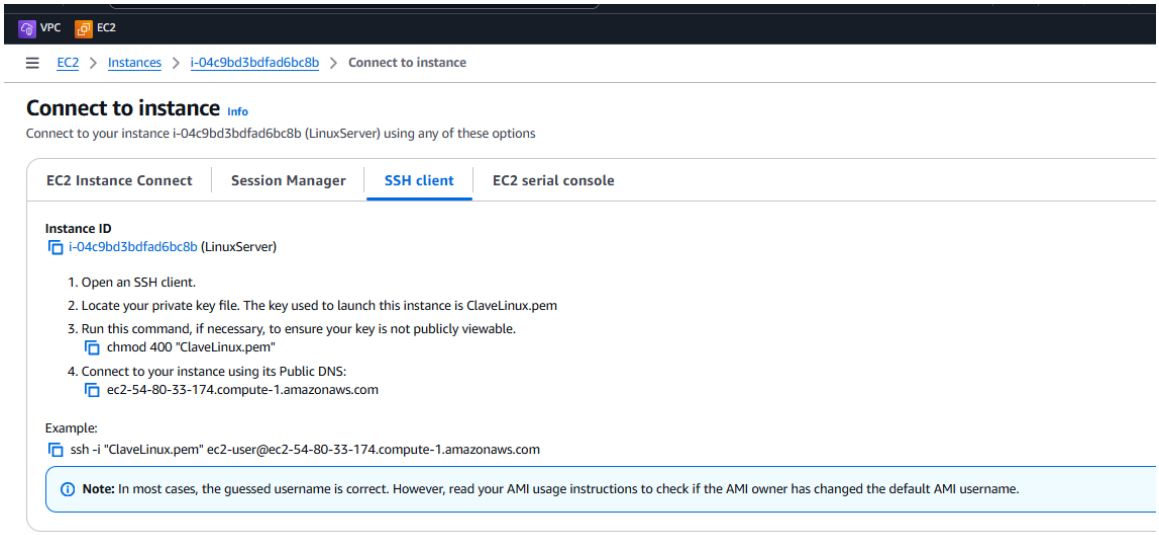
Crearemos una regla para permitir el tráfico por el puerto 80 de la siguiente manera:



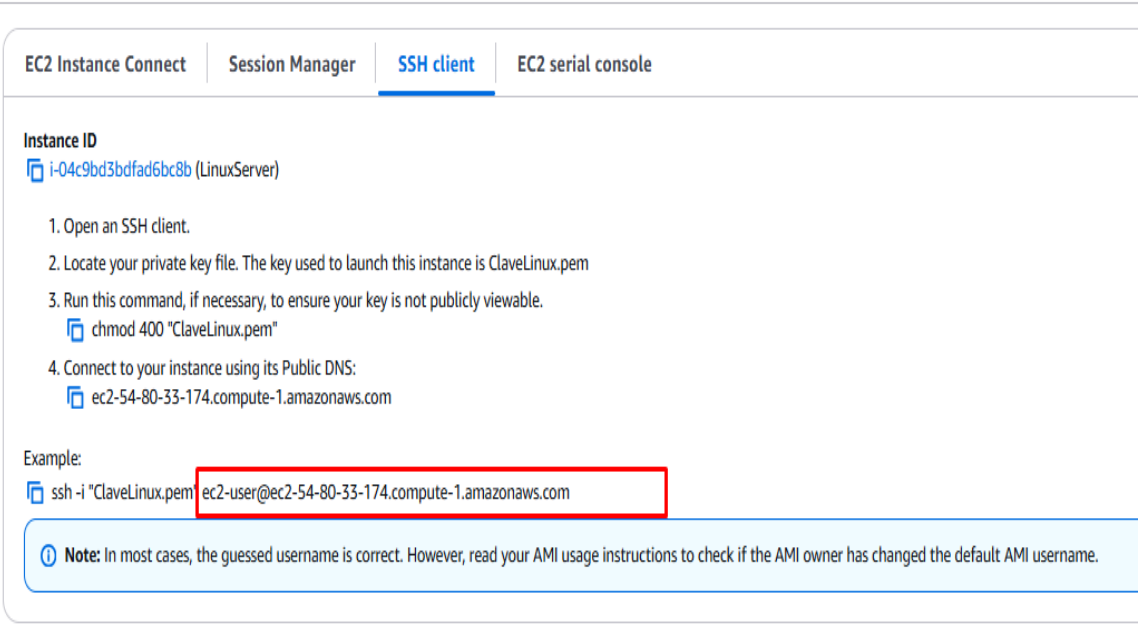
Validaremos si la instancia se creó correctamente, si todo va bien debería de verse reflejado igual que en la imagen:



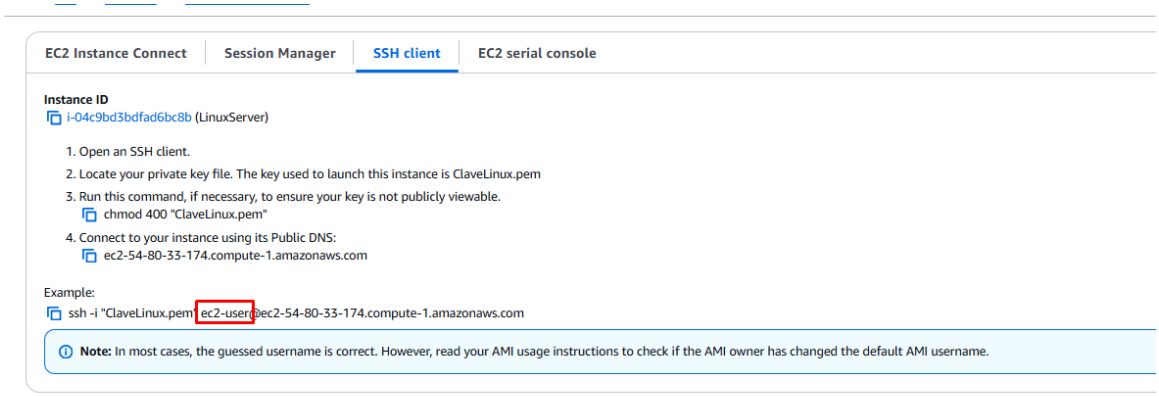
Para conectarnos a nuestra instancia y comenzar a trabajar con ella utilizáremos el apartado SSH client como se muestra en la imagen, previo a esto podés descargar un software que nos permita realizar conexiones con clientes SSH.



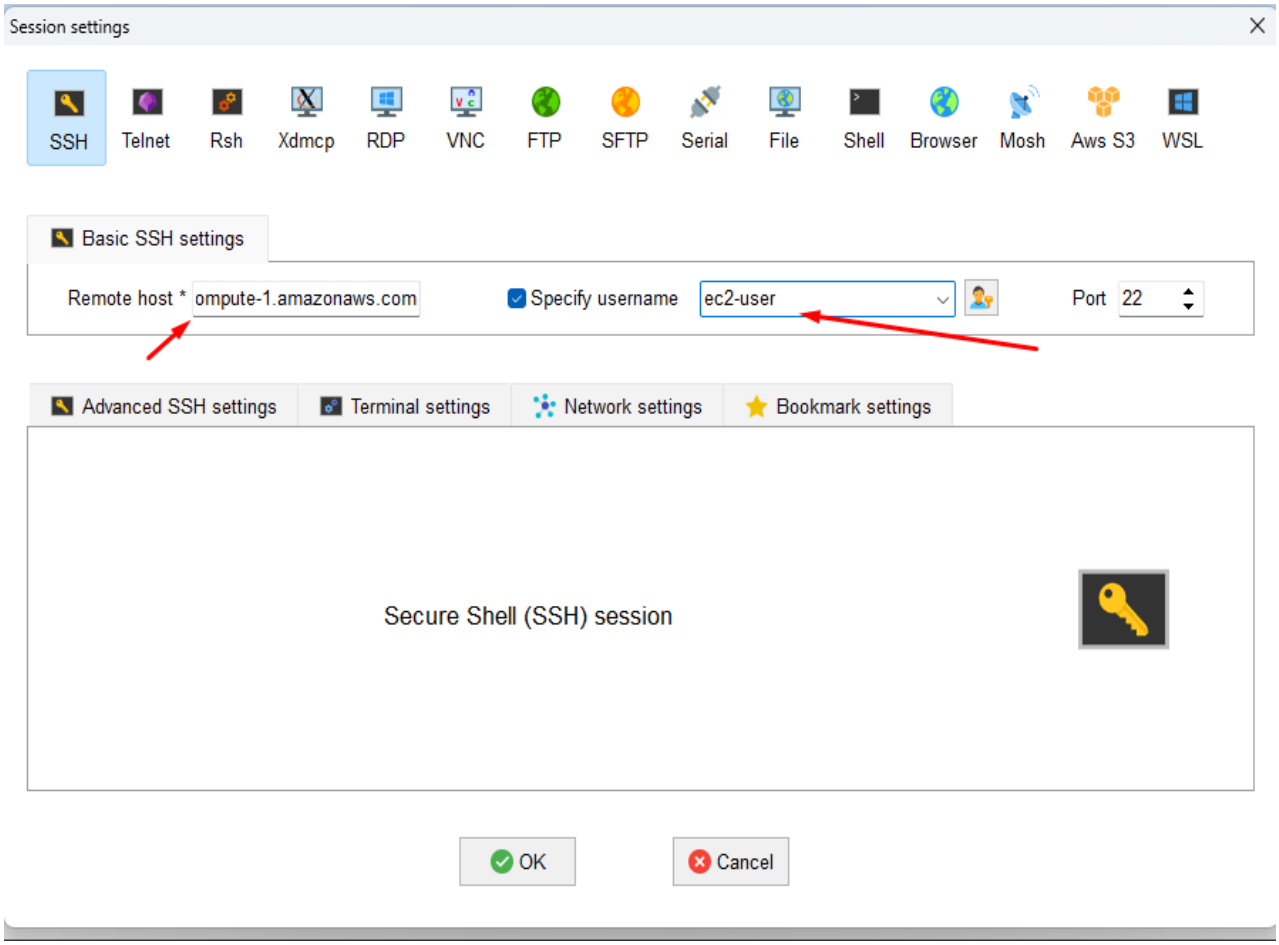
Después de descargar el SSH Client , nuestro Remote host ya estaría operando y seria el siguiente:



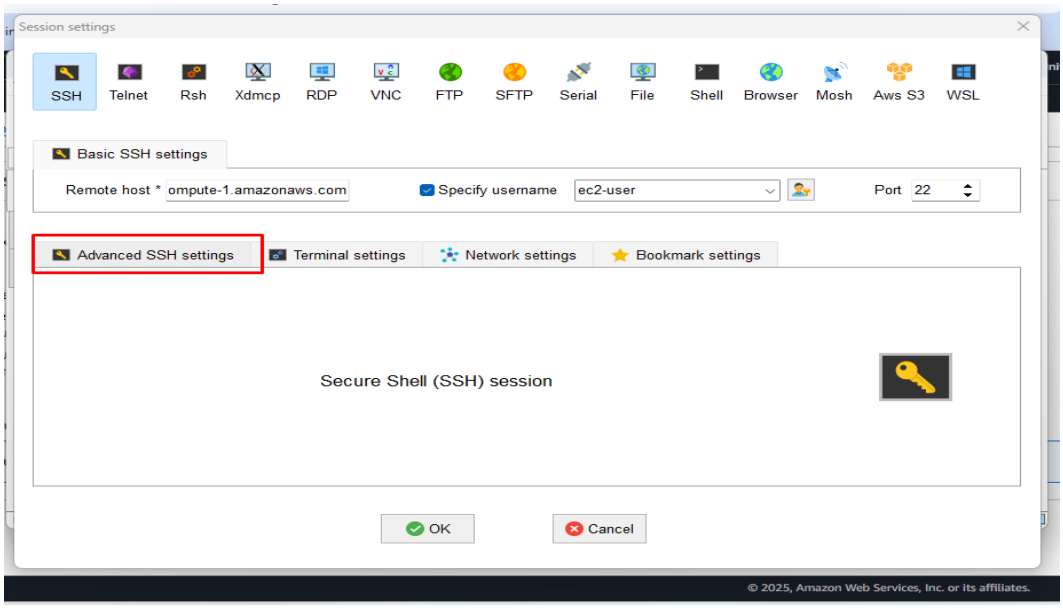
Nuestro user (usuario) sería el siguiente:



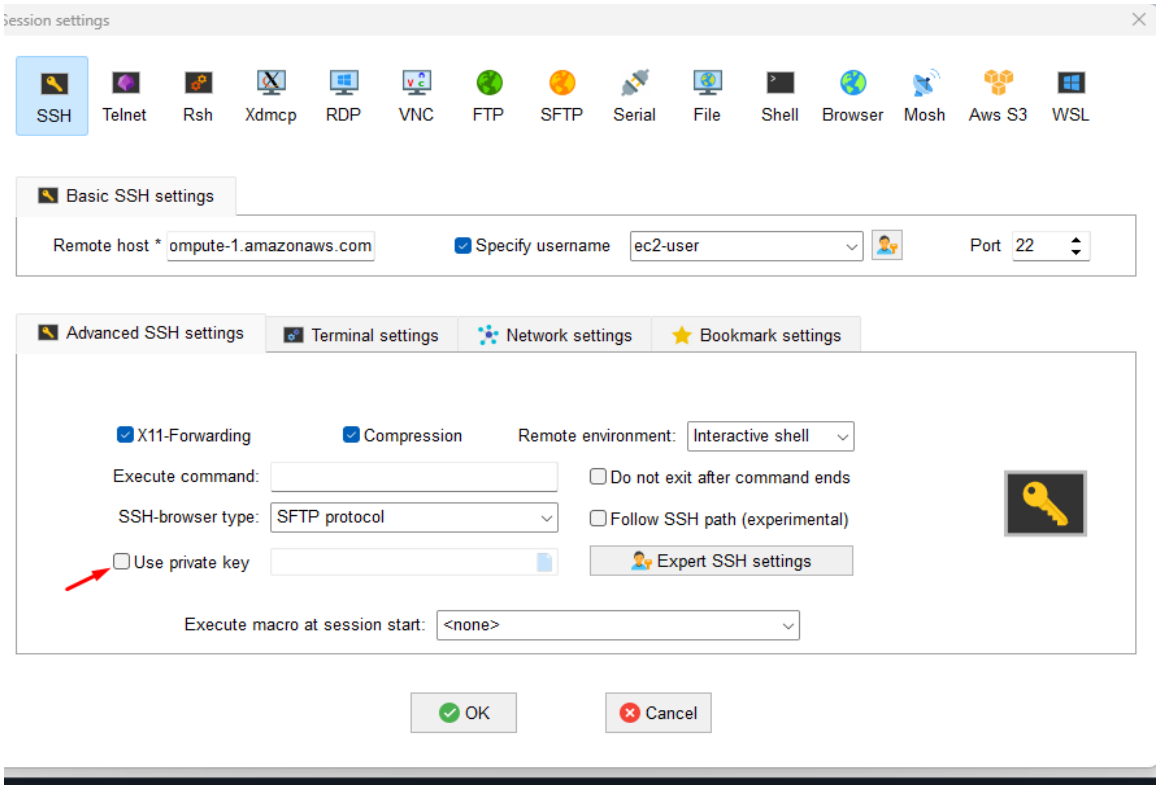
Utilizando el software SSH client en Windows nos encontraríamos con una imagen tal como se aprecia para poder realizar la conexión.



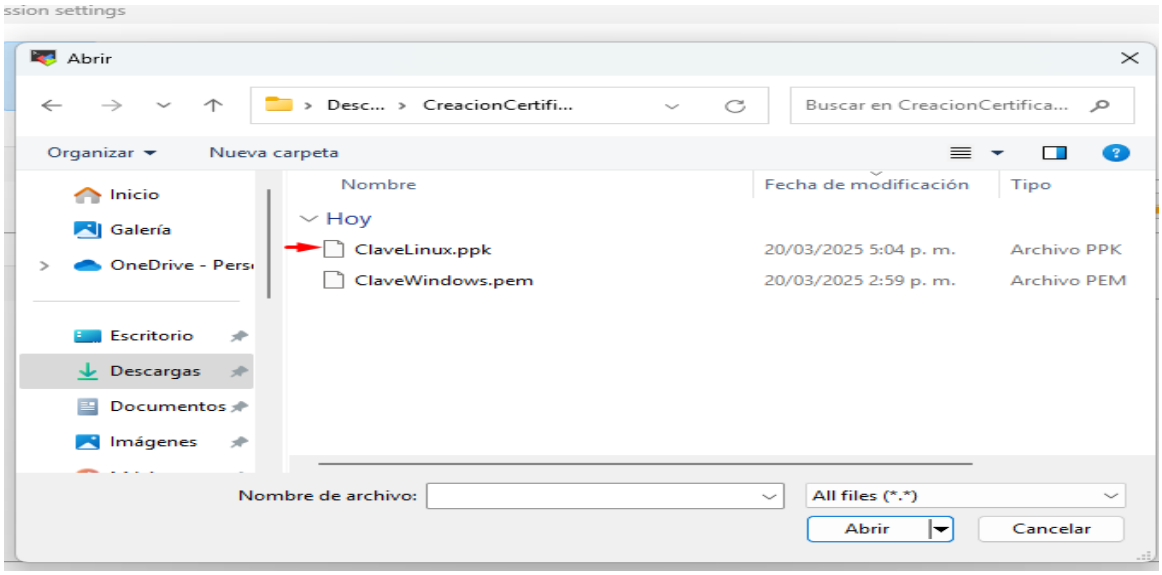
Paso seguido seleccionamos el apartado Advanced SSH Settings:



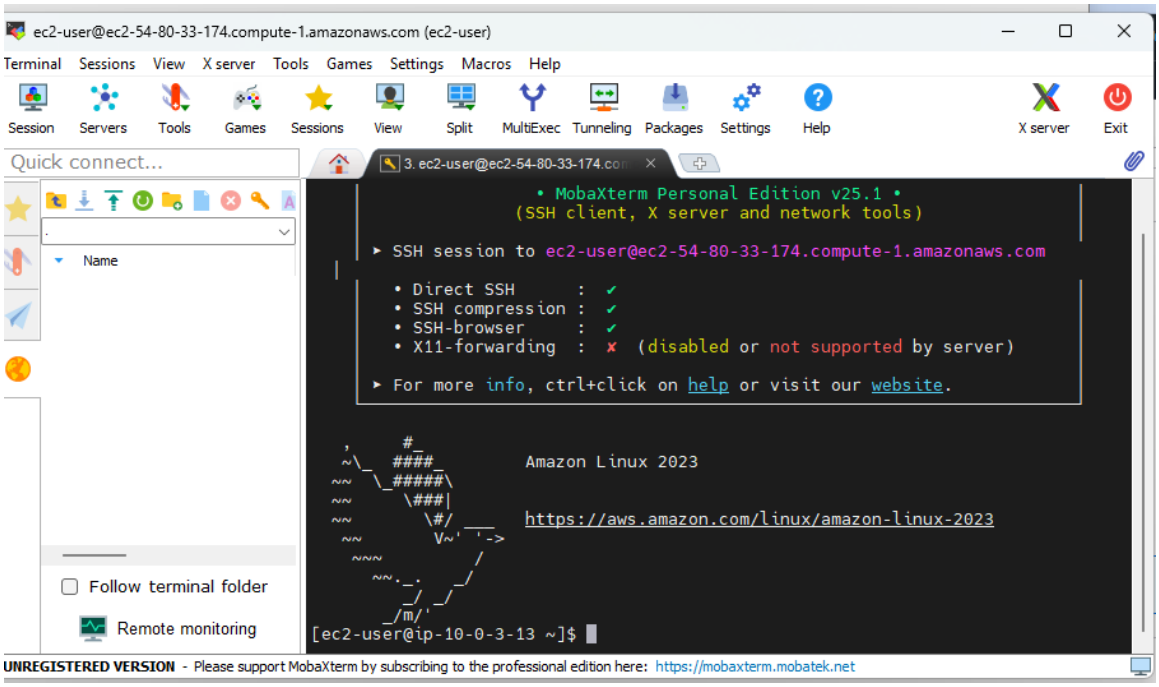
Seleccionamos el check para poder ingresar la contraseña anteriormente creada en el proceso:



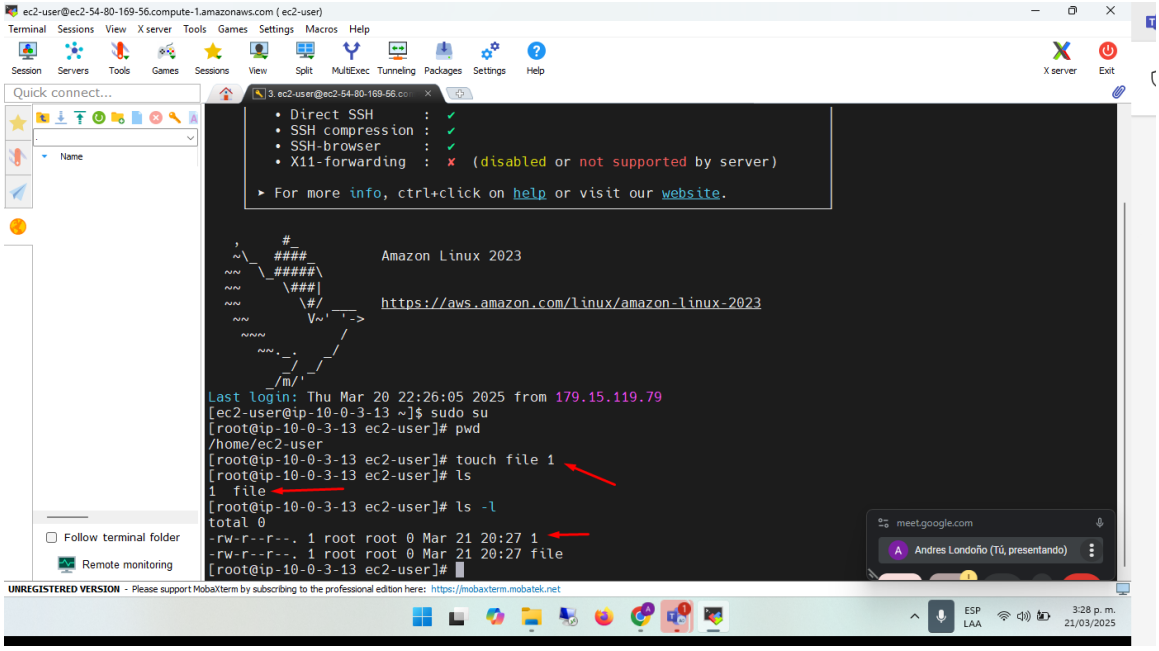
Seleccionamos la ppk para linuk que se guardó en nuestro PC, es de suma importancia guardar muy bien este key ya que si lo pierdes o extravías no podrás recuperar tu maquina ya que ni siquiera soporte de AWS te va ayudar:



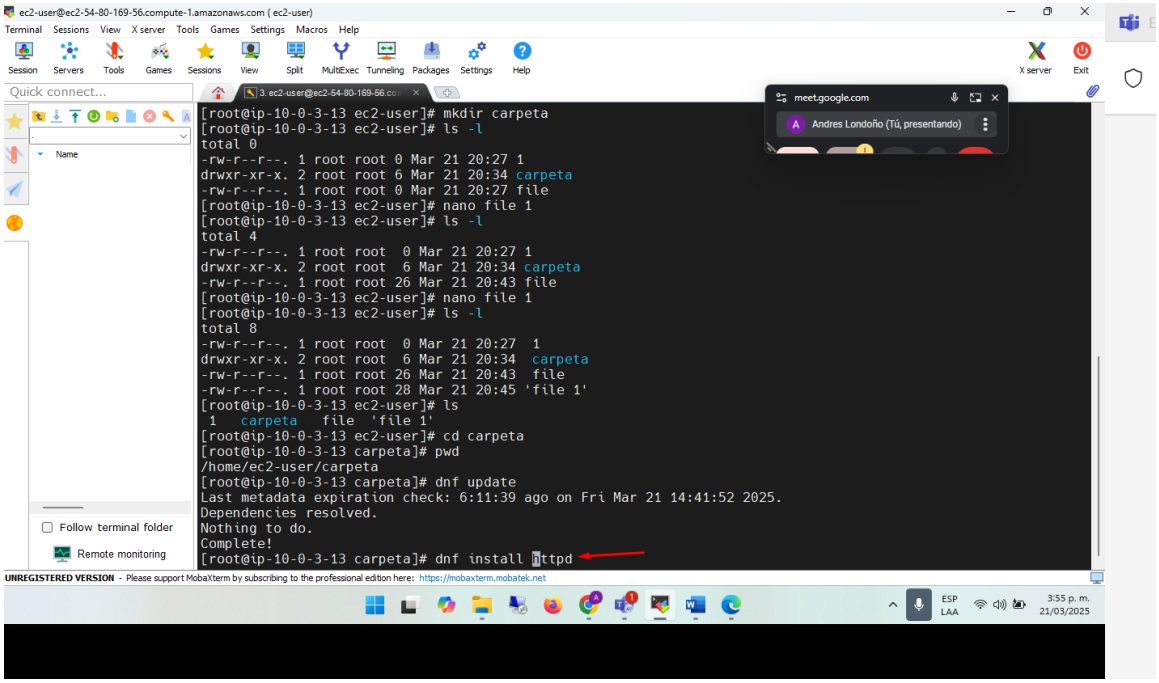
Validamos que después de los pasos anteriores nos resulte la siguiente pantalla si todo esta:



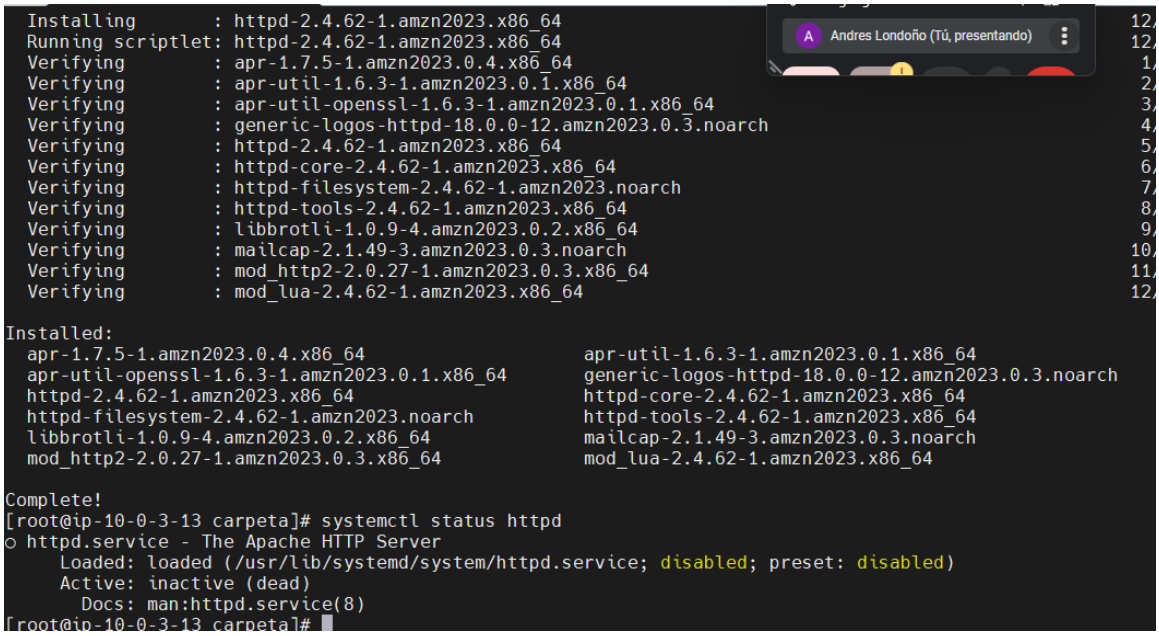
Creamos Un archivo nombrado File 1:



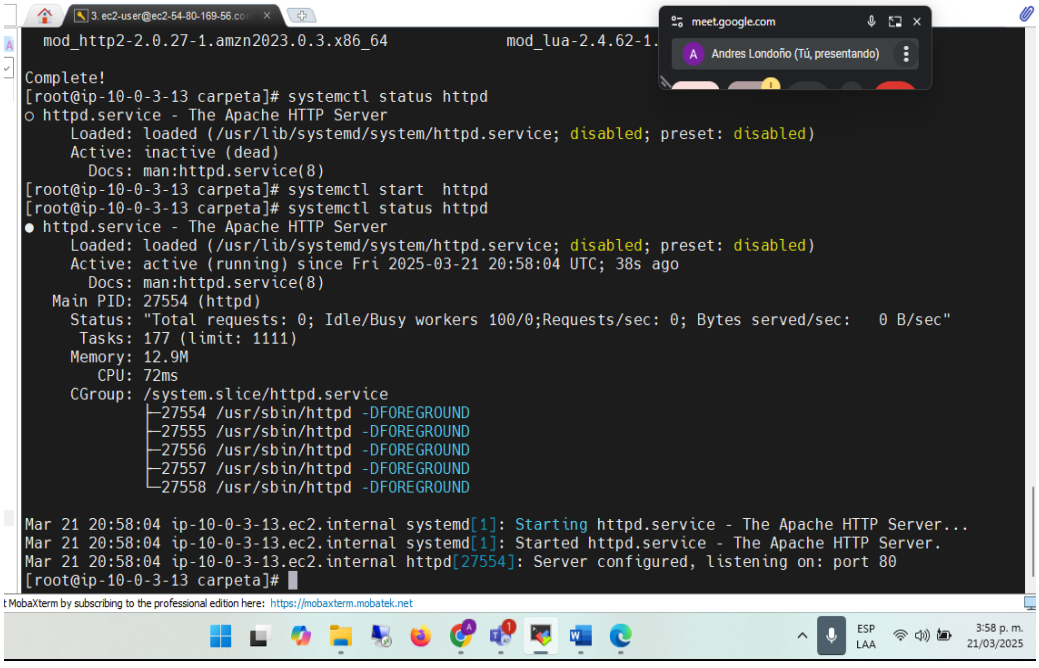
Para poder continuar con nuestro trabajo necesitaremos instalar en nuestra maquina un server, que para este caso utilizaremos el apache, este se instala vía comandos como se puede evidenciar en la imagen:



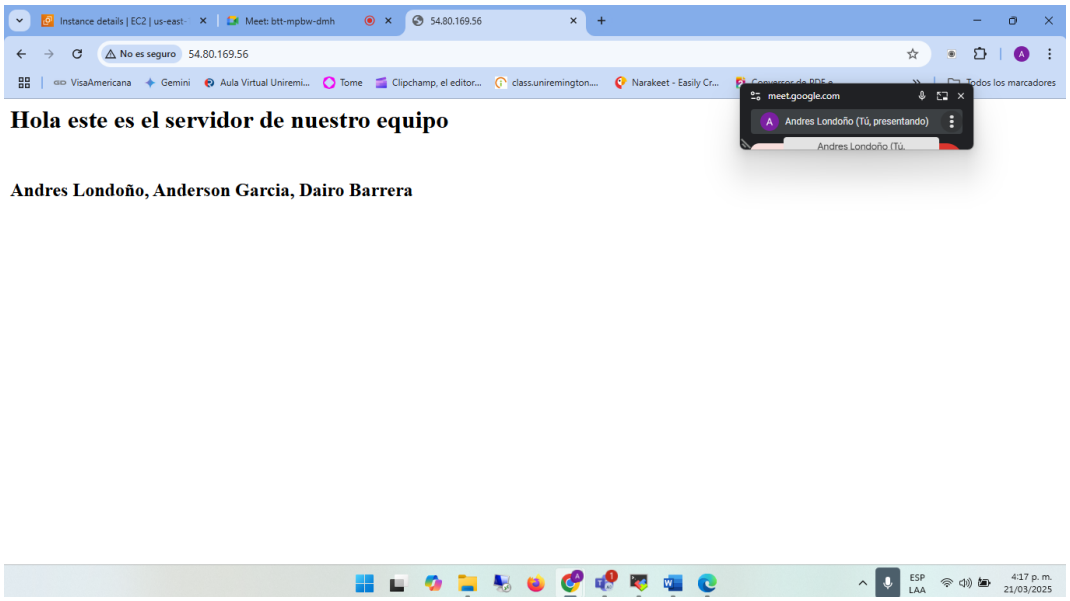
En esta imagen estaremos evidenciando el estado de nuestro server:



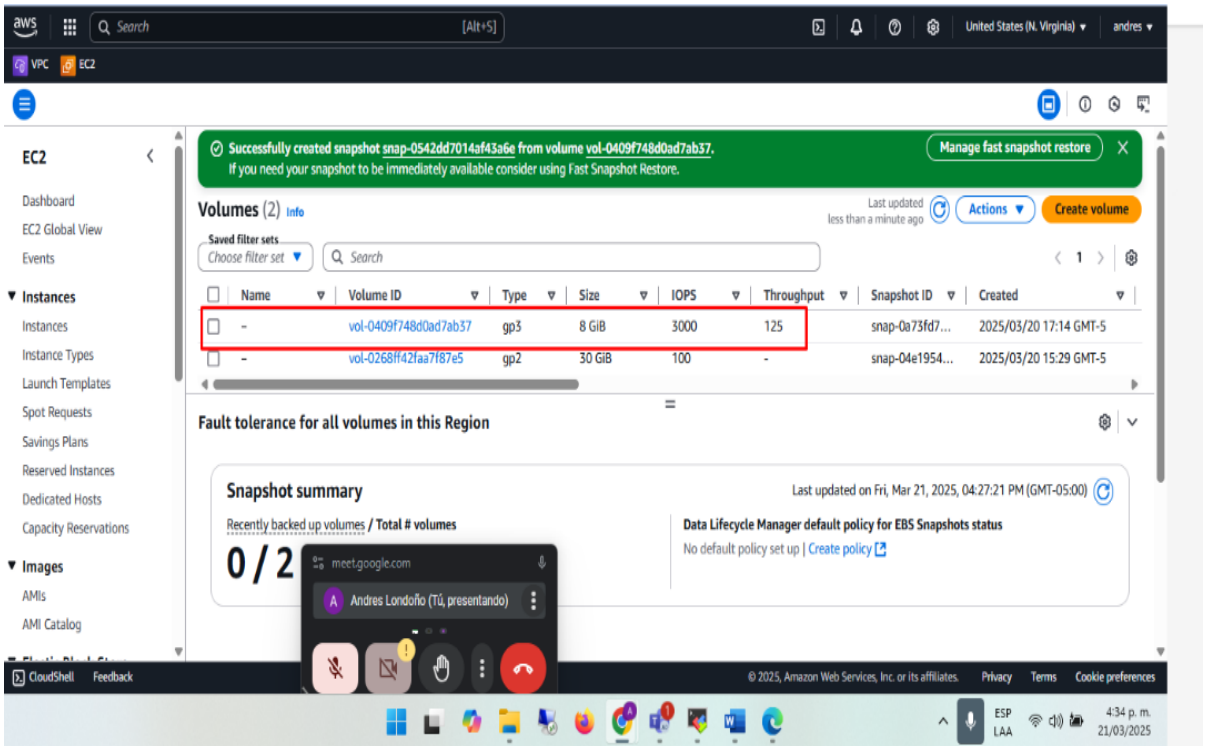
Validaremos que el servicio este corriendo de forma normal, si todo va bien deberías de estar viendo una imagen similar a esta:



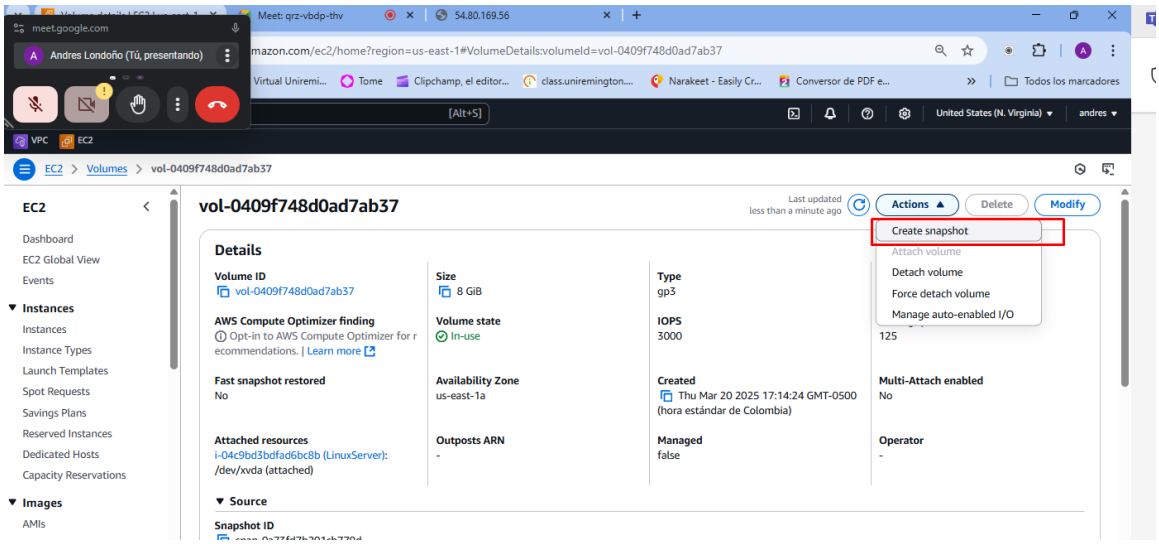
Utilizando un navegador web y digitado la ip asignada por AWS más, deberíamos poder observas navegación, con un mensaje diferente ya que en nuestro caso creamos un archivo índex para que el navegador mostrara información.



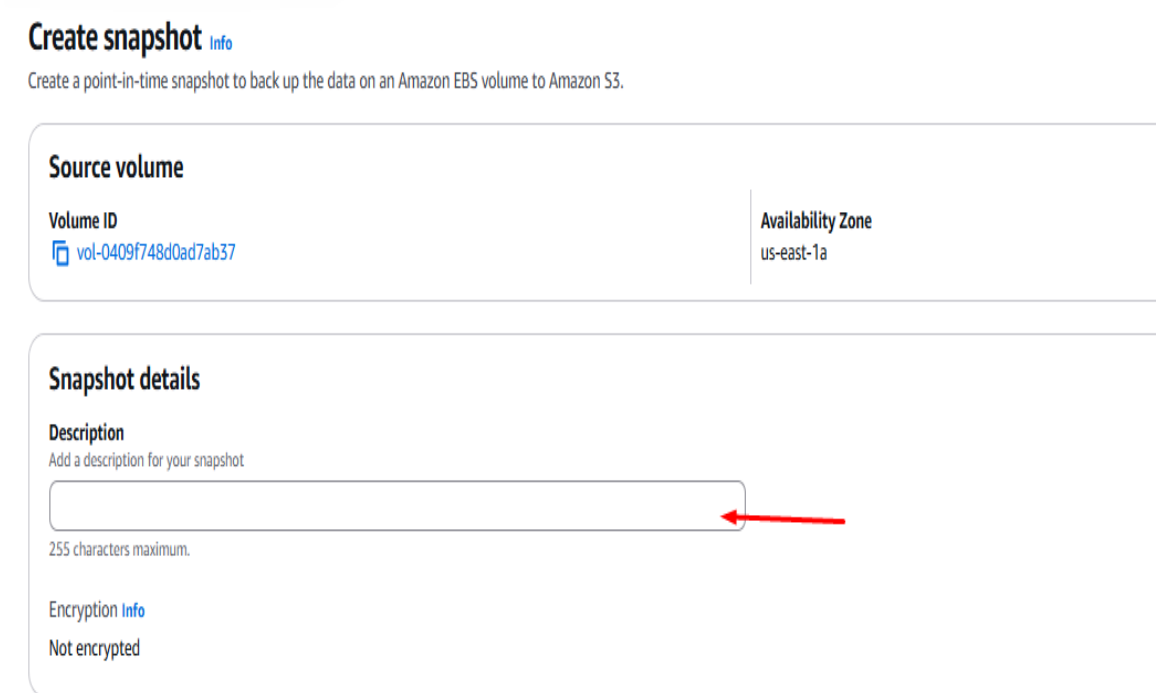
Seleccionamos el disco del server(instancia) Linux:



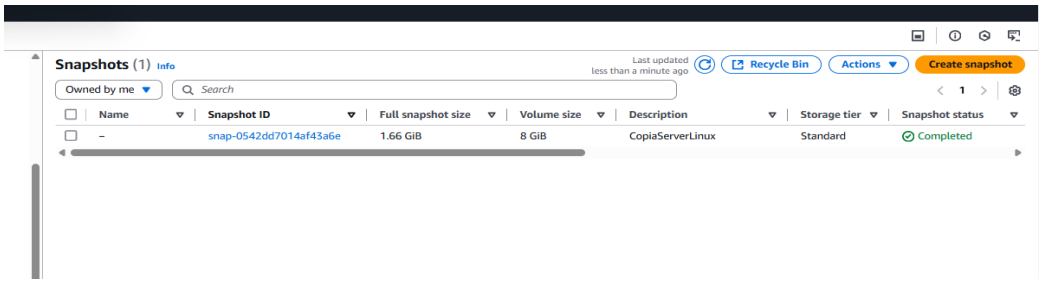
Podemos crear un Snapshots (copia de seguridad) de la siguiente manera:



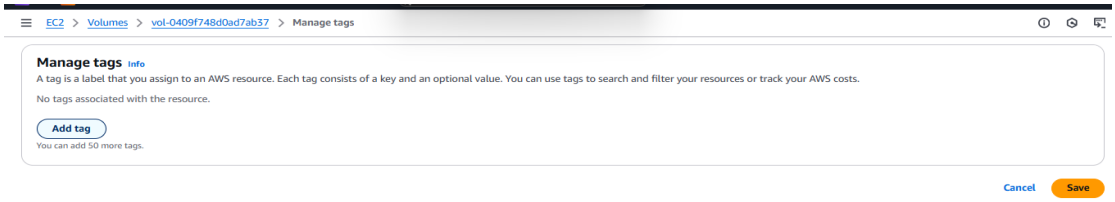
Le ponemos un nombre representativo que nos dé alusión a lo que estamos realizando:



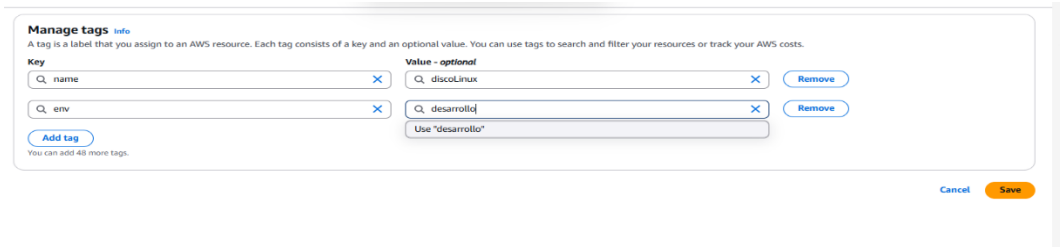
Podemos validar que ya fue creado:



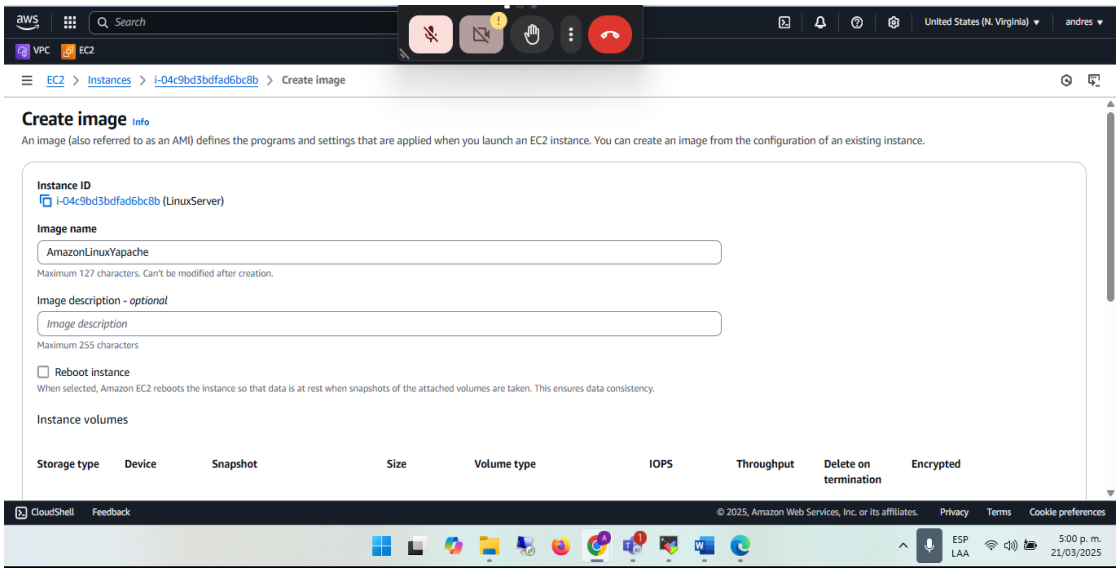
Creamos un tag para el disco de la instancia Linux de la siguiente manera:



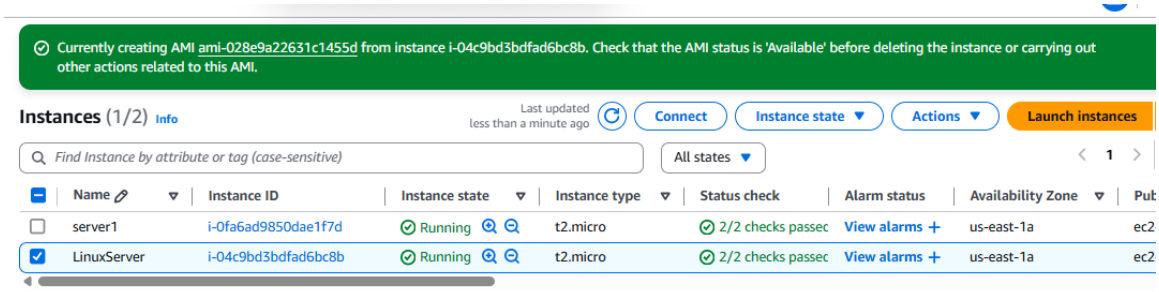
Creamos los que sean necesarios para nuestro ambiente y hacer el backup de los discos:



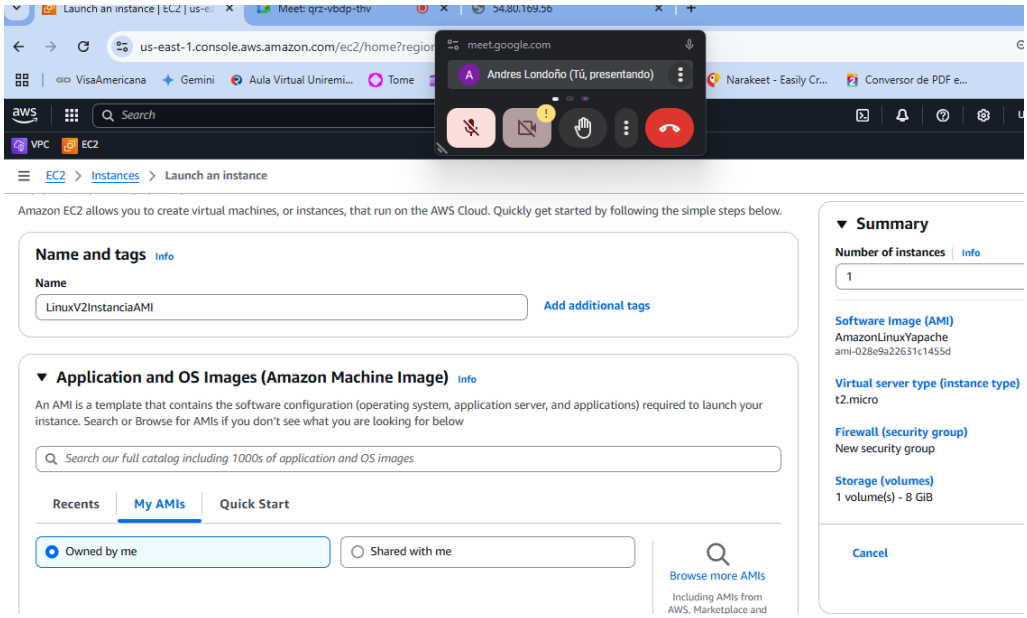
Creamos una imagen tipo (backup) utilizando los snapshots:



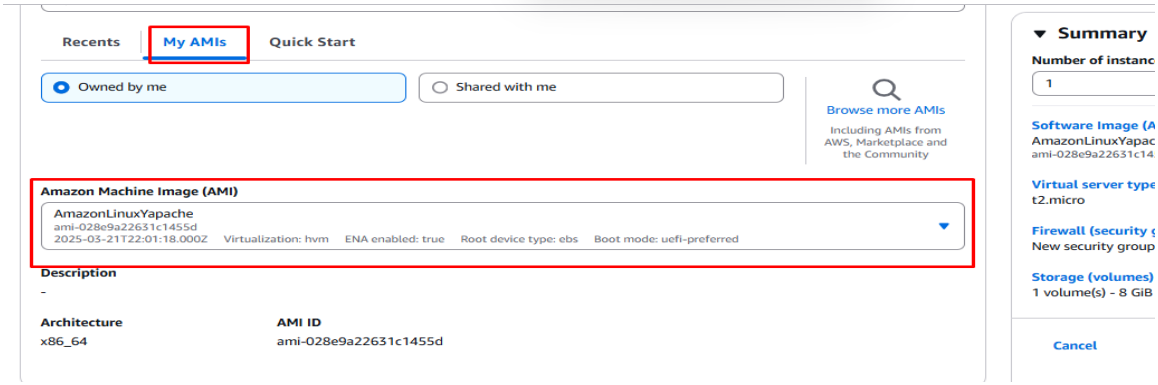
Quedaría la AMI de la siguiente manera (clonaría toda la instancia), se ahorra todos los parámetros de configuración, solo seleccionando la AMI cuando crea la instancia:



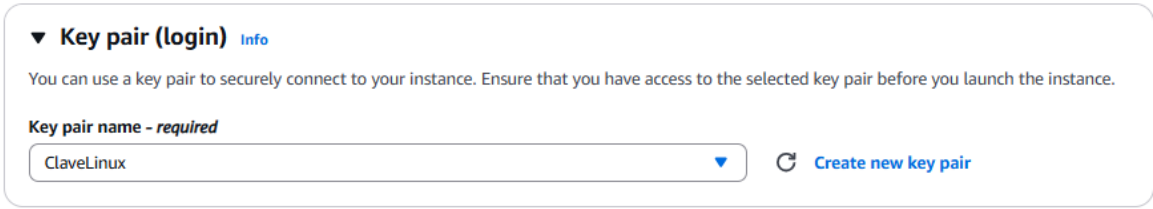
Para crear una AMI en una nueva instancia hacemos lo siguiente:



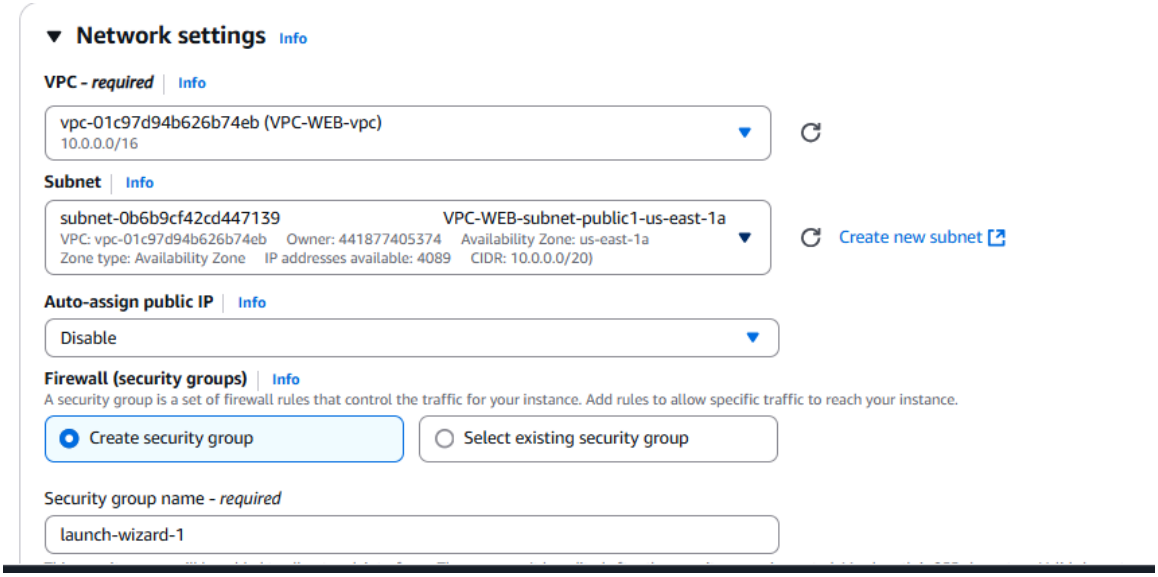
Seleccionamos la AMI anteriormente creada de la siguiente manera:

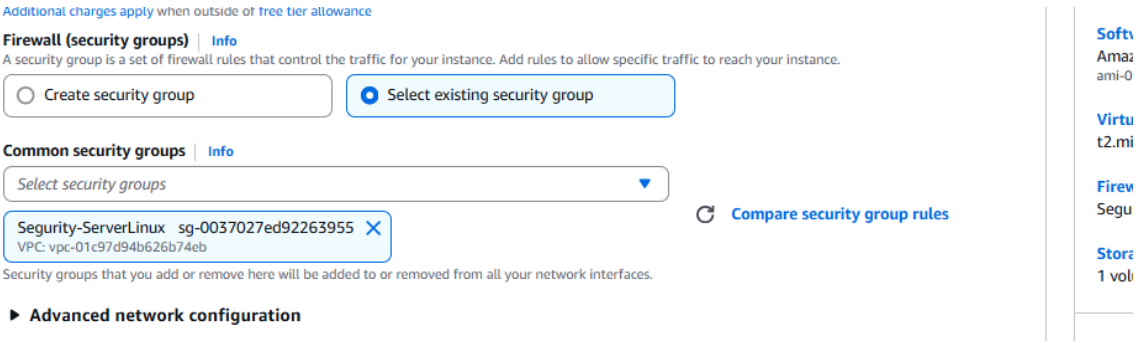


Volvemos a utilizar la misma clave de la instancia anterior:

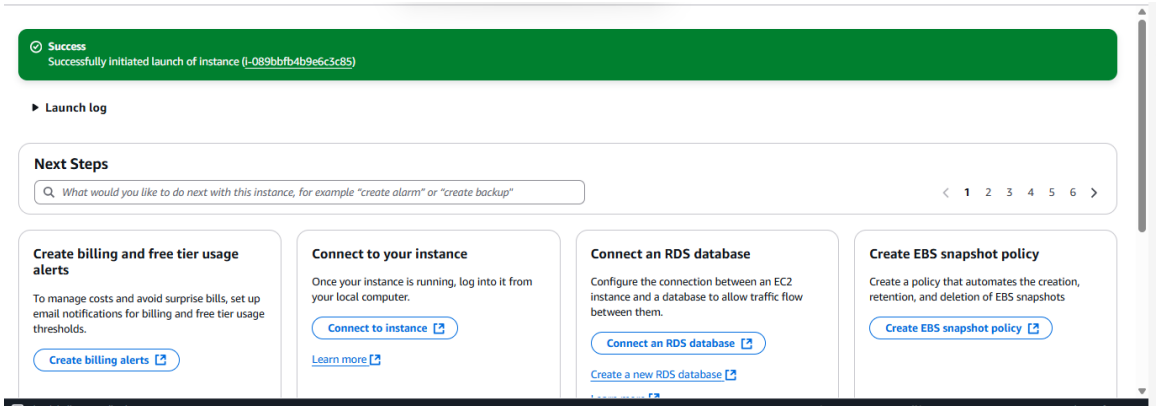


Configuramos nuevamente la red y el VPC:

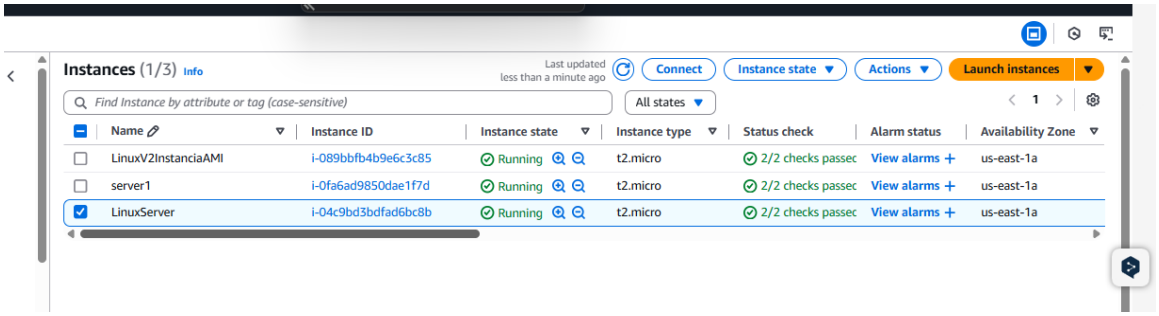




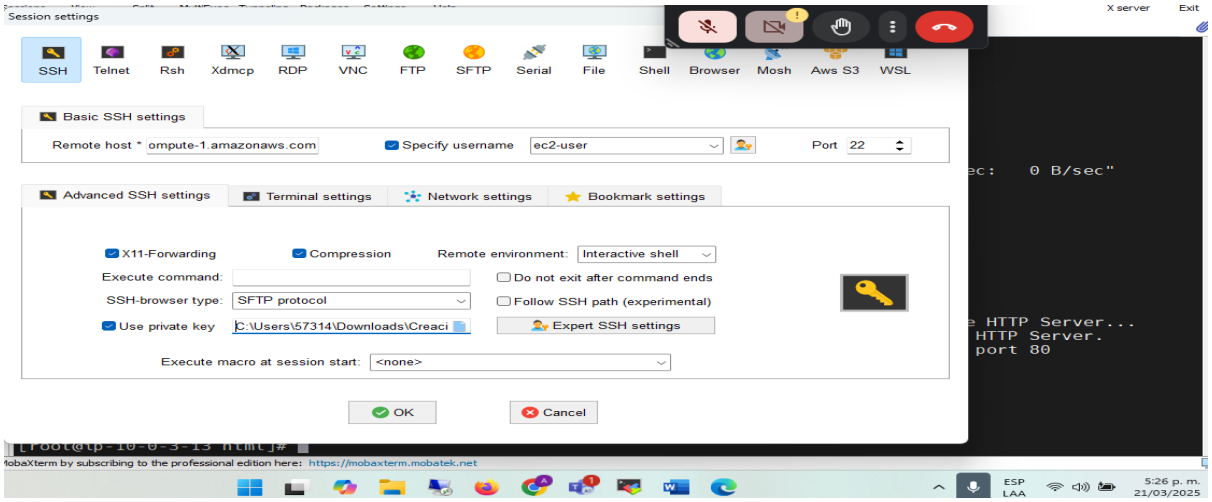
Queda de la siguiente manera:



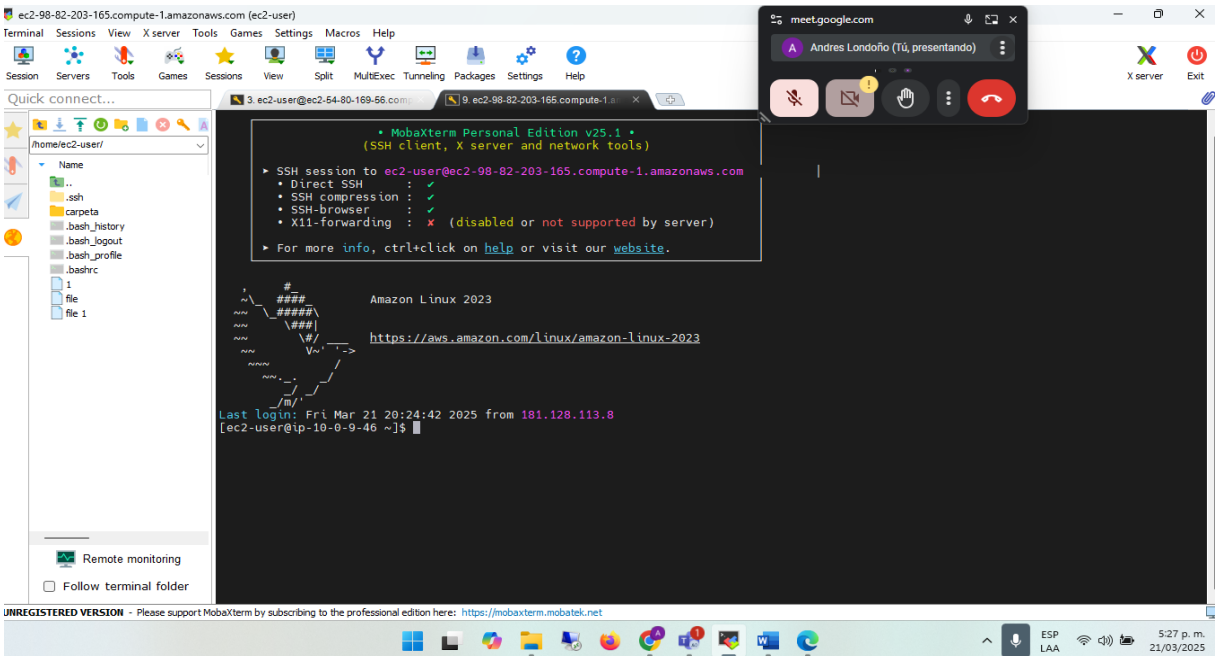
Como podemos observar en la siguiente imagen quedaría de la siguiente manera nuestra nueva Instancia AMI:



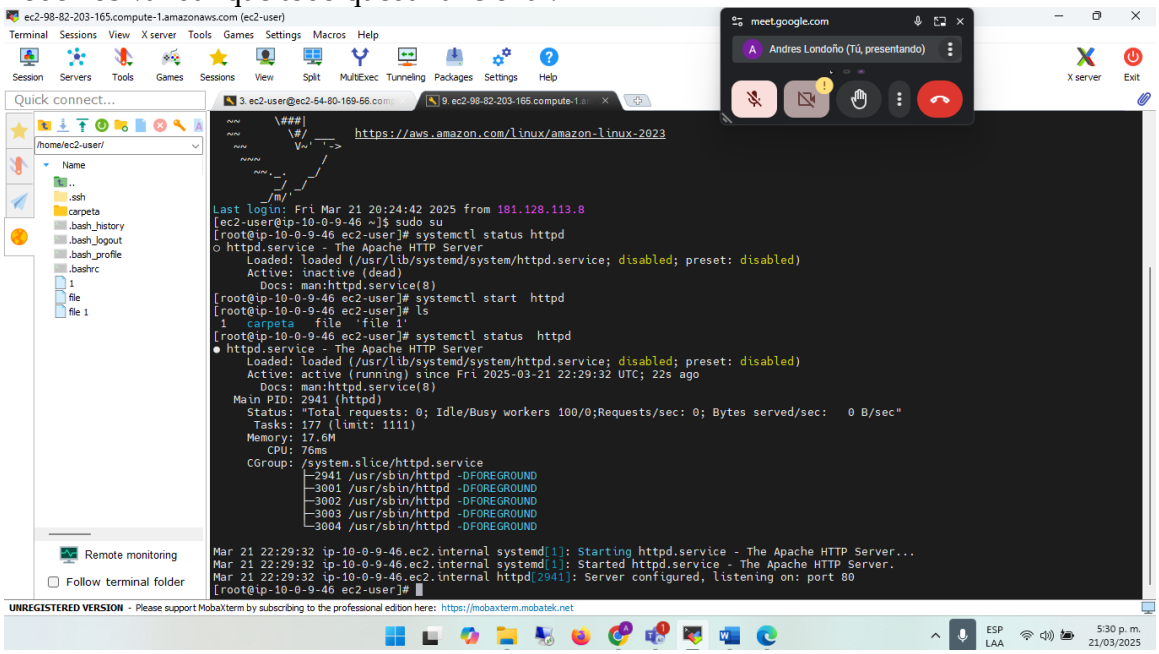
Nos conectamos a la instancia AMI creada de la siguiente manera:



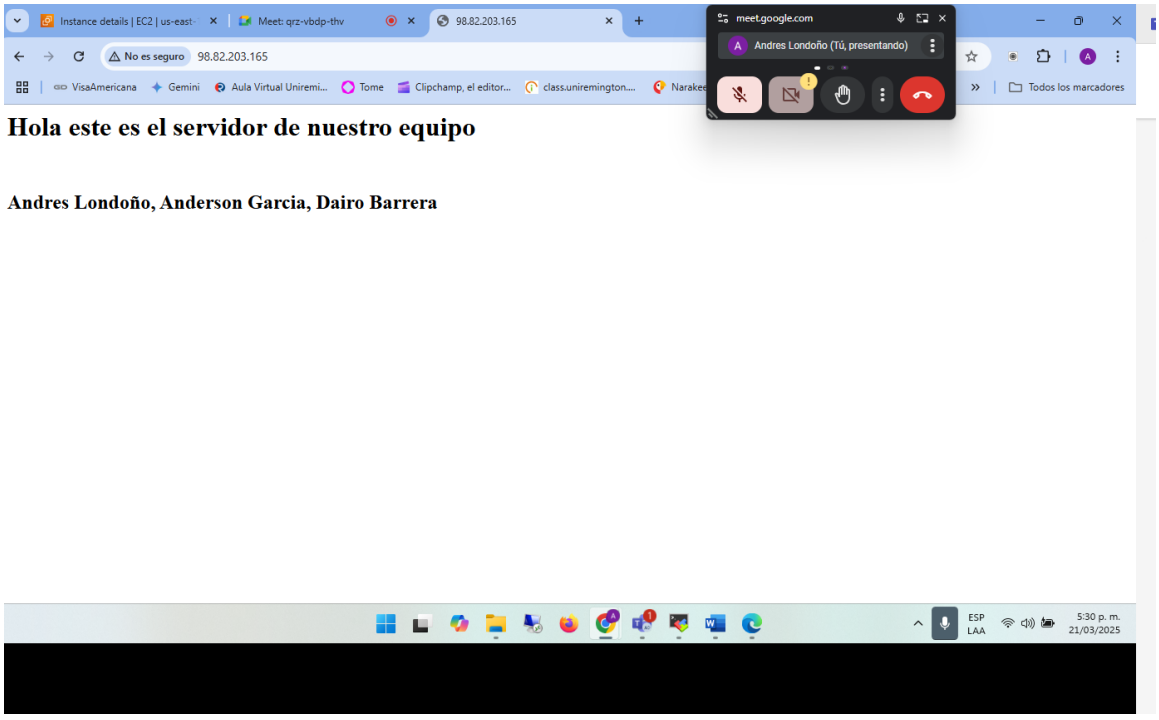
Quedaría de la siguiente manera:



Podemos validar que todo queda funcional:

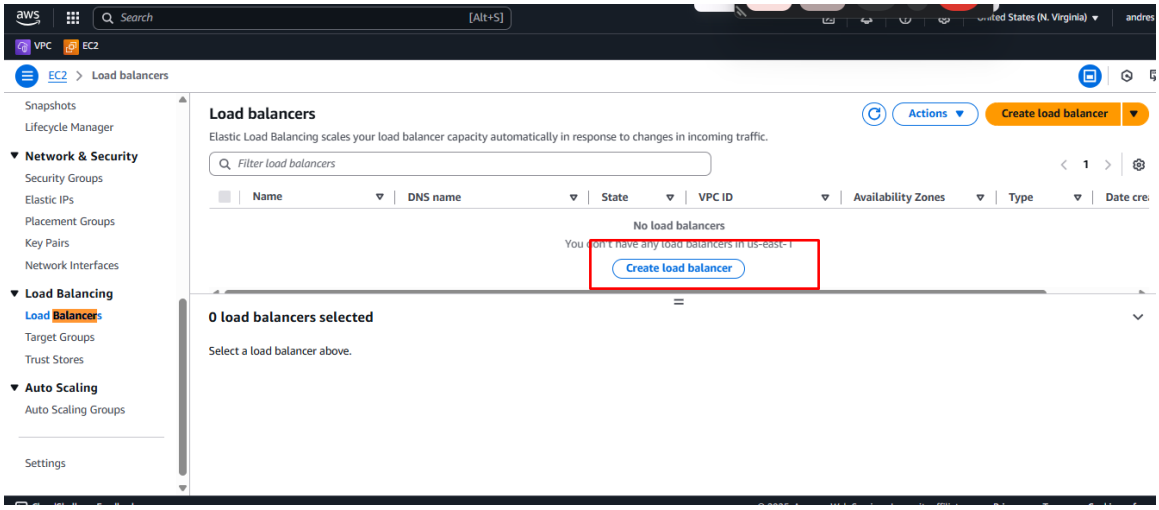


En esta imagen estamos mostrando nuestro servidor corriendo:

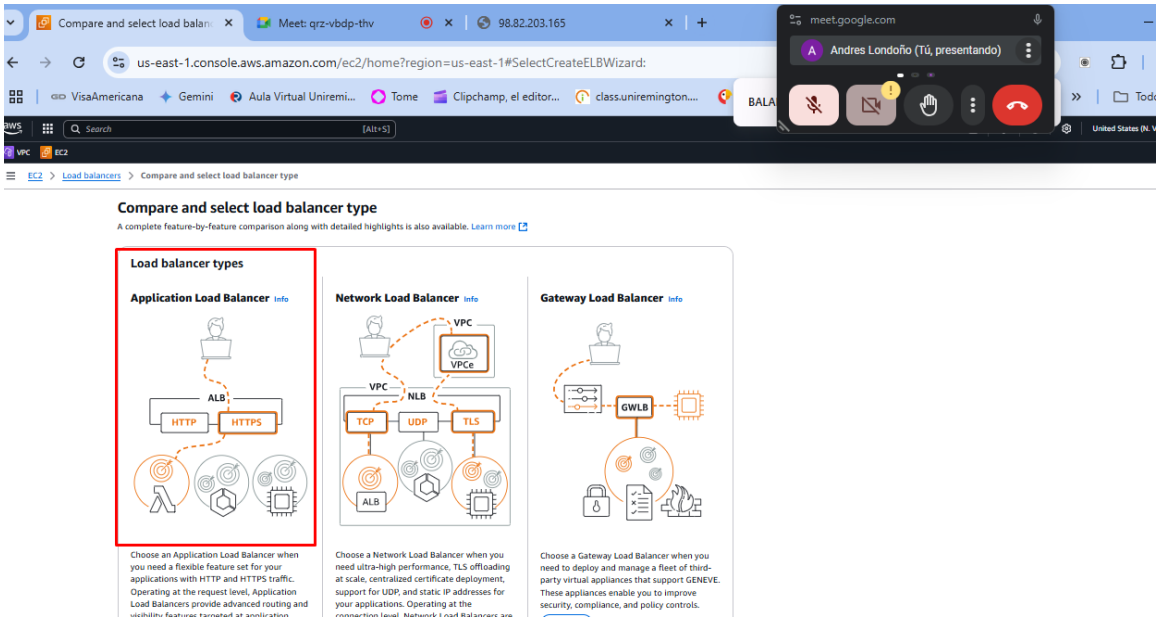


BALANCEADOR DE CARGA AUTOSCALING

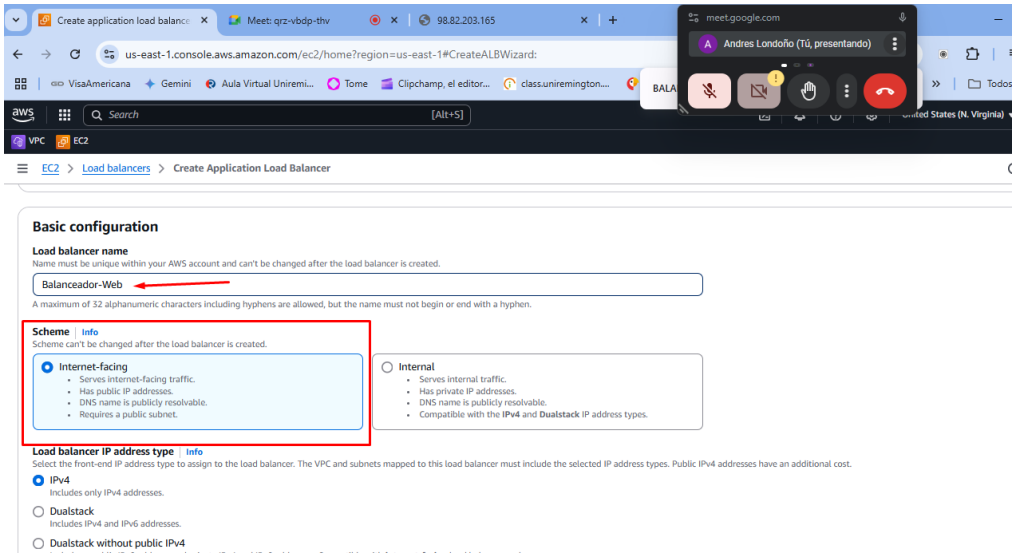
Creamos nuestro balanceador:



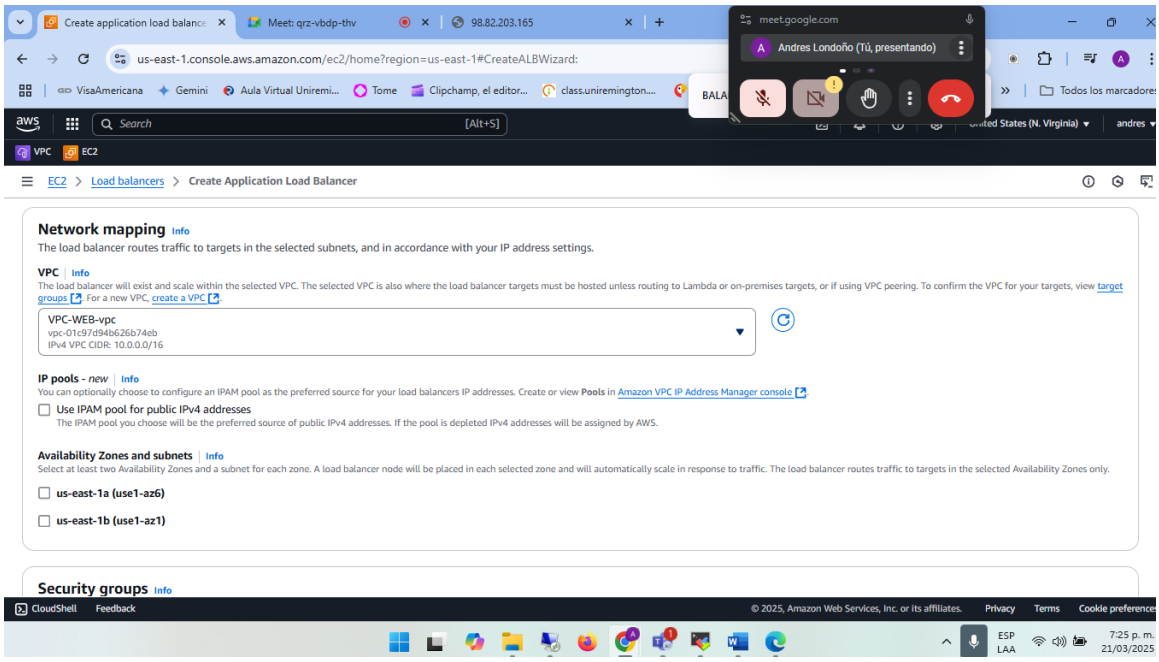
Seleccionamos el primer balanceador trae HTTP HTTPS :



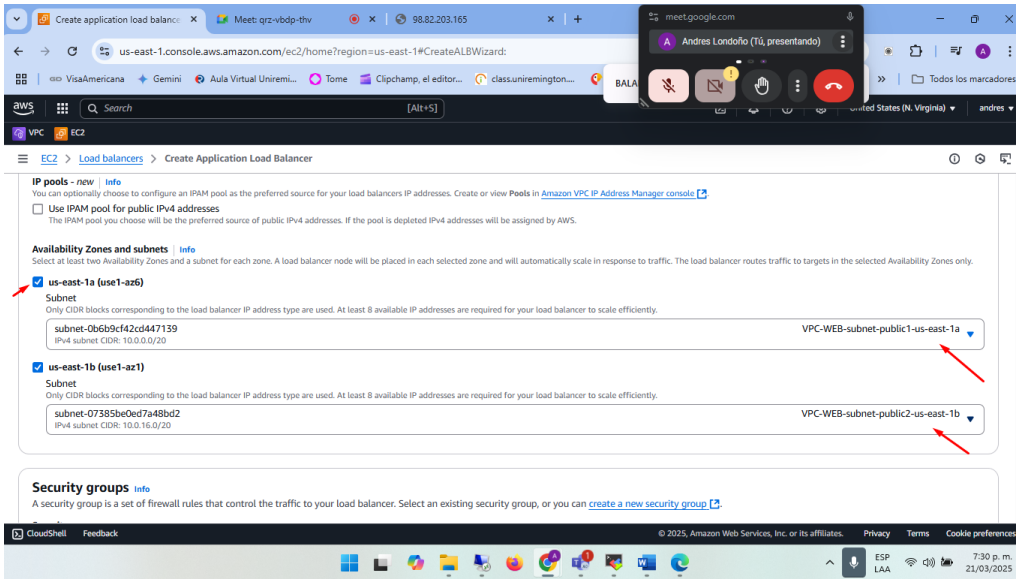
Le damos un nombre Ideal Acorde a las condiciones y seleccionamos El balanceador Externo que se puede acceder desde afuera(Internet):



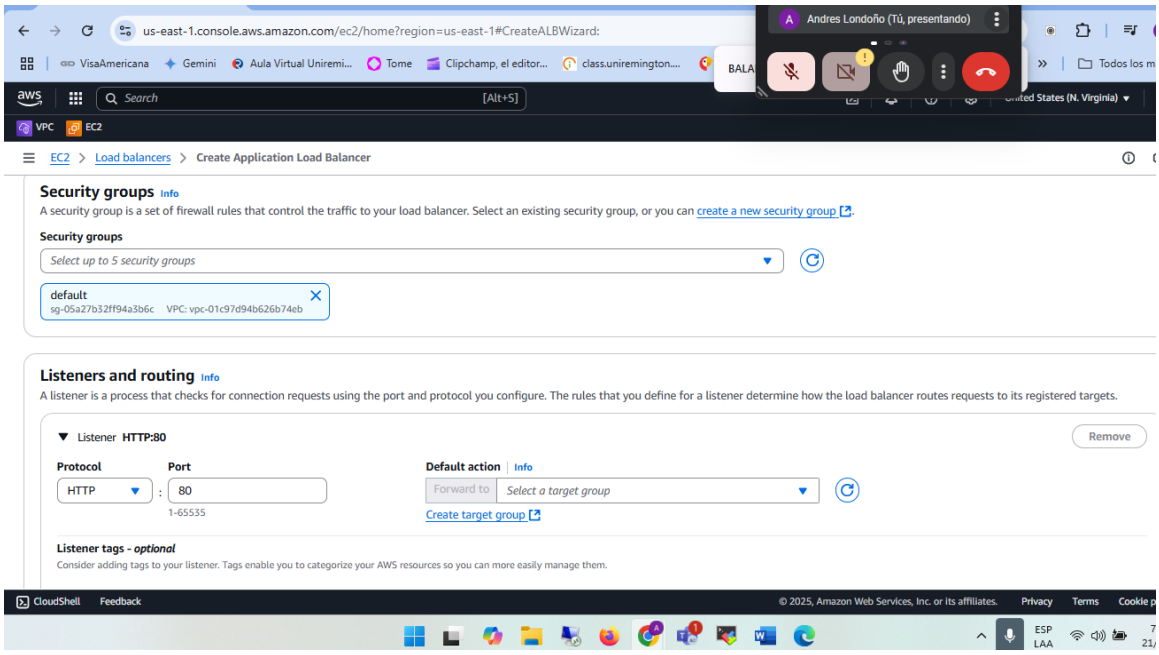
Seleccionamos nuestro VPC ya creado anteriormente:



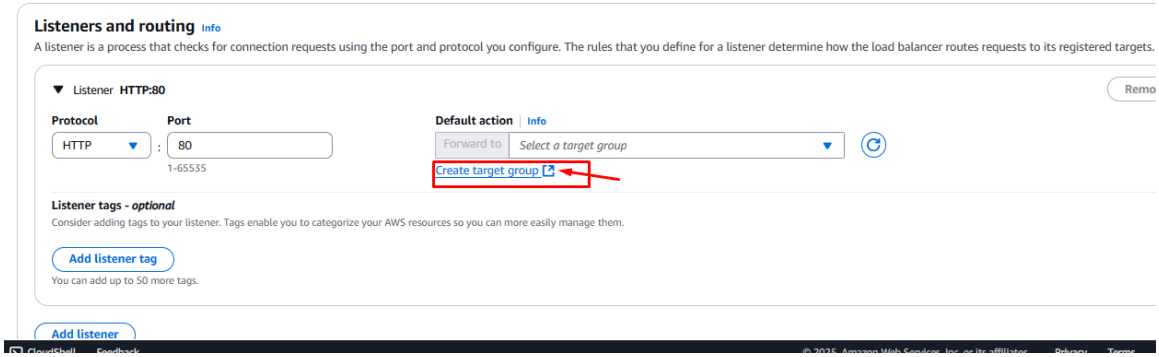
Seleccionamos las subredes públicas de los datacenter para tener un dinamismo, esto con el fin de evitar que la información este en un solo datacenter:

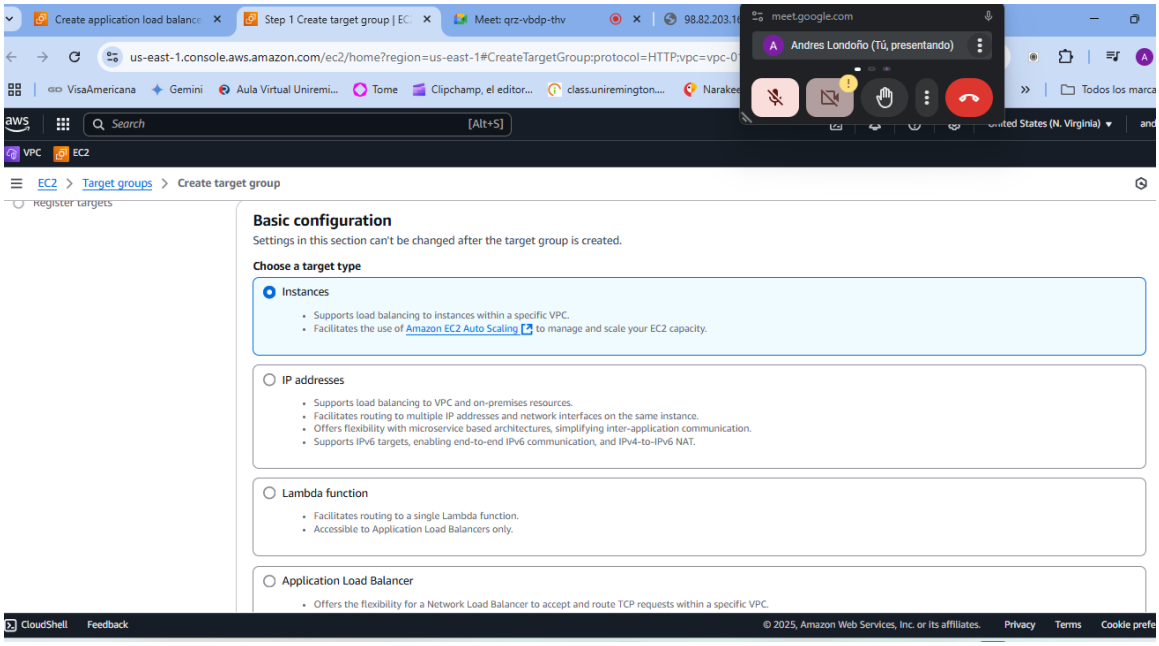


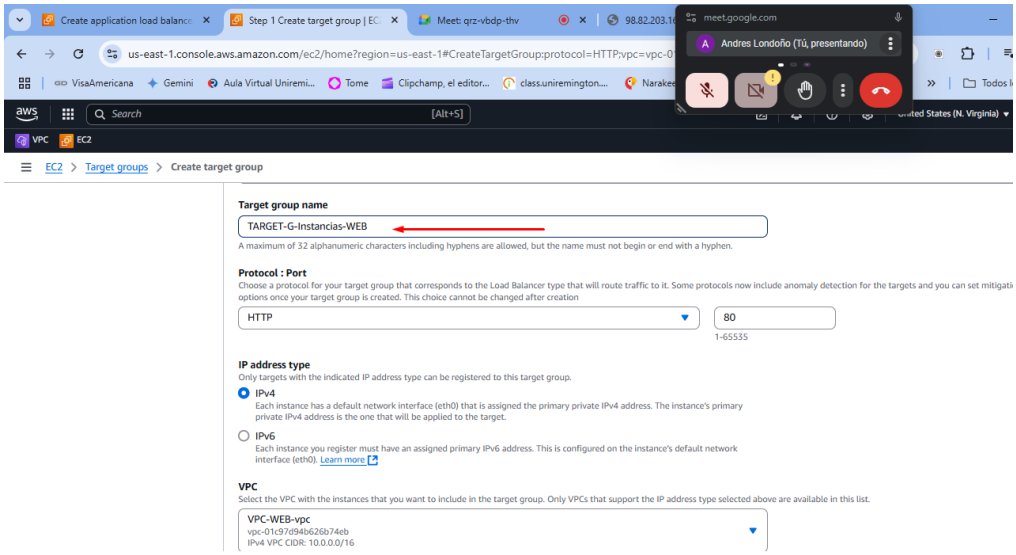
Las peticiones las recibe el balanceador de carga en este caso por el puerto 80:



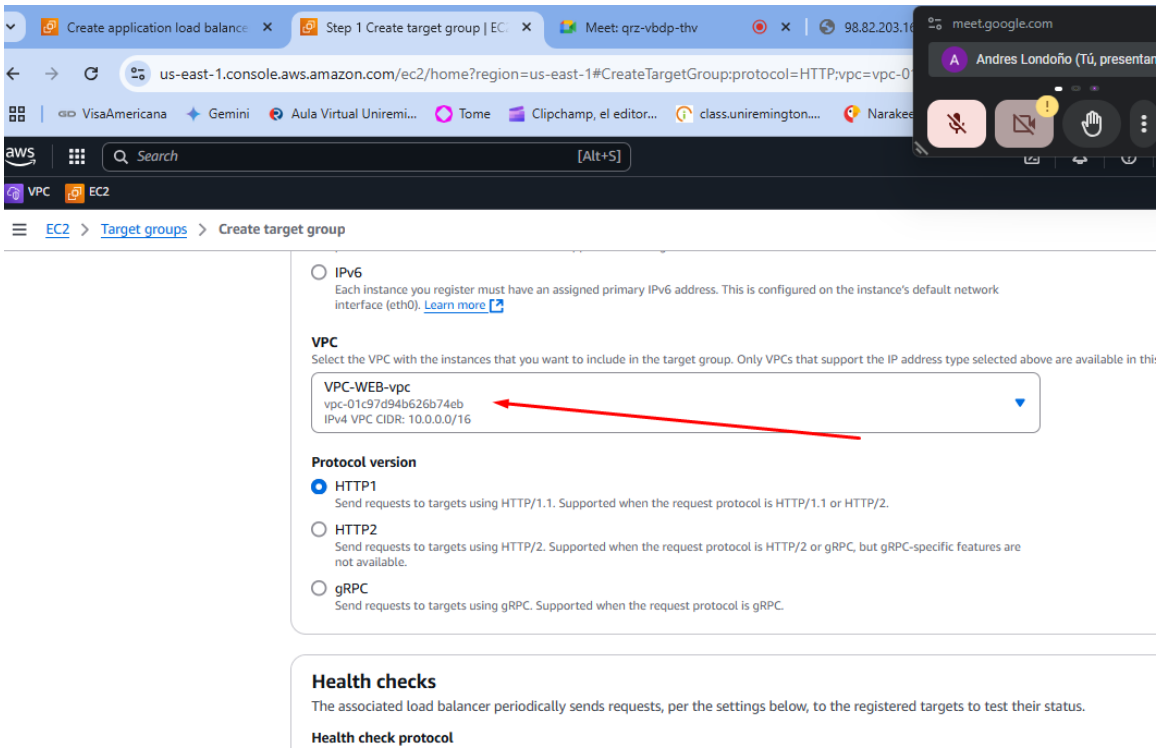
Creamos el target group:



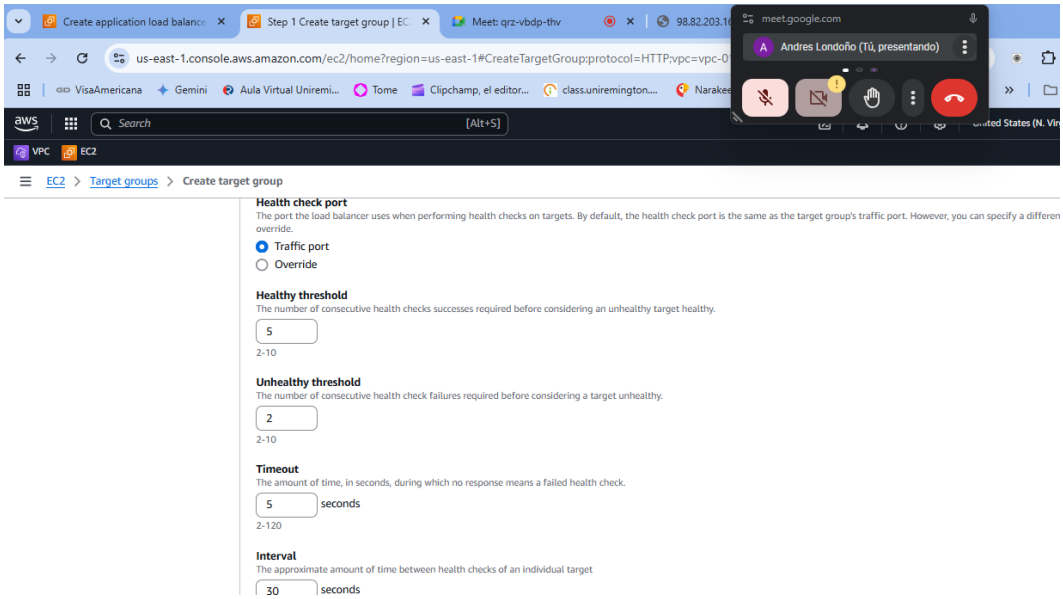




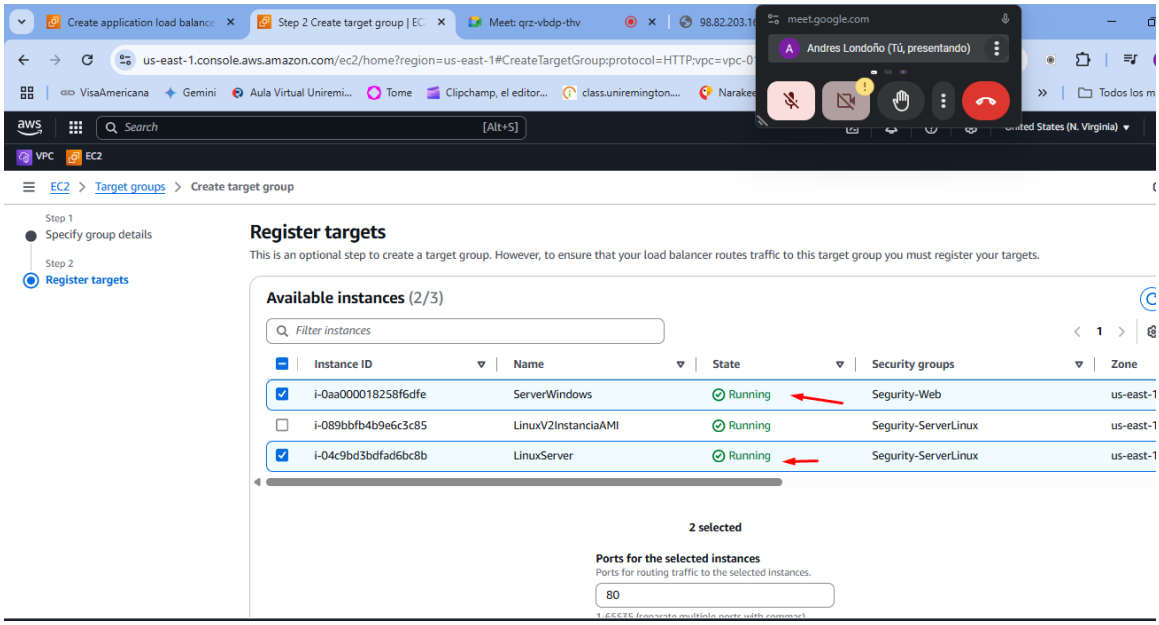
Se busca el VPC de las instancias que para nuestro caso es:

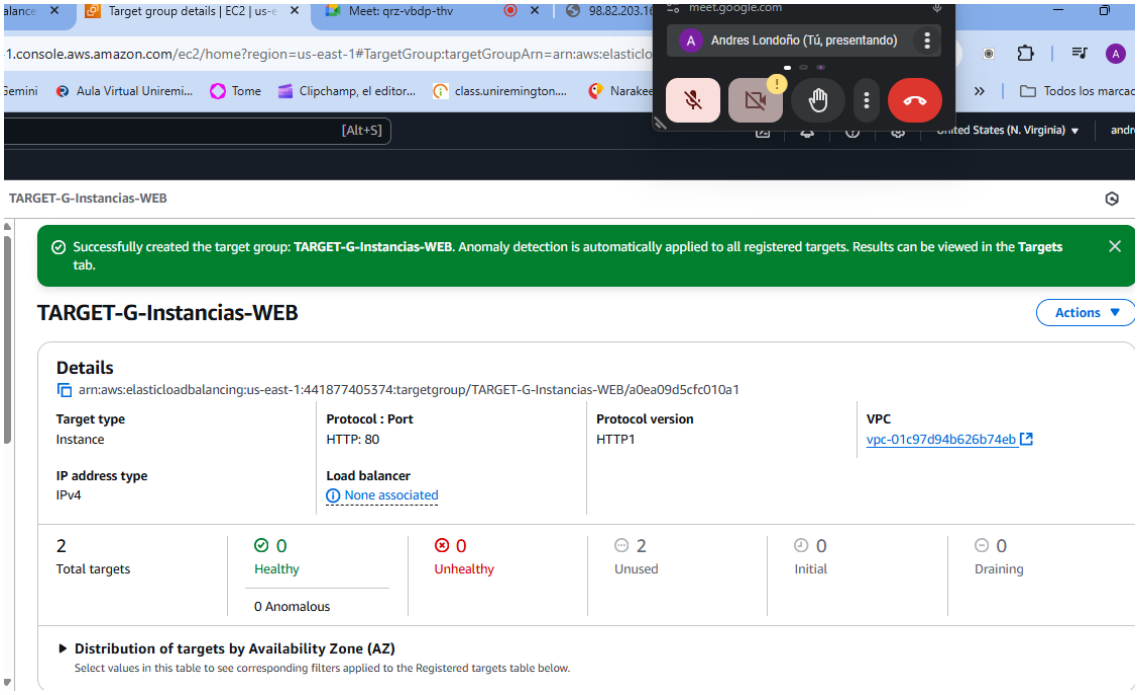


Valida por tiempos cada cuanto las instancias están funcionando(Activas) valida con el código 200, se configura según su criterio:

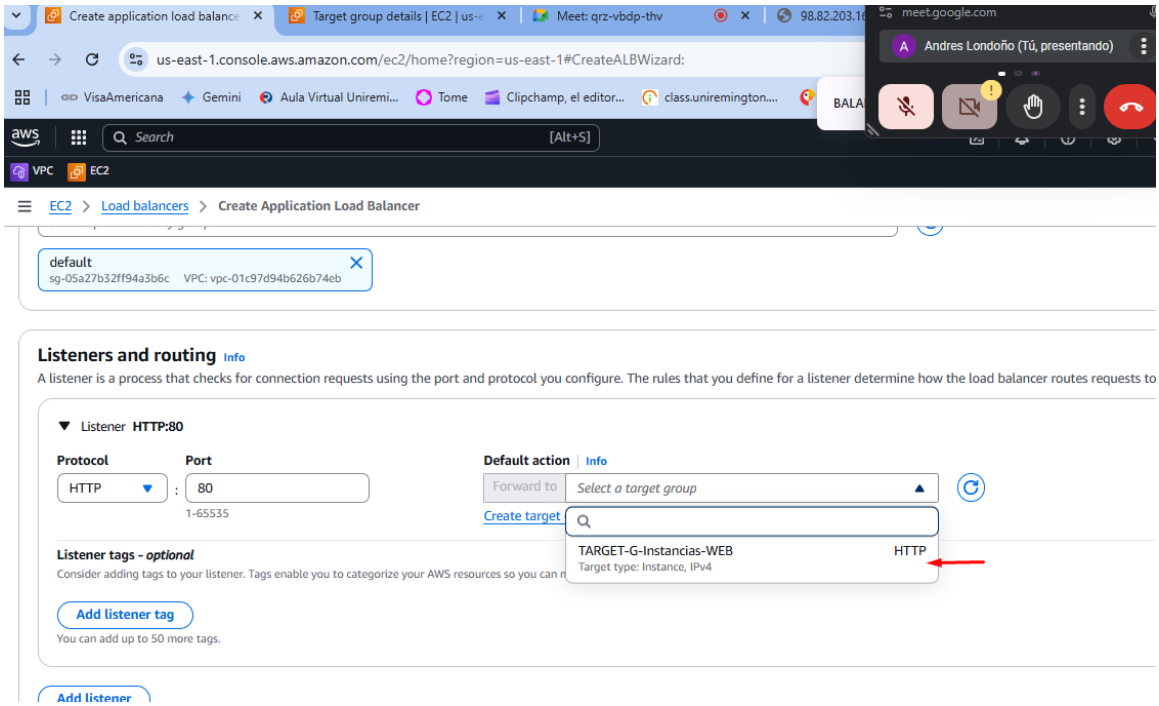


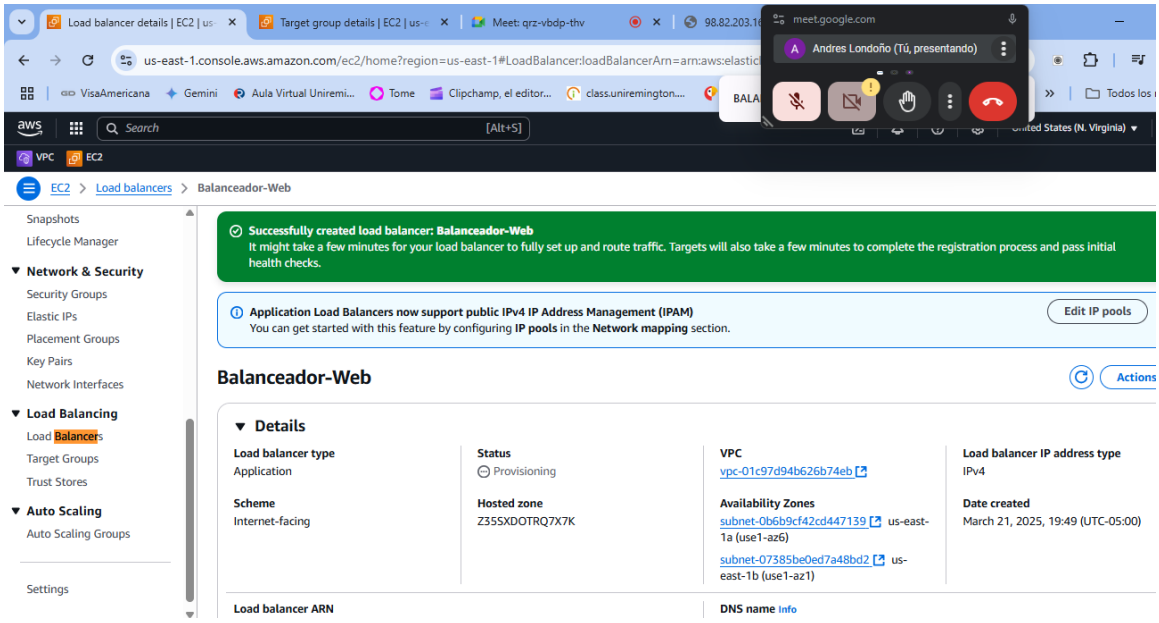
Seleccionamos las instancias a las cuales le aplicaremos la validación:



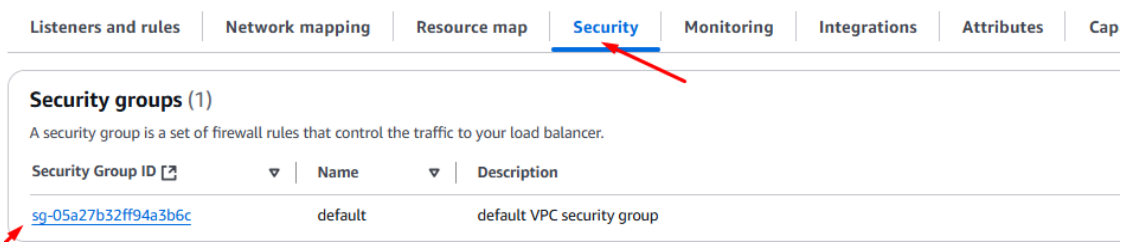


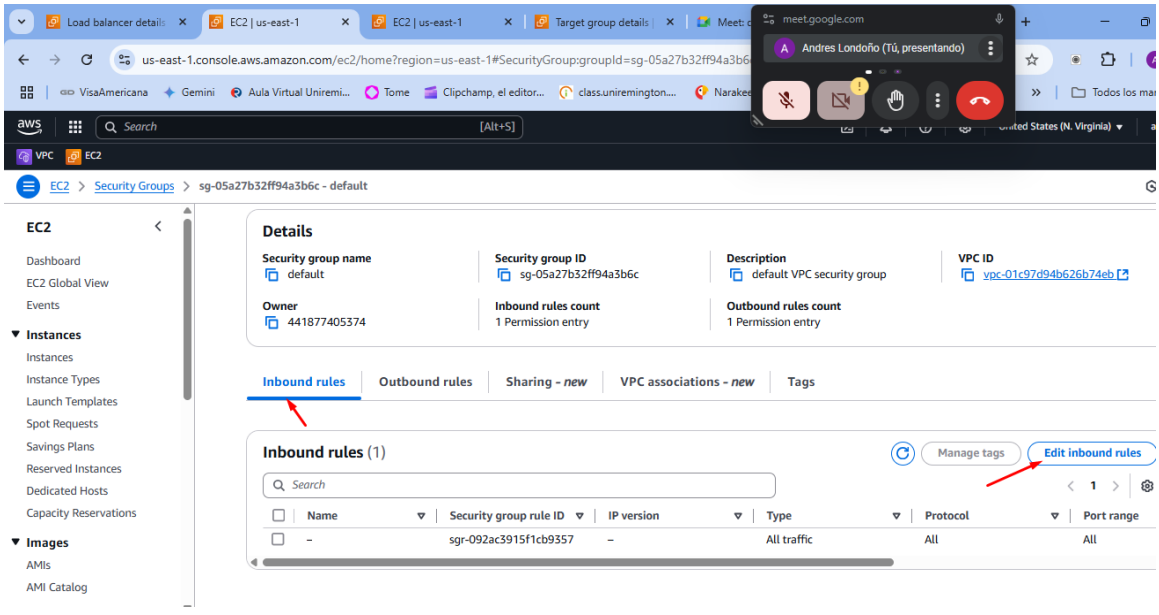
Volvemos al balanceador y escogemos el target anteriormente creado y quedaría de la siguiente manera:



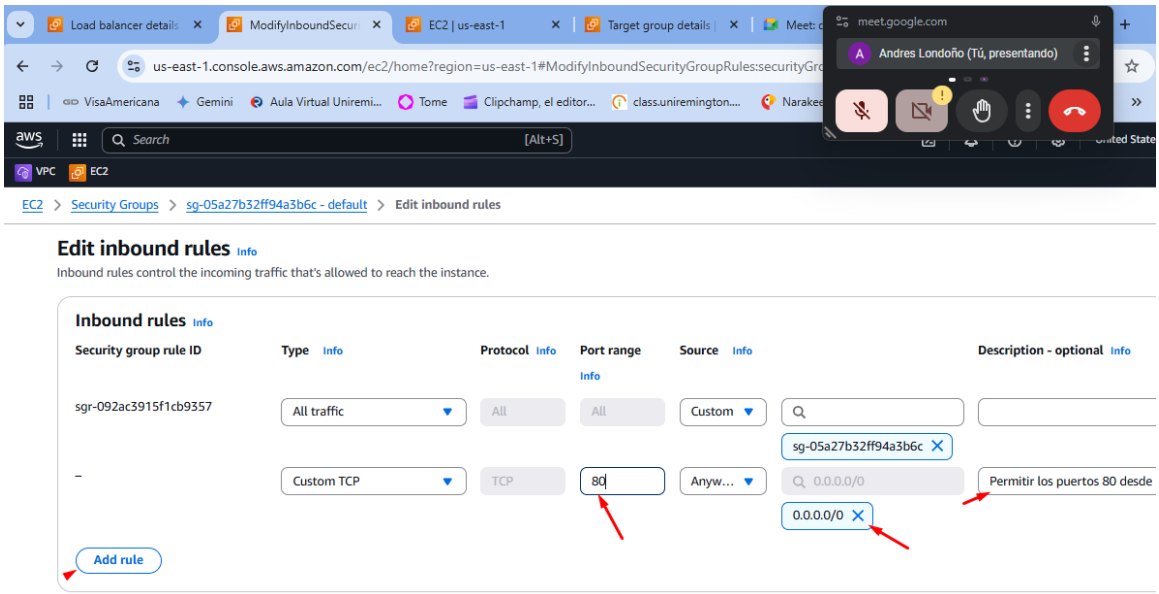


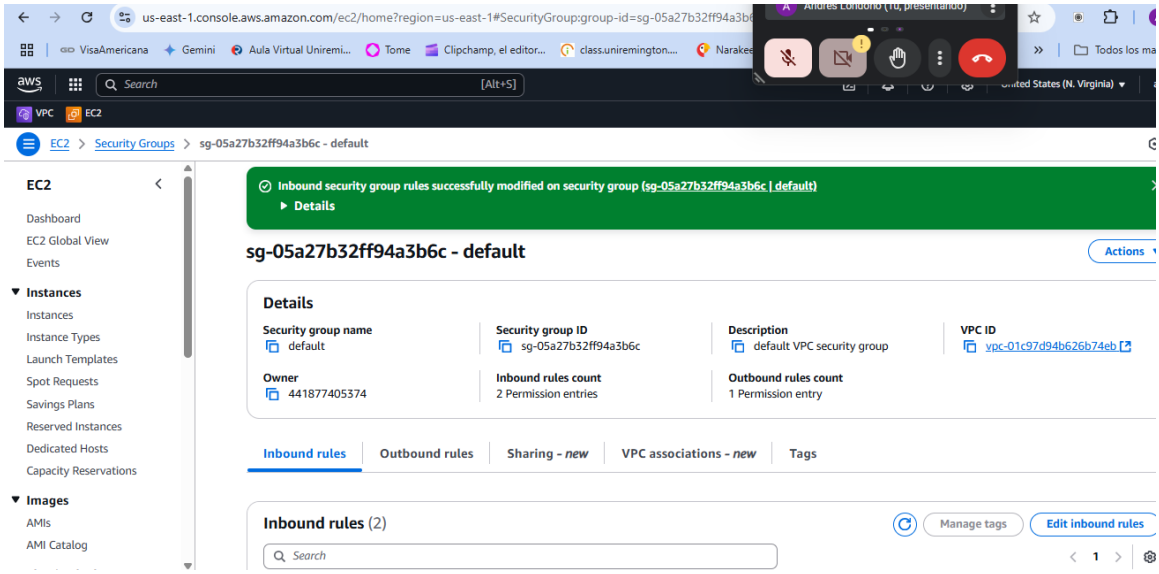
Vamos a las reglas de seguridad :



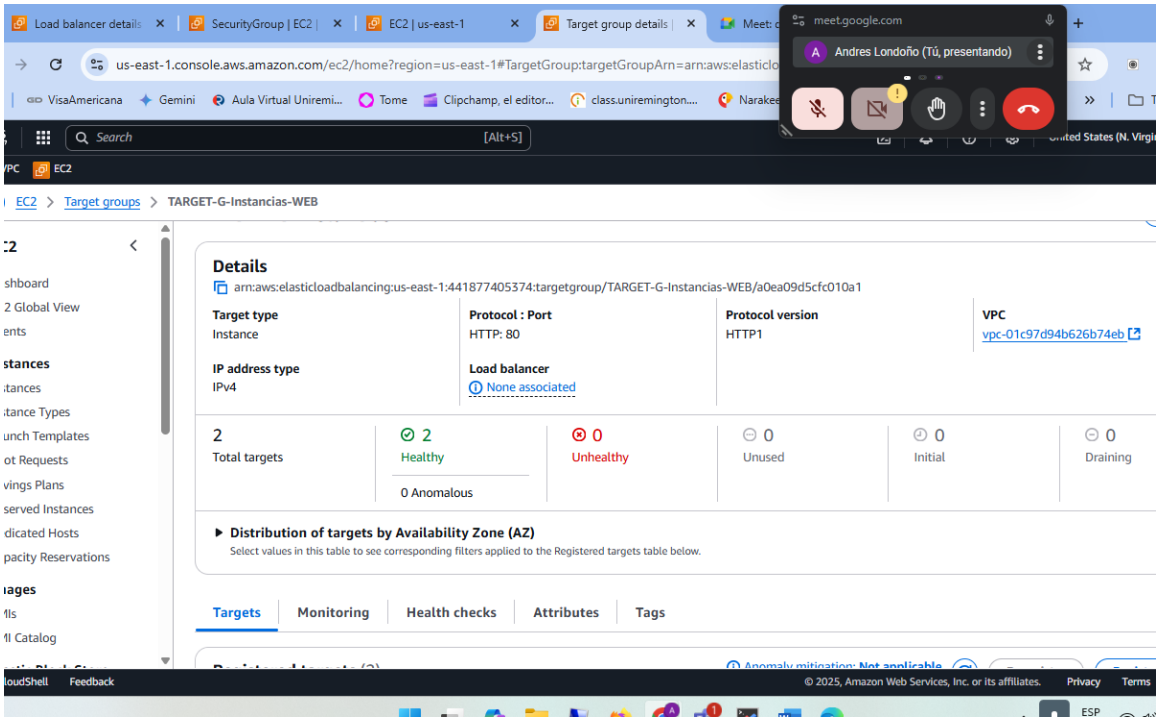


Creamos una regla para que el balanceador quede de la siguiente manera para permitir el tráfico por el puerto 80:

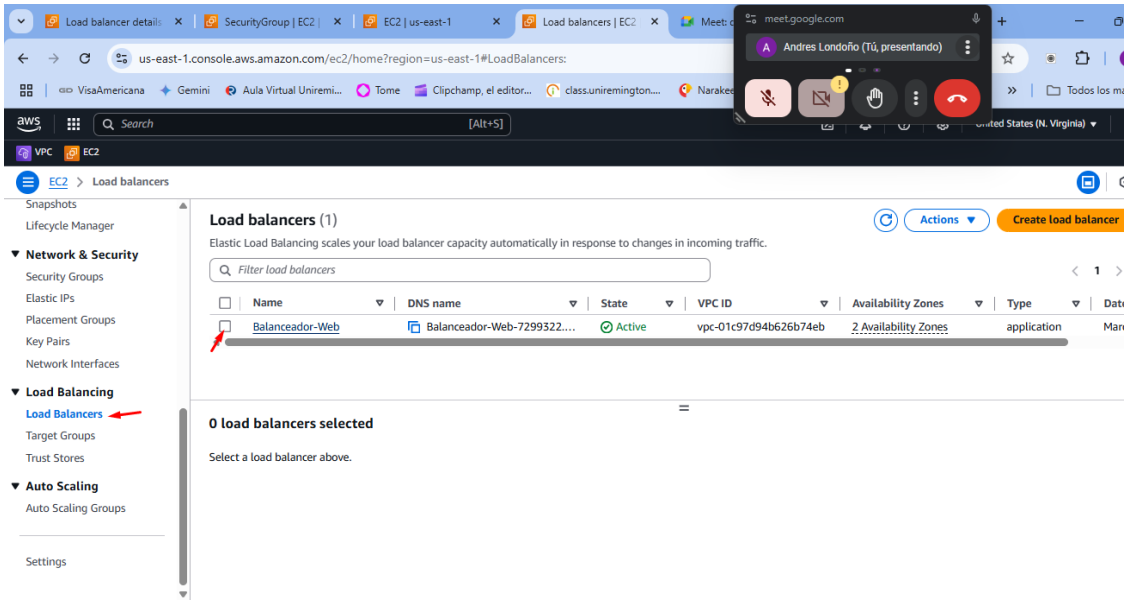


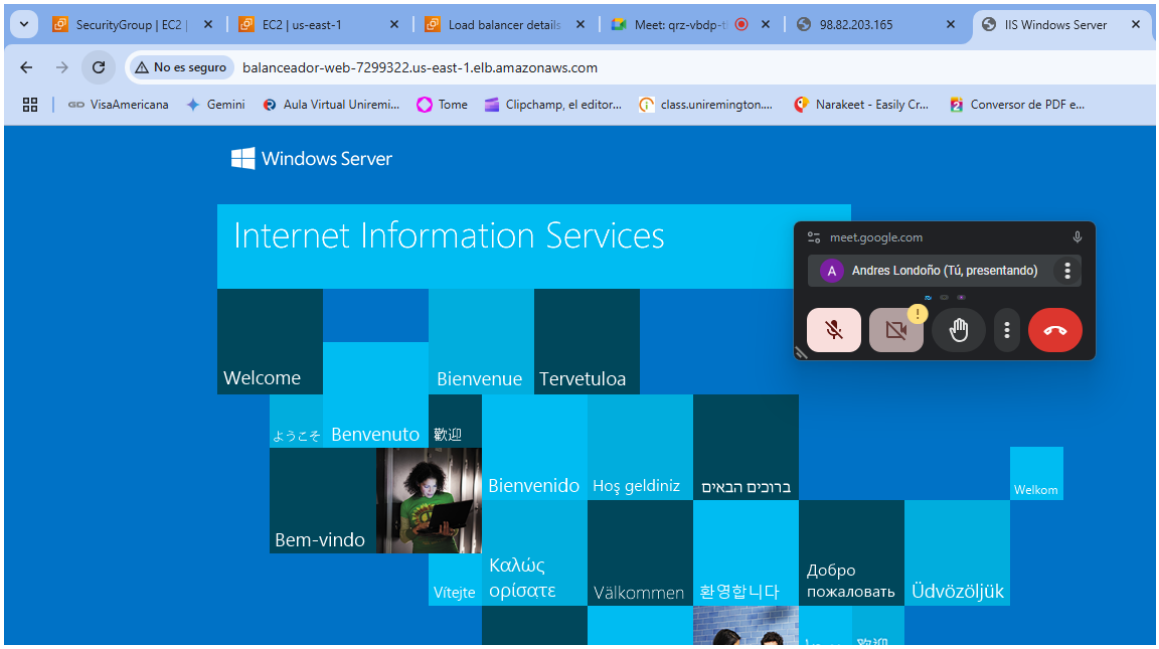


Así podemos garantizar que todos los servicios se encuentran activos y el balanceador activo:

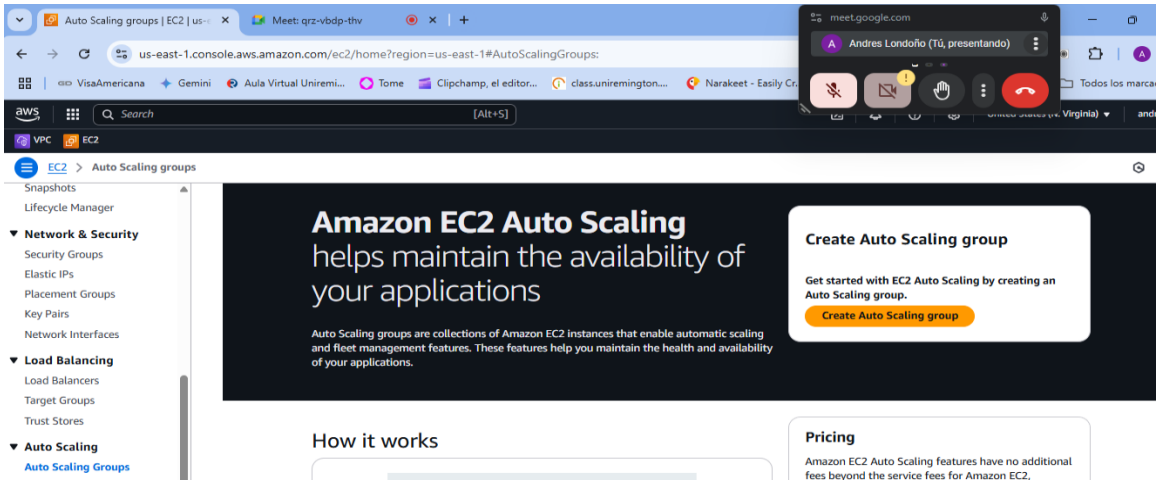


Seleccionamos Nuestro balanceador para poderlo identificar y sacar el DNS, dicho balanceador en este momento lo realizamos de manera manual: Balanceador-Web-7299322.us-east-1.elb.amazonaws.com:

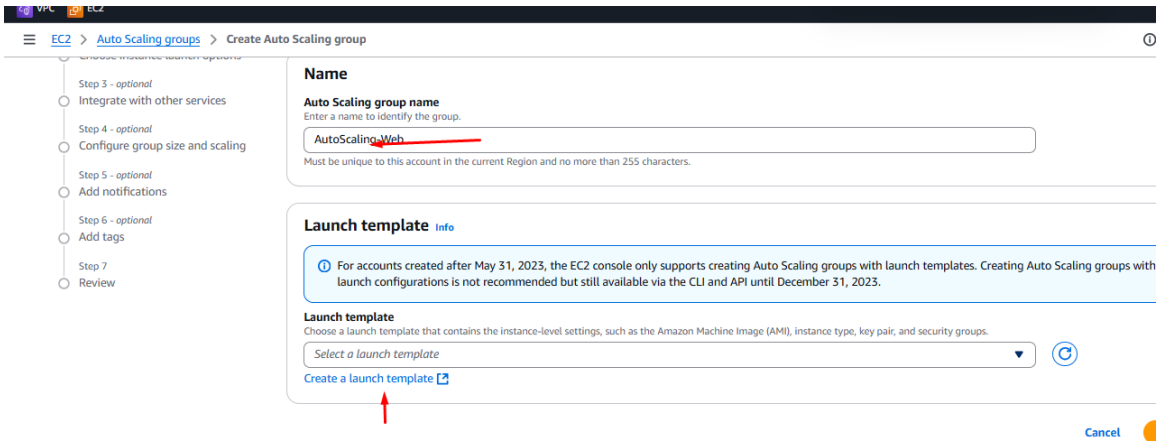


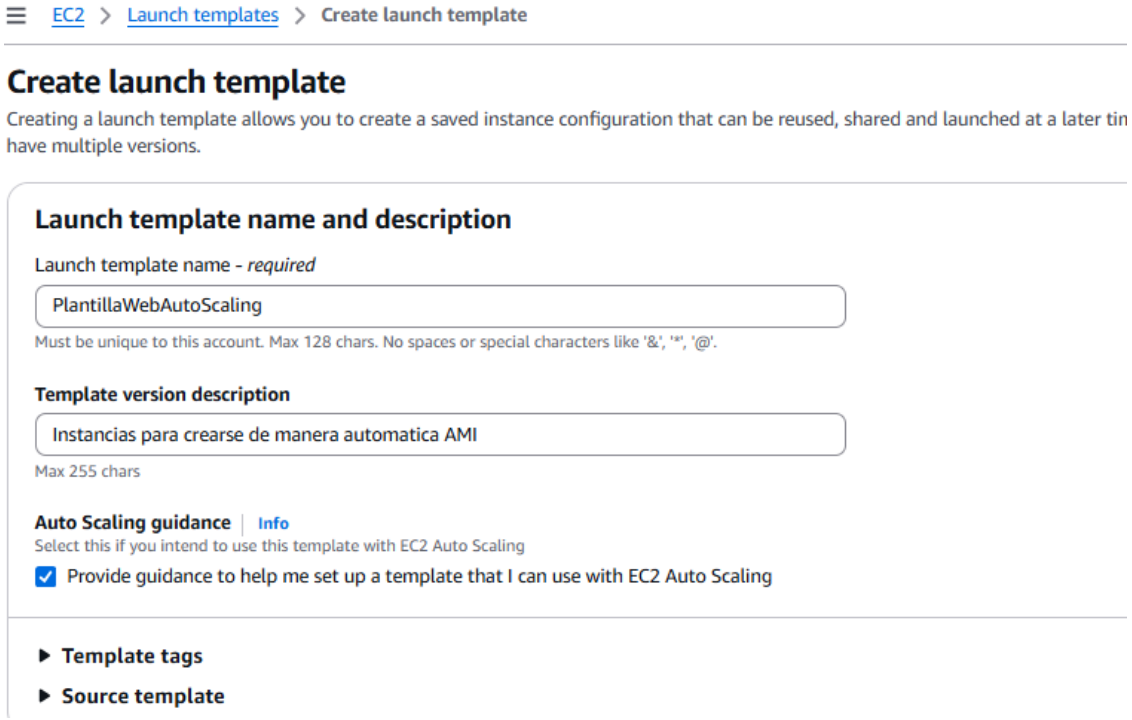


Para hacerlo de manera automática con la ayuda de AWS se realizaría de la siguiente manera(AUTOSCALING):

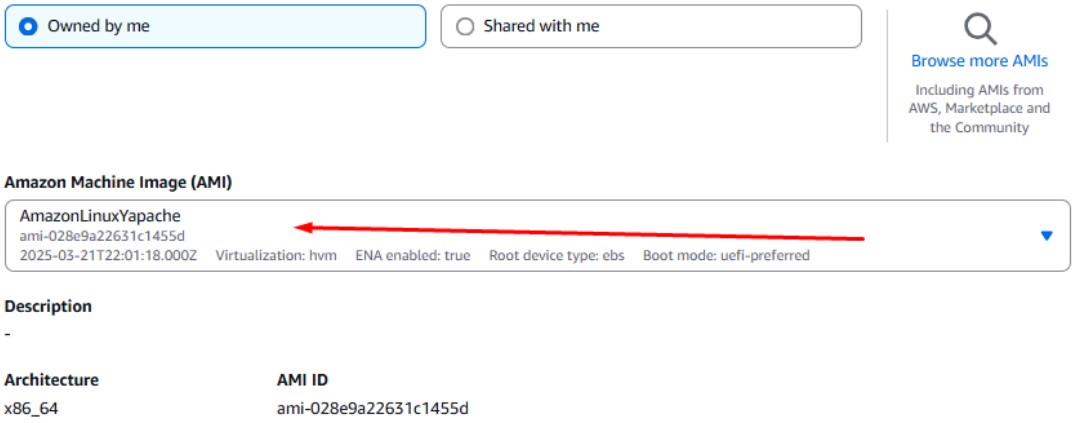


Creamos un nombre y adicional creamos un template ya que no tenemos ninguno por el momento para brindarle los parámetros que se requieran:

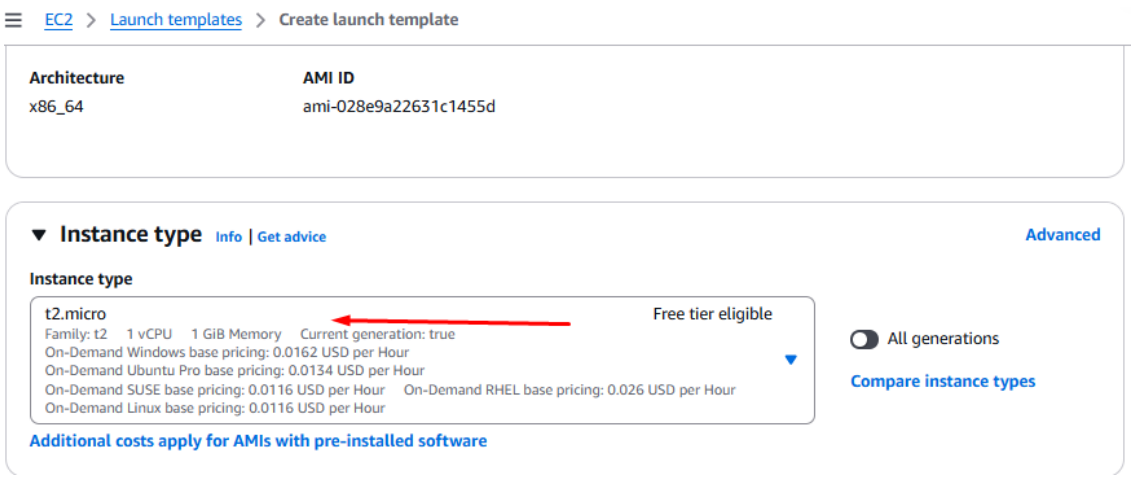




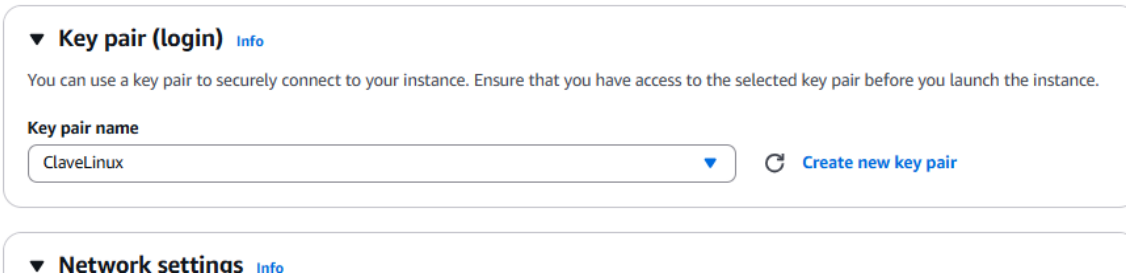
Seleccionamos la AMI correspondiente:



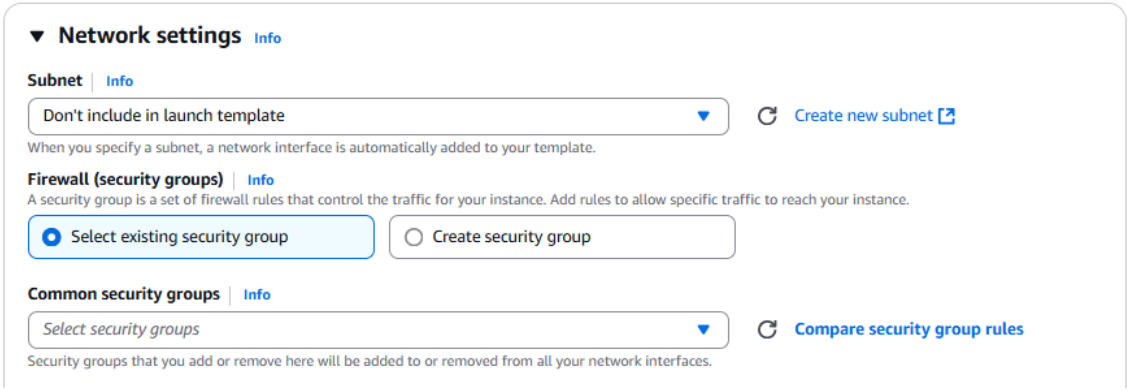
Seleccionamos la instancia(server):

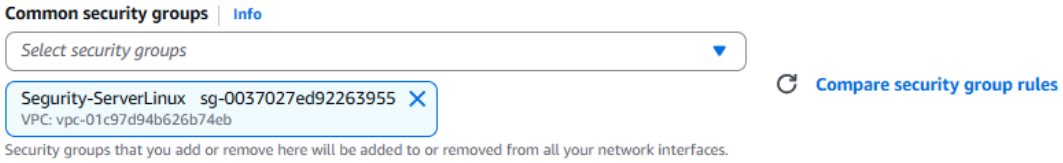


Seleccionamos la clave anteriormente creada:

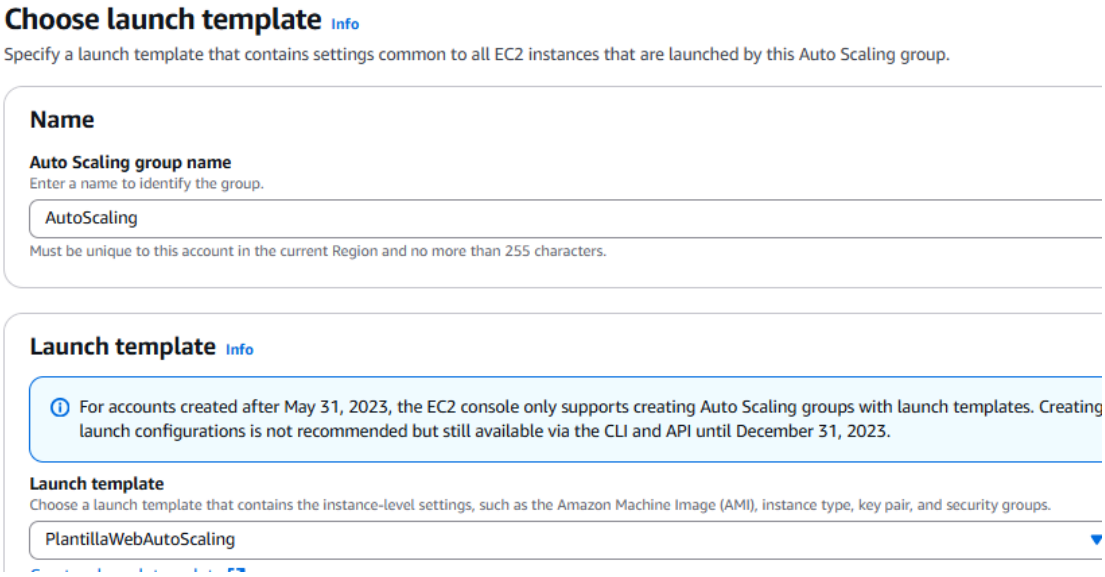


No seleccionamos ninguna sub red:

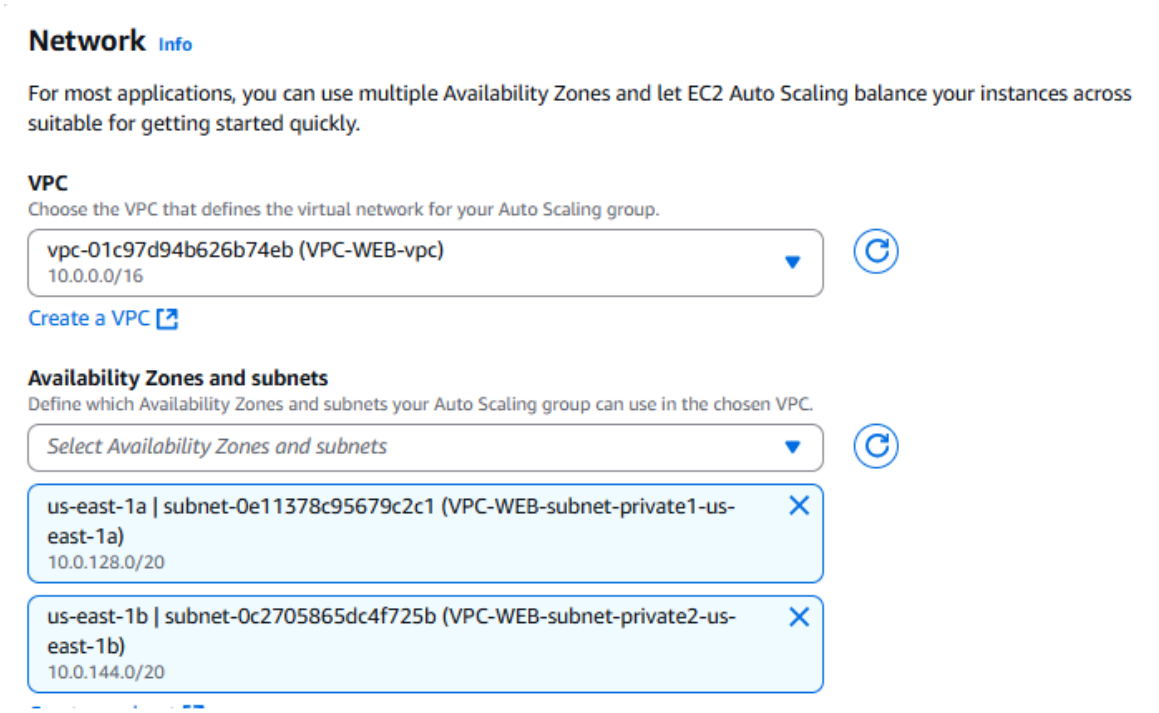




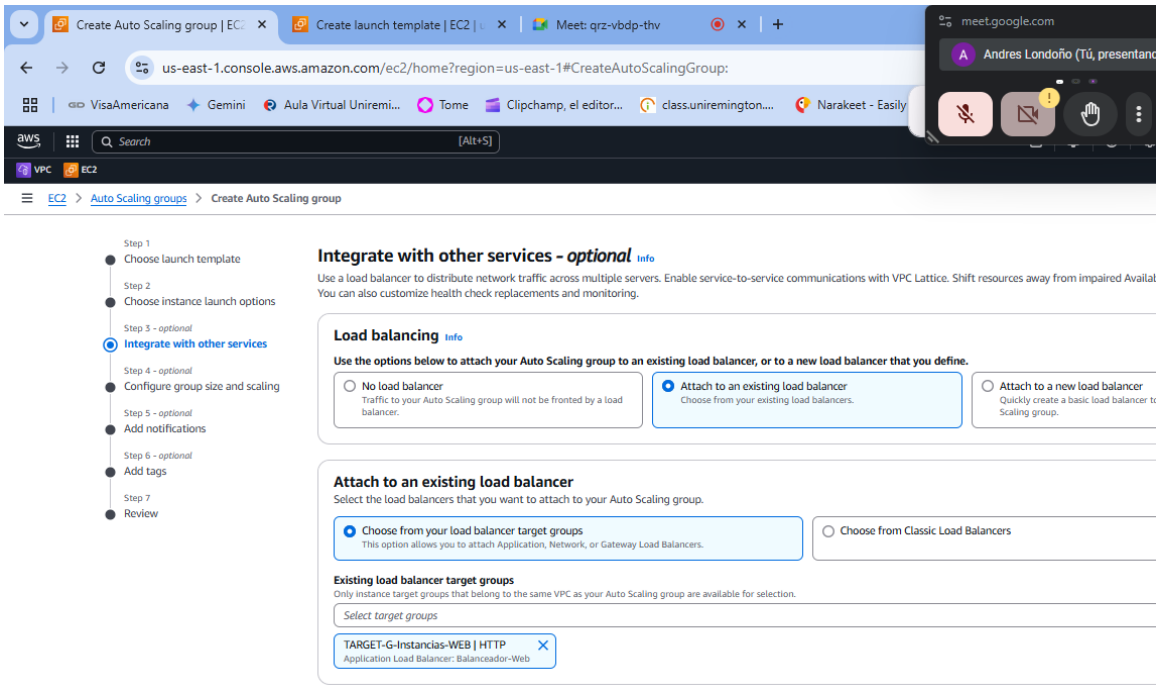
Volvemos al template:



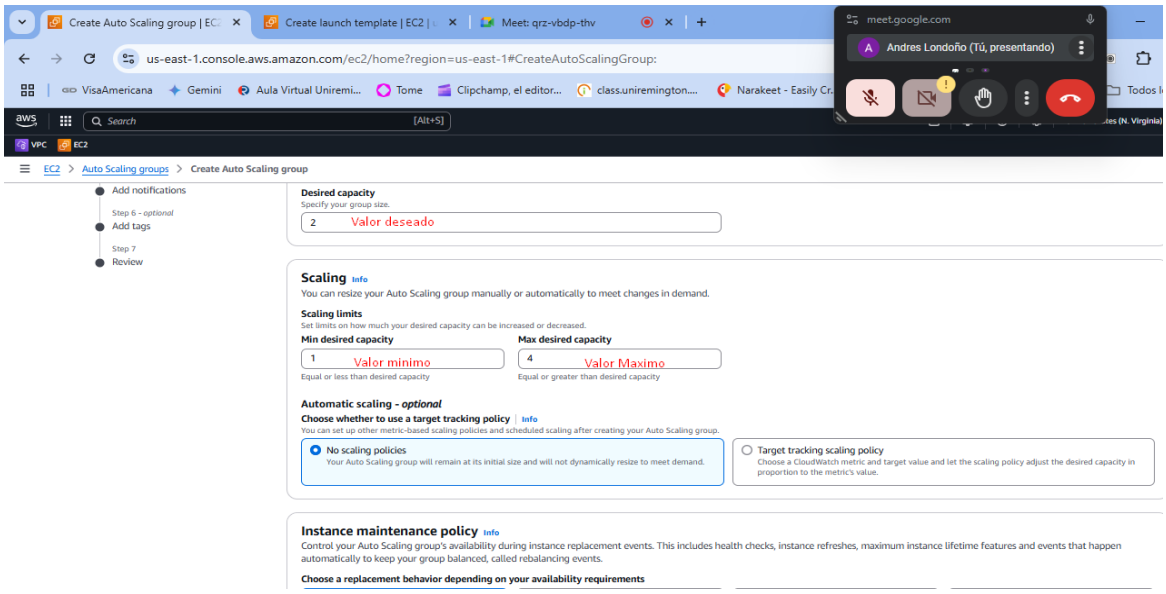
Seleccionamos el VPC creado anteriormente y adicional agregamos las subredes privadas:

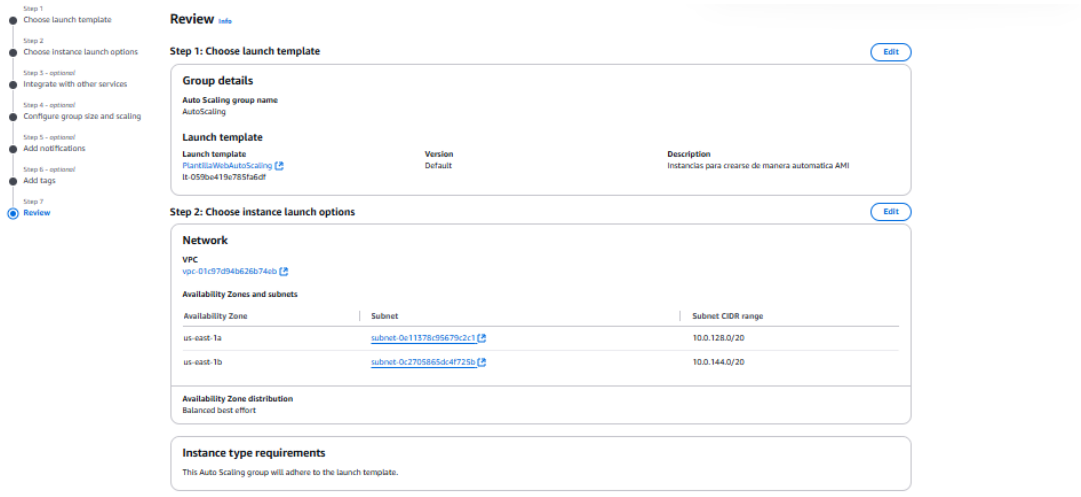


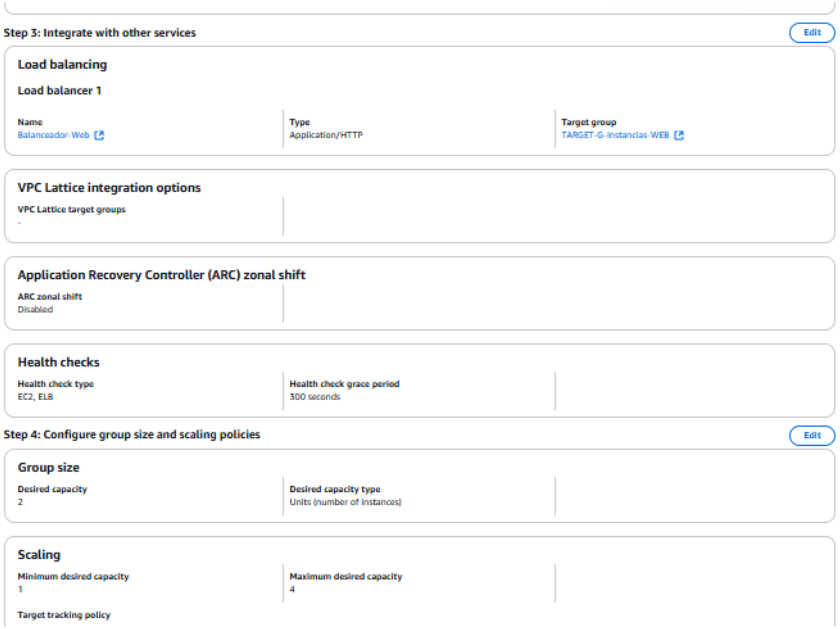
Asociamos el balanceador de carga:

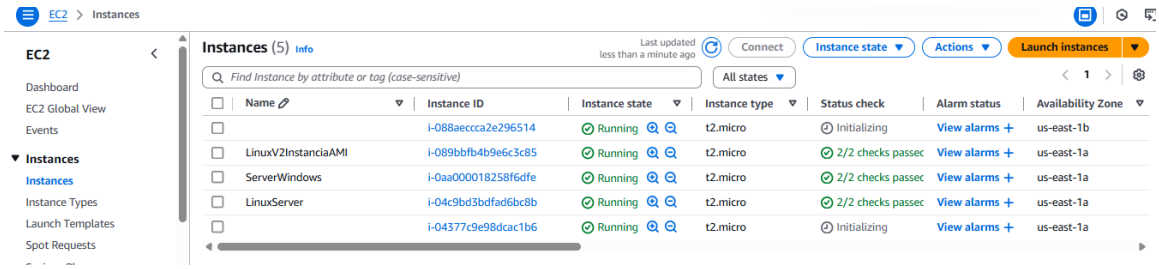


Acá miramos las variables y las cantidades de instancias mínimas, Máximas y deseadas. En el balanceador de carga:

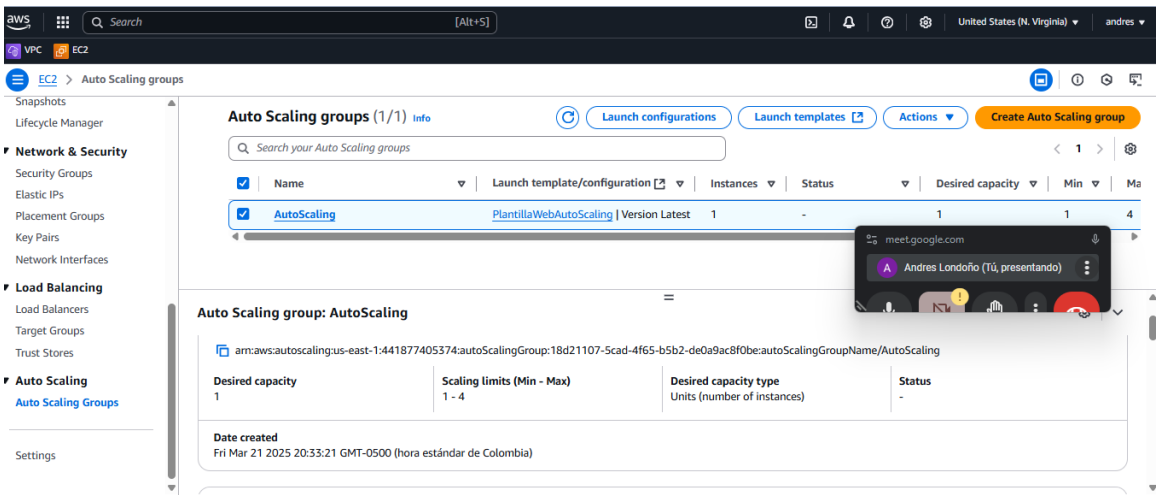


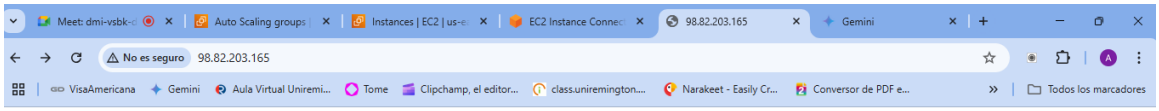






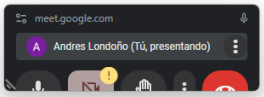
Podemos validar que el autoScaling funciona correctamente:

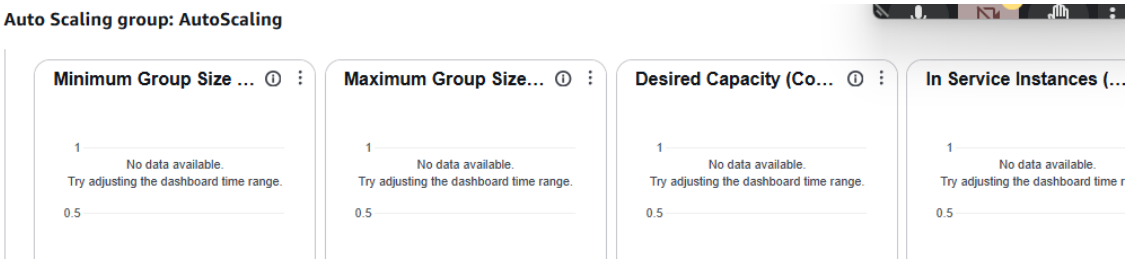




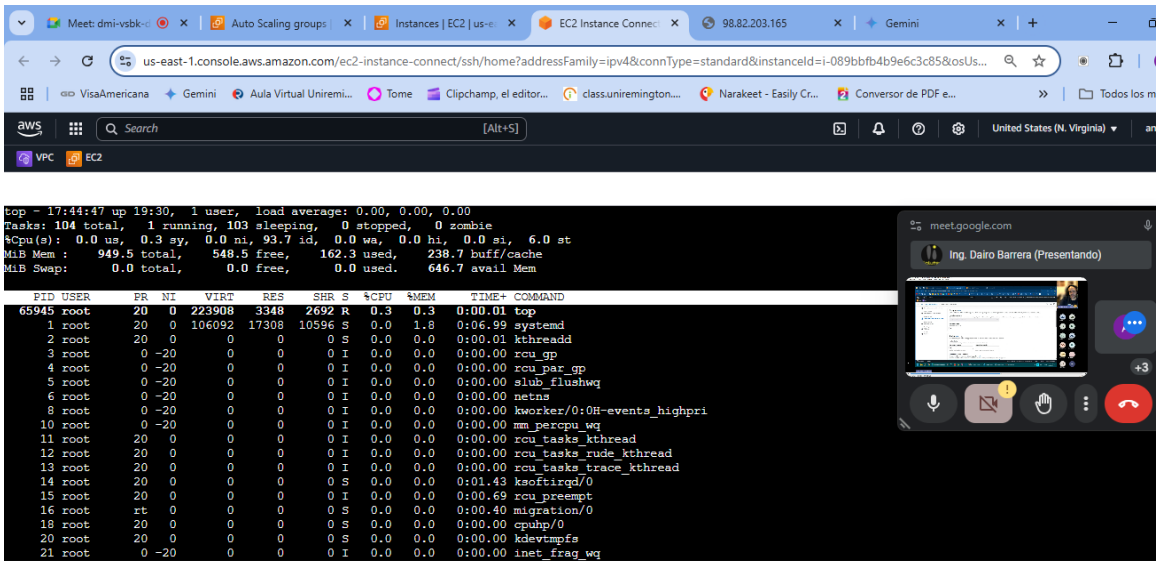
Hola este es el servidor de nuestro equipo

Andres Londoño, Anderson Garcia, Dairo Barrera

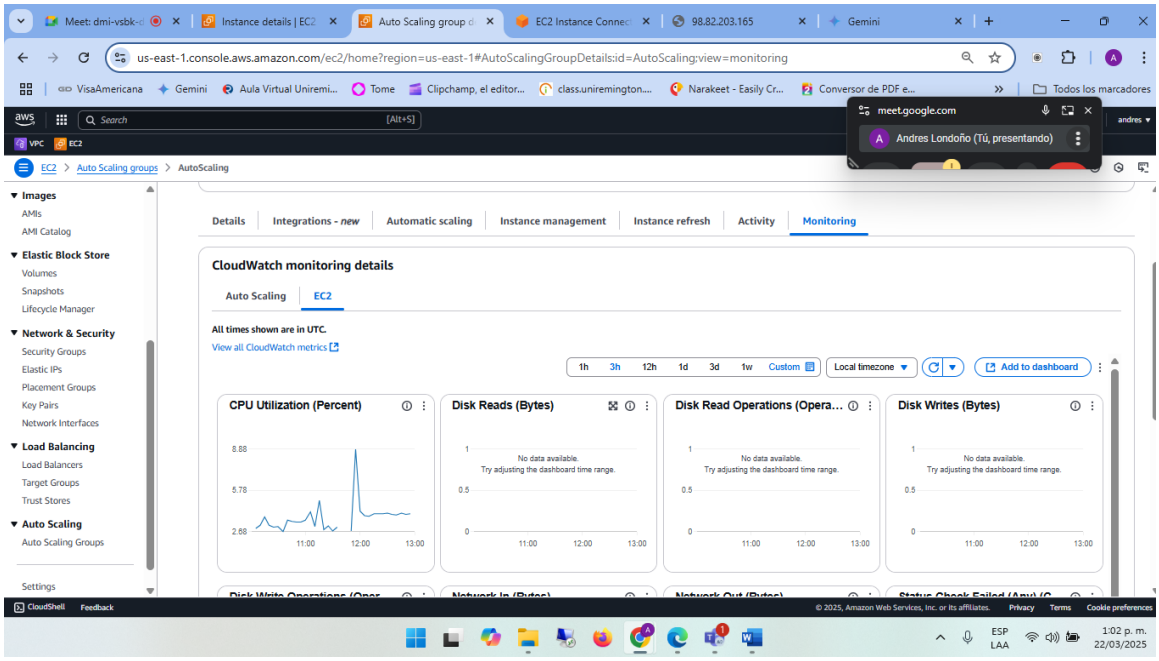




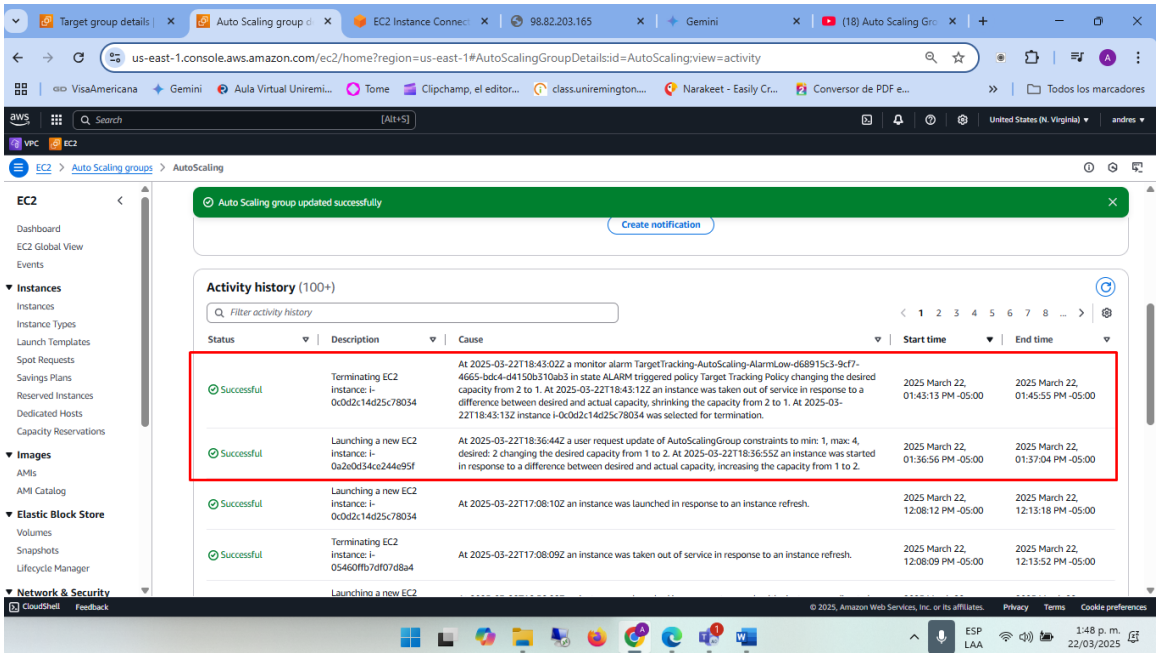
Validamos el rendimiento de la maquina:

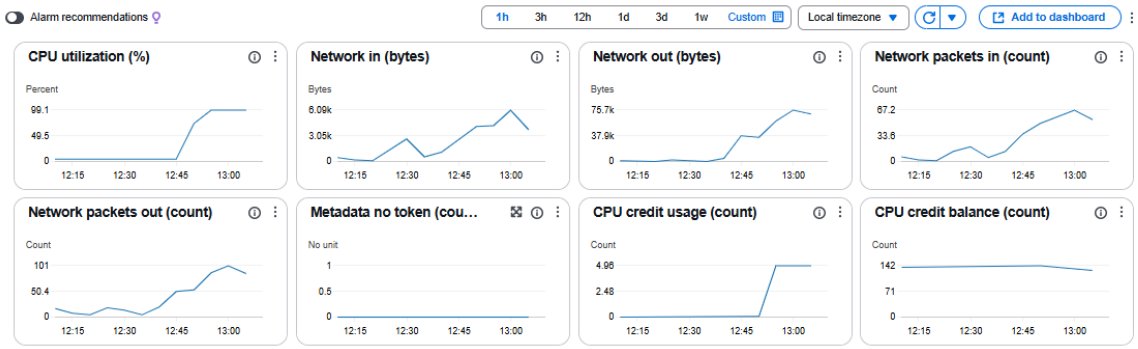


i-089bbfb4b9e6c3c85 (LinuxV2InstanciaAMI)



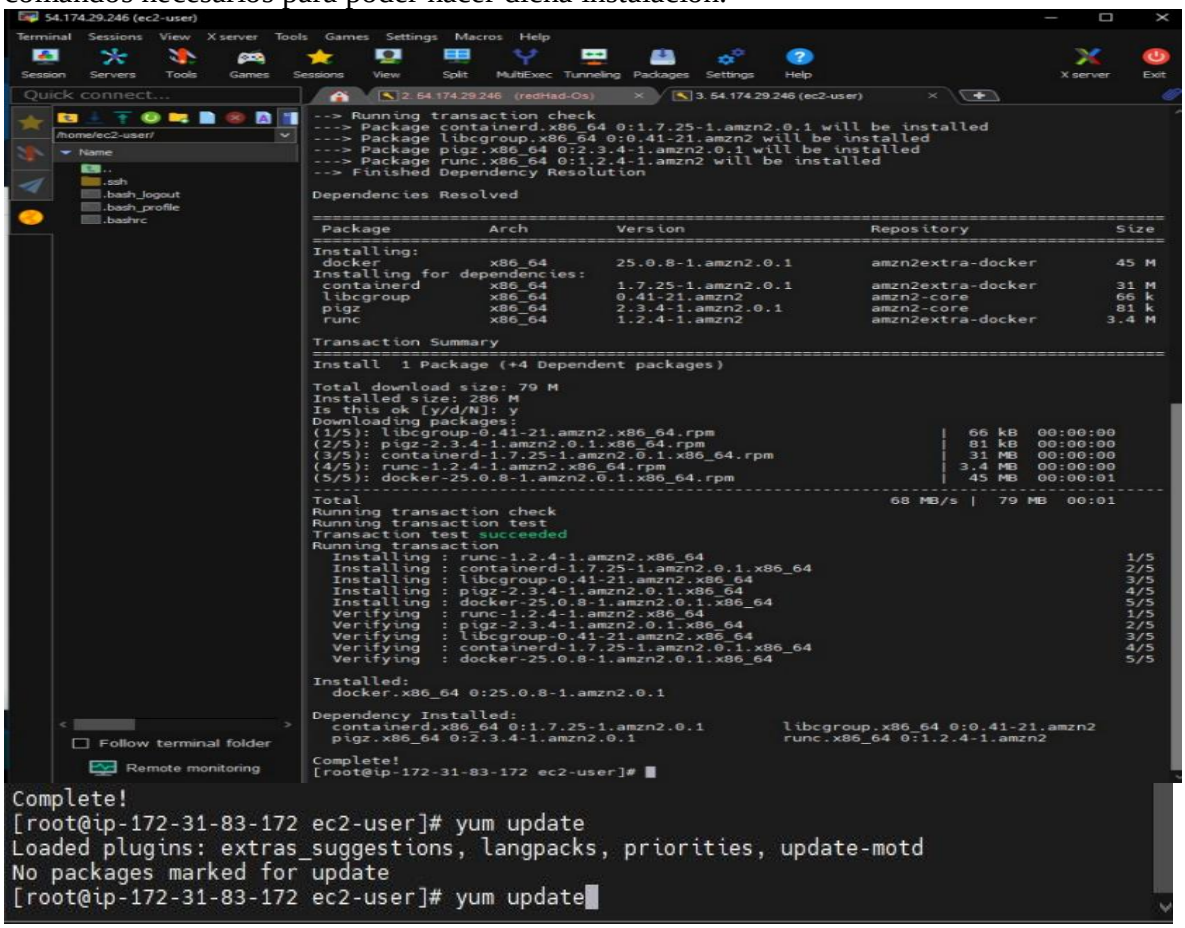
Como podemos observar después de enviar el comando para saturar las instancias:



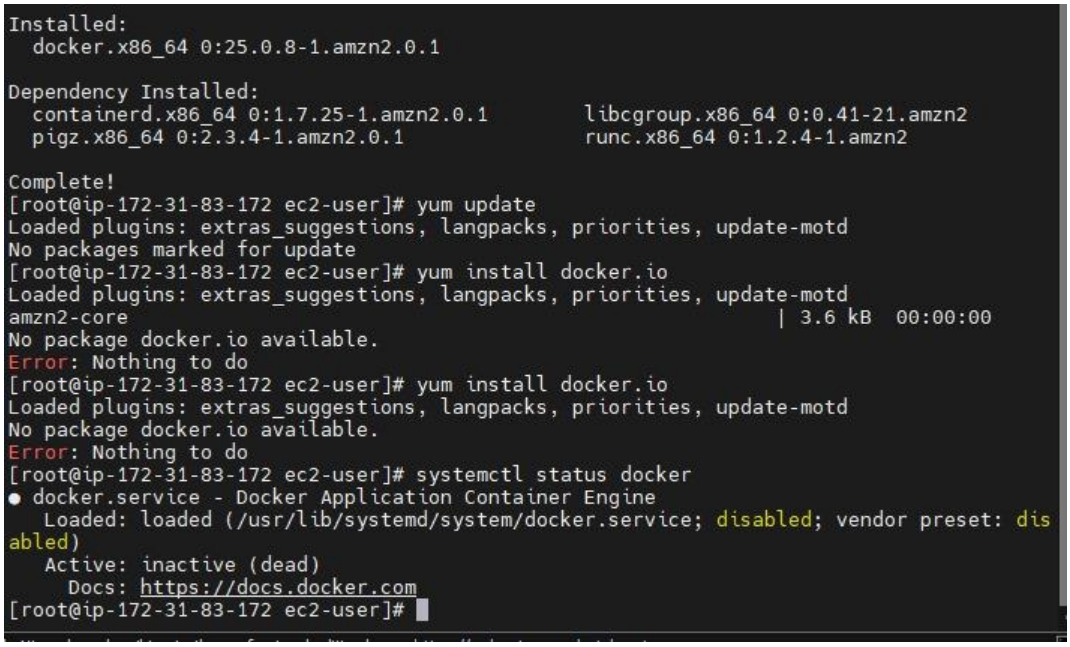

Alarm recommendations
1h
3h
12h
1d
3d
1w
Custom
Local timezone
Add to dashboard
CPU utilization (%)
Percent
99.1
49.5
0
12:15
12:30
12:45
13:00
Network in (bytes)
Bytes
6.09k
3.05k
0
Network out (bytes)
Bytes
75.7k
37.9k
0
Network packets in (count)
Count
67.2
33.6
0
Network packets out (count)
Count
101
50.4
0
Metadata no token (cou…
No unit
1
0.5
0
CPU credit usage (count)
Count
4.96
2.48
0
CPU credit balance (count)
Count
142
71
0

Segundo entregable practico **Docker**

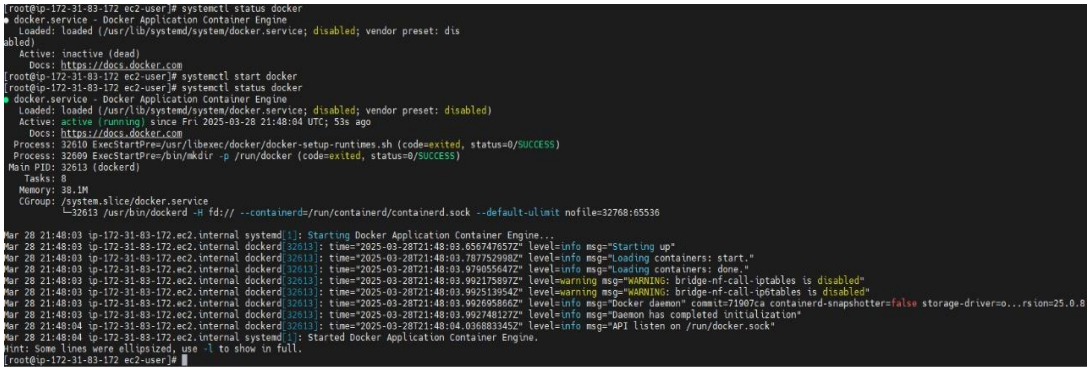
Ingresamos como administrador para poder ejecutar los comandos necesarios para la instalación de Docker y la desinstalación del Http, en la siguiente imagen se muestran los comandos necesarios para poder hacer dicha instalación.



Actualizamos los paquetes con el fin de validar que todos los componentes se cargaron de manera exitosa:

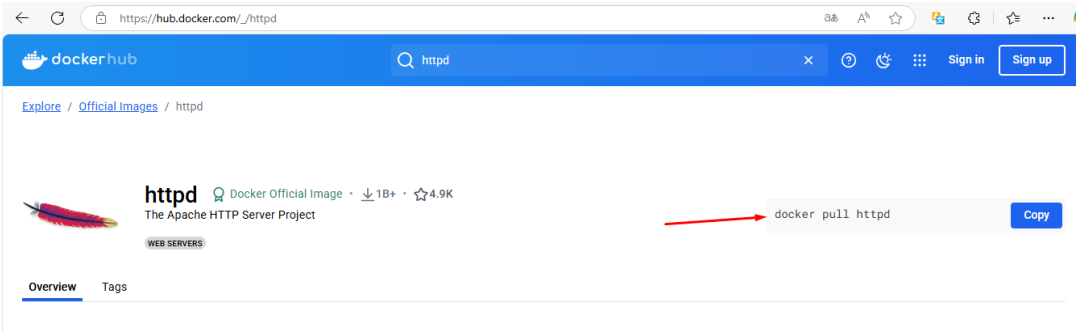


Después de dicha actualización Validamos que este corriendo de manera correcta, el servicio debe de estar activo y funcionando, El comando para correrlo es Start status :



Para activar el servicio y que arranque de manera automática ya debe de estar Corriendo, copiamos el siguiente comando systemctl enable docker:

Para utilizar la imagen de http la cual encontramos en la siguiente ruta [httpd - Official Image | Docker Hub](#) de dicha página necesitamos ingresar el siguiente comando docker pull httpd, en este caso la imagen es de httpd:

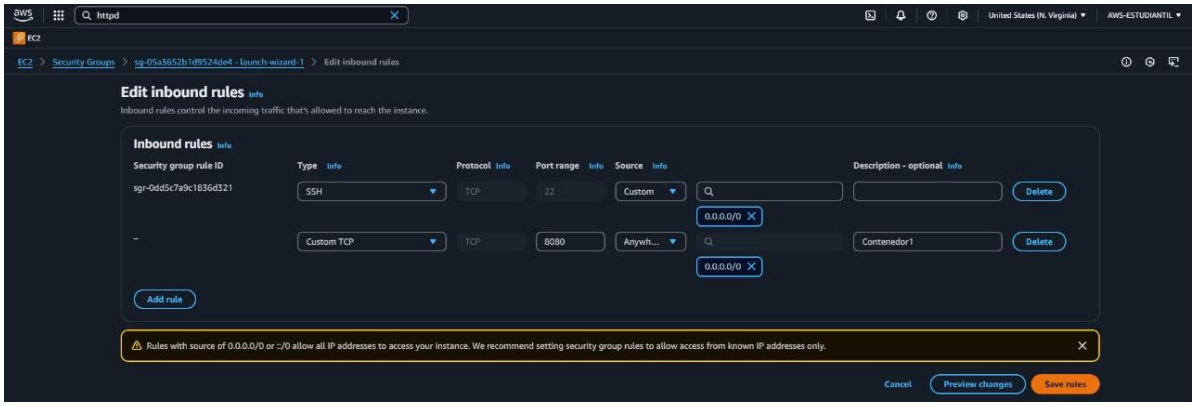



```
[root@ip-172-31-83-172 ec2-user]# docker pull httpd
Using default tag: latest
latest: Pulling from library/httpd
6e909acdb790: Pull complete
9f9b03a66afb: Pull complete
4f4fb700ef54: Pull complete
09bf08b13dbd: Pull complete
084c58879b9a: Pull complete
c61868f0ad74: Pull complete
Digest: sha256:391a8eb0c1ed464163da46099606a5ec293705118f3054d6c60f5957e2485bd0
Status: Downloaded newer image for httpd:latest
docker.io/library/httpd:latest
[root@ip-172-31-83-172 ec2-user]#
```

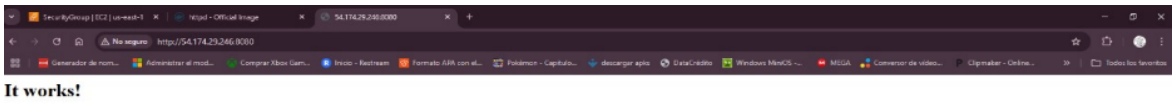
Después de haber utilizado el servicio de la imagen docker+httpd, Para crear el contenedor utilizamos el comando run, cada contenedor tiene un nombre único por lo tanto el puerto 8080 en este caso no se puede repetir en otro contenedor, ya que también es único y lo que se podría repetir es el puerto 80(si desea):

```
[root@ip-172-31-83-172 ec2-user]# docker run -dit --name dockerV1 -p 8080:80 httpd
b070adeee040a7b3d81c98a42da885ce5f65084b2c46810e98f4cc2f21d3ded4
[root@ip-172-31-83-172 ec2-user]#
```

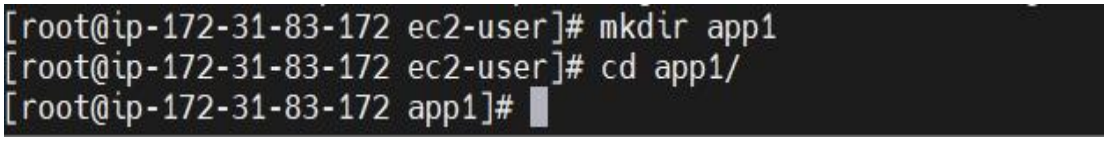
Habilitamos en nuestra instancia (server linux) el puerto 8080 del primer contenedor esto con el fin de poder garantizar el tráfico libre por dicho puerto, es decir, sin ninguna restricción:



Abrimos un navegador para validar que todo funciona perfectamente, es decir, validar que la IP de la instancia + el puerto en este caso 8080:80 no tenga ninguna restricción y nos redireccione a un contenedor en específico, para demostrarlo nos saldrá en la página el siguiente mensaje:

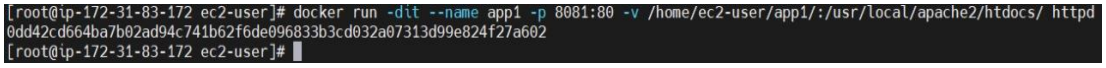


Volvemos a nuestra consola de administración para Crear una carpeta en la ruta del usuario ec2 de la siguiente manera, utilizamos el comando relacionado en la siguiente imagen:

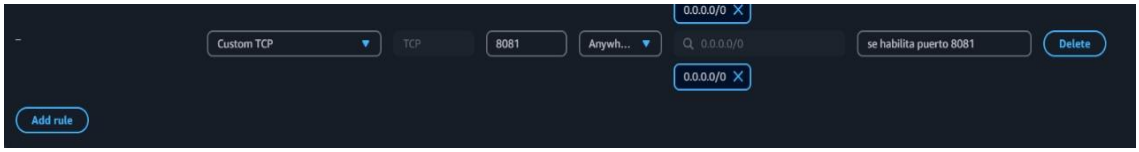


Después de estar posicionados en la carpeta App1, dentro de ella Creamos nuestro primer archivo el cual será denominado index.html , dicho archivo llevará nuestra primera página para el 1 contenedor de la siguiente manera:

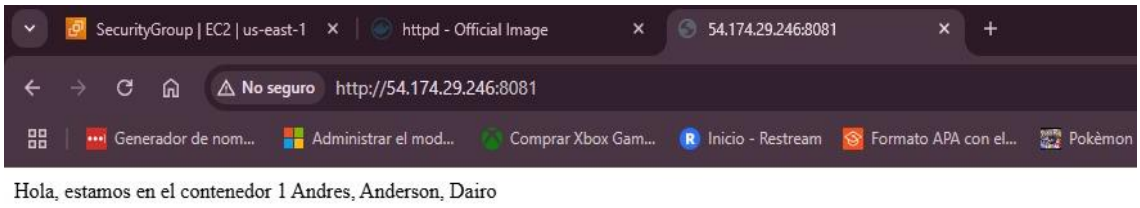
Creamos un enlace simbólico en el cual corremos el docker haciendo referencia a la carpeta anteriormente mencionada (App1) y a su puerto respectivamente, para que apache lo encuentre y poder tener nuestra rutas(carpetas) separada del contenedor y tener los archivos en un lugar diferente, tenerlo en el aws, Esto nos ayuda a tener un contenedor dinámico ya que tenemos los archivos separados.



Se habilita el tráfico en el puerto 8081 esto con el fin de dar vía libre a todo el tráfico sin que se genere alguna restricción de la siguiente manera:



Acá creamos nuestro primer contenedor con un archivo dinámico creado por nosotros el cual apunta a la IP “cabe aclarar que la IP puede variar ya que es un servicio que requiere de un pago, según políticas de aws” de la instancia con el puerto 8081 anteriormente habilitado, el resultado es:

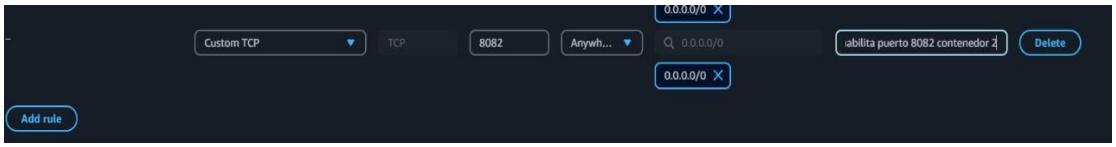


Volvemos a la consola y esta vez volvemos a pararnos en la ruta del usuario Ec2, Creamos otra carpeta llamada app2 con todos los permisos para evitar errores más adelante, en dicha carpeta debemos crear un archivo el cual llamaremos index.html” al hacer validaciones el index2.html nos genera error, por lo cual decidimos renombrarla a index.html”, el cual tiene un contenido totalmente diferente y apunta a otro contenedor:

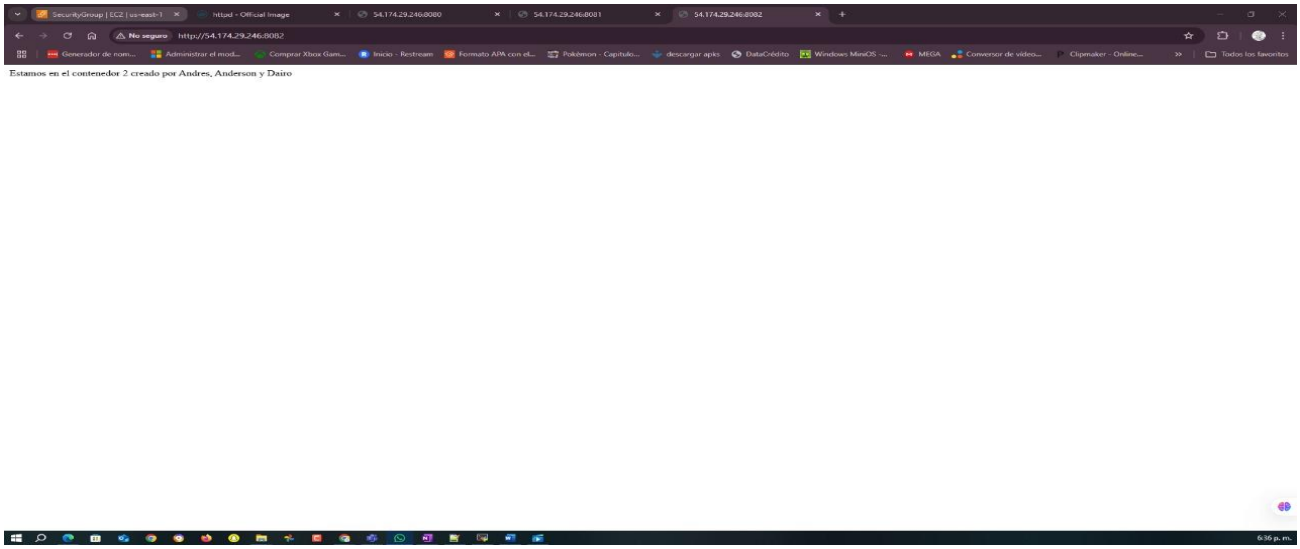
Creamos un enlace simbólico en el cual corremos el docker haciendo referencia a la carpeta anteriormente mencionada (App2) y a su puerto respectivamente, para que apache lo encuentre y poder tener nuestra ruta(carpetas) separada del contenedor y tener los archivos en un lugar diferente, tenerlo en el aws, Esto nos ayuda a tener un contenedor dinámico ya que tenemos los archivos separados.

```
[root@ip-172-31-83-172 app2]# cd ..
[root@ip-172-31-83-172 ec2-user]# chmod -R 777 app2/
[root@ip-172-31-83-172 ec2-user]# docker run -dit --name app2 -p 8082:80 -v /home/ec2-user/app2/:/usr/local/apache2/htdocs/ httpd
bbfe716c2bd02e9adcb5f4e69a9c9ebc6e6eafb28a5f7f2acfb0f0279a91626a
[root@ip-172-31-83-172 ec2-user]#
```

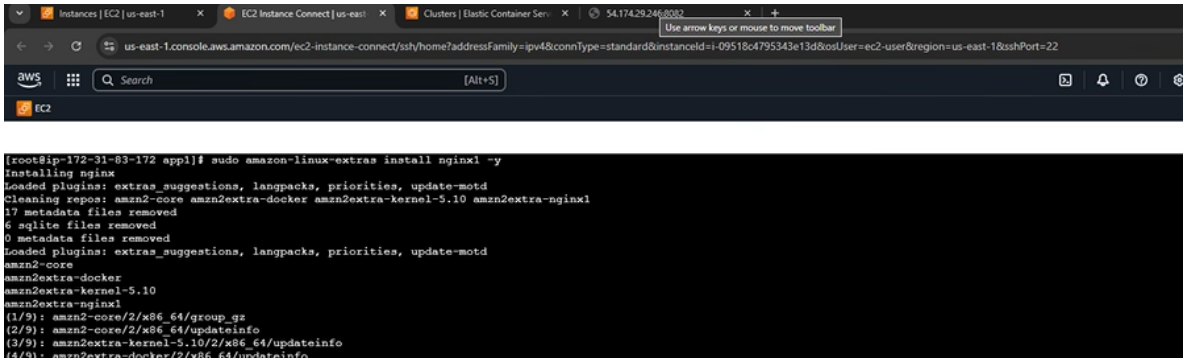
Habilitamos el puerto 8082 esto con el fin de habilitar el tráfico sin ningún problema y sin ninguna restricción para el siguiente contenedor el cual muestra un index.html totalmente diferente:



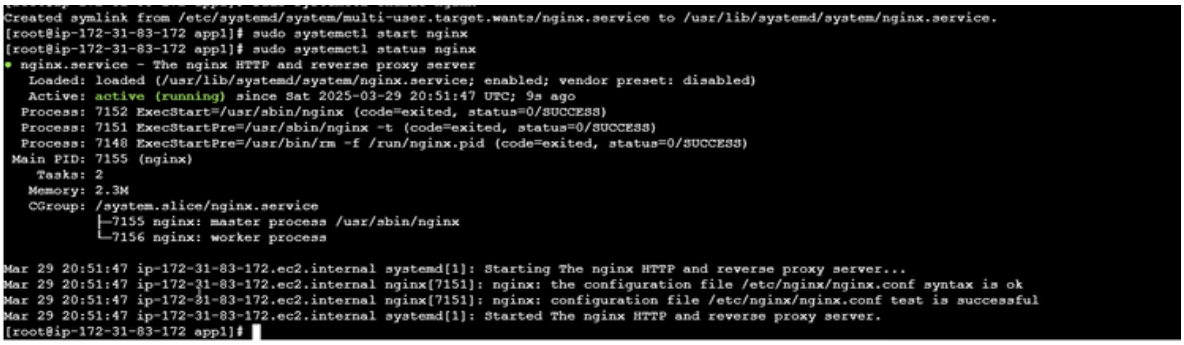
Al abrir la IP con el puerto 8082 podemos visualizar que la página abre con toda ormalidad y Quedaría de la siguiente manera:



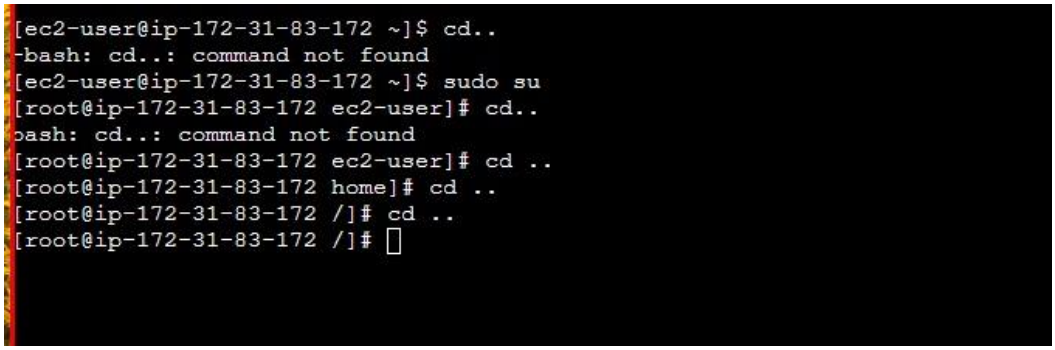
Instalamos ngnex esto con el fin de poder generar el proxy reverso, es decir, poder utilizar las rutas sin necesidad de ingresar con un puerto en específico, se haría de la siguiente manera:



Después lo habilitamos en el sistema para que corra de manera automática cuando la maquina se apague suban los servicios sin necesidad de líneas de código:

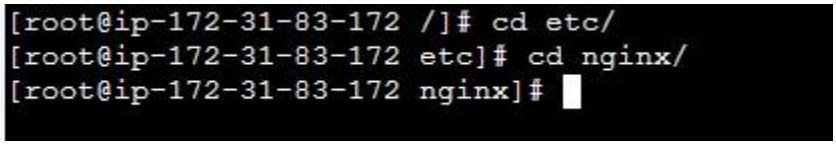


Ingresamos nuevamente a nuestra consola con el usuario root esto con el fin de Navegar hacia la carpeta raíz para acceder a la carpeta de configuración como se muestra en las siguientes imágenes:

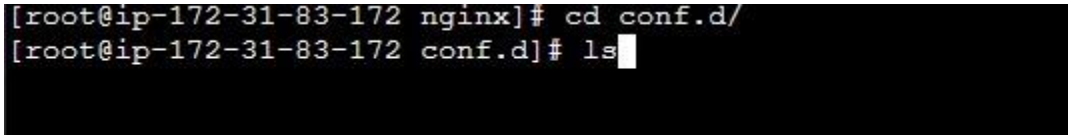


La carpeta es etc como se puede visualizar en la imagen, navegamos para poder ingresar a esa:

Después de estar parados en la carpeta etc nos movemos hacia la carpeta llamada nginx utilizando el comando Cd como se muestra en la siguiente imagen:



Después de la imagen anterior ingresamos a otra carpeta llamada conf.d la cual es la última carpeta a la cual nos debemos dirigir:



Creamos un archivo de configuración “dicho archivo puede llevar el nombre que más les guste ”en nuestro caso le llamamos proxy-containers.conf , cabe resaltar que la extensión .conf debe de ir si o si con nuestro archivo, la siguiente manera con el virtualizador(editor de archivos) vim :

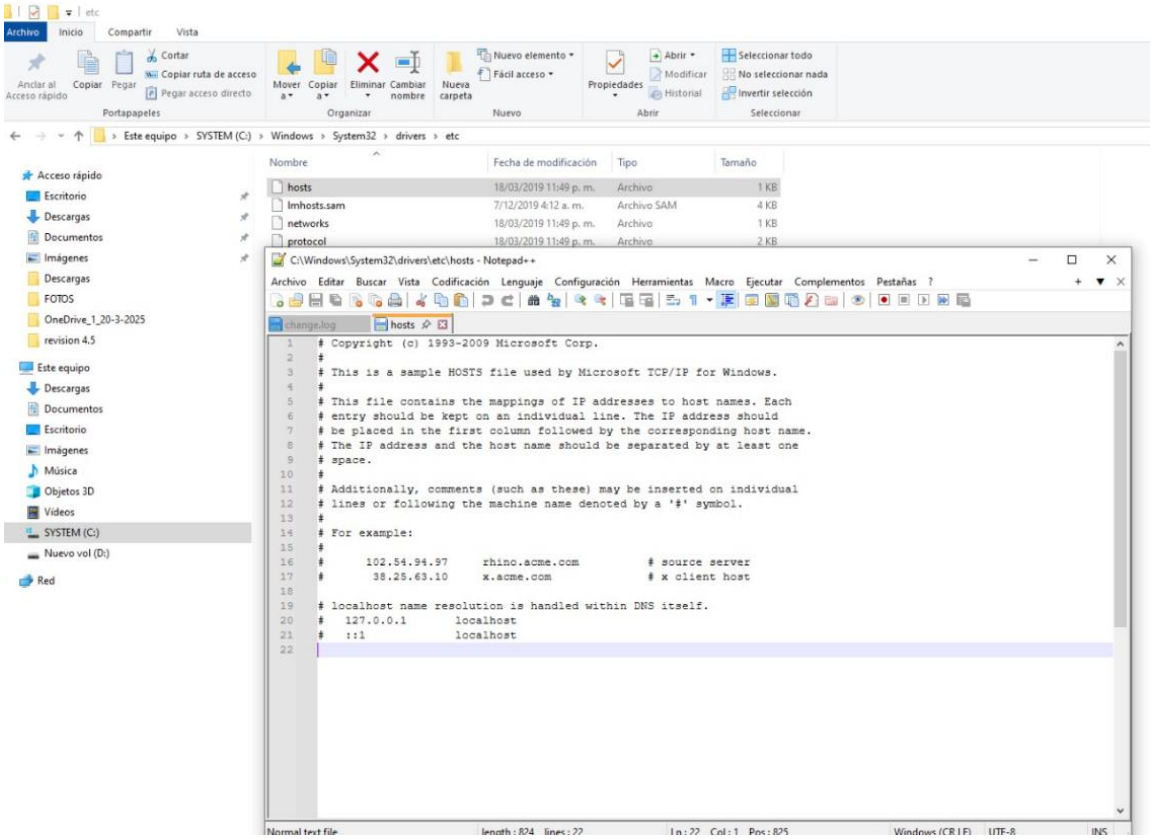
```
[root@ip-172-31-83-172 conf.d]# vim proxy-containers.conf
```

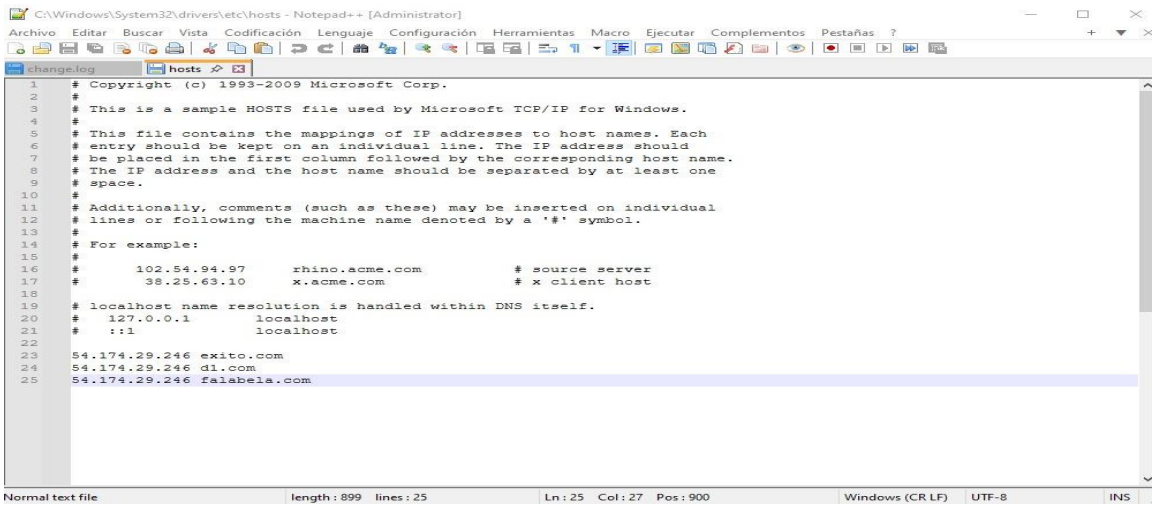
La configuración que mostraremos a continuación es la de “escucha sin necesidad de poner puertos” del archivo proxy-containers.conf para los contenedores:

```
server {
listen 80;
server_name exito.com;

location / {
        proxy_pass http://54.174.29.246:8081;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
}
}

server {
listen 80;
server_name falabela.com;

location / {
        proxy_pass http://54.174.29.246:8082;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
}
}
~
~
~
~
~
~
~
~
~
~
~
```

- Utilizamos el server para abrir nuestro contenido.
- Listen es para escuchar por el puerto 80;
- Server_name: es para ponerle la “URL” ,“DNS” a la cual nos va a redirigir. Esto lo debemos configurar en el host del pc local para poder que se visualice el ejercicio.
- Proxy_Passs: es para redirigir a otro servidor en este caso la IP del server la redirige con el puerto 8081,8082, etc para el dominio en este caso éxito o Falabella, etc .
- Hacemos el mismo procedimiento para todos nuestros contenedores

Después de tener esto configurado vamos a buscar el archivo host del pc que se encuentra en la siguiente ruta C:\Windows\System32\drivers\etc acá ponemos la IP de la instancia más el nombre que se anexo al archivo proxy-containers.conf :





Validamos que el puerto 80 en nuestra instancia se encuentre abierto para la salida a internet y sin ninguna restricción para que los ejercicios anteriormente mencionados puedan tener los resultados esperados:

Name	Security group rule ID	Port range	Protocol	Source	Security groups	Description
–	sgr-0113d20b71a1eb78b	80	TCP	0.0.0.0/0	launch-wizard-1	From internet
–	sgr-0185e8e61affe4948	8081	TCP	0.0.0.0/0	launch-wizard-1	se habilita puerto 8081
–	sgr-0dd5c7a9c1836d321	22	TCP	0.0.0.0/0	launch-wizard-1	–
–	sgr-098f7dd9da0d092ca	8080	TCP	0.0.0.0/0	launch-wizard-1	Contenedor1
–	sgr-0251d9e3b0f0a4e0d	8082	TCP	0.0.0.0/0	launch-wizard-1	se habilita puerto 8082 contene

Si se siguieron los pasos correctamente el ejercicio debería de quedar de la siguiente manera:

Estamos en el contenedor 2 de FALABELA creado por Andres, Anderson y Dairo

Hola, estamos en el contenedor 1 de SURA Andres, Anderson, Dairo

Estamos en el conetenedor 3 del EXITO Andres, Anderson Dairo

Conclusiones

La práctica realizada permitió comprender los conceptos fundamentales de la virtualización, la contenerización de aplicaciones y la migración hacia la nube. La utilización de herramientas como Docker y Portainer facilita el manejo de servicios en contenedores y permite replicar entornos de forma eficiente. Asimismo, se evidenció que AWS proporciona una plataforma robusta para ejecutar entornos previamente desarrollados en local, favoreciendo la escalabilidad y disponibilidad de los servicios, volviendo se una excelente opción para las empresas que desean invertir en nuevas tecnologías buscando así una interacción sencilla y desnaturalizada de la tecnología por lo cual no es necesario preocuparse de administrar servidores o inclusive las aplicaciones de manera física corriendo así el riesgo de una perdida de información ya sea por malos manejos, empleados inexpertos, saqueo, daños naturales, apagones, etc. Todo esto se traduce en administrar de una manera más escalable y sinfónica.

La virtualización de nuestras tecnologías llego para quedarse, AWS es una buena opción para ello, solo con dar unos cuantos clicks podemos administrar de manera sencilla y eficaz nuestros productos y servicios sin perder la productividad o correr el riesgo de perderla

Referencias

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

Portainer.io. (2023, 12 julio). *Kubernetes and Docker Container Management Software*. https://www.portainer.io/

Ubuntu Server documentation. (s. f.). Ubuntu Server. https://ubuntu.com/server/docs

.