



Instituto Politécnico Nacional



Escuela Superior de Cómputo

Desarrollo de Sistemas Distribuidos

*Tarea 4. Transferencia de archivos utilizando sockets
seguros*

López Sánchez Kevin Ian

Grupo: 4CV12

Profesor: Pineda Guerrero Carlos

05 de abril de 2023

Desarrollo

Creación de máquina virtual de Ubuntu Server en Azure

Para realizar esta práctica es necesario crear una máquina virtual de Ubuntu Server en nuestro portal de Azure y a continuación explicaremos los pasos para este proceso:

1. Entramos al portal de Azure y vamos al apartado de “Máquinas Virtuales”.

The screenshot shows the Microsoft Azure portal interface. In the top left, there's a search bar with the placeholder "Search resources, services, and docs (G+)". Below it, the "Azure services" sidebar is visible, with "Virtual machines" selected. The main content area is titled "Virtual machines" and contains a "Create" button and a "View" button. To the right, there are links to "Quickstart Center", "App Services", "SQL databases", "Kubernetes services", and "More services". On the left, under "Resources", there are sections for "Recent" and "Favorite" resources, and a "Name" filter. A "Description" section provides instructions for creating a virtual machine. Below this, there are three training video thumbnails: "Introduction to Azure virtual machines", "Create a Windows virtual machine in Azure", and "Create a Linux virtual machine in Azure". At the bottom, there are "Useful links" for "Overview" and "Compute". The browser address bar at the bottom shows the URL: <https://portal.azure.com/#blade/HubsExtension/BrowseResourceBlade/resourceType/Microsoft.Compute%2FVirtualMachines>. The status bar at the bottom right shows the date and time: 4/4/2023 23:29.

2. Seleccionamos la opción de “crear” y luego “Azure virtual machine”.

The screenshot shows the Microsoft Azure portal interface. The user is on the 'Virtual machines' page. A dropdown menu is open under the '+ Create' button, listing several options for creating a virtual machine:

- Azure virtual machine
- Azure virtual machine with preset configuration
- Azure Arc virtual machine
- Azure VMware Solution virtual machine

Below the dropdown, there is a link to 'Learn more about Windows virtual machines'.

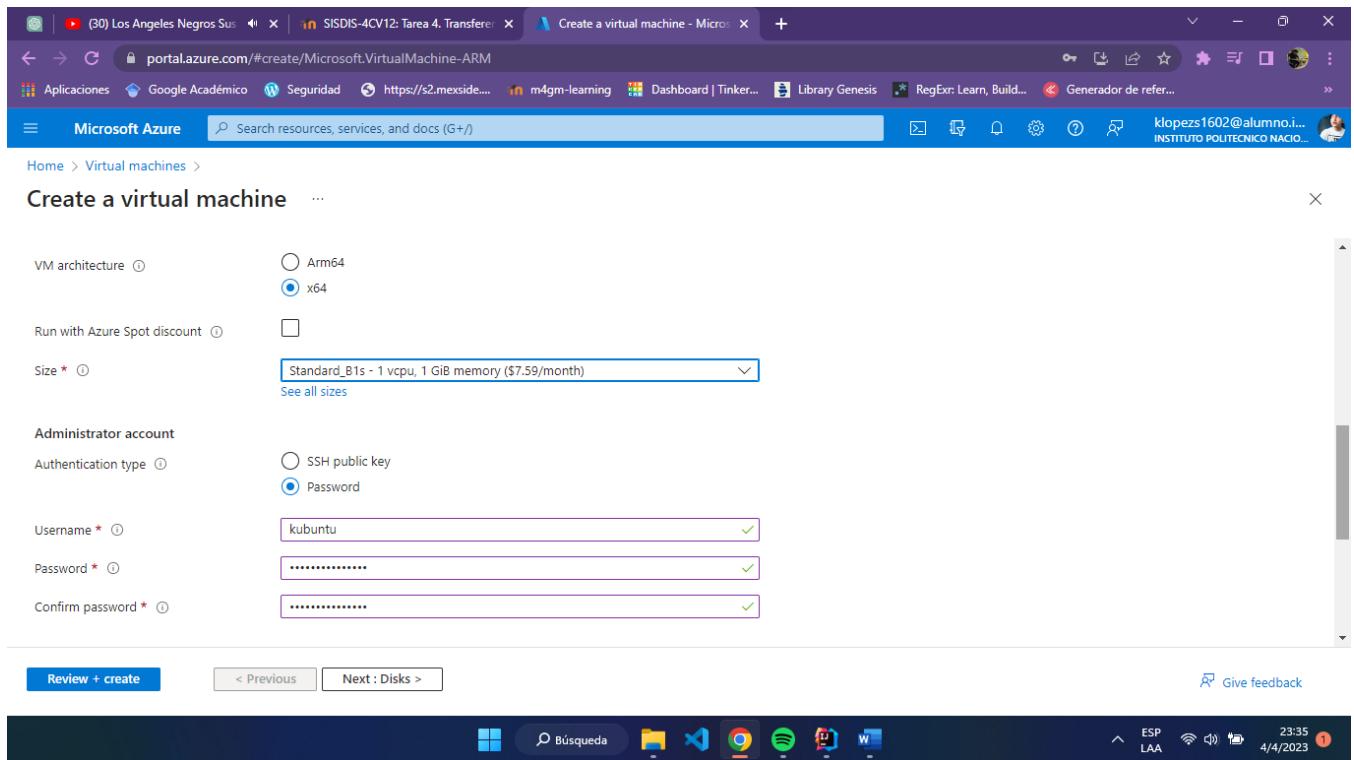
3. Seleccionamos el grupo de recursos o creamos uno nuevo, escribimos el nombre de nuestra máquina virtual y seleccionamos la opción de Ubuntu Server 18.04 LTS

The screenshot shows the 'Create a virtual machine' wizard in the Microsoft Azure portal. The user is on the 'Instance details' step. The form fields are filled as follows:

- Subscription: Azure for Students
- Resource group: (New) ESCOM
- Virtual machine name: T4-2020630205
- Region: (Europe) Switzerland North
- Availability options: No infrastructure redundancy required
- Security type: Standard
- Image: Ubuntu Server 18.04 LTS - x64 Gen2
- VM architecture: x64

At the bottom, there are navigation buttons: 'Review + create' (highlighted in blue), '< Previous', 'Next : Disks >', and 'Give feedback'.

4. Seleccionamos 1 Gb para la memoria RAM de nuestra máquina e ingresamos el usuario y la contraseña para ingresar posteriormente.



5. Seleccionamos el puerto de entrada por el que accederemos a la máquina virtual, en este caso el de SSH.

Create a virtual machine

Password * ⓘ
Confirm password * ⓘ

Inbound port rules

Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.

Public inbound ports * ⓘ None Allow selected ports

Select inbound ports * SSH (22)

⚠️ This will allow all IP addresses to access your virtual machine. This is only recommended for testing. Use the Advanced controls in the Networking tab to create rules to limit inbound traffic to known IP addresses.

Review + create < Previous Next : Disks > Give feedback

6. Seleccionamos como almacenamiento un HDD estándar

Create a virtual machine

VM disk encryption

Azure disk storage encryption automatically encrypts your data stored on Azure managed disks (OS and data disks) at rest by default when persisting it to the cloud.

Encryption at host ⓘ Encryption at host is not registered for the selected subscription. [Learn more about enabling this feature](#)

OS disk

OS disk size * ⓘ Default size (30 GiB)

OS disk type * ⓘ Standard HDD (locally-redundant storage)

The selected VM size supports premium disks. We recommend Premium SSD for high IOPS workloads. Virtual machines with Premium SSD disks qualify for the 99.9% connectivity SLA.

Delete with VM ⓘ

Key management ⓘ Platform-managed key

Review + create < Previous Next : Networking > Give feedback

7. Desactivamos el diagnóstico de inicio.

The screenshot shows the Microsoft Azure portal with the URL portal.azure.com/#create/Microsoft.VirtualMachine-ARM. The page is titled 'Create a virtual machine' and is on the 'Monitoring' tab. It includes sections for 'Alerts' (checkbox for 'Enable recommended alert rules') and 'Diagnostics' (radio buttons for 'Boot diagnostics' and 'Enable OS guest diagnostics'). At the bottom, there are 'Review + create' and 'Next : Advanced >' buttons.

8. Nos aseguramos de que todo se haya configurado correctamente y creamos la máquina virtual

The screenshot shows the Microsoft Azure portal with the URL portal.azure.com/#create/Microsoft.VirtualMachine-ARM. The page is titled 'Create a virtual machine' and shows a green header bar with a checkmark and the text 'Validation passed'. Below it, the 'Basics' section lists various configuration details:

Setting	Value
Subscription	Azure for Students
Resource group	(new) ESCOM
Virtual machine name	T4-2020630205
Region	East US
Availability options	No infrastructure redundancy required
Security type	Standard
Image	Ubuntu Server 18.04 LTS - Gen2
VM architecture	x64
Size	Standard B1s (1 vcpu, 1 GiB memory)
Authentication type	Password
Username	kubuntu
Public inbound ports	SSH
Azure Spot	No

At the bottom, there are 'Create' and 'Next >' buttons.

9. Esperamos a que nuestra máquina termine de crearse.

CreateVm-Canonical.UbuntuServer-18_04-Its-gen2-20230404233149 | Overview

Deployment is in progress

Deployment name: CreateVm-Canonical.UbuntuServer-18_04-Its-g... Start time: 4/4/2023, 11:39:06 PM
Subscription: Azure for Students Correlation ID: 182657e1-8213-4e4b-9c53-2ca8b7
Resource group: ESCOM

Deployment details

Resource	Type	Status	Operation details
No results.			

Give feedback [Tell us about your experience with deployment](#)

Microsoft Defender for Cloud
Secure your apps and infrastructure [Go to Microsoft Defender for Cloud >](#)

Free Microsoft tutorials
[Start learning today >](#)

Work with an expert
Azure experts are service provider partners who can help manage your assets on Azure and be your first line of support. [Find an Azure expert >](#)

10. Una vez creada, probamos la conexión.

CreateVm-Canonical.UbuntuServer-18_04-Its-gen2-20230404233149 | Overview

Your deployment is complete

Deployment name: CreateVm-Canonical.UbuntuServer-18_04-Its-g... Start time: 4/4/2023, 11:39:06 PM
Subscription: Azure for Students Correlation ID: 182657e1-8213-4e4b-9c53-2ca8b7
Resource group: ESCOM

Deployment details

Next steps

- Setup auto-shutdown Recommended
- Monitor VM health, performance and network dependencies Recommended
- Run a script inside the virtual machine Recommended

[Go to resource](#) [Create another VM](#)

Give feedback [Tell us about your experience with deployment](#)

Cost Management
Get notified to stay within your budget and prevent unexpected charges on your bill. [Set up cost alerts >](#)

Microsoft Defender for Cloud
Secure your apps and infrastructure [Go to Microsoft Defender for Cloud >](#)

Free Microsoft tutorials
[Start learning today >](#)

Work with an expert

```

kubuntu@T4-2020630205: ~ + 
* Support: https://ubuntu.com/advantage

System information as of Wed Apr 5 05:41:18 UTC 2023

System load: 0.24 Processes: 102
Usage of /: 4.5% of 28.89GB Users logged in: 0
Memory usage: 20% IP address for eth0: 10.0.0.4
Swap usage: 0%

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

kubuntu@T4-2020630205: $ 
kubuntu@T4-2020630205: $ 

```

11. Para realizar la implementación debemos crear una regla de puerto de entrada para abrir el puerto 50000.

Priority	Name	Port	Protocol
300	SSH	22	TCP
310	AllowAnyCustom5000...	50000	TCP
65000	AllowVnetInBound	Any	Any
65001	AllowAzureLoadBal...	Any	Any
65500	DenyAllInBound	Any	Any

Conversación con Chat GPT

YO:

Necesito realizar en java 3 programas implementando sockets seguros. Sin embargo ahora sólo requiero que escribas el servidor para posteriormente implementar los clientes. El primer programa es un servidor multi-thread al cuál se conectarán 2 tipos de cliente: Cliente PUT y Cliente GET. La petición GET seguida del nombre de un archivo (sin directorios). El servidor deberá leer el archivo del disco local, si el archivo se pudo leer el servidor deberá enviar al cliente OK, la longitud del archivo y el contenido del archivo, de otra manera deberá enviar al cliente ERROR.

La petición PUT seguida del nombre de un archivo (sin directorios), la longitud del archivo y el contenido del archivo. El servidor deberá escribir el archivo en el disco local, si el archivo se pudo escribir en el disco, el servidor enviará al cliente OK de otra manera enviará ERROR.

ChatGPT:

Claro, puedo ayudarte con eso. Te proporcionaré un ejemplo de servidor multi-thread que implementa sockets seguros en Java.

Antes de comenzar a escribir el código, asegúrate de tener un certificado SSL válido para el servidor.

Aquí está el código del servidor:

```
import java.io.*;  
import java.net.*;  
import javax.net.ssl.*;  
  
public class ServidorSSL {  
  
    public static void main(String[] args){  
        System.setProperty("javax.net.ssl.keyStore","C:/Users/kevin/keystore_servidor.jks");  
        System.setProperty("javax.net.ssl.keyStorePassword","1234567");  
    }  
}
```

```
SSLServerSocketFactory      socket_factory      =      (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();

ServerSocket socket_servidor;

try {
    socket_servidor = socket_factory.createServerSocket(50000);

    while (true) {
        Socket conexion = socket_servidor.accept();

        System.out.println("Cliente           conectado: " +

conexion.getInetAddress().getHostName());

        Thread cliente = new ClienteThread(conexion);

        cliente.start();
    }
} catch (IOException e) {
    e.printStackTrace();
}
}
```

```
class ClienteThread extends Thread {

    private Socket conexion;

    public ClienteThread(Socket conexion) {
        this.conexion = conexion;
    }
}
```

```
}
```

```
@Override
```

```
public void run() {  
    try {  
        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());  
        DataInputStream entrada = new DataInputStream(conexion.getInputStream());  
        String peticion = entrada.readUTF();  
        String[] partes_peticion = peticion.split(" ");  
        String tipo_peticion = partes_peticion[0];  
        String nombre_archivo = partes_peticion[1];  
        File archivo = new File(nombre_archivo);  
        if (tipo_peticion.equals("PUT")) {  
            long longitud_archivo = entrada.readLong();  
            byte[] buffer = new byte[1024];  
            int leidos = 0;  
            FileOutputStream archivo_salida = new FileOutputStream(archivo);  
            while (longitud_archivo > 0 && (leidos = entrada.read(buffer, 0, (int) Math.min(buffer.length, longitud_archivo))) != -1) {  
                archivo_salida.write(buffer, 0, leidos);  
                longitud_archivo -= leidos;  
            }  
            archivo_salida.close();  
            if (longitud_archivo == 0) {  
                salida.writeUTF("OK");  
            }  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

```
        System.out.println("Archivo recibido con éxito: " + archivo.getAbsolutePath());  
    } else {  
        salida.writeUTF("ERROR");  
        System.out.println("Error al recibir archivo: " + archivo.getAbsolutePath());  
    }  
}  
} else if (tipo_peticion.equals("GET")) {  
    if (archivo.exists() && archivo.isFile()) {  
        FileInputStream archivo_entrada = new FileInputStream(archivo);  
        byte[] buffer = new byte[1024];  
        int leidos = 0;  
        salida.writeUTF("OK");  
        salida.writeLong(archivo.length());  
        while ((leidos = archivo_entrada.read(buffer)) > 0) {  
            salida.write(buffer, 0, leidos);  
        }  
        archivo_entrada.close();  
        System.out.println("Archivo enviado con éxito: " + archivo.getAbsolutePath());  
    } else {  
        salida.writeUTF("ERROR");  
        System.out.println("Error al enviar archivo: " + archivo.getAbsolutePath());  
    }  
}  
conexion.close();  
}  
} catch (IOException e) {
```

```
        e.printStackTrace();  
    }  
}  
}
```

En este código, primero establecemos las propiedades del sistema necesarias para usar el keystore y la contraseña del keystore. Luego, creamos un objeto SSLSocketFactory y un objeto SSLSocket utilizando ese factory.

El servidor entra en un bucle infinito donde espera nuevas conexiones entrantes.

YO:

Escribe el Cliente PUT con las siguientes características: deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a enviar.

Entonces el cliente leerá el archivo del disco local, si puede leer el archivo se conectará al servidor y enviará una petición PUT, el nombre del archivo a enviar, la longitud del archivo y el contenido del archivo. Si el cliente no puede leer el archivo del disco local desplegará un mensaje de error indicando esta situación.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá desplegar un mensaje indicando que el archivo fue recibido por el servidor con éxito, de otra manera desplegará un mensaje de error indicando que el servidor no pudo escribir el archivo en el disco local.

ChatGPT:

A continuación te muestro el código para el Cliente PUT en Java utilizando sockets seguros:

```
import java.io.*;  
import java.net.*;  
import javax.net.ssl.*;
```

```
public class ClientePUTSSL {
```

```
public static void main (String[] args){

    String serverIP = args[0];
    int serverPort = Integer.parseInt(args[1]);
    String fileName = args[2];

    // Leemos el archivo del disco local
    File file = new File(fileName);
    byte[] fileContent = new byte[(int) file.length()];
    try {
        FileInputStream fis = new FileInputStream(file);
        fis.read(fileContent);
        fis.close();
    } catch (IOException e) {
        System.out.println("Error al leer el archivo del disco local");
        return;
    }

    // Creamos el socket seguro y nos conectamos al servidor
    System.setProperty("javax.net.ssl.trustStore","C:/Users/kevin/keystore_cliente.jks");
    System.setProperty("javax.net.ssl.trustStorePassword","1234567");
    SSLSocketFactory cliente = (SSLSocketFactory) SSLSocketFactory.getDefault();

    Socket conexion;
    try {
```

```
conexion = cliente.createSocket(serverIP, serverPort);

DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
DataInputStream entrada = new DataInputStream(conexion.getInputStream());

// Enviamos la petición PUT al servidor con el nombre del archivo, la longitud y el
contenido

salida.writeUTF("PUT " + file.getName() + " " + file.length());
salida.write(fileContent);

// Esperamos la respuesta del servidor

String response = entrada.readUTF();

// Mostramos el resultado al usuario

if (response.equals("OK")) {
    System.out.println("El archivo fue enviado al servidor con éxito");
} else {
    System.out.println("Error al enviar el archivo al servidor");
}

conexion.close();

} catch (IOException e) {
    System.out.println("Error al conectarse al servidor");
    e.printStackTrace();
}
}
```

}

Este código utiliza los argumentos de línea de comandos para obtener la dirección IP del servidor, el puerto y el nombre del archivo a enviar. Luego, lee el contenido del archivo del disco local y se conecta al servidor utilizando un socket seguro. Envía una petición PUT al servidor con el nombre del archivo, la longitud y el contenido, y espera la respuesta del servidor. Si la respuesta es "OK", muestra un mensaje indicando que el archivo fue enviado con éxito, de lo contrario muestra un mensaje de error.

Yo:

Cómo se debe ejecutar el cliente en línea de comandos?

ChatGPT:

Para ejecutar el cliente desde la línea de comandos en Java, debes abrir la terminal o el símbolo del sistema y navegar hasta la ubicación donde se encuentra el archivo compilado .class del programa. Luego, debes escribir el siguiente comando y presionar Enter:

```
java ClientePUT <ip_servidor> <puerto_servidor> <nombre_archivo>
```

Donde:

<ip_servidor> es la dirección IP del servidor.

<puerto_servidor> es el puerto en el que el servidor está escuchando las conexiones.

<nombre_archivo> es el nombre del archivo que deseas enviar al servidor.

Por ejemplo, si el archivo compilado del cliente se encuentra en la carpeta /home/usuario/sockets/ClientePUT.class, el servidor está en la dirección IP 192.168.1.100 y está escuchando en el puerto 50000, y deseas enviar el archivo datos.txt, debes ejecutar el siguiente comando:

```
java /home/usuario/sockets/ClientePUT 192.168.1.100 50000 datos.txt
```

Recuerda que debes tener el archivo datos.txt en la misma carpeta desde la que estás ejecutando el comando o especificar la ruta completa del archivo.

Yo:

Hay un problema y es que el servidor no escribe el archivo completamente y no regresa la respuesta al cliente.

ChatGPT:

El problema parece estar en la lectura y escritura del archivo en el servidor. En el método run() de la clase ClienteThread, cuando el cliente envía una petición PUT, el servidor lee la longitud del archivo y luego lee el archivo en bloques usando un buffer. Sin embargo, en el bucle de lectura, la longitud restante del archivo no se actualiza correctamente, lo que puede resultar en que el servidor no lea todo el archivo. Además, cuando el archivo se escribe en el disco, no se comprueba si la escritura fue exitosa o no antes de enviar una respuesta al cliente.

Para solucionar este problema, se pueden realizar los siguientes cambios en el método run():

@Override

```
public void run() {  
    try {  
        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());  
        DataInputStream entrada = new DataInputStream(conexion.getInputStream());  
        String peticion = entrada.readUTF();  
        String[] partes_peticion = peticion.split(" ");  
        String tipo_peticion = partes_peticion[0];  
        String nombre_archivo = partes_peticion[1];  
        File archivo = new File(nombre_archivo);  
        if (tipo_peticion.equals("PUT")) {  
            long longitud_archivo = Long.parseLong(partes_peticion[2]); // obtener la longitud  
            del archivo del mensaje  
            byte[] buffer = new byte[4096]; // Aumentar tamaño del buffer  
            int leidos = 0;  
            FileOutputStream archivo_salida = new FileOutputStream(archivo);  
            while (longitud_archivo > 0 && (leidos = entrada.read(buffer, 0, (int)  
Math.min(buffer.length, longitud_archivo))) != -1) {  
                archivo_salida.write(buffer, 0, leidos);  
            }  
            archivo_salida.close();  
        }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

```

        longitud_archivo -= leidos;

    }

    archivo_salida.close();

    if (longitud_archivo == 0 && archivo.length() ==
Long.parseLong(partes_peticion[2])) { // verificar si la escritura fue exitosa

        salida.writeUTF("OK");

        System.out.println("Archivo recibido con éxito: " + archivo.getAbsolutePath());

    } else {

        salida.writeUTF("ERROR");

        System.out.println("Error al recibir archivo: " + archivo.getAbsolutePath());

    }

} else if (tipo_peticion.equals("GET")) {

    if (archivo.exists() && archivo.isFile()) {

        FileInputStream archivo_entrada = new FileInputStream(archivo);

        byte[] buffer = new byte[1024];

        int leidos = 0;

        salida.writeUTF("OK");

        salida.writeLong(archivo.length());

        while ((leidos = archivo_entrada.read(buffer)) > 0) {

            salida.write(buffer, 0, leidos);

        }

        archivo_entrada.close();

        System.out.println("Archivo enviado con éxito: " + archivo.getAbsolutePath());

    } else {

        salida.writeUTF("ERROR");

    }

}

```

```
        System.out.println("Error al enviar archivo: " + archivo.getAbsolutePath());  
    }  
}  
conexion.close();  
} catch (IOException e) {  
    e.printStackTrace();  
}  
}
```

Con estos cambios, se debería poder recibir y enviar archivos correctamente en el servidor y recibir una respuesta en el cliente.

YO:

Ahora escribe el ClienteGET con las siguientes características en el lenguaje Java: deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a recibir.

Entonces el cliente se conectará al servidor y enviará la petición GET y el nombre del archivo.

El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá recibir la longitud del archivo y el contenido del archivo. Entonces el cliente deberá escribir el archivo en el disco local, si puede escribir el archivo deberá desplegar un mensaje indicando que el archivo se recibió con éxito, de otra manera deberá desplegar un mensaje de error.

ChatGPT:

Aquí tienes un ejemplo del ClienteGET en Java:

```
import java.io.*;  
  
import java.net.*;  
  
import javax.net.ssl.*;
```

```
public class ClienteGET {
```

```
public static void main(String[] args) {  
  
    String serverIP = args[0];  
    int serverPort = Integer.parseInt(args[1]);  
    String fileName = args[2];  
  
    // Creamos el socket seguro y nos conectamos al servidor  
    System.setProperty("javax.net.ssl.trustStore", "C:/Users/kevin/keystore_cliente.jks");  
    System.setProperty("javax.net.ssl.trustStorePassword", "1234567");  
    SSLSocketFactory cliente = (SSLSocketFactory) SSLSocketFactory.getDefault();  
    Socket conexion;  
    try {  
        conexion = cliente.createSocket(serverIP, serverPort);  
        DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());  
        DataInputStream entrada = new DataInputStream(conexion.getInputStream());  
        System.out.println("Conexión realizada con el servidor. Solicitando archivo...");  
  
        // Enviamos la petición GET al servidor con el nombre del archivo  
        salida.writeUTF("GET " + fileName);  
  
        // Esperamos la respuesta del servidor  
        String response = entrada.readUTF();  
        if (response.equals("ERROR")) {  
    }  
}
```

```
        System.out.println("Error al solicitar el archivo al servidor");
        conexion.close();
        return;
    }

    // Recibimos el archivo del servidor y lo guardamos en disco local
    FileOutputStream fos = new FileOutputStream(new File(fileName));
    byte[] buffer = new byte[4096];
    int bytesRead;
    while ((bytesRead = entrada.read(buffer)) != -1) {
        fos.write(buffer, 0, bytesRead);
    }
    fos.close();

    System.out.println("El archivo " + fileName + " fue descargado del servidor con éxito");

    conexion.close();
} catch (IOException e) {
    System.out.println("Error al conectarse al servidor");
    e.printStackTrace();
}
}
```

En este código, la solicitud GET se envía al servidor con el nombre del archivo que se desea descargar. Luego, el cliente espera la respuesta del servidor. Si el servidor devuelve un mensaje de "ERROR", significa que no se pudo encontrar el archivo y se cierra la conexión.

Si el archivo existe en el servidor, el cliente recibe los datos del archivo en bloques y los escribe en un archivo local. En este ejemplo, los datos se escriben en un archivo con el mismo nombre que se especificó en la solicitud GET, pero se puede modificar el código para guardar el archivo con un nombre diferente.

Finalmente, el cliente cierra la conexión con el servidor después de descargar el archivo o si se produce un error en la conexión.

YO:

¿Si quisiera ejecutar el servidor en una máquina virtual y los clientes de manera local, debo crear otro certificado en la máquina virtual?

ChatGPT:

Si quieres ejecutar el servidor en una máquina virtual y los clientes de manera local, necesitarás crear otro certificado en la máquina virtual.

La razón de esto es que cada máquina tiene su propia identidad y, por lo tanto, debe tener su propio certificado SSL. Si utilizas el mismo certificado en la máquina virtual y en los clientes locales, los clientes no podrán establecer una conexión segura con el servidor debido a la discrepancia en las identidades.

Por lo tanto, para solucionar este problema, deberás crear otro certificado en la máquina virtual y configurar el servidor para utilizar ese certificado en lugar del certificado utilizado en la máquina local. También deberás actualizar los clientes para confiar en el nuevo certificado de la máquina virtual.

YO:

¿Como actualizo los clientes para confiar en el nuevo certificado de la máquina virtual?

ChatGPT:

Para actualizar los clientes y hacer que confíen en el nuevo certificado de la máquina virtual, debes seguir los siguientes pasos:

Genera un nuevo certificado para el servidor en la máquina virtual utilizando una autoridad de certificación confiable.

Exporta el certificado del servidor desde la máquina virtual y guárdalo en un archivo. Asegúrate de que el archivo contenga únicamente el certificado y no la clave privada.

En cada uno de los clientes, agrega el nuevo certificado del servidor al truststore del cliente.

Si el truststore del cliente ya tiene un certificado para el servidor, revoca ese certificado.

Reinicia los clientes.

Para agregar el nuevo certificado del servidor al truststore del cliente, sigue estos pasos:

Copia el archivo con el certificado del servidor en la máquina del cliente.

Abre una terminal y ejecuta el siguiente comando:

```
keytool -import -alias <alias> -file <cert-file> -keystore <truststore-file>
```

Donde:

<alias> es un nombre descriptivo para el certificado.

<cert-file> es el archivo que contiene el certificado del servidor.

<truststore-file> es el archivo del truststore del cliente.

Se te solicitará la contraseña del truststore. Ingresa la contraseña correcta.

Si se te pide confirmación para confiar en el certificado, ingresa "yes" o "si".

Verifica que el certificado se haya agregado correctamente al truststore del cliente ejecutando el siguiente comando:

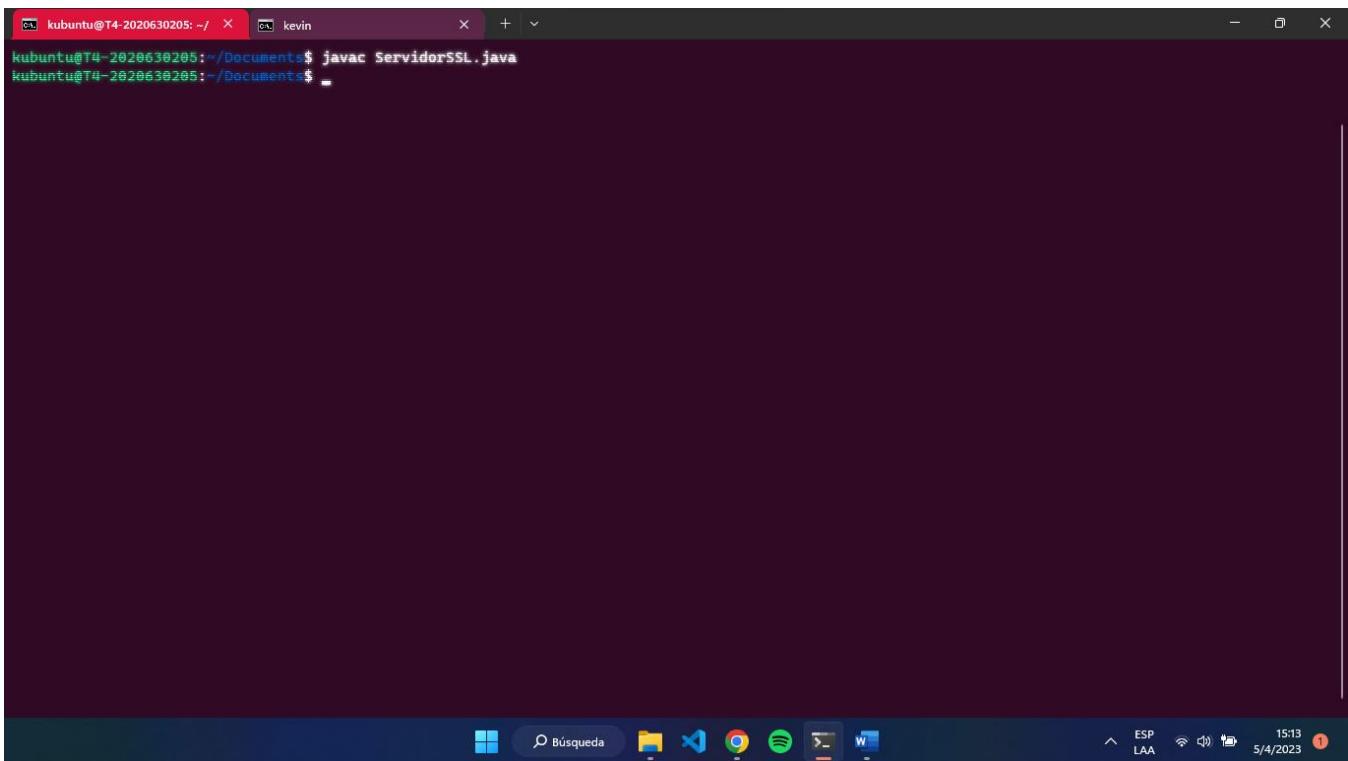
```
keytool -list -v -keystore <truststore-file>
```

Reemplaza <truststore-file> con el archivo de tu truststore.

Reinicia los clientes para que carguen el nuevo truststore con el certificado actualizado.

Implementación

1. Enviamos el servidor a la máquina virtual de Ubuntu y creamos el certificado que posteriormente enviamos a la computadora local, también creamos el keystore del servidor. Posteriormente compilamos el ServidorSSL.java

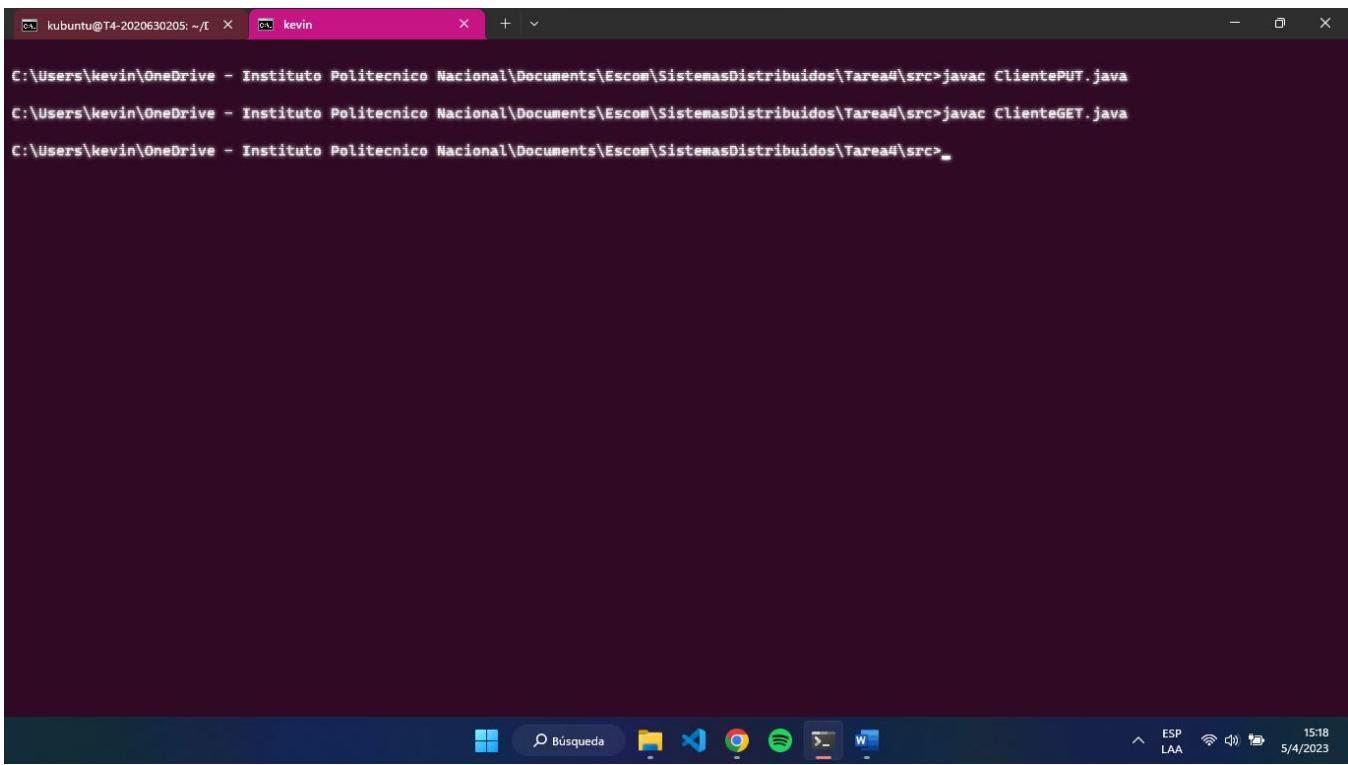


A screenshot of a Linux desktop environment, likely Kubuntu, showing a terminal window. The terminal window has a dark background and contains the following text:

```
kubuntu@T4-2020630205: ~/kevin$ javac ServidorSSL.java
kubuntu@T4-2020630205: ~/kevin$
```

The desktop interface includes a dock at the bottom with icons for various applications like a search bar, file explorer, and code editor. The system tray shows network status, battery level (ESP LAA), and the date and time (15:18, 5/4/2023).

2. Una vez que tenemos de manera local el certificado del servidor, creamos con él la keystore del cliente y compilamos ambos: ClientePUT.java y ClienteGET.java



A screenshot of a Windows desktop environment showing a terminal window. The terminal window has a dark background and contains the following text:

```
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>javac ClientePUT.java
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>javac ClienteGET.java
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>_
```

The desktop interface includes a taskbar at the bottom with icons for a search bar, file explorer, and code editor. The system tray shows network status, battery level (ESP LAA), and the date and time (15:18, 5/4/2023).

3. Ejecutamos el ServidorSSL para que pueda recibir las peticiones.

```
kubuntu@T4-2020630205:~/kevin$ javac ServidorSSL.java
kubuntu@T4-2020630205:~/kevin$ java ServidorSSL
```

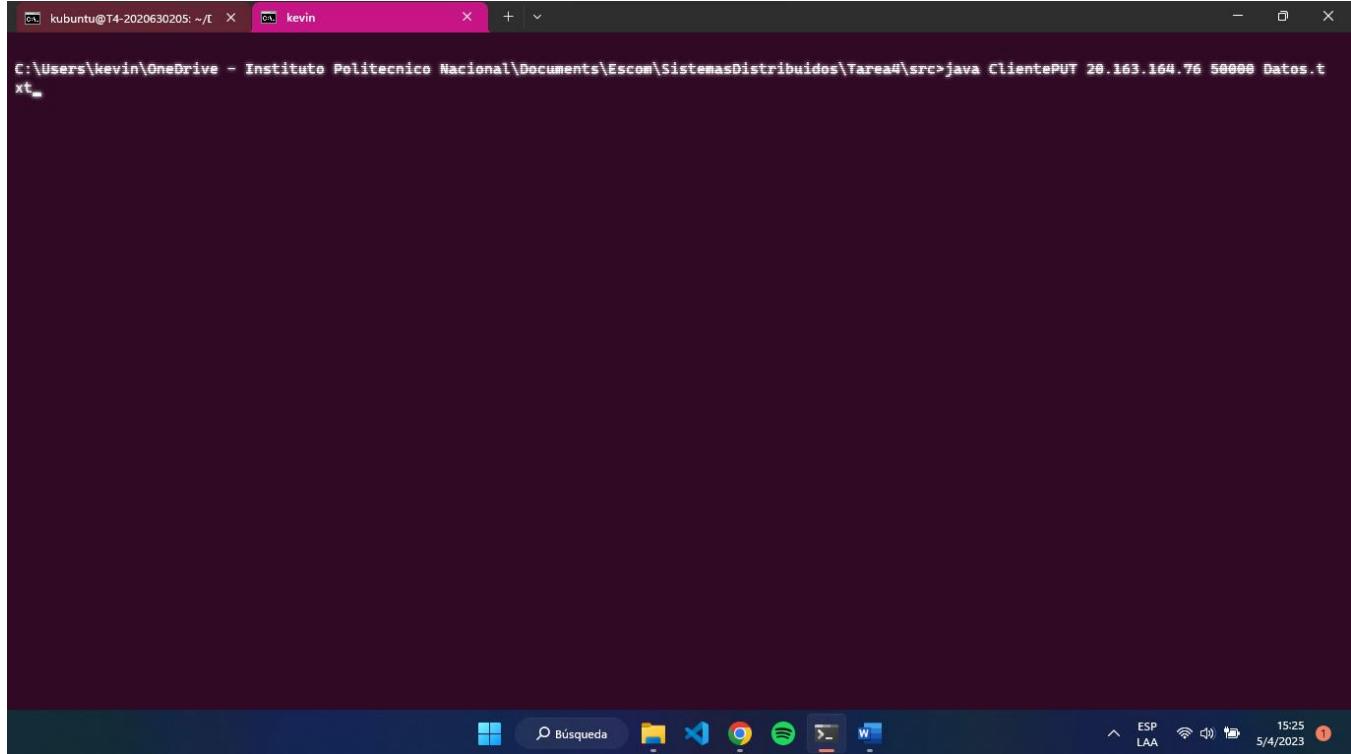
The screenshot shows a terminal window on a Kubuntu desktop. The terminal has two tabs: one for the user 'kevin' and another for the command line. The command 'javac ServidorSSL.java' is run in the terminal, followed by 'java ServidorSSL'. The desktop interface includes a taskbar with icons for search, file explorer, and other applications like Spotify and Microsoft Word.

4. Creamos un archivo de manera local para probar la funcionalidad de la petición PUT, lo llamaremos Datos.txt

```
GNU nano 5.6.1          Datos.txt
```

The screenshot shows a terminal window running the 'nano' text editor. The title bar indicates 'GNU nano 5.6.1' and the file name 'Datos.txt'. The editor displays the text: 'Este es un archivo de texto enviado desde el cliente'. The bottom of the screen shows the nano command-line interface with various keyboard shortcuts.

5. Ya que las propiedades de SSL se encuentran en el código, sólo debemos pasar como parámetro la IP de la máquina virtual de Ubuntu, el puerto por el que nos vamos a conectar y el archivo a enviar o recibir según sea el caso.



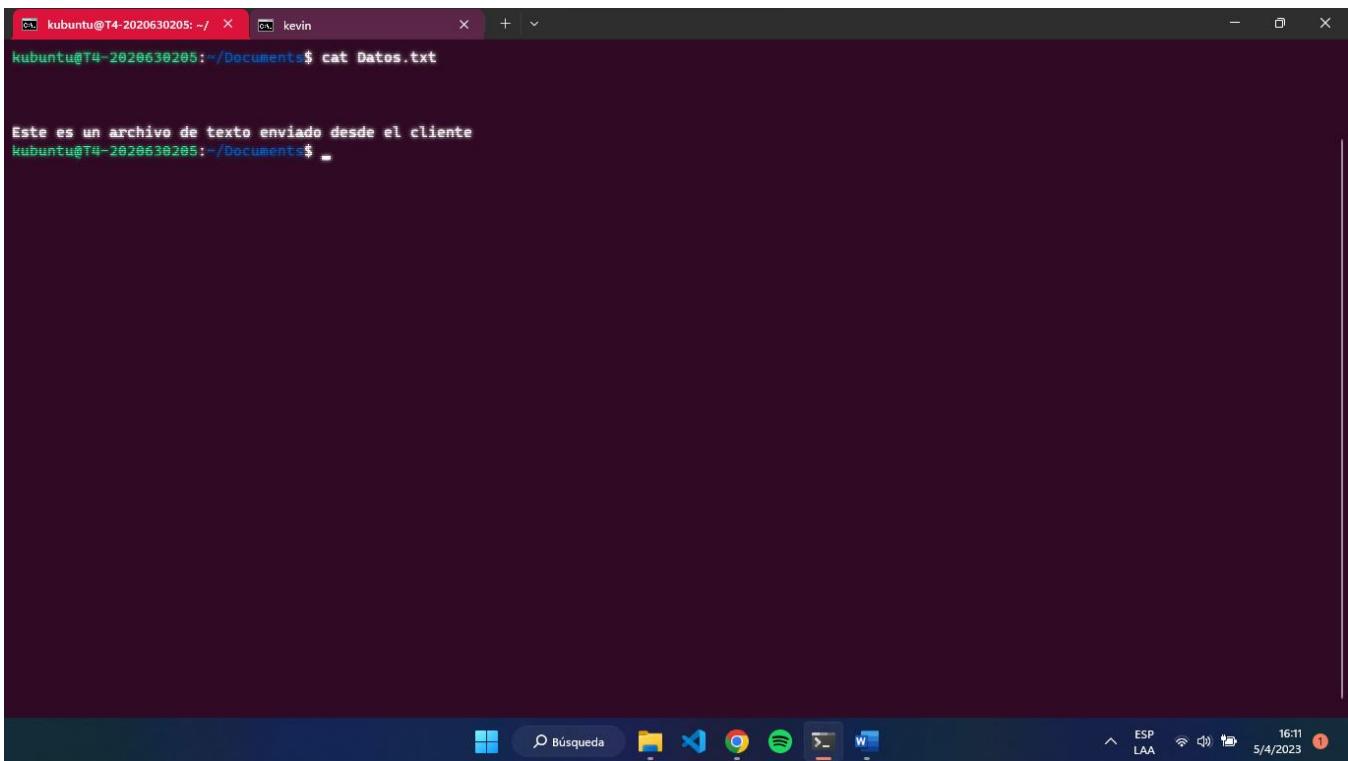
A screenshot of a terminal window titled "kevin". The command entered is "java ClientePUT 20.163.164.76 50000 Datos.txt". The terminal is running on a Windows operating system, as evidenced by the taskbar icons at the bottom.

```
kubuntu@T4-2020630205: ~/T > java ClientePUT 20.163.164.76 50000 Datos.txt
```

6. Ejecutamos y podemos ver tanto en la máquina local como en la virtual que el archivo ha sido recibido y guardado en la máquina virtual.

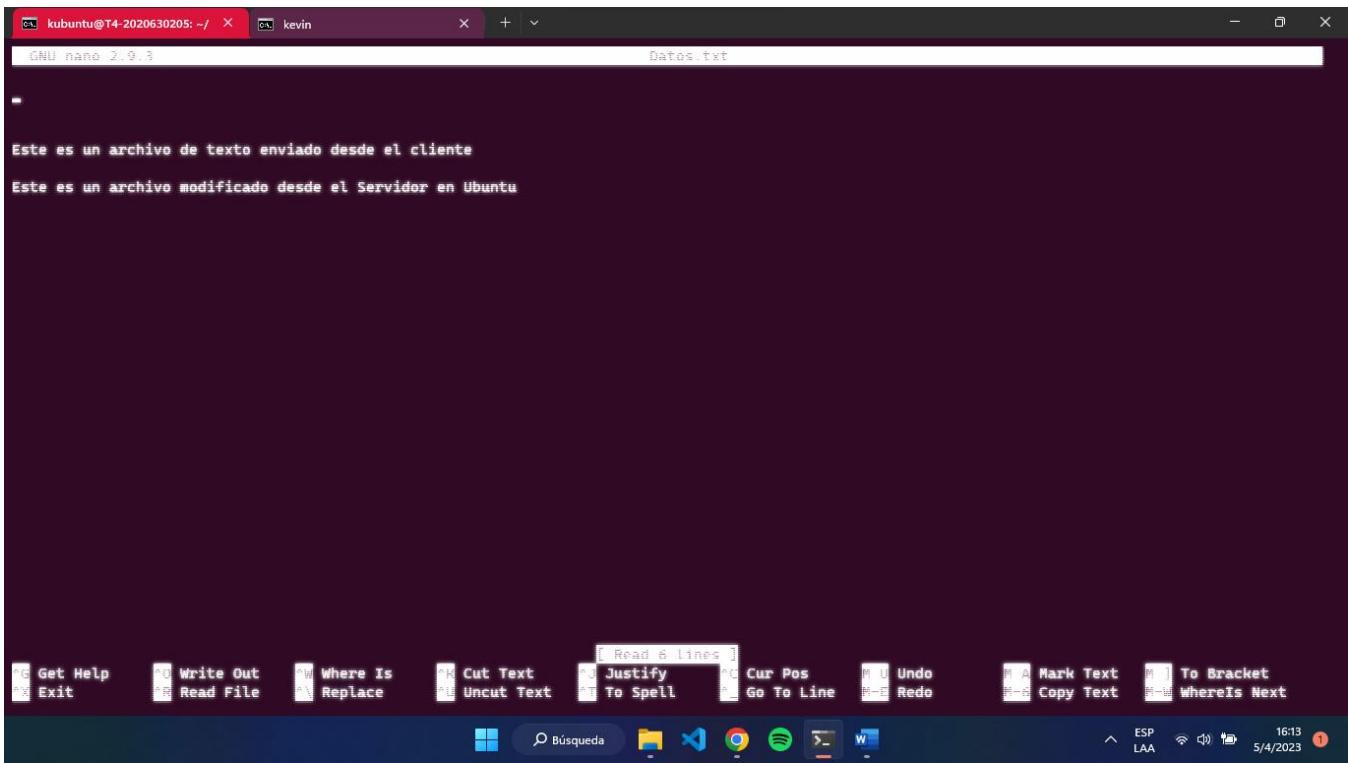
```
kubuntu@T4-2020630205: ~/kevin ~ kevin + - x
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>java ClientePUT 26.163.164.76 50000 Datos.txt
Conexion realizada con el servidor. Enviando archivo...
El archivo fue enviado al servidor con exito
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>
```

```
kubuntu@T4-2020630205: ~/kevin ~ kevin + - x
kubuntu@T4-2020630205:~/Documentos$ java ServidorSSL
Cliente conectado: fixed-187-191-39-8.totalplay.net
Archivo recibido con éxito: /home/kubuntu/Documentos/Datos.txt
=
```



```
kubuntu@T4-2020630205: ~/kevin$ cat Datos.txt
Este es un archivo de texto enviado desde el cliente
kubuntu@T4-2020630205:~/Documents$
```

7. Ahora para probar la petición GET, modificaremos el archivo desde la máquina virtual de Ubuntu.



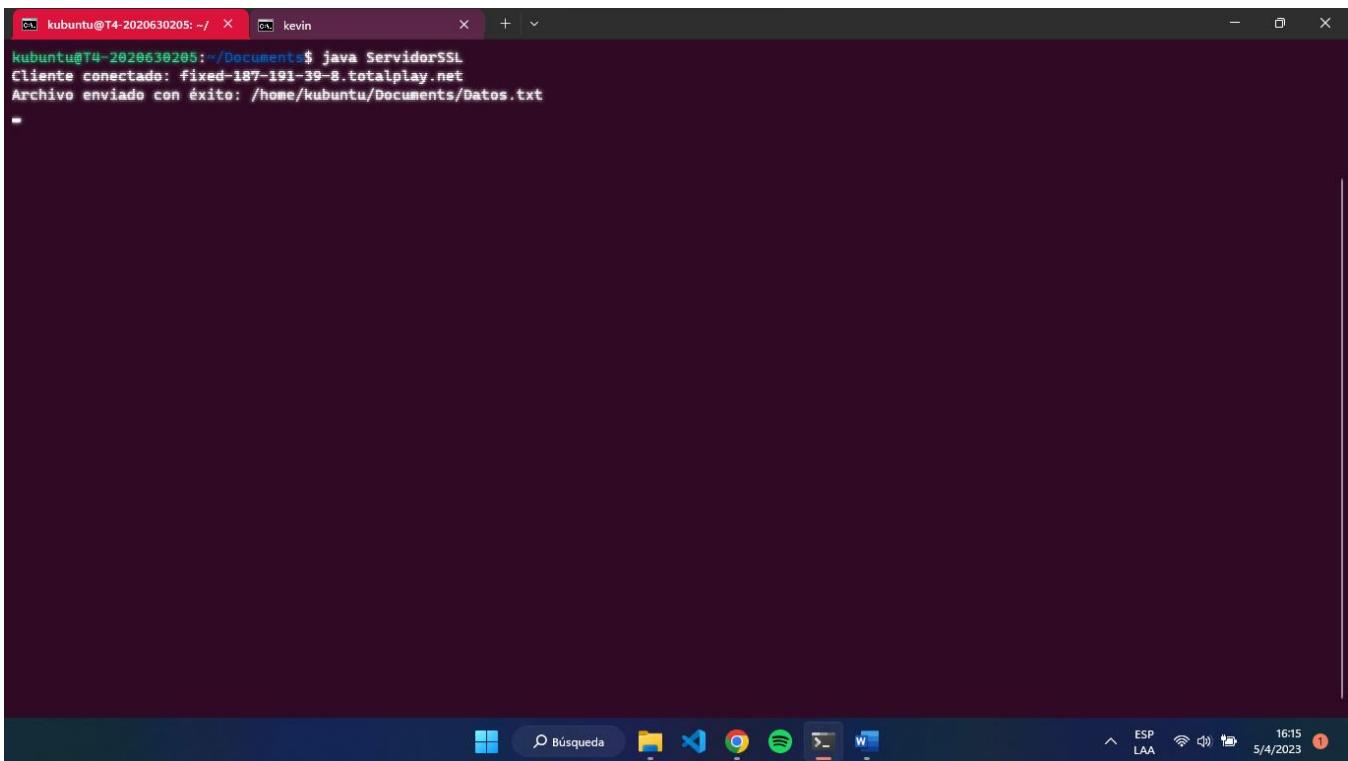
```
GNU nano 2.0.3          Datos.txt
-
Este es un archivo de texto enviado desde el cliente
Este es un archivo modificado desde el Servidor en Ubuntu
```

8. Y ejecutamos el ClienteGET con los mismos parámetros.

```
kubuntu@T4-2020630205: ~/T ~ kevin + - x
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>java ClienteGET 20.163.164.76 50000 Datos.txt
```

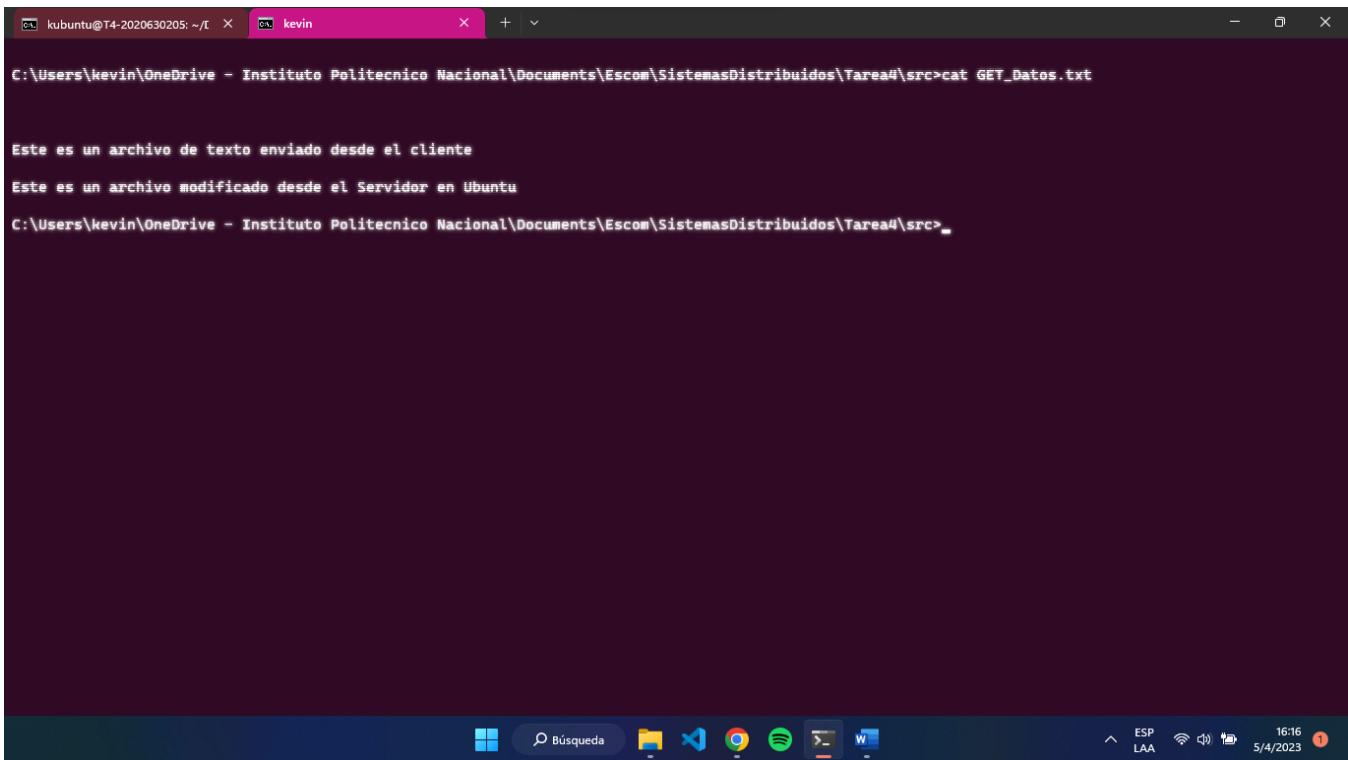
9. Tanto en el cliente como en el servidor podemos ver que la acción se realizó correctamente

```
kubuntu@T4-2020630205: ~/T ~ kevin + - x
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>java ClienteGET 20.163.164.76 50000 Datos.txt
Conexion realizada con el servidor. Solicitando archivo...
El archivo Datos.txt fue descargado del servidor con exito
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>
```



```
kubuntu@T4-2020630205: ~/kevin$ java ServidorSSL
Cliente conectado: fixed-187-191-39-8.totalplay.net
Archivo enviado con éxito: /home/kubuntu/Documents/Datos.txt
```

10. Si vemos el contenido del archivo, ahora el archivo descargado se llama GET_Datos.txt, vemos los cambios hechos desde la máquina virtual de Ubuntu.



```
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>cat GET_Datos.txt

Este es un archivo de texto enviado desde el cliente
Este es un archivo modificado desde el Servidor en Ubuntu
C:\Users\kevin\OneDrive - Instituto Politecnico Nacional\Documents\Escritorio\SistemasDistribuidos\Tarea4\src>
```

Conclusiones

La tarea de desarrollar los programas ServidorSSL, ClientePUT y ClienteGET me ha permitido profundizar en el uso de sockets seguros y certificados digitales para garantizar la confidencialidad, integridad y autenticidad de la comunicación entre aplicaciones.

Para lograr esto, fue necesario crear un certificado digital para el servidor y agregarlo a un keystore, y luego utilizarlo en la configuración del ServidorSSL para permitir conexiones seguras con los clientes. Asimismo, se creó localmente un keystore para el cliente y se agregó el certificado del servidor remoto a la lista de certificados de confianza para permitir la conexión segura con el servidor en la máquina virtual de Ubuntu.

Además, se implementaron los programas ClientePUT y ClienteGET para realizar solicitudes seguras al servidor y enviar o recibir archivos de manera confiable asimilando al protocolo SFTP.

En resumen, la tarea de desarrollo de los programas ServidorSSL, ClientePUT y ClienteGET fue una experiencia valiosa para comprender el uso de sockets seguros y certificados digitales en la creación de aplicaciones que requieren una comunicación segura y confiable.