Instituto Politécnico Nacional

Escuela Superior de Cómputo

Sistemas Distribuidos

Profesor:

Carlos Pineda Guerrero

Asignatura:

Tarea 1. Implementación de un proxy inverso HTTPS y servidores HTTP en la nube.

Grupo:

7CV1

Alumna:

Mateo García Alejandra Michelle

Introducción

El presente documento aborda la implementación práctica de una arquitectura de sistemas distribuidos centrada en un proxy inverso HTTPS y múltiples servidores HTTP de backend, desplegada en un entorno de computación en la nube (Microsoft Azure).

Un proxy inverso es un componente fundamental que actúa como intermediario a nivel de aplicación (Capa 7 de OSI), situándose estratégicamente frente a los servidores de origen. Su función principal es aceptar las conexiones entrantes de los clientes y redirigirlas a uno o varios servidores backend. Esta arquitectura centralizada ofrece beneficios críticos en entornos de producción:

- Seguridad: Oculta la topología interna de la red y la identidad de los servidores de origen, actuando como una primera capa de defensa contra ataques cibernéticos. Además, facilita la terminación SSL/TLS (SSL Offloading), centralizando la gestión de certificados y liberando recursos de cómputo en los servidores de contenido.
- Balanceo de Carga: Permite la distribución eficiente del tráfico entre múltiples servidores, asegurando la alta disponibilidad y la escalabilidad horizontal del servicio.
- Optimización del Rendimiento: Soporta el almacenamiento en caché de contenido estático y la compresión de respuestas para acelerar la entrega de datos al cliente.

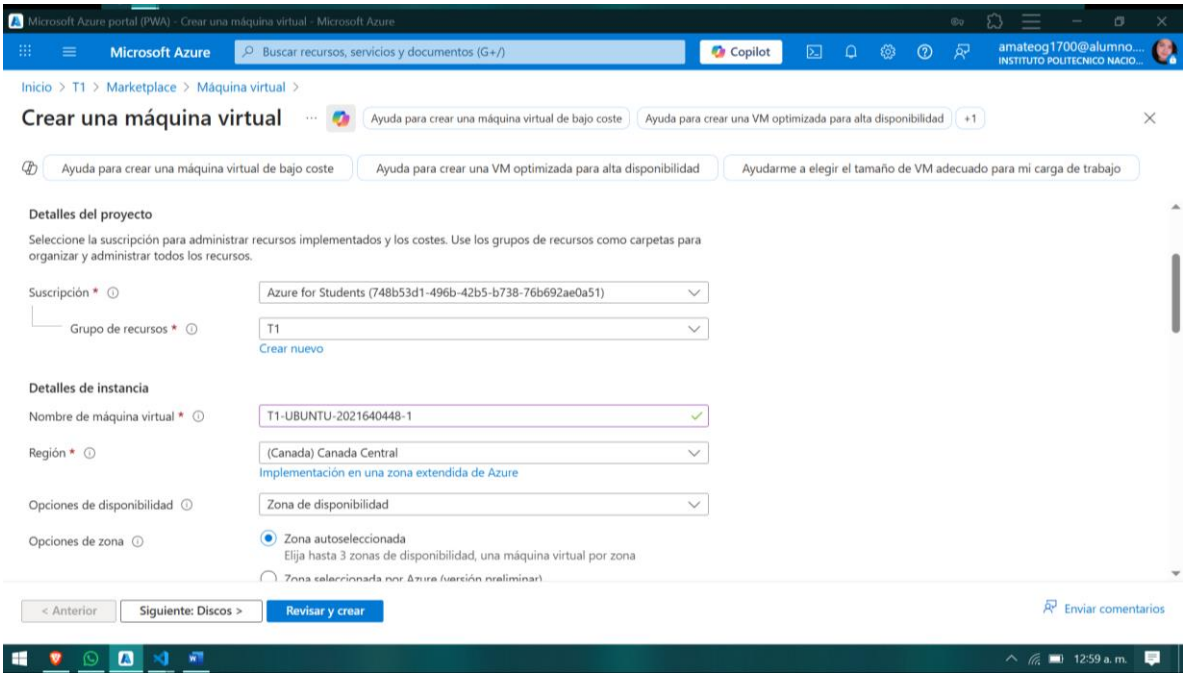
La tarea requirió el desarrollo y despliegue de: 1) un servidor HTTP/1.1 con soporte para el mecanismo de caché condicional (utilizando los encabezados Last-Modified e If-Modified-Since); 2) un proxy inverso en Java (AdministradorTrafico.java y AdministradorTraficoSSL.java) para gestionar peticiones HTTP y HTTPS y demostrar la capacidad de redirección de tráfico selectiva a dos backends ; y 3) la configuración de un entorno de máquinas virtuales con Ubuntu Server y Windows Server en la plataforma Azure, incluyendo la configuración de reglas de puerto (iptables) y certificados autofirmados.

Desarrollo

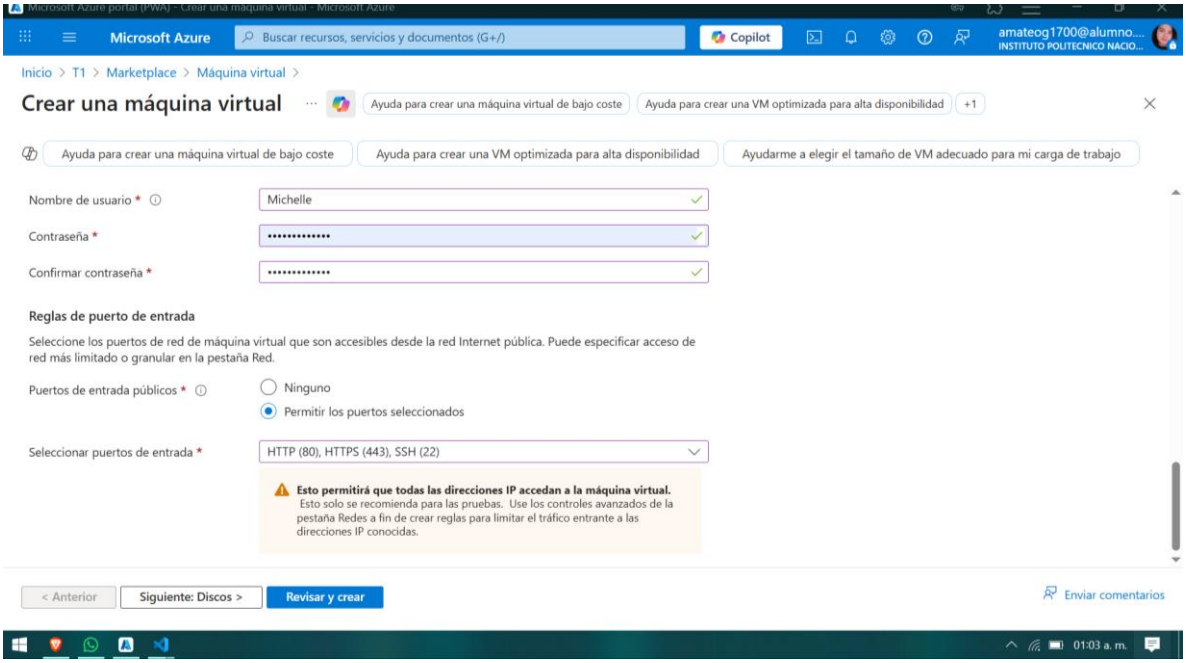
Para el desarrollo de esta práctica es necesario hacer la Implementación del uso de la caché. El servidor deberá enviar al navegador en cada respuesta el encabezado Last-Modified y deberá procesar el encabezado If-Modified-Since que envíe al navegador e implementar la funcionalidad de servidor web. Cuando el servidor reciba una petición "GET /archivo.html HTTP/1.1" se deberá leer el archivo.html del disco local y se deberá regresar al navegador el contenido del archivo con el tipo mime "text/html". En este caso nos apoyamos de la herramienta Chatgpt que nos modifico el archivo de servidor HTTP: https://chatgpt.com/share/68cf9460-a9d0-8002-91fd-03ddd0c4f708.

Para el siguiente programa AdministradorTrafico.java es un proxy inverso. Lo que hace es ponerse en medio entre el navegador y dos servidores. Cuando el navegador pide algo (con un GET), el proxy recibe esa petición y la manda a los dos servidores que tiene configurados. Luego, el proxy recibe las respuestas de ambos, pero solo le devuelve al navegador la del Servidor-1; la del Servidor-2 no la manda, aunque sí la recibe. La idea es que aprendas cómo un proxy puede manejar el tráfico, redirigir peticiones y decidir qué respuesta darle al cliente. De igual manera se realizó con chatgpt: https://chatgpt.com/share/68cf9460-a9d0-8002-91fd-03ddd0c4f708.

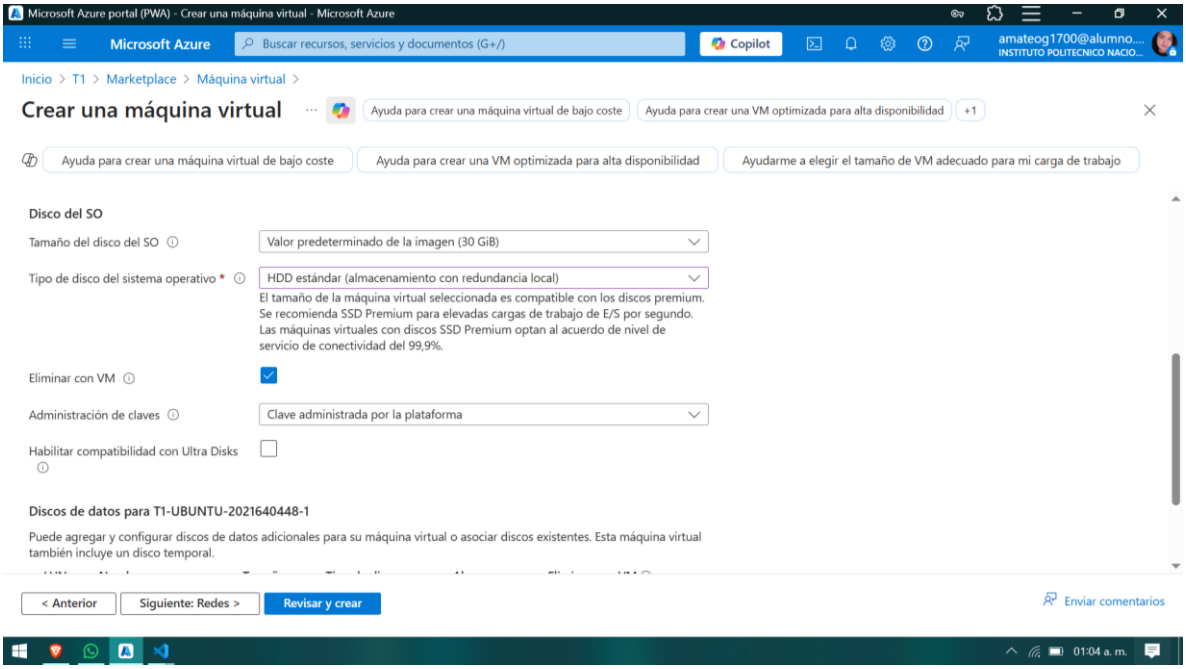
Como se indica en la tarea, se creo las 3 máquinas virtuales, todas con Ubuntu server, la maquina 1 fue nuestro cliente, este se conectaba a las máquinas 2 y 3 que fungían como los servidores HTTPS.



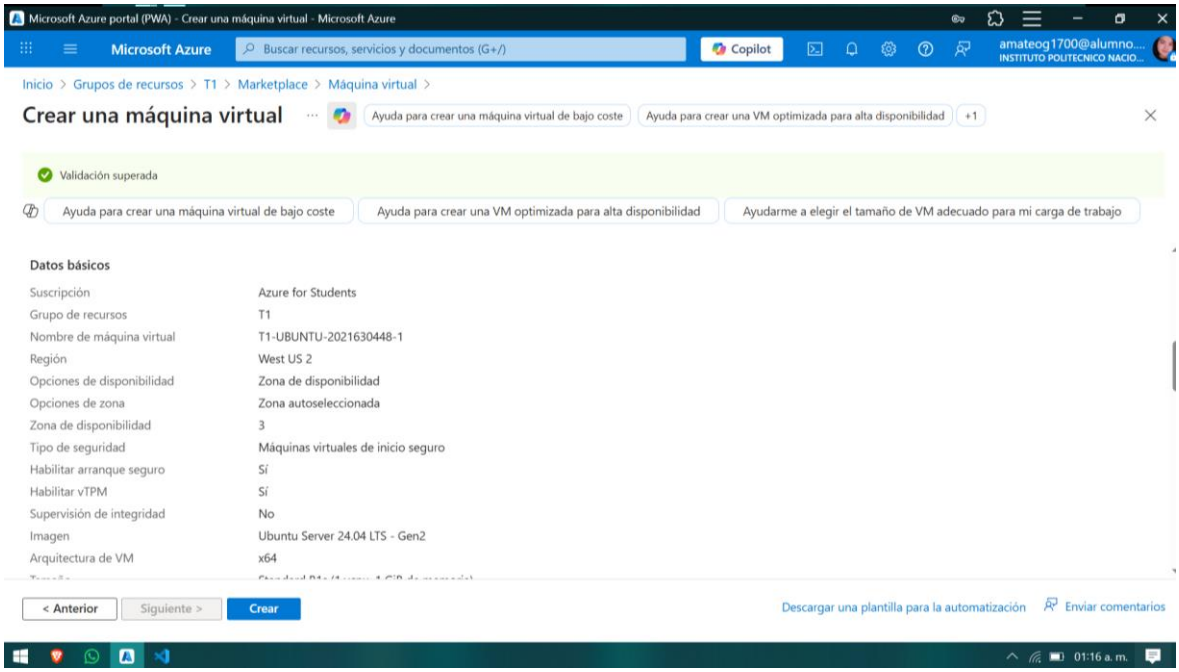
Se muestra que se abrieron los puertos 443 y 20.



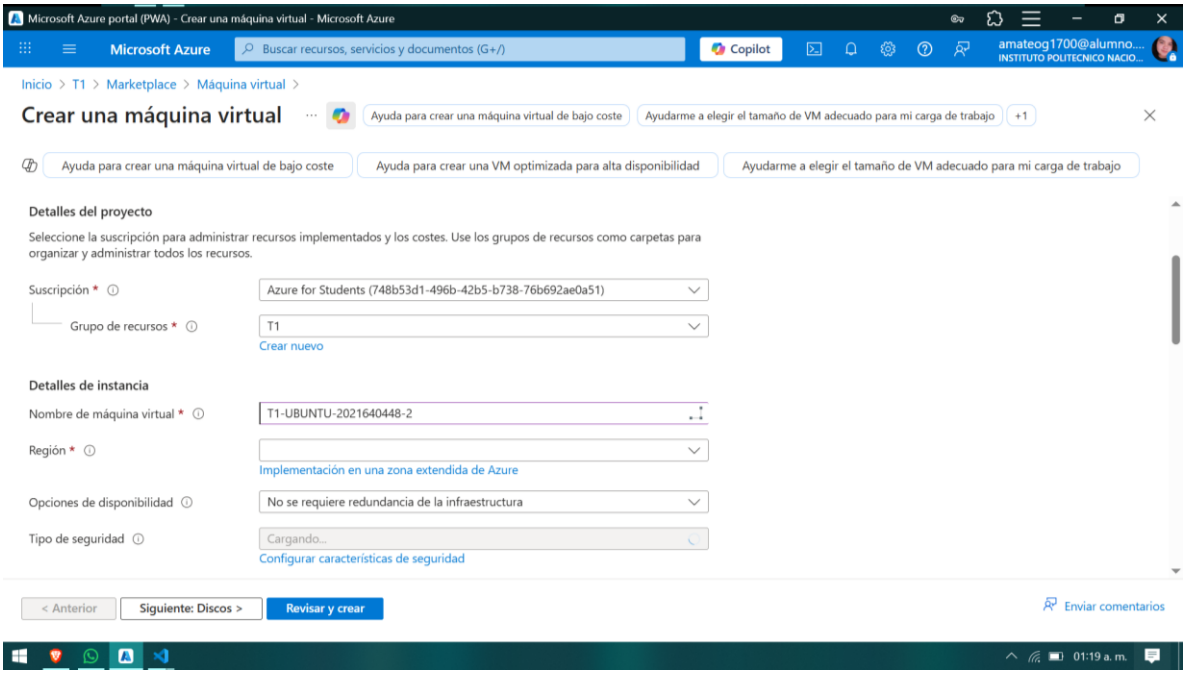
Se seleccionaron discos HDD de 30 GiB



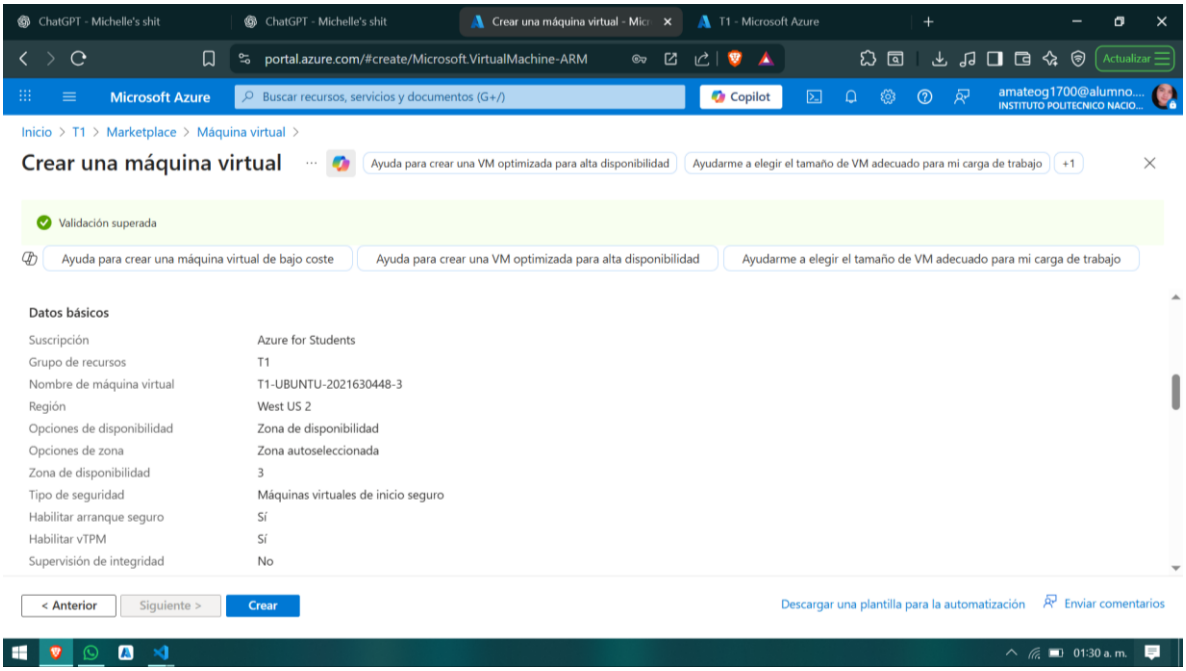
Resumen de las configuraciones de la máquina Virtual 1.



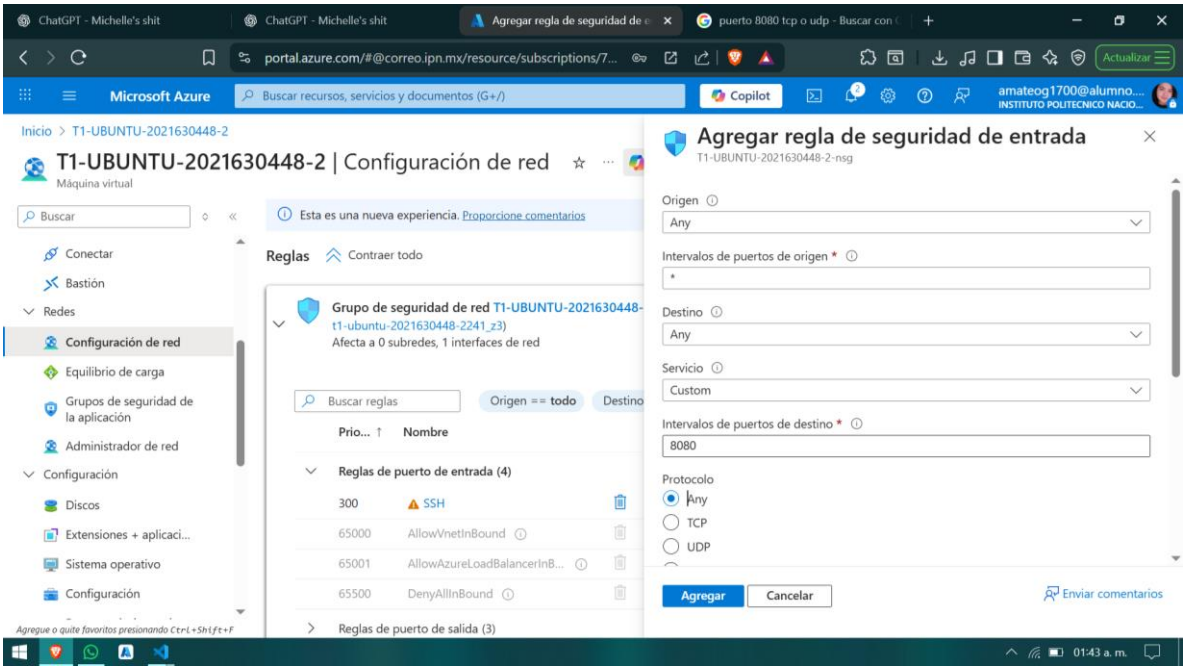
Creación de la máquina virtual 2 con Ubuntu Server, este es el servidor 1.



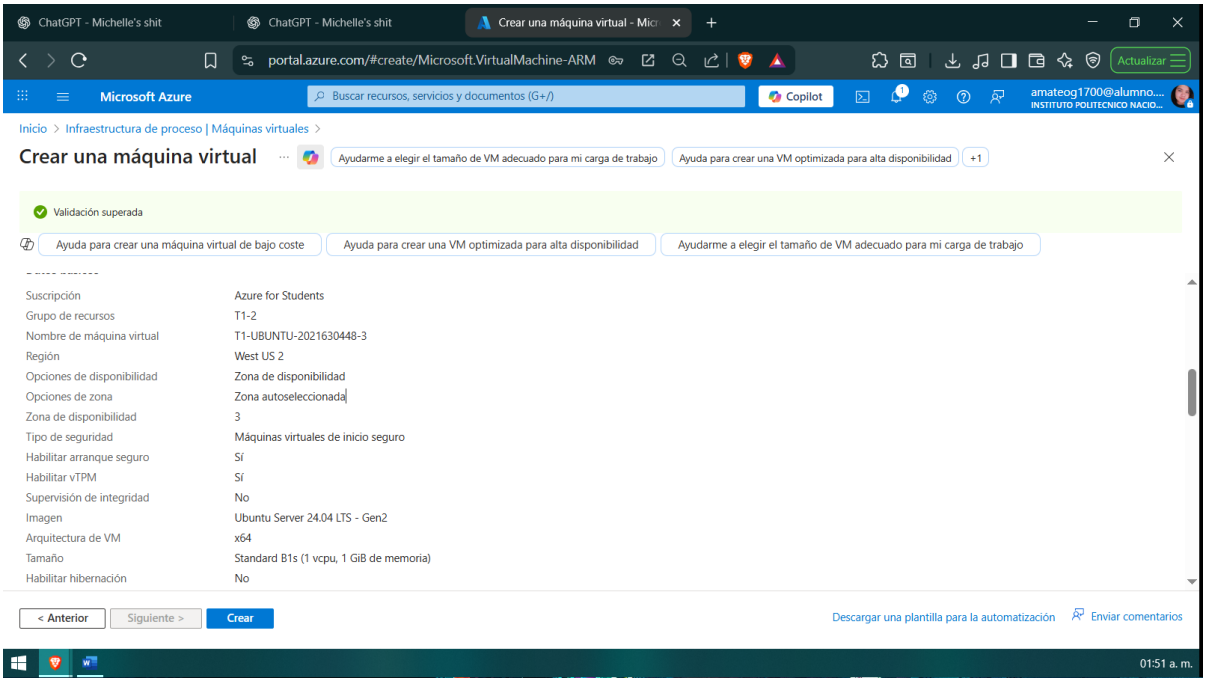
Resumen general de la máquina virtual 2 que este sería el servidor 1.



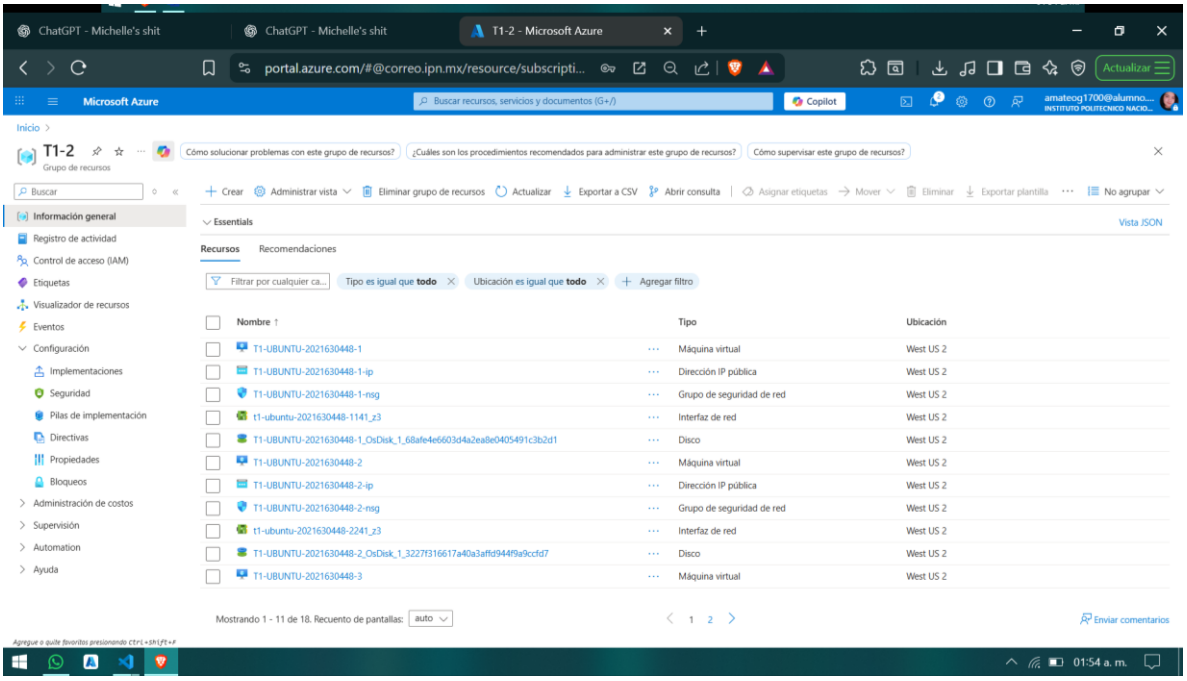
Para esto se tuvo que abrir el puerto 8080 en las configuraciones de red y permitir la entrada del puerto 8080.



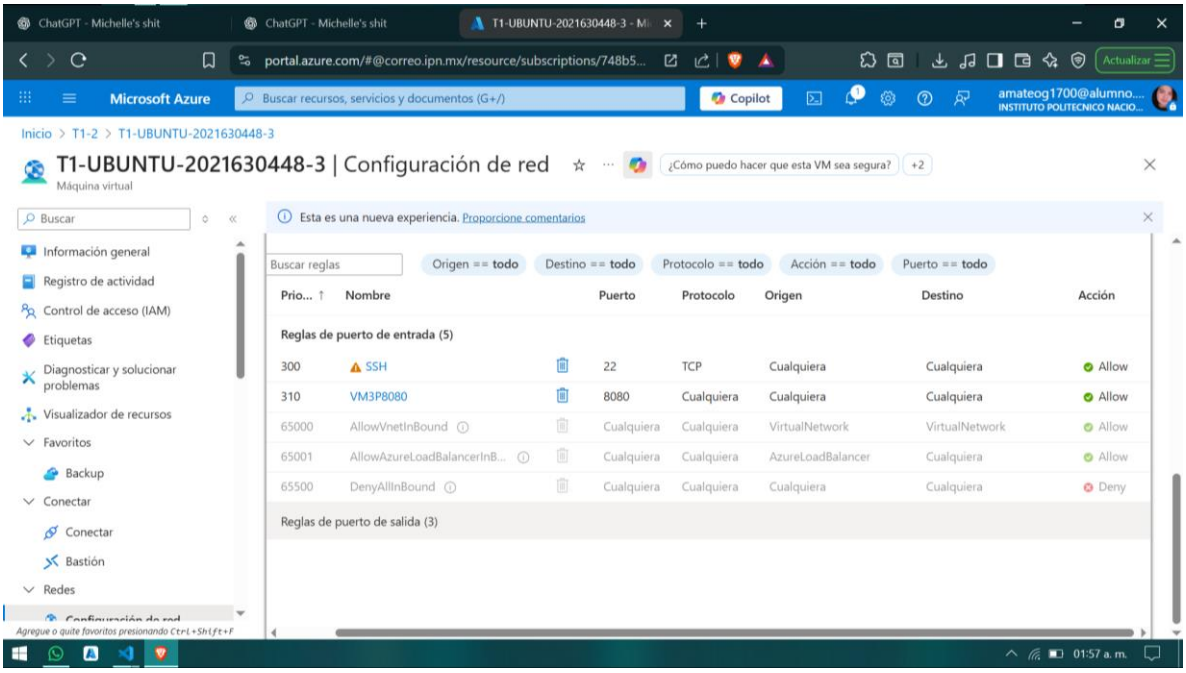
Por último, tenemos la Máquina virtual 3, igual siendo este el segundo servidor de la tarea, en este caso se hicieron 3 máquinas virtuales con las mismas características como se muestra en las imágenes.



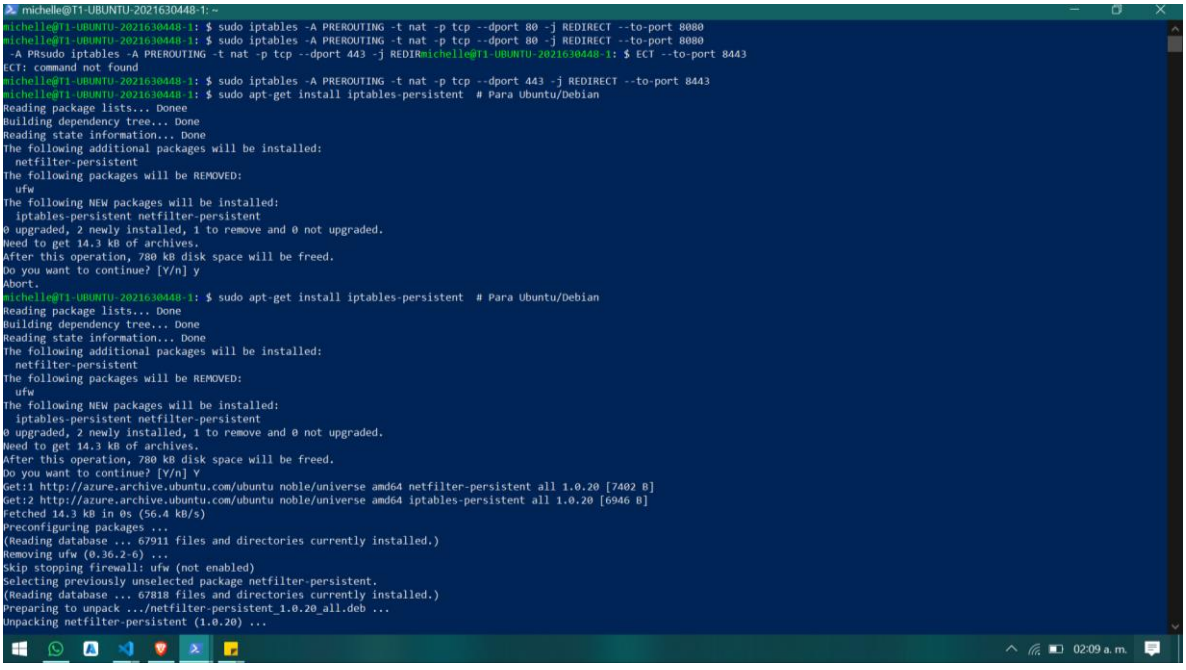
Por último, muestro las 3 VM creadas en el mismo grupo de recursos con sus respectivas interfaces de red, ips, NSG, entre otros más.



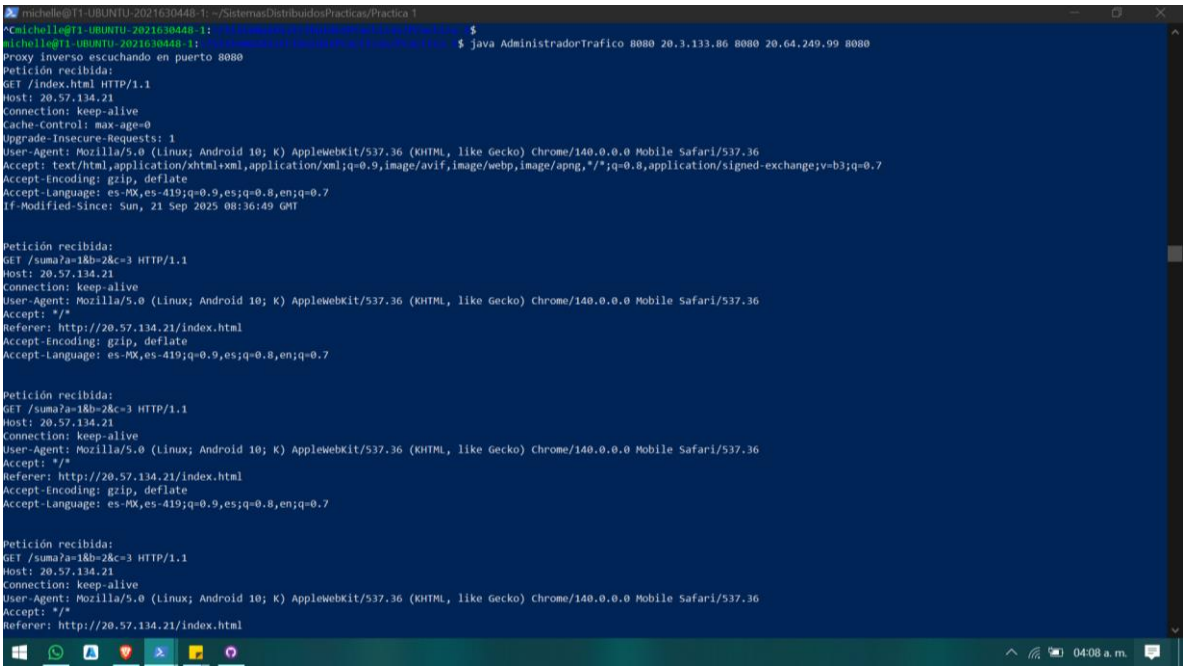
De igual forma se abrió el puerto 8080 en las configuración de red.

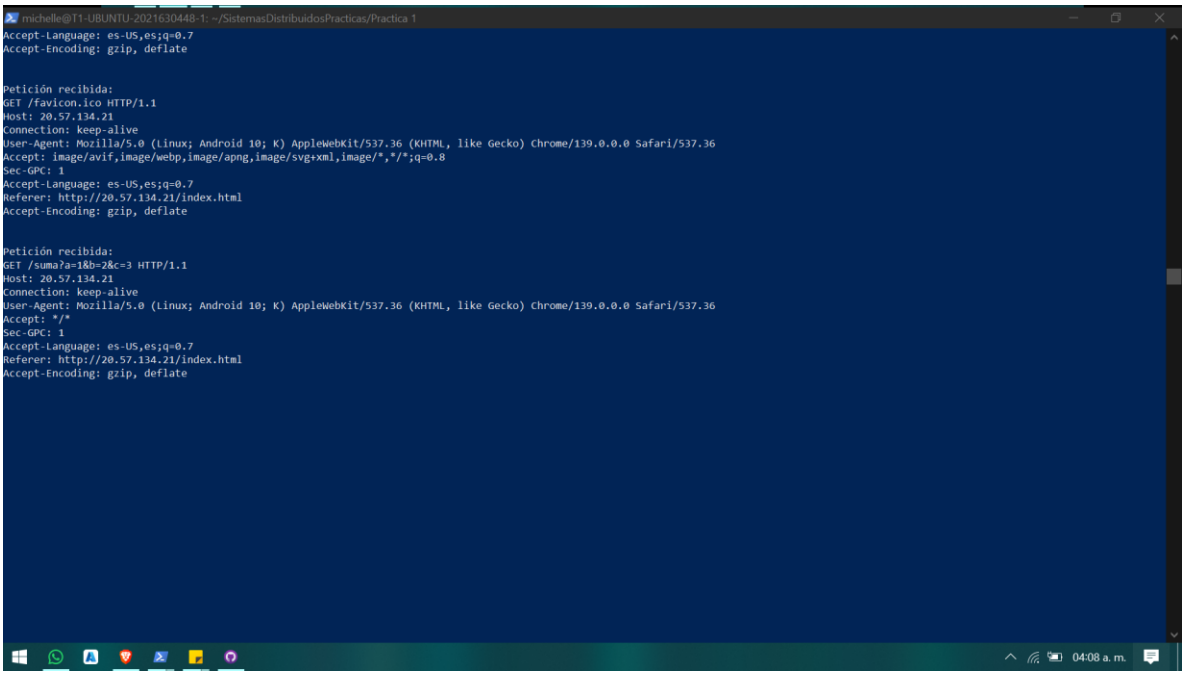


En la primera máquina virtual se mapearon los puertos 80 y 443 a los puertos 8080 y 8443 respectivamente como se muestra en la imagen. Esto redirige las peticiones que llegan a los primeros puertos que los segundos puertos sean los que tengan el tráfico.

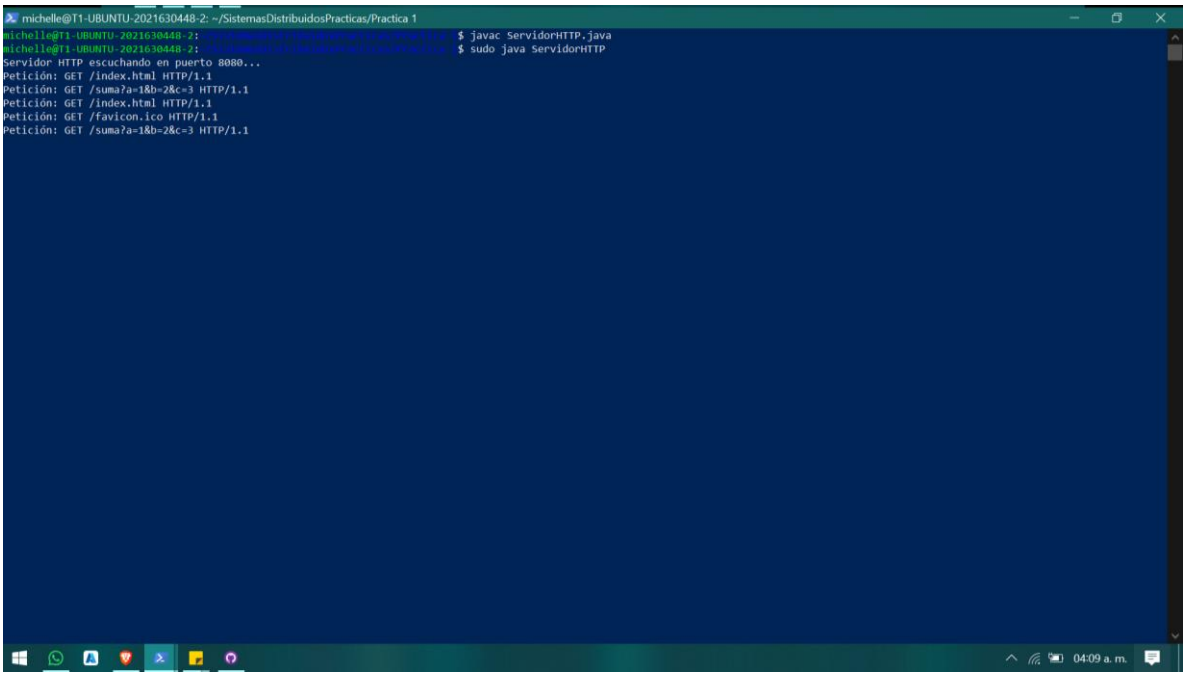


En la máquina virtual 1 se ejecutó el administrador de tráfico ejecutando el puerto 8080 adicionando las maquinas virtuales 2 y 3 con sus Ips públicas con el puerto 8080, esto permitiendo las conexiones con los sockets, además de recibiendo las peticiones get.

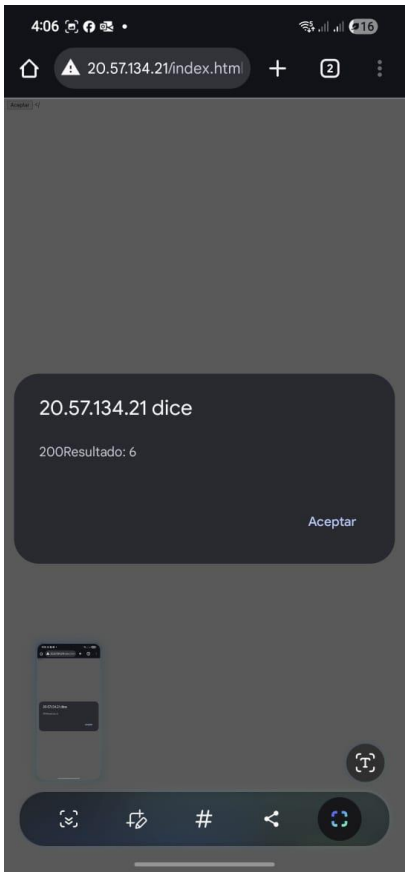




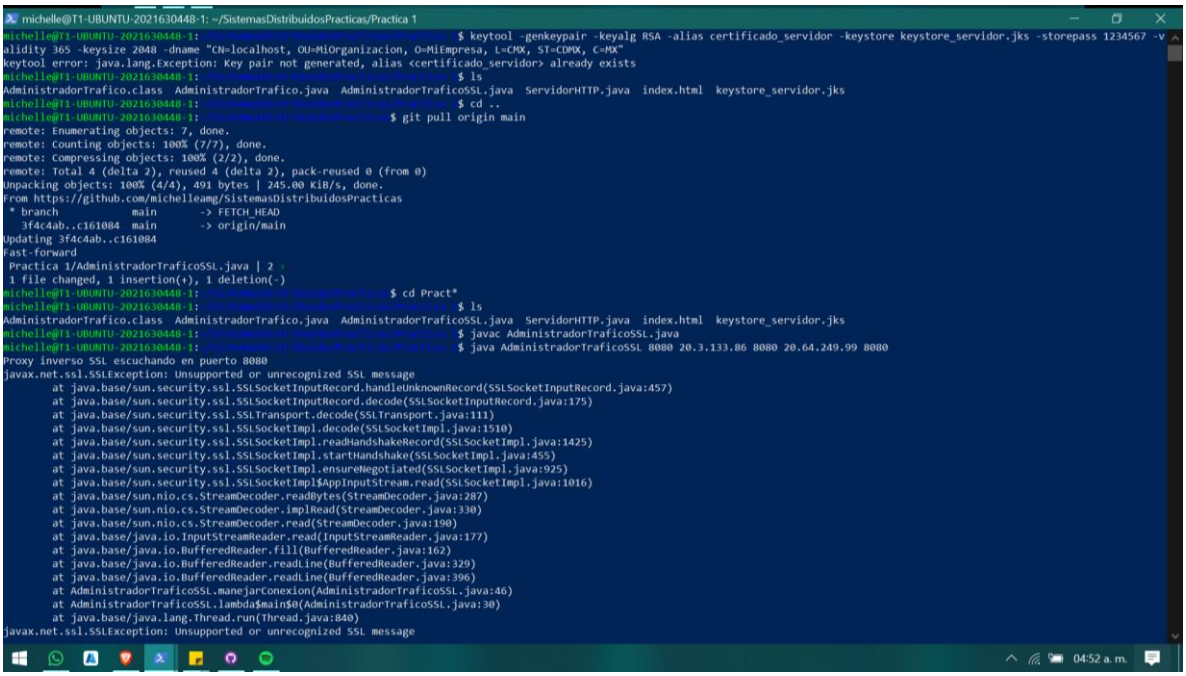
Tenemos ahora los 2 servidores ejecutándose como se muestra en la maquina virtual 2 y 3, este servidor fue modificado con ChatGPT, esta muestra las peticiones get.

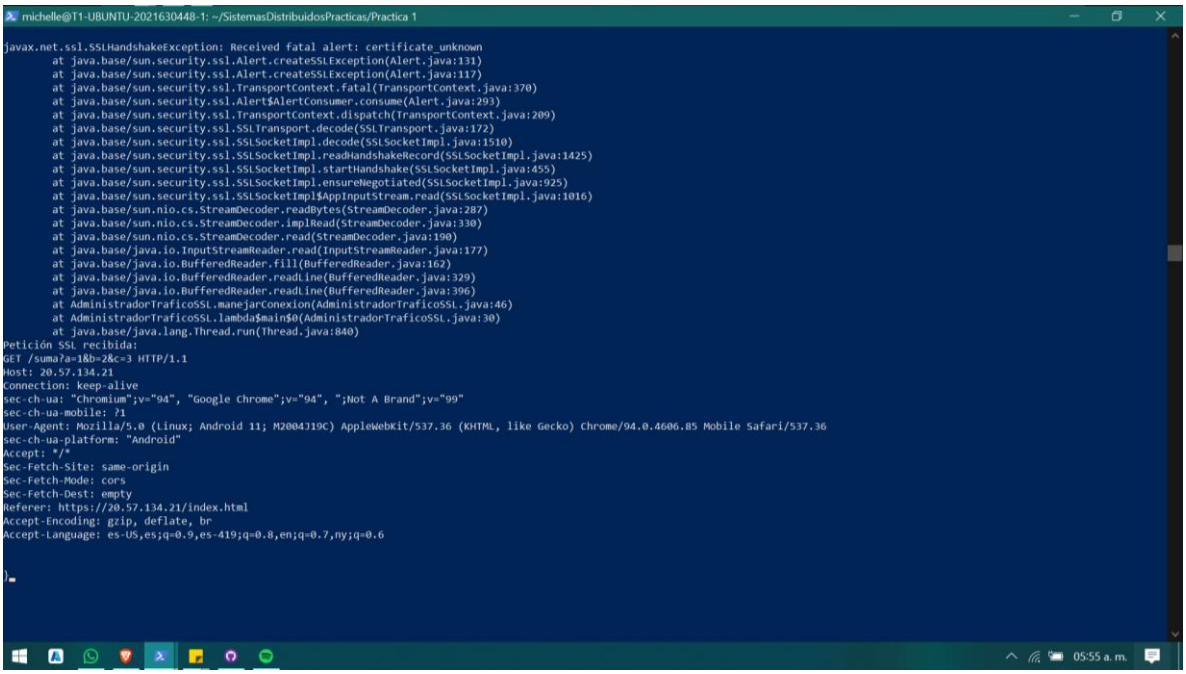


Como se muestra a continuación, en el teléfono inteligente en este caso mi celular personal al darle en el botón aceptar y agregar con la IP publica en este caso es http://20.57.134.21/index.html



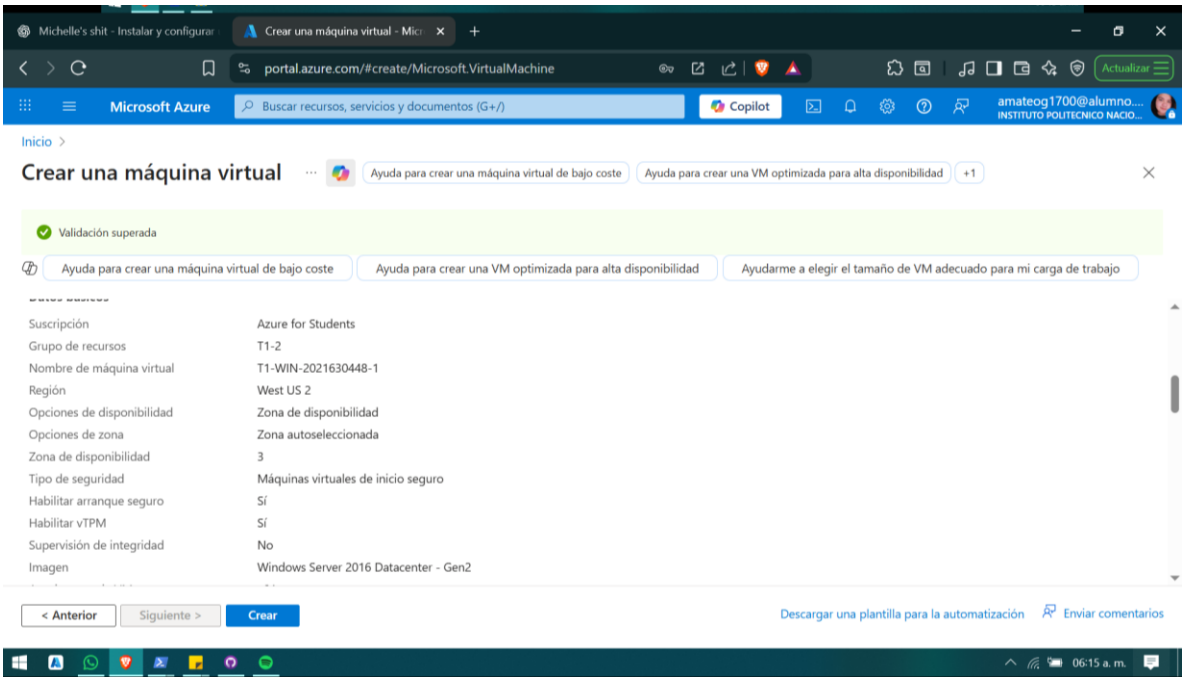
En el programa AdministradorTraficoSSL que se modificó con ChatGPT, primero se generaron los certificados auto firmados para poder realizar las peticiones HTTP, después de generarla, después se ejecuta como se muestra en las siguientes imágenes.



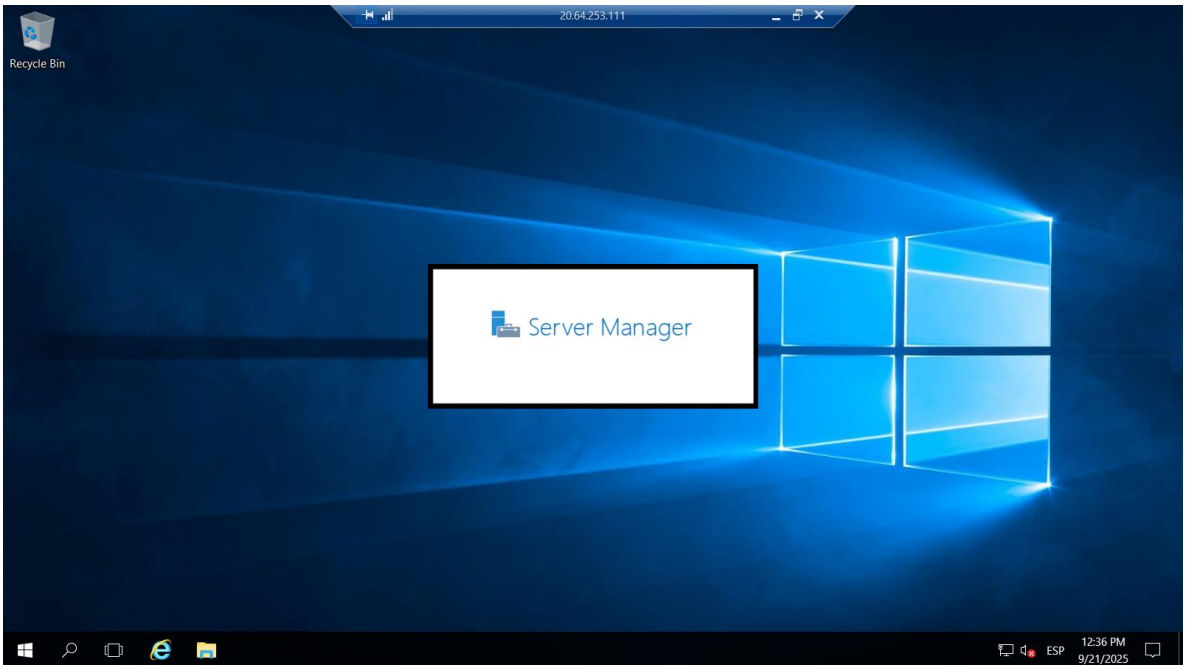


En este caso se hizo la prueba en otro celular, con el protocolo HTTPS sale con advertencia, sin embargo, al entrar y dar en aceptar, nos da el resultado 6.

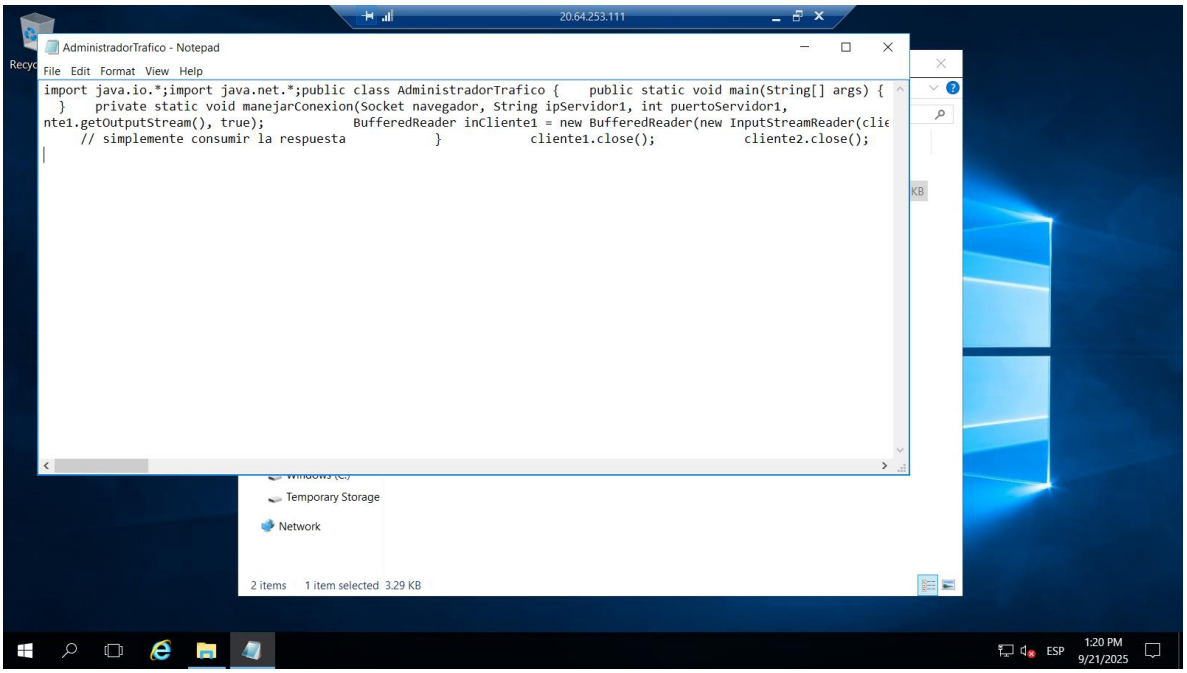
En este caso se eliminó la maquina virtual 1 de Ubuntu server y ahora se hace una máquina virtual con Windows server, con data center de la generación 2, con un disco normal HDD.

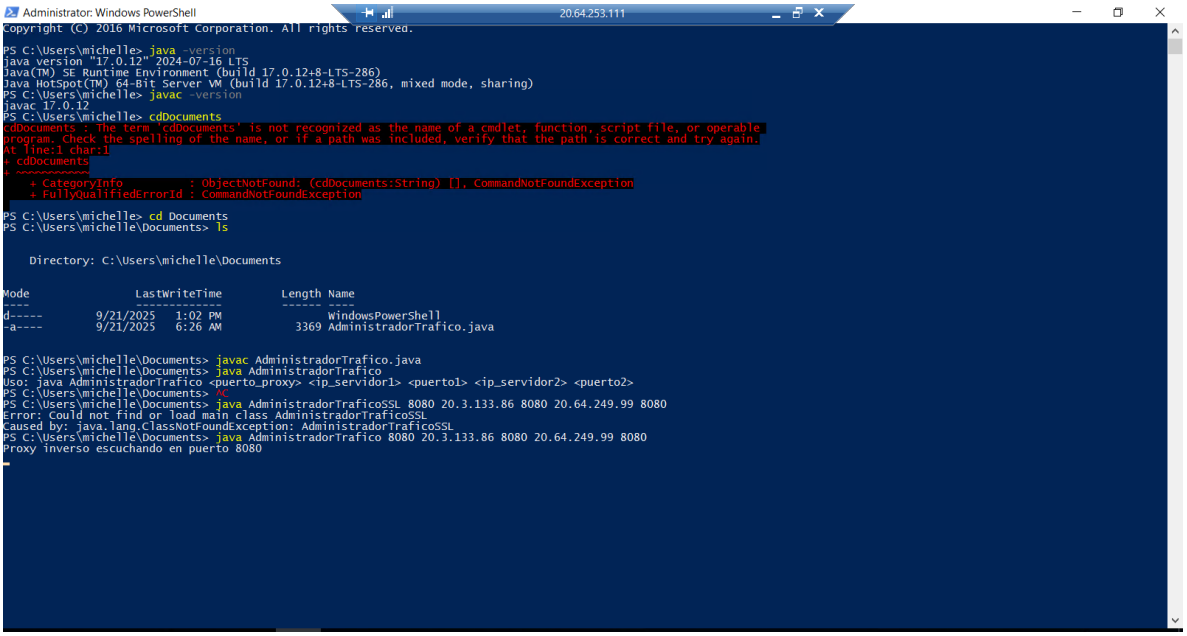


Por medio del protocolo RDP y montando el disco local F qe generamos desde local para que pudiéramos agregar los archivos necesarios, en este caso ingresamos con el archivo que bajamos de la nube, después ingresamos por medio de contraseña e ingresamos.



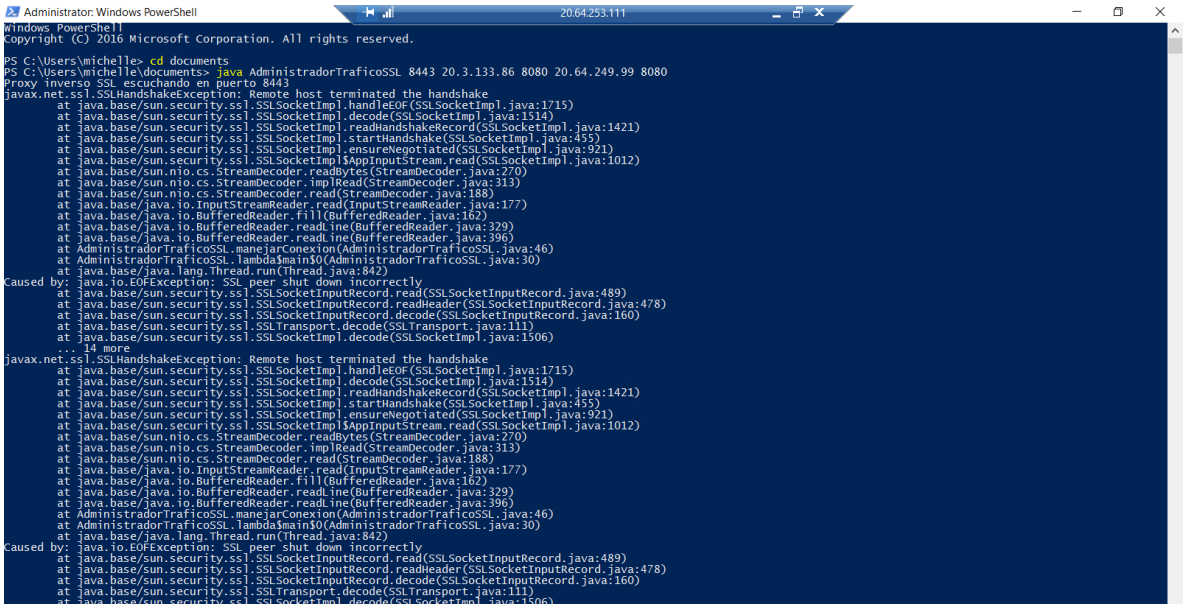
Para las siguientes imágenes se muestra la ejecución del administrador de trafico, este fue creado por la IA. Aquí se muestra que se hizo la ejecución.





En la siguiente imagen se muestra que se da el resultado al apretar el botón aceptar, esto, solamente la diferencia que tuve que poner el puerto ya que no dejaba ingresar ya que bloqueaba la conexión.

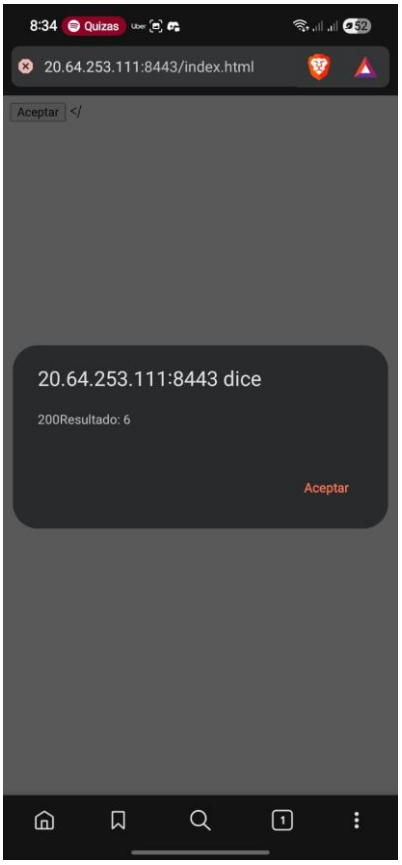
En la siguiente imagen se ejecuta el administrador de tráfico SSL, primero se generó el certificado auto firmado y después se ejecutó.



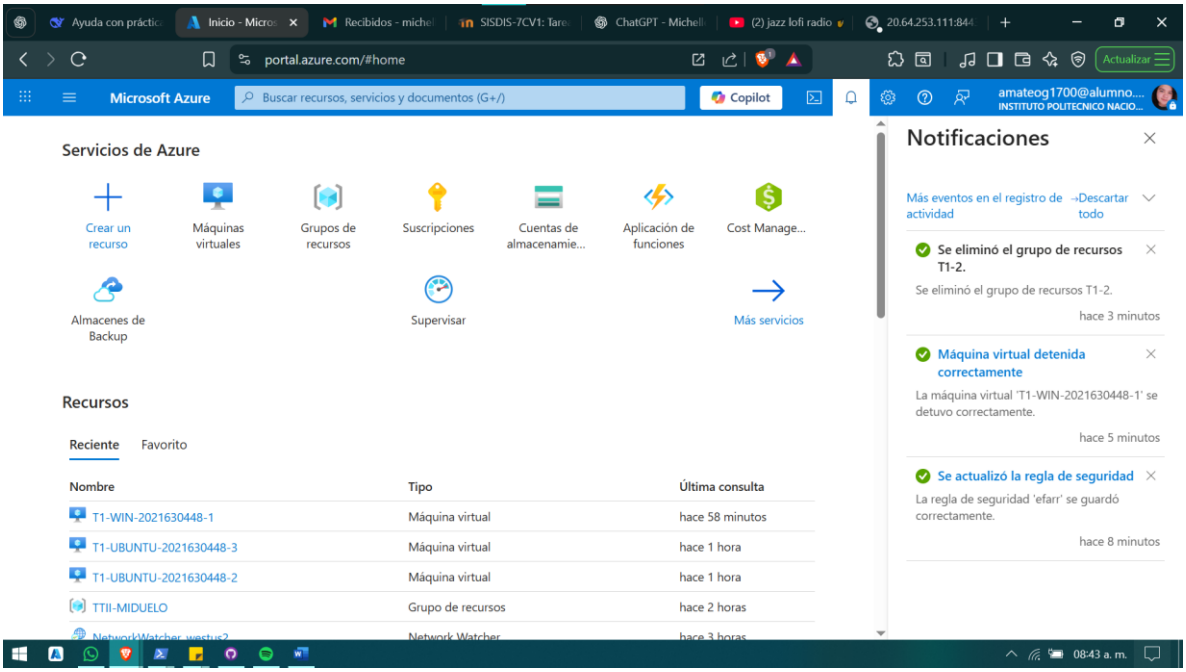
```
        at AdministradorTraficoSSL.lambda$main$0(AdministradorTraficoSSL.java:30)
        at java.base/java.lang.Thread.run(Thread.java:842)
Petici??n SSL recibida:
GET /suma?a=1&b=2&c=3 HTTP/1.1
Host: 20.64.253.111:8443
Connection: keep-alive
sec-ch-ua-platform: "Android"
User-Agent: Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Mobile Safari/537
.36
sec-ch-ua: "Chromium";v="140", "Not=A?Brand";v="24", "Brave";v="140"
sec-ch-ua-mobile: ?1
Accept: */*
Sec-GPC: 1
Accept-Language: es-US,es;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://20.64.253.111:8443/index.html
Accept-Encoding: gzip, deflate, br, zstd


Petici??n SSL recibida:
GET /suma?a=1&b=2&c=3 HTTP/1.1
Host: 20.64.253.111:8443
Connection: keep-alive
sec-ch-ua-platform: "Android"
User-Agent: Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/140.0.0.0 Mobile Safari/537
.36
sec-ch-ua: "Chromium";v="140", "Not=A?Brand";v="24", "Brave";v="140"
sec-ch-ua-mobile: ?1
Accept: */*
Sec-GPC: 1
Accept-Language: es-US,es;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: https://20.64.253.111:8443/index.html
Accept-Encoding: gzip, deflate, br, zstd
```

Para la última captura, se ingresó de manera normal con el HTTPS, de igual manera nos decía que la conexión era insegura, sin embargo, pude ingresar, de igual manera tuve que poner el puerto por las mismas razones de que la conexión me lo rechazaba.



Al final se eliminaron el grupo de recursos.



Conclusiones

El proyecto culminó exitosamente con la implementación y verificación de la arquitectura de proxy inverso en la nube. La práctica demostró la operatividad de los principios de los sistemas distribuidos a través de una aplicación funcional de balanceo de carga rudimentario y terminación SSL.

Técnicamente, se logró la configuración de la infraestructura como servicio (IaaS) en Azure, asegurando la conectividad mediante la apertura y redirección de puertos (80 y 443 a 8080 y 8443) a nivel del sistema operativo con iptables. El desarrollo de la lógica del proxy inverso en Java (AdministradorTrafico.java) ilustró cómo se manejan y reenvían sockets para lograr la mediación de las peticiones, destacando la gestión de la solicitud a dos servidores y el reenvío selectivo de la respuesta del Servidor-1 al cliente.

La implementación de la variante HTTPS del proxy probó la capacidad de manejar peticiones seguras mediante el uso de keys y certificados autofirmados, aunque en un entorno de prueba, lo que generó la advertencia de conexión insegura en el cliente. Finalmente, la integración del manejo de caché HTTP (Last-Modified/If-Modified-Since) en los servidores backend validó el control sobre la eficiencia en la transferencia de datos y la reducción de la carga del servidor.