

“INSTITUTO POLITECNICO NACIONAL”
ESCUELA SUPERIOR DE COMPUTO



**Desarrollo e Implementación de un Proxy HTTPS Inverso con
Servidores HTTP en la Nube con Azure.**

5CV2 | Sistemas Distribuidos | Mtro. Pineda Guerrero Carlos

Autor: Alejandro Yahir Osorio Olivares

Indice.

1. Prefacio.....	1
2. Introducción	2
3. Servidores “http”	3
3.1. Modificación: “ServidorHTTP.java”	4
4. Proxy Inverso - Administrador de Trafico	6
4.1. Implementación: “AdministradorTrafico.java”	7
4.1.1. Parámetros de configuración.....	7
4.1.2. Servidor Proxy Escuchando	7
4.1.3. Clase Worker (Hilo de trabajo)	8
4.1.4. Manejo de excepciones	10
5. Ejecución del Trafico en Ubuntu.....	11
5.1. Creación de Máquinas Virtuales en Azure	11
5.2. Configuración de las Máquinas Virtuales	12
5.3. Subir ServidorHTTP.java a las VMs 2 & 3	13
5.4. Subir y ejecutar AdministradorTrafico.java en la primera VM	14
5.5. Prueba de acceso al servidor desde un dispositivo externo.....	17
5.6. Resumen	18
6. Implementación de Sockets Seguros (SSL)	19
6.1. Implementación & Configuración del “keystore”	19
6.2. Modificación del Administrador de Trafico para soportar HTTPS	20
6.3. Ejecución del Administrador de Trafico con SSL.....	23
6.4. Prueba de Comunicación Segura	25
6.5. Resumen	27
7. Conclusión	28

1. Prefacio.

Dada la expansión de la infraestructura digital, el uso de la computación en la nube ha transformado la forma en que las organizaciones gestionan sus infraestructuras tecnológicas. Microsoft Azure se ha consolidado como una de las principales plataformas de servicios en la nube, ofreciendo herramientas avanzadas para el despliegue y administración de aplicaciones distribuidas. Su flexibilidad y escalabilidad permiten a los desarrolladores implementar soluciones eficientes y seguras, optimizando recursos y mejorando el rendimiento de los sistemas. Gracias a su amplia gama de servicios, es posible diseñar arquitecturas personalizadas que respondan a las necesidades específicas de cada entorno, garantizando alta disponibilidad, redundancia y un sólido esquema de seguridad.

Uno de los elementos clave en la arquitectura de redes y seguridad en la nube es el uso de proxies, en particular, los proxies inversos HTTPS. Estas tecnologías permiten gestionar y redirigir el tráfico de manera eficiente, asegurando que las solicitudes de los clientes sean procesadas correctamente antes de llegar a los servidores backend. Además, los proxies inversos ofrecen ventajas como la optimización del tráfico, la distribución de carga y la implementación de medidas de seguridad avanzadas, tales como el cifrado mediante certificados SSL/TLS. Su implementación en entornos distribuidos permite mejorar la resiliencia del sistema, minimizando los riesgos de ataques cibernéticos y asegurando la integridad y confidencialidad de los datos transmitidos.

Los servidores HTTP desempeñan un papel fundamental en la provisión de contenido y servicios a los usuarios finales. A través de estos servidores, es posible gestionar solicitudes web, ejecutar aplicaciones y distribuir datos en entornos distribuidos. La combinación de un proxy inverso con servidores HTTP en la nube permite mejorar la eficiencia y confiabilidad de los sistemas, garantizando una comunicación segura y fluida entre los distintos componentes de la infraestructura. Además, la implementación de múltiples servidores HTTP permite balancear la carga, evitando sobrecargas en un solo nodo y mejorando el tiempo de respuesta de los servicios.

En este contexto, la correcta configuración y administración de un proxy inverso con servidores HTTP en la nube representa un reto importante en la gestión de sistemas distribuidos. Aspectos como la configuración de reglas de reenvío, la gestión de certificados de seguridad y la integración con otros servicios en la nube son fundamentales para garantizar una implementación efectiva.

Este documento aborda el desarrollo e implementación de un Proxy HTTPS Inverso con servidores HTTP en Microsoft Azure, proporcionando una guía detallada sobre su configuración y funcionamiento. Se explorarán las ventajas de utilizar máquinas virtuales con diferentes sistemas operativos, analizando su rendimiento y compatibilidad en la nube. Con este estudio, se pretende fortalecer los conocimientos en el diseño e implementación de infraestructuras distribuidas, fomentando el uso de tecnologías avanzadas para la gestión de redes y seguridad en entornos digitales.

2. Introducción.

En el presente documento se detalla el desarrollo e implementación de un Proxy HTTPS Inverso con servidores HTTP en la nube utilizando la plataforma Microsoft Azure. Este trabajo se enmarca dentro de la asignatura de Sistemas Distribuidos, correspondiente al plan de estudios de la carrera de Ingeniería en Sistemas Computacionales.

El propósito de esta asignación es comprender y aplicar los conceptos fundamentales de los sistemas distribuidos mediante la configuración y despliegue de una infraestructura en la nube que permita gestionar solicitudes HTTPS de manera eficiente. Para ello, se hará uso de un servidor proxy inverso, el cual se encargará de recibir las peticiones de los clientes, distribuirlas a los servidores HTTP backend y reenviar las respuestas de manera segura y optimizada. El desarrollo de la asignación se llevará a cabo en Microsoft Azure, donde se implementará una arquitectura que incluya instancias de máquinas virtuales configuradas como servidores HTTP, junto con un servicio de proxy inverso que gestione el tráfico y garantice la seguridad de las comunicaciones. Para ello, se utilizarán exclusivamente máquinas virtuales con sistema operativo Ubuntu, permitiendo así evaluar el desempeño y realizar las configuraciones necesarias en este entorno. Esta actividad permitirá explorar la implementación de un entorno distribuido en la nube, configurando correctamente la comunicación entre los distintos servidores y asegurando la disponibilidad y estabilidad del servicio. Además, se analizarán las ventajas y desafíos de utilizar Ubuntu en la configuración del proxy inverso, considerando aspectos de compatibilidad, rendimiento y seguridad.

El estudio de esta implementación facilitará la comprensión del funcionamiento de los proxies inversos y su relevancia en la administración de tráfico en arquitecturas distribuidas. Asimismo, permitirá identificar buenas prácticas en el manejo de certificados SSL/TLS, estrategias de balanceo de carga y mecanismos de optimización para mejorar la eficiencia del sistema. Se evaluará la escalabilidad del entorno, asegurando que pueda adaptarse a distintas cargas de trabajo y necesidades de los clientes.

Finalmente, este documento servirá como referencia para futuras implementaciones de infraestructuras distribuidas en la nube, proporcionando un análisis detallado de las configuraciones realizadas y los resultados obtenidos. Se destacarán las mejores prácticas identificadas durante la ejecución de la práctica, así como las posibles mejoras que podrían implementarse en escenarios similares, garantizando una gestión eficiente y segura del tráfico en sistemas distribuidos.

3. Servidores “http”.

El Servidor HTTP implementado en esta práctica es un servidor básico capaz de manejar solicitudes HTTP entrantes, procesarlas y generar respuestas adecuadas. Su función principal es atender las peticiones del Administrador de Tráfico, que actúa como un proxy para redirigir el tráfico entre el cliente y los servidores.

El servidor opera en un puerto específico, definido al iniciar la aplicación. Utiliza un **ServerSocket** para aceptar conexiones y crea un nuevo hilo para cada solicitud recibida, lo que permite manejar múltiples peticiones de forma concurrente. Al recibir una solicitud, el servidor lee la línea de petición y los encabezados correspondientes, analizando el tipo de solicitud recibida. Si la petición es de tipo **GET** y corresponde a la ruta principal ("/"), el servidor responde con una página HTML que incluye un botón para enviar solicitudes adicionales al servidor. Para ello, la página contiene un script en JavaScript que permite realizar una solicitud asíncrona utilizando **XMLHttpRequest**. Por otro lado, si la solicitud corresponde a la ruta **"/suma"**, el servidor analiza los parámetros de la URL y calcula la suma de los valores proporcionados. Luego, genera una respuesta en texto plano con el resultado de la operación. En caso de recibir una solicitud no reconocida, el servidor responde con un código de estado **404 Not Found**, indicando que el recurso solicitado no está disponible.

El servidor también incorpora medidas para optimizar la comunicación y reducir la cantidad de datos transmitidos. Por ejemplo, cuando un cliente incluye el encabezado **If-Modified-Since** en su petición, el servidor compara la fecha proporcionada con su propia marca de tiempo de última modificación. Si el recurso no ha cambiado desde esa fecha, en lugar de reenviar toda la respuesta, el servidor responde con un código **304 Not Modified**, lo que permite a los clientes reutilizar contenido almacenado en caché.

Además, el servidor ha sido diseñado para manejar múltiples conexiones de manera eficiente, evitando bloqueos o retrasos en la atención de solicitudes. Para lograr esto, cada petición se gestiona dentro de un nuevo hilo, lo que permite procesar varias solicitudes de manera simultánea sin afectar el rendimiento general del sistema. Esta arquitectura multihilo es esencial para garantizar que el servidor pueda responder rápidamente a las peticiones del Administrador de Tráfico y distribuir adecuadamente las solicitudes entre los distintos componentes de la red.

En términos de implementación, el servidor también se asegura de manejar adecuadamente los errores y excepciones que puedan ocurrir durante la ejecución. Se incluyen mecanismos para capturar posibles fallos en la lectura de datos o en la generación de respuestas, evitando que una falla individual afecte el funcionamiento global del servicio. Asimismo, se puede extender su funcionalidad añadiendo más rutas y métodos HTTP para mejorar su capacidad de respuesta ante diferentes tipos de solicitudes, lo que lo convierte en una base flexible para futuras mejoras y ampliaciones dentro del sistema distribuido.

3.1. Modificación: “ServidorHTTP.java”.

En el siguiente apartado se detallarán las instrucciones necesarias para modificar el código de **ServidorHTTP.java**, con el objetivo de preparar el servidor para futuras implementaciones de conexiones seguras. En esta fase inicial, se realizarán ajustes para optimizar el manejo de solicitudes HTTP y la interacción con el *Administrador de Tráfico*, sin alterar la funcionalidad básica del servidor. Aunque la implementación de sockets seguros mediante SSL no se abordará en esta etapa, los cambios establecidos servirán como base para integrar seguridad en etapas posteriores, permitiendo una transición hacia una infraestructura de comunicación cifrada.

1. **Adición de la variable `LAST_MODIFIED`:** Para simular la fecha de última modificación del recurso, se agregó esta constante.

```
1 static final String LAST_MODIFIED = "Fri, 01 Mar 2024 12:00:00 GMT";
```

2. **Lectura del encabezado *If-Modified-Since* y comparación con `LAST_MODIFIED`:** Se verificó si el encabezado *If-Modified-Since* está presente en la solicitud y se comparó con la fecha de última modificación del recurso.

```
1 String ifModifiedSince = null;
2 while (!(encabezado = entrada.readLine()).equals("")) {
3     System.out.println("Encabezado: " + encabezado);
4     if (encabezado.startsWith("If-Modified-Since: ")) {
5         ifModifiedSince = encabezado.substring(19);
6     }
7 }
```

3. **Condicional para responder con 304 Not Modified si el recurso no ha cambiado:** Si el valor del encabezado ***If-Modified-Since*** coincide con la fecha de modificación, se devuelve un código 304 Not Modified.

```
1  if (req.startsWith("GET / ")) {
2      if (ifModifiedSince != null && ifModifiedSince.equals(LAST_MODIFIED)) {
3          salida.println("HTTP/1.1 304 Not Modified");
4          salida.println("Connection: close");
5          salida.println();
6      } else {
7
8          // Proceso (else): Respuesta
9
10     }
```

4. **Agregando el encabezado *Last-Modified* en la respuesta:** El servidor incluye el encabezado ***Last-Modified*** en la respuesta para permitir la verificación de caché en las solicitudes futuras.

```
1  salida.println("Last-Modified: " + LAST_MODIFIED);
```

4. Proxy Inverso – Administrador de Trafico.

El **AdministradorTrafico.java** es el componente central de esta práctica, funcionando como un **Proxy HTTPS Inverso**. Su propósito es recibir solicitudes de clientes (navegadores web), distribuirlas a dos servidores HTTP y reenviar la respuesta de uno de ellos al cliente.

Este proxy recibe tres parámetros en la línea de comandos al momento de su ejecución:

1. **Puerto local del proxy:** Puerto en el que el proxy escucha las solicitudes entrantes.
2. **Dirección IP y puerto del Servidor-1:** Configuración de conexión para el primer servidor HTTP.
3. **Dirección IP y puerto del Servidor-2:** Configuración de conexión para el segundo servidor HTTP.

Los servidores HTTP están implementados en **ServidorHTTP.java** y se ejecutan en diferentes instancias en la nube.

Funcionamiento del Proxy

El proxy sigue este flujo para procesar las solicitudes:


1. **Escucha en el puerto local** y acepta conexiones de clientes.
2. **Lee la solicitud HTTP** enviada por el navegador.
3. **Abre conexiones con ambos servidores HTTP** y reenvía la solicitud a cada uno.
4. **Recibe la respuesta de Servidor-1** y la reenvía al navegador.
5. **Descarta la respuesta de Servidor-2** (solo se conecta para mantener la estructura del sistema).
6. **Cierra la conexión con el cliente** una vez enviada la respuesta.

4.1. Implementación: “AdministradorTrafico.java”.

Este código se desarrolló desde cero para gestionar las solicitudes de los clientes y distribuirlas a los servidores de acuerdo con la configuración proporcionada. Se puede encontrar el código completo de esta Practica junto a la dependencia de este Documento.

Descripción de la Funcionalidad:

- 4.1.1. Parámetros de configuración:** El programa toma como parámetros la dirección IP y los puertos de los dos servidores HTTP, así como el puerto local en el que escuchará las solicitudes de los clientes.



```
1 puertoLocal = Integer.parseInt(args[0]);
2 servidor1 = args[1];
3 puertoServidor1 = Integer.parseInt(args[2]);
4 servidor2 = args[3];
5 puertoServidor2 = Integer.parseInt(args[4]);
```

- 4.1.2. Servidor Proxy Escuchando:** El proxy se pone en escucha en el puerto local proporcionado, esperando las conexiones de los clientes. Cuando recibe una solicitud, crea un nuevo hilo (**Worker**) para manejar la petición.



```
1 ServerSocket serverSocket = new ServerSocket(puertoLocal);
2 while (true) {
3     Socket cliente = serverSocket.accept();
4     new Worker(cliente).start();
5 }
```

4.1.3. Clase Worker (Hilo de trabajo): Cada conexión de cliente es manejada por un hilo en la clase **Worker**. En este hilo se realiza lo siguiente:

- a) **Conexión a ambos servidores:** El hilo se conecta a **Servidor-1** y **Servidor-2** utilizando los parámetros de configuración, lo que permite enviar la misma solicitud a ambos servidores.

```
1 // Conectar a ambos servidores
2 Socket servidor1Socket = new Socket(servidor1, puertoServidor1);
3 Socket servidor2Socket = new Socket(servidor2, puertoServidor2);
```

- b) **Lectura de la solicitud del cliente:** El proxy lee la solicitud del cliente, incluyendo el encabezado If-Modified-Since si está presente. Esta parte también maneja la creación del cuerpo de la solicitud que será enviada a los servidores.

```
1 while ((linea = entradaCliente.readLine()) != null && !linea.isEmpty()) {
2     if (primerLinea) {
3         System.out.println("Petición recibida en el proxy: " + linea);
4         primerLinea = false;
5     }
6     if (linea.startsWith("If-Modified-Since: ")) {
7         ifModifiedSince = linea.substring(19);
8     }
9     petition.append(linea).append("\r\n");
10 }
11 petition.append("\r\n");
```

- c) **Envío de la solicitud a ambos servidores:** El proxy envía la solicitud a ambos servidores HTTP.

```
1 // Enviar la petición a ambos servidores
2 salidaServidor1.print(peticion.toString());
3 salidaServidor1.flush();
4 salidaServidor2.print(peticion.toString());
5 salidaServidor2.flush();
```

- d) **Recepción de las respuestas:** Se leen las respuestas de ambos servidores. Sin embargo, solo la respuesta de **Servidor-1** es reenviada al cliente, mientras que la respuesta de **Servidor-2** es descartada.

```
1 // Leer la respuesta de Servidor-1
2 ByteArrayOutputStream respuestaServidor1 = new ByteArrayOutputStream();
3 byte[] buffer = new byte[4096];
4 int bytesRead;
5 while ((bytesRead = entradaServidor1Binaria.read(buffer)) != -1) {
6     respuestaServidor1.write(buffer, 0, bytesRead);
7 }
8
9 // Leer la respuesta de Servidor-2 (sin enviarla al cliente)
10 while ((bytesRead = entradaServidor2Binaria.read(buffer)) != -1) {
11     // Solo la leemos, pero no la enviamos
12 }
```

- e) **Envío de la respuesta al cliente:** Finalmente, el proxy envía la respuesta de **Servidor-1** al cliente.

```
1 // Enviar la respuesta del Servidor-1 al cliente
2 salidaCliente.write(respuestaServidor1.toByteArray());
3 salidaCliente.flush();
```

- 4.1.4. **Manejo de excepciones:** El código maneja posibles errores en la conexión con los clientes o servidores mediante bloques try-catch. Si ocurre un error, se imprime un mensaje en la consola.

```
1 } catch (IOException e) {
2     System.err.println("Error en la conexión: " + e.getMessage());
3     } finally {
4         try {
5             if (cliente != null) cliente.close();
6         } catch (IOException e2) {
7             e2.printStackTrace();
8         }
9     }
10 }
```

En resumen, el **AdministradorTráfico.java** implementa un proxy inverso que distribuye las solicitudes de los clientes entre dos servidores HTTP, gestionando eficazmente la comunicación y optimizando el tráfico al descartar las respuestas no necesarias. La conexión y lectura de datos se realiza de manera asíncrona utilizando hilos, lo que permite manejar múltiples solicitudes de manera concurrente.

5. Ejecución del Trafico en Ubuntu.

5.1. Creación de Máquinas Virtuales en Azure.

Para la implementación del Administrador de Tráfico, es necesario crear tres máquinas virtuales en Azure con la siguiente configuración:

- **Sistema Operativo:** Ubuntu 24
- **Tamaño:** B1s (1 GB de memoria RAM, 1 CPU virtual)
- **Almacenamiento:** Disco tipo HDD con capacidad de 30 GB

El nombre de cada máquina virtual debe seguir la nomenclatura: T1-U-<número de boleta>-<número de máquina>. Por ejemplo, en este caso el número de boleta será: **2022630095**, los nombres de las máquinas quedarían de la siguiente manera:

- Primera máquina virtual: **T1-U-2022630095-1** | usuario: **adminHTTP**
- Segunda máquina virtual: **T1-U-2022630095-2** | usuario: **server1**
- Tercera máquina virtual: **T1-U-2022630095-3** | usuario: **server2**

Inicio >




Máquinas virtuales

Instituto Politécnico Nacional

+ Crear ▾ Cambiar al modo clásico ⌚ Reservas ▾ ⚙ Administrar vista ▾ ↻ Actualizar ⬇ Exportar a CSV 🔗 Abrir consulta | 🏷 Asignar etiquetas ▶ Inicio ↺ Reiniciar ⋮

Filtrar por cualquier ca... Suscripción es igual a **todo** Tipo es igual a **todo** Grupo de recursos es igual a **todo** × Ubicación es igual a **todo** × 🔍 Agregar filtro

Mostrando de 1 a 3 de 3 registros. Sin agrupar ▾ ≡ Vista de lista ▾

<input type="checkbox"/>	Nombre ↑↓	Suscripción ↑↓	Grupo de recursos ↑↓	Ubicación ↑↓	Estado ↑↓	Sistema operativo ↑↓	Cambiar el tamaño ↑↓	Dirección IP públ...
<input type="checkbox"/>	 T1-U-2022630095-1	Azure for Students	Assignment_1	West US 3	En ejecución	Linux	Standard_B1s	20.38.47.162
<input type="checkbox"/>	 T1-U-2022630095-2	Azure for Students	Assignment_1	West US 3	En ejecución	Linux	Standard_B1s	20.163.75.95
<input type="checkbox"/>	 T1-U-2022630095-3	Azure for Students	Assignment_1	West US 3	En ejecución	Linux	Standard_B1s	20.38.45.194

Para garantizar la conectividad y comunicación entre estas máquinas virtuales, es fundamental que sus direcciones IP privadas pertenezcan a la misma subred. En este caso, se asignarán las siguientes direcciones IP privadas dentro de la misma subred: **10.0.0.4** para la primera máquina, **10.0.0.5** para la segunda y **10.0.0.6** para la tercera. Esto facilitará la comunicación entre las instancias, asegurando que puedan intercambiar información de manera eficiente sin restricciones adicionales dentro de la red interna de Azure.

5.2. Configuración de las Máquinas Virtuales.

En la primera máquina virtual (**T1-U-2022630095-1**), se debe subir y ejecutar el programa `AdministradorTrafico.java`. Este programa deberá utilizar el puerto 8080 para su funcionamiento. En la segunda y tercera máquina virtual (**T1-U-2022630095-2** & **T1-U-2022630095-3**), se debe subir el archivo `ServidorHTTP.java`, el cual actuará como servidor web.

Para subir un archivo desde tu directorio local a una máquina virtual en Azure mediante la línea de comandos usando `scp`, sigue estos pasos:

Requisitos previos:

1. **Tener la dirección IP pública** de la máquina virtual en Azure.
2. **Contar con un usuario habilitado** en la VM de Azure.
3. **Tener configurada la clave SSH** en la VM de Azure si el acceso es por clave pública/privada.
4. **Asegurarse de que el puerto 22 está abierto** en el grupo de seguridad de la VM.

Comando `scp` para subir un archivo

Desde tu terminal en la máquina local, navega hasta el directorio donde se encuentra el archivo que deseas subir y ejecuta:



```
Administrator: Windows PowerShell
scp archivo.ext usuario@ip-publica:~
```

- **archivo.ext:** Nombre del archivo que deseas subir.
- **usuario:** Usuario con acceso a la máquina virtual en Azure.
- **ip-publica/privada:** Dirección IP pública de la máquina virtual.
- **~:** Directorio home del usuario en la VM.

Nota: Es importante realizar mapeo de puertos es necesario para redirigir las solicitudes entrantes en los puertos estándar **80** (HTTP) y **443** (HTTPS) a los puertos internos **8080** y **8443**, donde realmente se están ejecutando los servidores. Esto permite que los usuarios accedan al servicio sin necesidad de especificar puertos no convencionales en la URL.

a) Para redirigir el tráfico del puerto **80** al puerto **8080**, ejecutar el siguiente comando:

```
sudo iptables -A PREROUTING -t nat -p tcp --dport 80 -j REDIRECT --to-port 8080
```

b) Para redirigir el tráfico del puerto **443** al **8443**, ejecutar el siguiente comando:

```
sudo iptables -A PREROUTING -t nat -p tcp --dport 443 -j REDIRECT --to-port 8443
```

5.3. Subir ServidorHTTP.java a las VMs 2 & 3.

En las máquinas **T1-U-2022630095-2** & **T1-U-2022630095-3**, se debe subir **ServidorHTTP.java**, que actuará como servidor web.

Para la segunda máquina virtual:

```
Administrator: Windows PowerShell
scp ServidorHTTP.java server1@10.0.0.5:~
```

Para la tercera máquina virtual:

```
Administrator: Windows PowerShell
scp ServidorHTTP.java server2@10.0.0.6:~
```

Después de subirlo, accede a cada VM con:

```
Administrator: Windows PowerShell
ssh server1@10.0.0.5
```

```
Administrator: Windows PowerShell
ssh server2@10.0.0.6
```

Compila y ejecuta en cada una:

```
Administrator: Windows PowerShell
javac ServidorHTTP.java
java ServidorHTTP
```

Observemos que, en ambas instancias se comienzan a ejecutar el servidor HTTP en Java. En la primera terminal, el servidor **ServidorHTTP.java** se compila y ejecuta en el puerto **8081**, mientras que en la segunda se ejecuta en el puerto **8082**. Ambos servidores están en espera de conexiones, listos para manejar solicitudes entrantes.

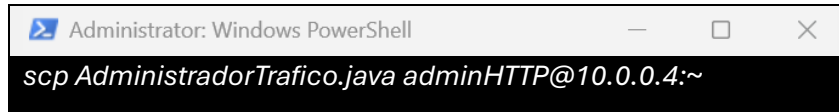
```
server1@T1-U-2022630095-2: ~$ ls
'ServidorHTTP$Worker.class'  ServidorHTTP.class  ServidorHTTP.java
server1@T1-U-2022630095-2: ~$ javac ServidorHTTP.java
server1@T1-U-2022630095-2: ~$ java ServidorHTTP.java 8081
Intentando iniciar servidor en puerto: 8081
Servidor HTTP escuchando en puerto: 8081
Esperando conexión...

server2@T1-U-2022630095-3: ~$ ls
'ServidorHTTP$Worker.class'  ServidorHTTP.class  ServidorHTTP.java
server2@T1-U-2022630095-3: ~$ javac ServidorHTTP.java
server2@T1-U-2022630095-3: ~$ java ServidorHTTP.java 8082
Intentando iniciar servidor en puerto: 8082
Servidor HTTP escuchando en puerto: 8082
Esperando conexión...
```

5.4. Subir y ejecutar AdministradorTrafico.java en la primera VM.

En la máquina virtual **T1-U-2022630095-1**, se debe subir el archivo `AdministradorTrafico.java`, el cual usará el puerto 8080.

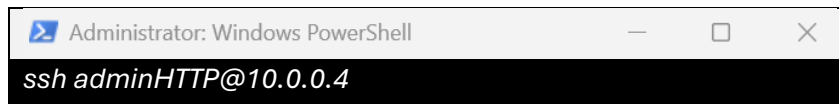
Desde tu máquina local, ejecuta:



```
Administrator: Windows PowerShell
scp AdministradorTrafico.java adminHTTP@10.0.0.4:~
```

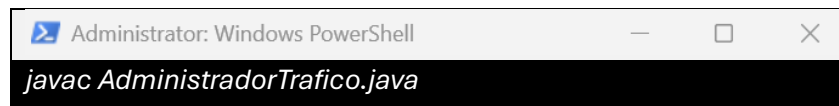
- **adminHTTP:** Nombre de usuario en la VM.
- **10.0.0.4:** Dirección IP pública de la VM 1.
- **~:** Indica que el archivo se copiará en el directorio home del usuario.

Para ejecutar el programa en la VM después de subirlo, accede a ella mediante SSH:



```
Administrator: Windows PowerShell
ssh adminHTTP@10.0.0.4
```

Y compílalo con:



```
Administrator: Windows PowerShell
javac AdministradorTrafico.java
```

Luego, ejecútalo asegurándote de usar el puerto 80:



```
Administrator: Windows PowerShell
java AdministradorTrafico 80 10.0.0.5 8081 10.0.0.6 8082
```

- **80:** Es el puerto asignado a la maquina virtual **T1-U-2022630095-1**
- **10.0.0.5:** Es la IP Privada de la maquina virtual **T1-U-2022630095-2**
- **8081:** Es el puerto asignado a la maquina virtual **T1-U-2022630095-2**
- **10.0.0.6:** Es la IP Privada de la maquina virtual **T1-U-2022630095-3**
- **8082:** Es el puerto asignado a la maquina virtual **T1-U-2022630095-3**

Antes de continuar con la implementación de este comando, utilizaremos la herramienta de comando **curl** para garantizar que la comunicación entre el cliente y los servidores se haya establecido correctamente en la configuración del Proxy HTTPS Inverso. Con el comando **curl http://ip-publica/privada:puerto**. Este comando permite verificar que se pueda acceder a los servidores internos a través de su IP Privada, antes de realizar pruebas más complejas. Al ejecutar **curl**, se puede comprobar si hay puertos bloqueados o problemas al establecer la conexión, ya que el comando devuelve un código de estado HTTP que indica si la solicitud fue exitosa o si hubo algún error, como problemas de red o de configuración del firewall. Esta verificación preliminar es clave para asegurarse de que no existan impedimentos en la comunicación entre los servidores y facilitar la resolución de problemas antes de continuar con el proceso de implementación.

Veamos en la siguiente imagen, la implementación del comando **curl**:

The image displays three terminal windows from a development environment. The top-left window, titled 'server1@T1-U-2022630095-2', shows the execution of 'ls', 'javac ServidorHTTP.class', and 'java ServidorHTTP.java' to start an HTTP server on port 8081. It successfully receives a GET request from 10.0.0.4. The top-right window, titled 'server2@T1-U-2022630095-3', shows similar steps for a second server on port 8082, which also receives a GET request from 10.0.0.4. The bottom window, titled 'adminHTTP@T1-U-2022630095-1', shows the compilation and execution of 'AdministradorTrafico.java'. It attempts to act as a proxy between 10.0.0.5:8081 and 10.0.0.6:8082 but fails with a 'java.net.BindException: Permission denied' error. Below the error, it shows a curl command being used to send a GET request to the proxy on port 8081.

Observemos que la imagen muestra tres terminales de comandos en un entorno de desarrollo.

- En la **primera terminal** (arriba a la izquierda), se ejecuta un servidor HTTP en el puerto 8081. Se observa la salida de comandos como **ls**, **javac ServidorHTTP.java**, y **java ServidorHTTP**. El servidor HTTP está escuchando en el puerto **8081** y ha recibido una petición **GET** desde la dirección IP **10.0.0.6**.
- En la **segunda terminal** (arriba a la derecha), se ejecuta otro servidor HTTP en el puerto 8082. Similarmente, se observan los comandos **ls**, **javac ServidorHTTP.java**, y **java ServidorHTTP**. Este servidor está escuchando en el puerto **8082** y también ha recibido una petición **GET** desde la dirección IP **10.0.0.6**.
- En la **tercera terminal** (abajo), se intenta ejecutar un programa llamado **AdministradorTrafico** con los comandos **javac AdministradorTrafico.java** y **java AdministradorTrafico 10.0.0.5 8081 10.0.0.6 8082**. Sin embargo, se encuentra con un error de **java.net.BindException: Permission denied**, lo que indica que el programa no tiene permisos para enlazar el socket en el puerto especificado. La salida muestra una traza de la pila de errores que incluye varias clases de Java relacionadas con sockets y networking.

Esta imagen es relevante porque muestra el proceso de configuración y ejecución de servidores HTTP y la gestión del tráfico entre ellos, así como la resolución de problemas comunes como errores de permisos en la vinculación de sockets.

Una vez destacada la importancia de verificar la correcta conexión entre las máquinas virtuales utilizando herramientas como curl, y solucionados los posibles inconvenientes relacionados con el establecimiento de la comunicación entre ellas, procedemos a ejecutar el siguiente comando: `java AdministradorTrafico 80 10.0.0.5 8081 10.0.0.6 8082`. Este comando permite iniciar la gestión del tráfico entre las máquinas virtuales, utilizando los puertos correspondientes para asegurar que los servidores estén correctamente configurados y que la comunicación fluya sin interrupciones.

```
server1@T1-U-2022630095-2: ~
Encabezado: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=0,i
Encabezado: Accept-Encoding: gzip, deflate
Encabezado: Connection: keep-alive
Conexión aceptada desde: /10.0.0.4
Esperando conexión...
Petición recibida: GET /favicon.ico HTTP/1.1
Conexión aceptada desde: /10.0.0.4
Esperando conexión...
Petición recibida: GET /suma?a=1&b=2&c=3 HTTP/1.1
Encabezado: Host: 20.38.47.162
Encabezado: Referer: http://20.38.47.162/
Encabezado: Accept: /*
Encabezado: User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 18_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.3 Mobile/15E148 Safari/604.1
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=3,i
Encabezado: Accept-Encoding: gzip, deflate
Encabezado: Connection: keep-alive

server2@T1-U-2022630095-3: ~
Encabezado: Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=0,i
Encabezado: Accept-Encoding: gzip, deflate
Encabezado: Connection: keep-alive
Conexión aceptada desde: /10.0.0.4
Esperando conexión...
Petición recibida: GET /favicon.ico HTTP/1.1
Conexión aceptada desde: /10.0.0.4
Esperando conexión...
Petición recibida: GET /suma?a=1&b=2&c=3 HTTP/1.1
Encabezado: Host: 20.38.47.162
Encabezado: Referer: http://20.38.47.162/
Encabezado: Accept: /*
Encabezado: User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 18_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.3 Mobile/15E148 Safari/604.1
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=3,i
Encabezado: Accept-Encoding: gzip, deflate
Encabezado: Connection: keep-alive

adminHTTP@T1-U-2022630095-1: ~
Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

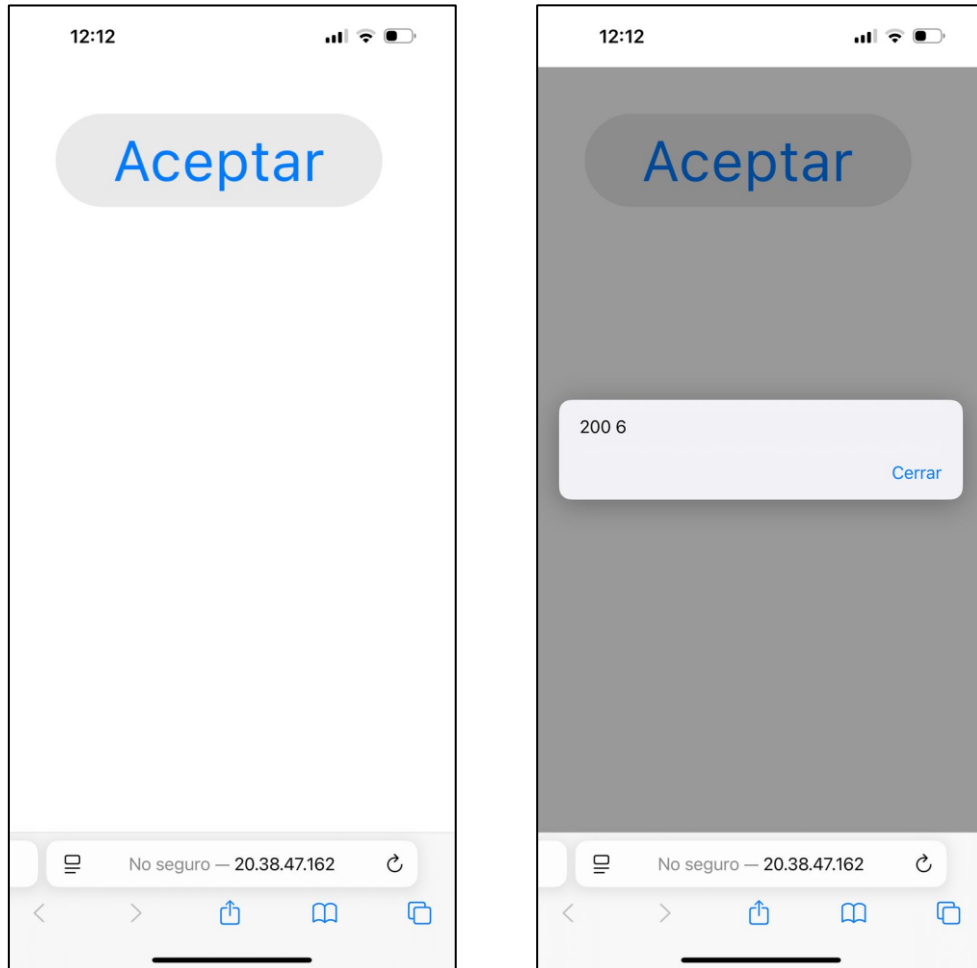
Last login: Tue Mar 11 03:53:14 2025 from 187.190.88.92
adminHTTP@T1-U-2022630095-1:~$ sudo ufw disable
Firewall stopped and disabled on system startup
adminHTTP@T1-U-2022630095-1:~$ sudo iptables -A PREROUTING -t nat -p tcp --dport 443 -j REDIRECT --to-port 8443
adminHTTP@T1-U-2022630095-1:~$ sudo javac AdministradorTrafico.java
adminHTTP@T1-U-2022630095-1:~$ sudo java AdministradorTrafico 80 10.0.0.5 8081 10.0.0.6 8082
Proxy escuchando en puerto: 80
Redirigiendo tráfico entre 10.0.0.5:8081 y 10.0.0.6:8082
Petición recibida en el proxy: GET / HTTP/1.1
Respuesta del Servidor-2 recibida, pero no enviada al cliente.
Petición recibida en el proxy: GET / HTTP/1.1
Respuesta del Servidor-2 recibida, pero no enviada al cliente.
Petición recibida en el proxy: GET /favicon.ico HTTP/1.1
Respuesta del Servidor-2 recibida, pero no enviada al cliente.
Petición recibida en el proxy: GET /suma?a=1&b=2&c=3 HTTP/1.1
Respuesta del Servidor-2 recibida, pero no enviada al cliente.
```

En las dos ventanas superiores, se ejecutan dos servidores diferentes. Ambas ventanas muestran encabezados HTTP de solicitudes **GET**, incluyendo detalles como "Accept", "Accept-Language", "Accept-Encoding", "Connection", "User-Agent", entre otros. También se observa que ambas conexiones aceptan solicitudes desde la dirección IP "10.0.0.4" y se reciben peticiones para `/favicon.ico HTTP/1.1` y `/suma?1b8=28c3 HTTP/1.1`. El navegador utilizado es **Mozilla/5.0** en un **iPhone con iOS 18_3_1**.

En la ventana inferior, se ejecutan comandos para administrar el tráfico HTTP. Se ha deshabilitado el firewall con el comando `sudo ufw disable` y se han añadido reglas de **iptables** para redirigir el tráfico de los puertos **443** y **8081** al puerto **8443**. Luego, se ejecuta un archivo Java llamado `AdministradorTrafico.java` y se inicia un proxy escuchando en el puerto **8082**. La ventana muestra mensajes indicando que se ha recibido tráfico en el puerto 8081 y que se ha enviado al cliente, así como solicitudes **GET** para `/favicon.ico HTTP/1.1` y `/suma?1b8=28c3 HTTP/1.1`.

5.5. Prueba de acceso al servidor desde un dispositivo externo.

Con el fin de verificar que la configuración se haya realizado de forma correcta, es necesario realizar una prueba de acceso desde un dispositivo externo, mientras los procesos se ejecutan en las máquinas virtuales. Para ello, se debe ingresar la siguiente URL en un navegador web desde un teléfono inteligente o tableta:



En la imagen se muestra un dispositivo móvil que navega en una página web con una conexión no segura, evidenciada por el mensaje "**No seguro**" en la barra de direcciones que muestra la IP **20.38.47.162**. En el centro de la pantalla hay un botón que dice "Aceptar". La información provista es que el navegador utilizado es **Mozilla/5.0** en un **iPhone con iOS 18_3_1**.

Esta imagen está relacionada con la imagen anterior de las terminales en varios aspectos:

1. **Dirección IP y Dispositivo:** La dirección IP mostrada en el dispositivo móvil (**20.38.47.162**) indica que el dispositivo está realizando conexiones a un servidor específico. En la imagen anterior, las terminales muestran cómo las peticiones son recibidas y procesadas por los servidores HTTP configurados en las máquinas virtuales con IPs privadas dentro de la misma subred, como **10.0.0.5** y **10.0.0.6**. La conexión no segura también indica que el tráfico no está encriptado, lo cual puede ser relevante al analizar las configuraciones de seguridad en la red.
2. **Administración del Tráfico y Proxy:** La ventana de comandos que muestra el "AdministradorTráfico" en la imagen anterior demuestra cómo el tráfico HTTP es gestionado y redirigido a diferentes servidores. Las peticiones realizadas desde el dispositivo móvil son redirigidas y balanceadas por este administrador de tráfico, facilitando la comunicación eficiente entre el cliente (el dispositivo móvil) y los servidores.

En resumen, la interacción del dispositivo móvil con la página web es un ejemplo práctico de cómo las configuraciones y procesos observados en la imagen de las terminales se reflejan en el uso real. Las conexiones, las peticiones HTTP y la gestión del tráfico son elementos clave que conectan ambas imágenes y demuestran la funcionalidad del entorno configurado.

5.6. Resumen.

Hasta este punto, se ha logrado avanzar de manera significativa en el desarrollo e implementación del Proxy HTTPS Inverso con Servidores HTTP en la Nube utilizando Microsoft Azure. Se ha completado la creación y configuración de las máquinas virtuales, la apertura de puertos necesarios, y la ejecución de los programas de servidor y administrador de tráfico. Además, se ha realizado con éxito la prueba de acceso al servidor desde un dispositivo móvil, lo cual indica que la configuración básica del sistema es funcional.

Sin embargo, es importante destacar que la conexión establecida hasta este punto es no segura, tal como se observa en la interfaz del dispositivo móvil, donde el navegador muestra el mensaje "No seguro" debido a la falta de un certificado HTTPS. Este aspecto será clave en las siguientes etapas, ya que es necesario implementar una conexión segura con HTTPS para garantizar la integridad y privacidad de las comunicaciones entre el cliente y los servidores.

En resumen, los resultados obtenidos hasta ahora confirman que la infraestructura de servidores y la gestión del tráfico están correctamente configuradas, pero se requiere una implementación adicional de certificados de seguridad para completar la configuración del Proxy HTTPS Inverso y asegurar el tráfico de la red.

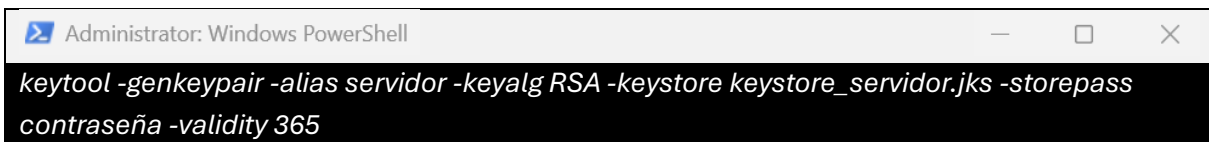
6. Implementación de Sockets Seguros (SSL).

Hasta este punto, la implementación del proxy inverso ha manejado las solicitudes HTTP de manera no segura. En esta sección, se procederá a la configuración de sockets seguros mediante SSL, permitiendo el cifrado de las comunicaciones entre el cliente y el servidor. Para ello, se modificará el programa **AdministradorTrafico.java** para que utilice conexiones seguras, lo que resultará en la creación de un nuevo programa denominado **AdministradorTraficoSSL.java**. Además, se empleará un keystore con un certificado autofirmado, específicamente **keystore_servidor.jks**, para habilitar la autenticación y el cifrado.

6.1. Implementación & Configuración del “keystore”.

Para habilitar la comunicación segura a través del protocolo HTTPS, es necesario crear un keystore en la primera máquina virtual. Este keystore funcionará como un repositorio seguro que almacenará la clave privada y el certificado digital autofirmado requerido para establecer conexiones cifradas. La utilización de este componente es esencial para habilitar la autenticación del servidor y proteger la integridad y confidencialidad de los datos transmitidos, especialmente ahora que se incorporará SSL al flujo de comunicación mediante la versión segura del programa, **AdministradorTraficoSSL.java**. La correcta configuración del keystore garantiza que las conexiones establecidas por el proxy inverso se realicen bajo un esquema seguro. Este procedimiento se compone de las siguientes etapas, que permitirán generar y preparar el keystore para su uso en el entorno de ejecución:

1. Acceder a la primera máquina virtual mediante SSH.
2. Utilizar el siguiente comando para generar un keystore con un certificado autofirmado:



```
Administrator: Windows PowerShell
keytool -genkeypair -alias servidor -keyalg RSA -keystore keystore_servidor.jks -storepass
contraseña -validity 365
```

Donde:

- **-alias servidor:** Define un alias para la clave.
- **-keyalg RSA:** Especifica el algoritmo de cifrado.
- **-keystore keystore_servidor.jks:** Crea el archivo del keystore.
- **-storepass contraseña:** Define la contraseña del keystore.
- **-validity 365:** Establece la validez del certificado en 365 días.

3. Se solicitará ingresar información como el nombre del propietario y la organización. Es importante recordar la contraseña establecida, ya que será necesaria para configurar el proxy HTTPS. Al aplicar las indicaciones previas, se observa que:

```
adminHTTP@T1-U-2022630095-1: ~  
adminHTTP@T1-U-2022630095-1:~$ keytool -genkeypair -alias servidor -keyalg RSA -keysize 2048 -validity 365 -keystore keystore_servidor.jks  
Enter keystore password:  
Re-enter new password:  
What is your first and last name?  
[Unknown]: Alejandro Osorio  
What is the name of your organizational unit?  
[Unknown]: IPN  
What is the name of your organization?  
[Unknown]: IPN  
What is the name of your City or Locality?  
[Unknown]: CDMX  
What is the name of your State or Province?  
[Unknown]: CDMX  
What is the two-letter country code for this unit?  
[Unknown]: MX  
Is CN=Alejandro Osorio, OU=IPN, O=IPN, L=CDMX, ST=CDMX, C=MX correct?  
[no]: yes  
  
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 365 days  
for: CN=Alejandro Osorio, OU=IPN, O=IPN, L=CDMX, ST=CDMX, C=MX  
adminHTTP@T1-U-2022630095-1:~$
```

Es importante verificar cuidadosamente los datos antes de generar el archivo del almacén de claves, ya que este paso resulta fundamental para habilitar comunicaciones seguras a través de cifrado RSA.

6.2. Modificación del Administrador de Tráfico para soportar HTTPS.

Con el objetivo de dotar al proxy de una capa adicional de seguridad mediante el uso de conexiones cifradas, se procede a adaptar el archivo original **AdministradorTráfico.java** para crear una nueva versión denominada **AdministradorTráficoSSL.java**. Esta adaptación implica una serie de ajustes y configuraciones específicas orientadas a habilitar el uso del protocolo **SSL/TLS** en la comunicación entre el cliente y el proxy. A continuación, se detallan los cambios más relevantes realizados durante este proceso:

1. Importación de bibliotecas adicionales

Se agregaron las siguientes clases para habilitar el uso de SSL/TLS:

```
1 import javax.net.ssl.*;  
2 import java.security.KeyStore;  
3 import javax.net.ssl.KeyManagerFactory;  
4 import javax.net.ssl.SSLContext;
```

2. Sustitución de sockets estándar por sockets seguros

Se reemplaza los objetos **ServerSocket** y **Socket** utilizados en la versión original por sus equivalentes seguros **SSLServerSocket** y **SSLSocket**, respectivamente. Esta modificación permite establecer canales de comunicación cifrados entre el cliente y el proxy, garantizando confidencialidad e integridad de los datos transmitidos.

3. Configuración de un contexto SSL (SSLContext)

Se incorpora la inicialización de un objeto **SSLContext**, el cual define el entorno criptográfico para las conexiones seguras. Este contexto es configurado para utilizar el protocolo "TLS", habilitando así mecanismos de cifrado modernos y robustos.

```
1 sslContext = SSLContext.getInstance("TLS");
```

4. Carga de un almacén de claves (keystore).

Para permitir la autenticación del servidor en el contexto SSL, se integra la carga de un archivo keystore en formato **PKCS12 (keystore_servidor.jks)**. Este almacén contiene el certificado y la clave privada que identifican al servidor ante los clientes. La carga se realizó a través de un objeto **KeyStore**, proporcionando la contraseña correspondiente para su acceso.

```
1 KeyStore keyStore = KeyStore.getInstance("PKCS12");
2 FileInputStream keyFile = new FileInputStream("keystore_servidor.jks");
3 keyStore.load(keyFile, "password".toCharArray());
```

5. Inicialización de un administrador de claves (KeyManagerFactory).

Se utilizó la clase **KeyManagerFactory** para asociar el almacén de claves al contexto SSL. Este paso fue esencial para que el servidor pudiera presentar su certificado digital al establecer la conexión segura.

```
1 KeyManagerFactory keyManagerFactory = KeyManagerFactory.getInstance(KeyManagerFactory.getDefaultAlgorithm());
2 keyManagerFactory.init(keyStore, "19052003".toCharArray());
3
4 sslContext.init(keyManagerFactory.getKeyManagers(), null, null);
```

6. Ajustes en la lógica de aceptación de conexiones.

Se modificó el ciclo principal del servidor para que acepte conexiones entrantes utilizando **SSLServerSocket.accept()**, retornando objetos **SSLSocket** que posteriormente son procesados por hilos individuales (**Worker**) diseñados para manejar sesiones SSL.

Código agregado/modificado:

```
1 SSLServerSocketFactory sslServerSocketFactory = sslContext.getServerSocketFactory();
2 SSLServerSocket serverSocket = (SSLServerSocket) sslServerSocketFactory.createServerSocket(puertoLocal);
```

Así mismo, para **SSLSocket** cliente:

```
1 SSLSocket cliente = (SSLSocket) serverSocket.accept();
```

Cambios en la clase **Worker** para aceptar **SSLSocket**.

```
1 static class Worker extends Thread {
2     SSLSocket cliente;
3     ...
4 }
```

7. Compatibilidad con la lógica de reenvío existente.

A pesar de los cambios introducidos para habilitar SSL, se mantuvo la lógica original de reenvío de peticiones a dos servidores remotos. El flujo de datos no sufrió modificaciones en su esencia, lo que garantizó la compatibilidad funcional con la versión anterior no cifrada.

Todos estos cambios en conjunto permiten al nuevo archivo `AdministradorTraficoSSL.java` operar como un proxy seguro con conexiones cifradas entre el cliente y el servidor proxy, utilizando certificados digitales y el protocolo TLS.

En la siguiente tabla se puede apreciar con mayor claridad y de forma más estructurada los cambios realizados entre ambas versiones del código.

Tabla Comparativa.

Elemento modificado	AdministradorTrafico.java	AdministradorTraficoSSL.java
Socket de cliente	Socket	SSLSocket
Socket del servidor local	ServerSocket	SSLServerSocket
Contexto de seguridad	No se utiliza	SSLContext con KeyStore y KeyManagerFactory
Almacén de claves	No requerido	Se carga keystore_servidor.jks
Cifrado de datos	No cifrado	Cifrado TLS habilitado

6.3. Ejecución del Administrador de Trafico con SSL.

Una vez que el programa **AdministradorTraficoSSL.java** ha sido configurado correctamente, así como, implementado y compilado en la primera máquina virtual, es momento de ejecutar el proxy con soporte para sockets seguros. Este programa tiene como objetivo establecer canales de comunicación cifrados mediante **SSL/TLS** entre el cliente y el proxy, permitiendo así una transferencia de datos protegida y confidencial.

Para su ejecución, es indispensable contar con el archivo del keystore previamente generado o cargado en el sistema de archivos de la primera máquina virtual. En esta práctica se ha utilizado un keystore denominado **keystore_servidor.jks**, que contiene un certificado autofirmado utilizado por el servidor SSL.

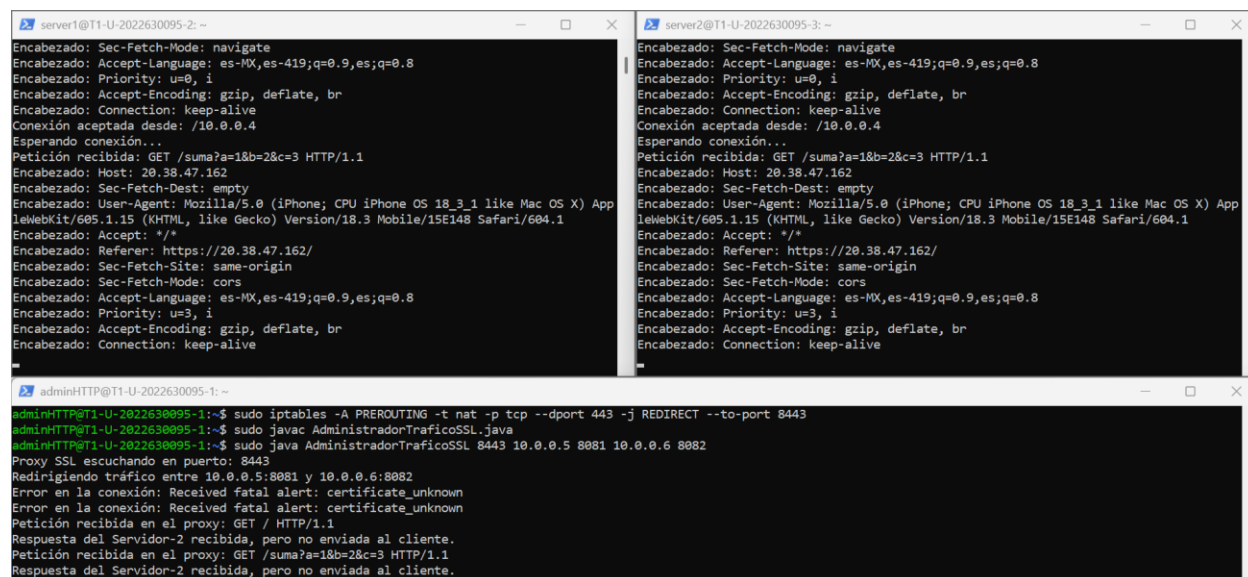
El archivo **.class** resultante de la compilación de AdministradorTraficoSSL.java deberá ejecutarse empleando el siguiente comando desde la terminal de la máquina virtual:

```
Administrator: Windows PowerShell
sudo java AdministradorTraficoSSL.java 8443 10.0.0.5 8081 10.0.0.6 8082
```

Nota: Asigne los puertos correspondientes para establecer una conexión segura **https (8443)**, así como, los puertos configurados previamente asignados para establecer conexión con las MV correspondientes.

Con este comando, se lanza el servidor proxy seguro en la primera máquina virtual, el cual se mantiene a la escucha del puerto **8443**, en espera de solicitudes HTTPS provenientes de clientes remotos. Esta implementación garantiza que toda la comunicación entre el cliente y el proxy se encuentra cifrada, cumpliendo con los principios fundamentales de la seguridad de la información: confidencialidad, integridad y autenticación.

A continuación, se muestra la imagen correspondiente donde se observa la ejecución de dicho comando y la posterior conexión establecida con las otras dos máquinas virtuales y el cliente.



```
server1@T1-U-2022630095-2: ~
Encabezado: Sec-Fetch-Mode: navigate
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=0, i
Encabezado: Accept-Encoding: gzip, deflate, br
Encabezado: Connection: keep-alive
Conexión aceptada desde: /10.0.0.4
Esperando conexión...
Petición recibida: GET /suma?a=1&b=2&c=3 HTTP/1.1
Encabezado: Host: 20.38.47.162
Encabezado: Sec-Fetch-Dest: empty
Encabezado: User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 18_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.3 Mobile/15E148 Safari/604.1
Encabezado: Accept: */*
Encabezado: Referer: https://20.38.47.162/
Encabezado: Sec-Fetch-Site: same-origin
Encabezado: Sec-Fetch-Mode: cors
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=3, i
Encabezado: Accept-Encoding: gzip, deflate, br
Encabezado: Connection: keep-alive

server2@T1-U-2022630095-3: ~
Encabezado: Sec-Fetch-Mode: navigate
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=0, i
Encabezado: Accept-Encoding: gzip, deflate, br
Encabezado: Connection: keep-alive
Conexión aceptada desde: /10.0.0.4
Esperando conexión...
Petición recibida: GET /suma?a=1&b=2&c=3 HTTP/1.1
Encabezado: Host: 20.38.47.162
Encabezado: Sec-Fetch-Dest: empty
Encabezado: User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 18_3_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/18.3 Mobile/15E148 Safari/604.1
Encabezado: Accept: */*
Encabezado: Referer: https://20.38.47.162/
Encabezado: Sec-Fetch-Site: same-origin
Encabezado: Sec-Fetch-Mode: cors
Encabezado: Accept-Language: es-MX,es-419;q=0.9,es;q=0.8
Encabezado: Priority: u=3, i
Encabezado: Accept-Encoding: gzip, deflate, br
Encabezado: Connection: keep-alive

adminHTTP@T1-U-2022630095-1: ~
adminHTTP@T1-U-2022630095-1:~$ sudo iptables -A PREROUTING -t nat -p tcp --dport 443 -j REDIRECT --to-port 8443
adminHTTP@T1-U-2022630095-1:~$ sudo javac AdministradorTráficoSSL.java
adminHTTP@T1-U-2022630095-1:~$ sudo java AdministradorTráficoSSL 8443 10.0.0.5 8081 10.0.0.6 8082
Proxy SSL escuchando en puerto: 8443
Redirigiendo tráfico entre 10.0.0.5:8081 y 10.0.0.6:8082
Error en la conexión: Received fatal alert: certificate_unknown
Error en la conexión: Received fatal alert: certificate_unknown
Petición recibida en el proxy: GET / HTTP/1.1
Respuesta del Servidor-2 recibida, pero no enviada al cliente.
Petición recibida en el proxy: GET /suma?a=1&b=2&c=3 HTTP/1.1
Respuesta del Servidor-2 recibida, pero no enviada al cliente.
```

En la visualización adjunta, se puede constatar la correcta ejecución del commando que inicia el servidor proxy seguro en la primera máquina virtual. En las terminales tanto de la máquina virtual como del cliente, se visualizan los encabezados HTTP y los parámetros de conexión. El tráfico se está redirigiendo de manera adecuada entre el cliente y el proxy, confirmando que este último está escuchando en el puerto 8443. Además, se observa el flujo de datos entre las máquinas virtuales y el cliente, lo que valida que el servidor proxy está operando correctamente, gestionando las solicitudes de manera cifrada y segura a través de HTTPS.

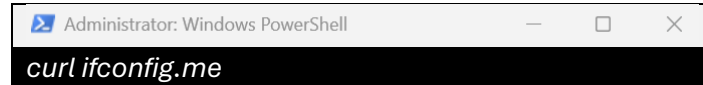
En el siguiente apartado, se abordará de manera detallada el procedimiento para establecer una conexión segura desde un dispositivo cliente externo. Se detallará el flujo de interacción entre el cliente y el servidor proxy, asegurando la protección de la comunicación mediante protocolos de cifrado y autenticación, y se explicará el uso del keystore para gestionar la seguridad de las conexiones.

6.4. Prueba de Comunicación Segura.

Una vez desplegado el proxy seguro en la primera máquina virtual, se procede a realizar la validación del funcionamiento del sistema accediendo desde un cliente externo a través de una conexión segura (HTTPS). Para ello, se utilizará un teléfono inteligente o tableta con conexión a Internet, el cual permitirá simular un entorno de acceso externo real, verificando así la correcta exposición del servicio y la respuesta del servidor ante solicitudes cifradas. Los pasos a seguir para llevar a cabo esta validación se describen a continuación:

1. Obtener la IP pública de la primera máquina virtual.

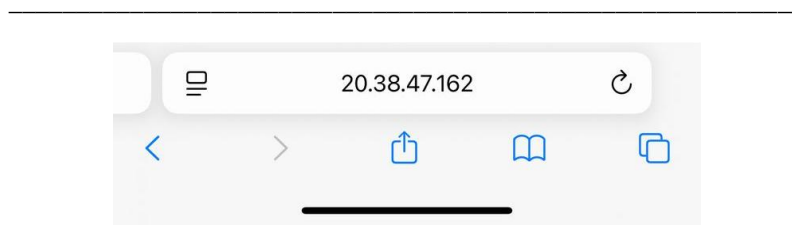
Esta dirección IP será proporcionada por Azure y se puede consultar desde el portal de administración de la plataforma o mediante el uso del comando ***curl ifconfig.me*** desde la terminal de la máquina virtual.



O bien, se puede acceder al recurso de la Máquina Virtual 1 a través del Portal de Azure. Dentro de la sección de información general del recurso, es posible visualizar la dirección IP pública asignada a dicha máquina.

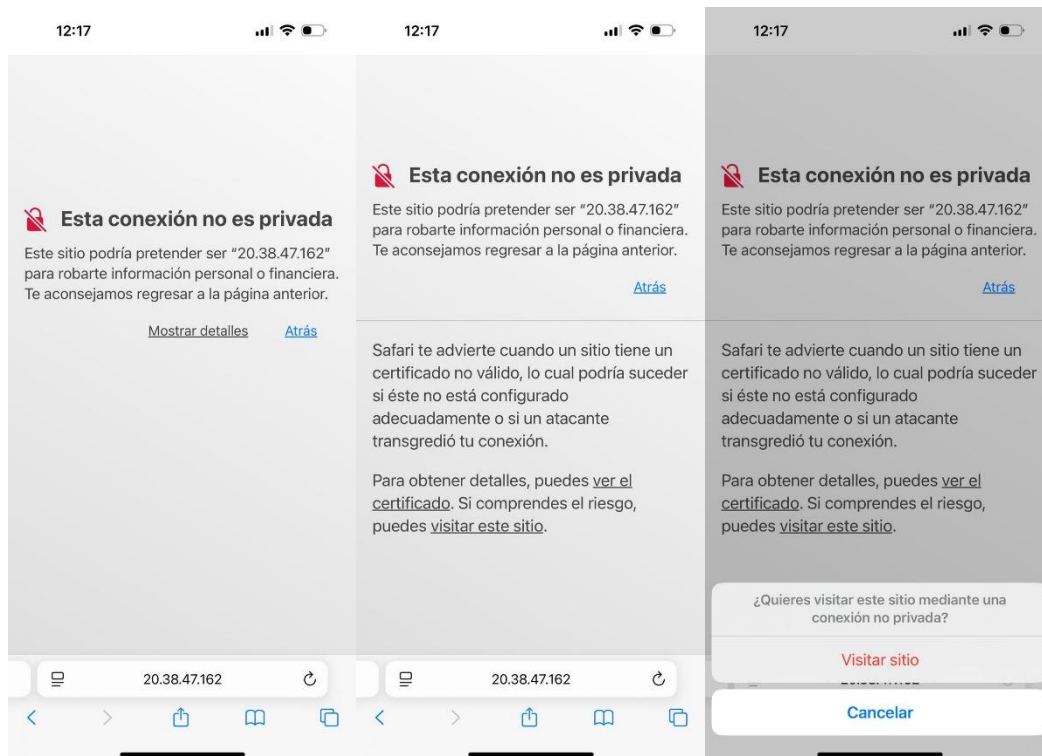
2. Ingresar la URL en el navegador del dispositivo móvil:

En el navegador web del teléfono o tableta, se deberá escribir la siguiente dirección, reemplazando ***ip-máquina-virtual*** por la dirección IP pública correspondiente. En el caso particular de la presente actividad, se deberá ingresar la siguiente URL: ***https://20.38.47.162***, ya que dicha dirección IP fue la asignada a la máquina virtual. Esta información también puede observarse en la imagen que se muestra a continuación.



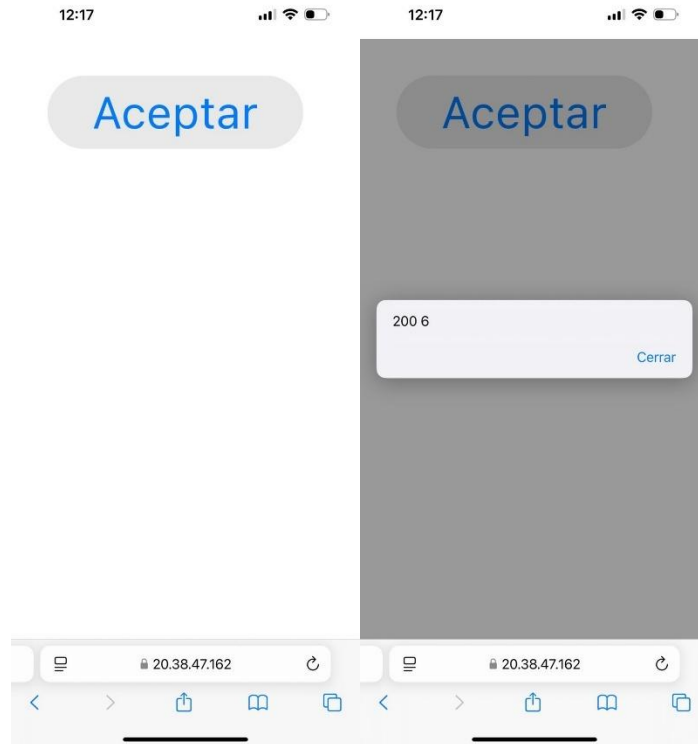
3. Aceptar el certificado autofirmado (si es requerido).

Debido a que se ha utilizado un certificado autofirmado en el keystore, es probable que el navegador emita una advertencia de seguridad. Se debe aceptar manualmente el riesgo para continuar con la conexión segura. Se adjuntan a continuación las capturas de pantalla que documentan la advertencia generada por el navegador al intentar establecer una conexión segura con un certificado autofirmado.



4. Visualizar el resultado del método web “suma”.

Una vez establecida la conexión, el navegador mostrará una interfaz con un botón. Al hacer clic en dicho botón, se enviará una solicitud HTTPS al proxy seguro, el cual se encargará de redirigirla a uno de los servidores HTTP (ubicados en la máquina virtual 2 y 3), obteniendo la respuesta y devolviéndola al cliente. Si la configuración ha sido realizada correctamente, el usuario visualizará el resultado del método web suma, lo cual confirmará que la comunicación cifrada mediante SSL entre el cliente y el proxy, y posteriormente entre el proxy y los servidores backend, se ha realizado con éxito.



6.5. Resumen.

Finalmente, se ha alcanzado la fase culminante de esta sección del documento, correspondiente a la implementación de seguridad en el proxy, integrando los componentes necesarios para habilitar una conexión segura mediante SSL. Esta etapa incluyó la creación y configuración del archivo **keystore_servidor.jks**, el cual permitió gestionar el certificado digital requerido para establecer comunicaciones cifradas. Posteriormente, se modificó el código del **AdministradorTrafico.java** para incorporar la funcionalidad HTTPS, dando paso a su ejecución en la primera máquina virtual bajo el nombre **AdministradorTraficoSSL.java**, con el objetivo de actuar como proxy seguro entre los clientes y los servidores remotos. Una vez desplegado, se procedió a validar el correcto funcionamiento del sistema mediante una prueba externa desde un dispositivo móvil, accediendo a la interfaz web mediante el protocolo seguro https y ejecutando la funcionalidad del método "suma". La exitosa respuesta a dicha solicitud confirmó no solo la operatividad del sistema bajo un canal seguro, sino también la correcta configuración y funcionamiento del keystore, lo que demuestra la eficacia de los sockets SSL en entornos distribuidos y el cumplimiento de los objetivos establecidos en esta etapa de la práctica.

7. Conclusión.

La implementación de un Proxy Inverso con soporte para HTTPS en un entorno distribuido, utilizando servidores HTTP desplegados en máquinas virtuales sobre Microsoft Azure, ha permitido validar de manera integral tanto los fundamentos teóricos como las competencias prácticas en el ámbito de la seguridad y administración del tráfico en sistemas distribuidos. Esta actividad se estructuró en etapas progresivas que abarcaron desde el desarrollo del servidor básico en Java hasta la configuración de un canal de comunicación cifrado mediante sockets SSL, consolidando el conocimiento técnico aplicado a escenarios reales en la nube.

Inicialmente, se abordó la creación de servidores HTTP funcionales mediante la modificación del programa `ServidorHTTP.java`, con lo cual se estableció la base para la recepción de peticiones y la ejecución de métodos web simples. Posteriormente, se desarrolló el componente central de esta práctica: el `AdministradorTráfico.java`, cuyo propósito fue actuar como proxy inverso entre el cliente y los servidores, administrando y redirigiendo el flujo de datos. A través de la incorporación de parámetros de configuración, clases multihilo (`Worker`) y un adecuado manejo de excepciones, este componente garantizó una operación eficiente y escalable.

La puesta en marcha de esta infraestructura sobre Azure implicó la creación, configuración y conexión entre máquinas virtuales distribuidas, permitiendo una emulación precisa de un entorno real. El despliegue del servidor en las máquinas virtuales 2 y 3, así como del Administrador de Tráfico en la primera, habilitó una topología funcional sobre la cual se realizaron pruebas de conectividad desde dispositivos externos. Estas pruebas confirmaron la operatividad del proxy en condiciones reales de red.

En una etapa posterior, se introdujo el concepto de comunicación segura mediante la implementación de sockets SSL. Esta fase supuso un nivel adicional de complejidad, requiriendo la generación e integración de un almacén de claves (`keystore_servidor.jks`) y la modificación del Administrador de Tráfico para soportar el protocolo HTTPS. La correcta ejecución del programa `AdministradorTráficoSSL.java` y la validación desde un dispositivo externo demostraron la eficacia de la solución desarrollada, garantizando confidencialidad, integridad y autenticación en la transmisión de datos.

Desde una perspectiva académica, esta práctica permitió consolidar los conocimientos adquiridos sobre el diseño y la implementación de soluciones distribuidas seguras, reforzando conceptos clave como el uso de proxies inversos, el manejo de conexiones concurrentes y la aplicación de criptografía en redes. A su vez, brindó una experiencia práctica con herramientas y servicios propios de la computación en la nube, como la plataforma Azure, afianzando competencias clave en entornos profesionales actuales.