Instituto Politécnico Nacional
Escuela Superior de Cómputo

Tarea extra: Transferencia de archivos utilizando sockets seguros

Olay Silis Jose Eduardo
4CV12
Prof. Carlos Pineda Guerrero

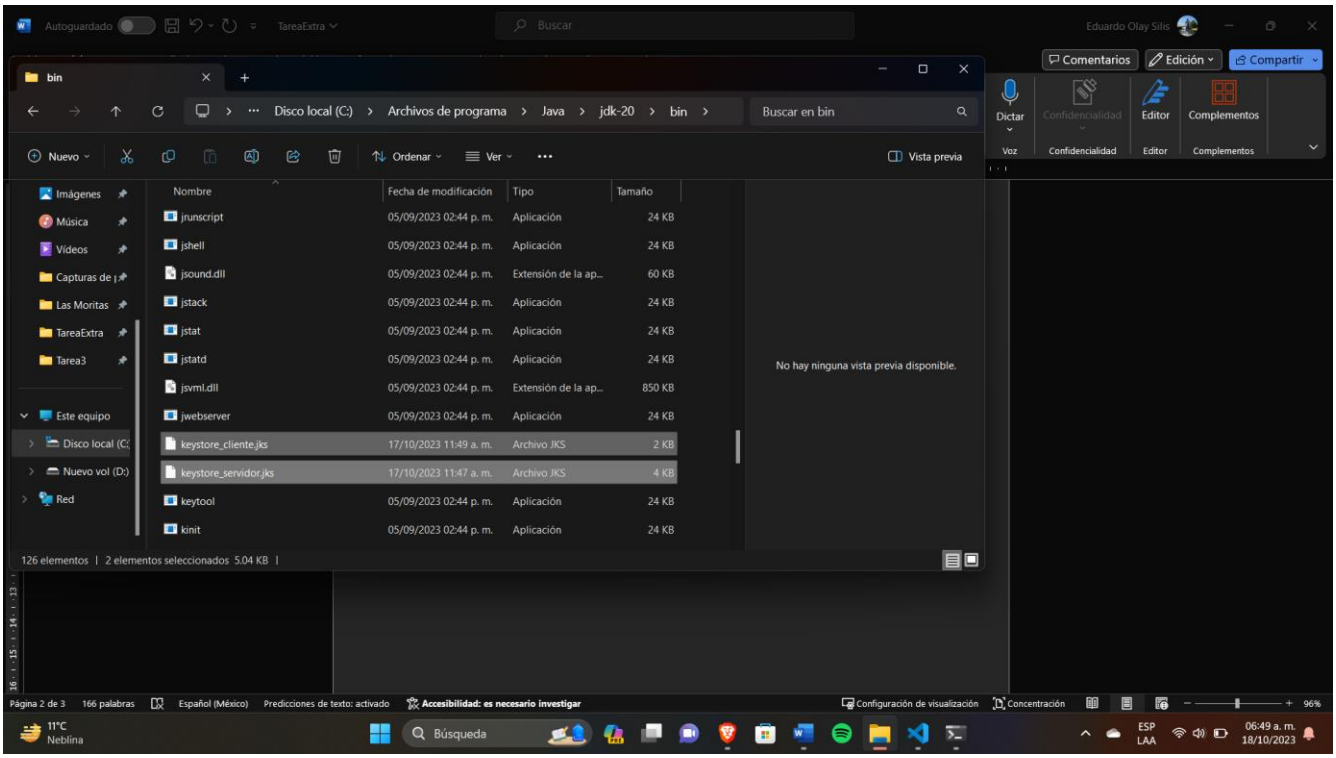
Introducción

Esta práctica se centra en la implementación de un sistema cliente-servidor seguro para la transferencia de archivos, donde se desarrolló un servidor multithread capaz de gestionar solicitudes GET y PUT, junto con clientes que utilizan conexiones SSL para asegurar la comunicación. Los clientes pueden recibir archivos del servidor mediante solicitudes GET y enviar archivos al servidor mediante solicitudes PUT. La práctica abarca aspectos de seguridad, transferencia de archivos y manejo de errores. Esta experiencia ofrece una comprensión completa de cómo funcionan los sistemas de transferencia de archivos seguros y cómo se aplican en situaciones del mundo real.

Desarrollo

Fase 1

En primera instancia comencé a realizar los certificados autofirmados para el cliente y el servidor mediante el programa keytool el cual fui siguiendo de acuerdo con la explicación que viene en la página de la materia quedando de la siguiente manera:



Una vez realizado los certificados comencé a realizar el código del servidor, para lo cual tome el ejemplo de servidorSSL de la plataforma y le dije al ChatGPT que me lo adaptara de acuerdo a lo que necesitaba que realizara el servidor, generando el código de la siguiente forma:

Tengo el siguiente código de un servidor con sockets seguros el cual recibe como parámetro un double:

```
System.setProperty("javax.net.ssl.trustStore","keystore_cliente.jks");

System.setProperty("javax.net.ssl.trustStorePassword","123456");

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

ServerSocket socket_servidor = socket_factory.createServerSocket(50000);

Socket conexion = socket_servidor.accept();

DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());

DataInputStream entrada = new DataInputStream(conexion.getInputStream());

double x = entrada.readDouble();

System.out.println(x);

conexion.close();
```

Volver a generar

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

```
conexion.close();
```

Necesito que basándote en el programa lo modifiques de manera que crees un servidor multithread el cual recibirá del cliente 2 tipos de peticiones:

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.

Para crear un servidor multithread que maneje las peticiones GET y PUT, puedes usar un enfoque como el siguiente. Asegúrate de tener en cuenta las siguientes consideraciones antes de implementar este código:

1. Asegúrate de que tu servidor tenga acceso de lectura/escritura a la ubicación en el disco donde se guardarán los archivos y que tengas un directorio apropiado para almacenar los archivos.
2. Asegúrate de que el cliente envíe las peticiones GET y PUT según el formato que esperas.

Volver a generar

Envía un mensaje

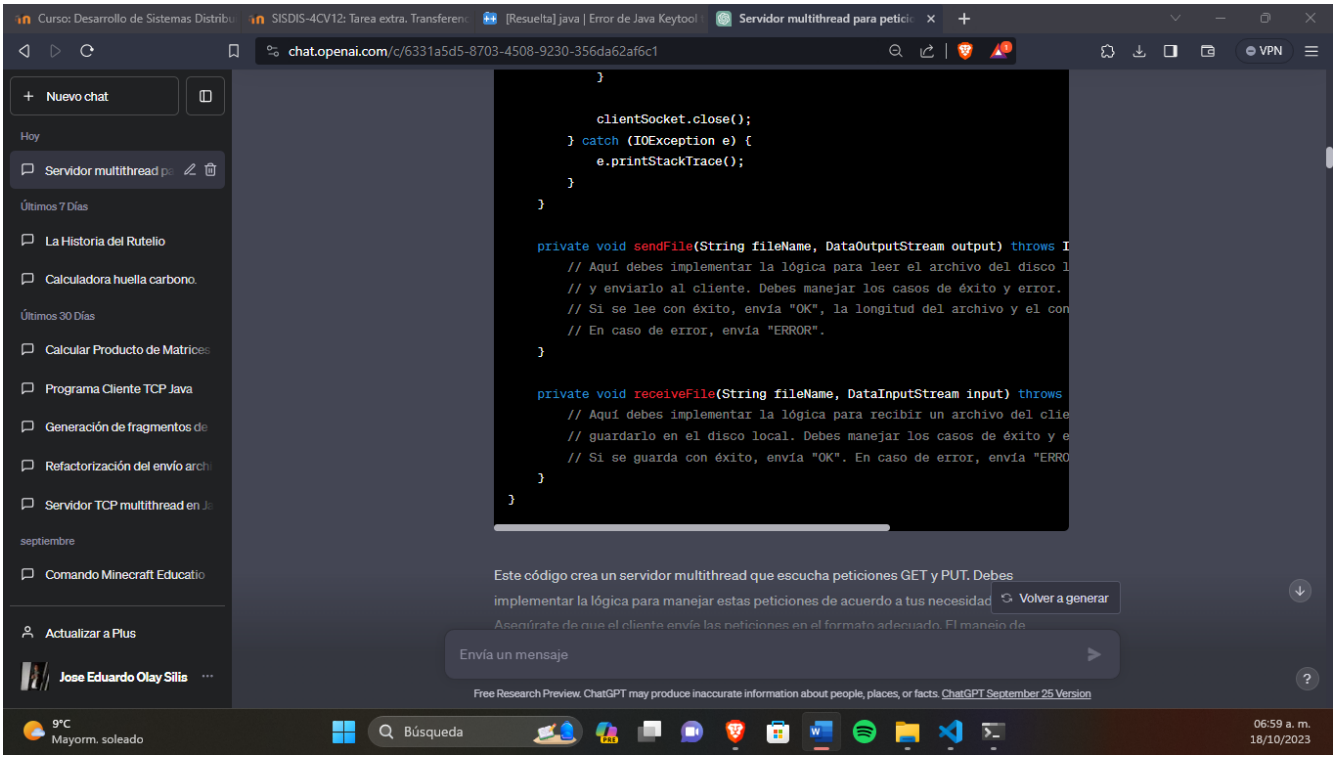
Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

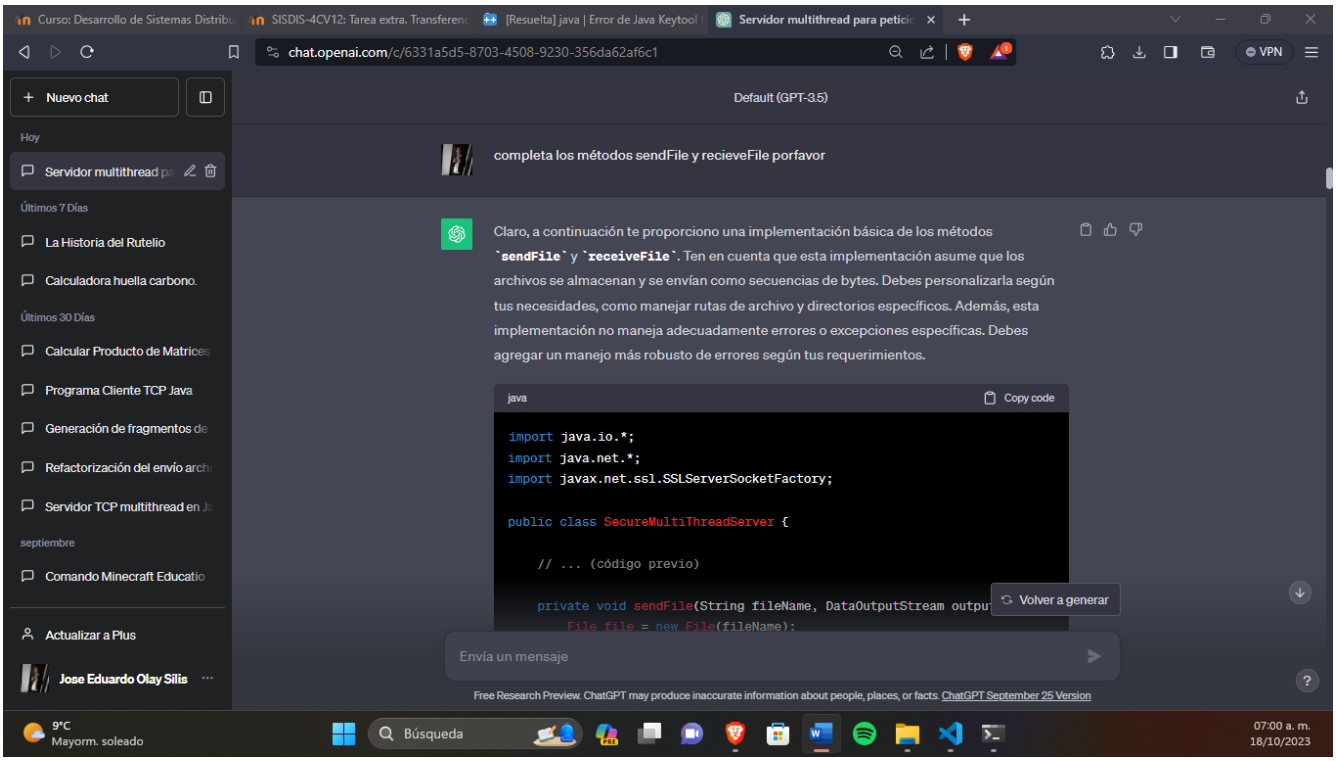
Aquí tienes un ejemplo de cómo modificar tu código para lograr esto:

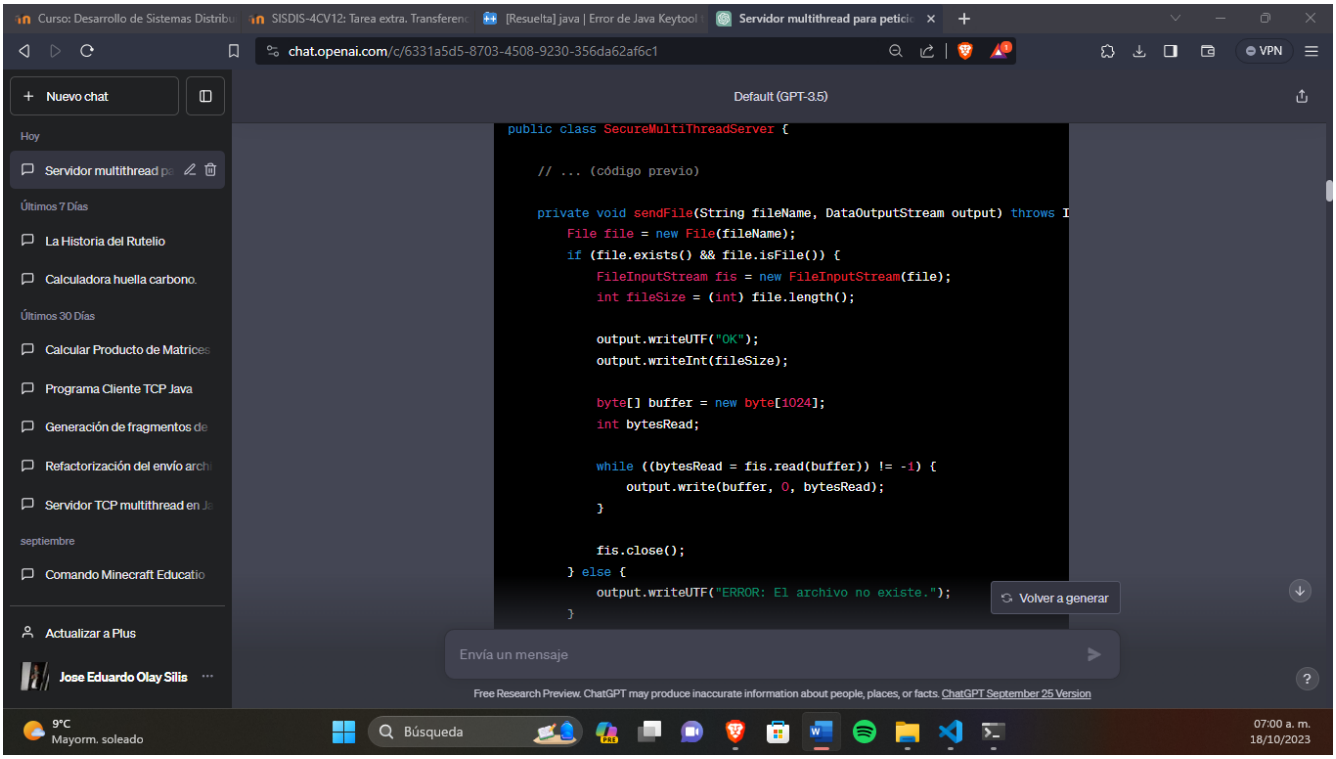
java

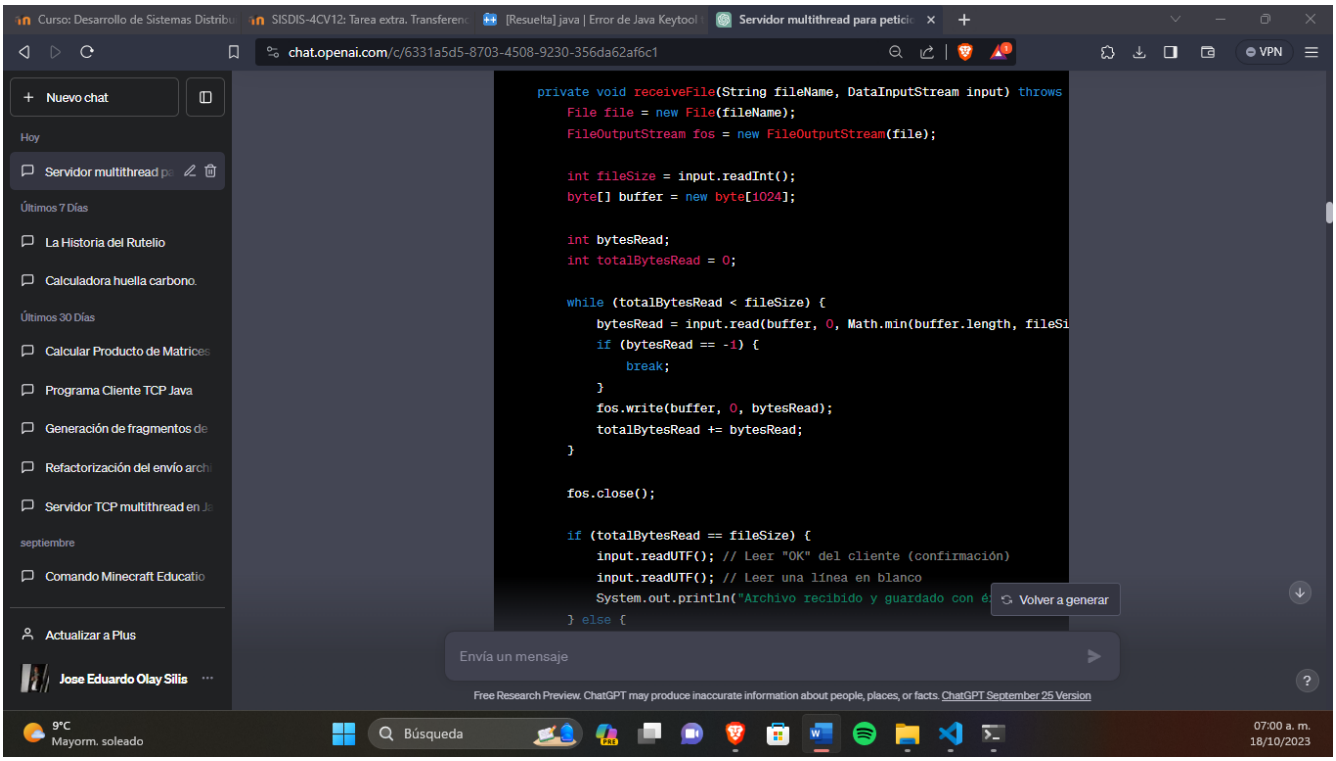
```java
import java.io.*;
import java.net.*;
import javax.net.ssl.SSLServerSocketFactory;

public class SecureMultiThreadServer {

    public static void main(String[] args) throws Exception {
        System.setProperty("javax.net.ssl.trustStore", "keystore_cliente.jks
        System.setProperty("javax.net.ssl.trustStorePassword", "123456");
        SSLServerSocketFactory socket_factory = (SSLServerSocketFactory) SSL
        ServerSocket socket_servidor = socket_factory.createServerSocket(500
        System.out.println("Servidor listo para recibir peticiones...");

        while (true) {
            Socket conexion = socket_servidor.accept();
            Thread clientThread = new ClientHandler(conexion);
            clientThread.start();
        }
    }
}

class ClientHandler extends Thread {
    private Socket clientSocket;
```

```java
    public ClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            DataOutputStream salida = new DataOutputStream(clientSocket.getO
            DataInputStream entrada = new DataInputStream(clientSocket.getIn

            String request = entrada.readUTF();
            String[] requestParts = request.split(" ");

            if (requestParts.length >= 2) {
                String action = requestParts[0];
                String fileName = requestParts[1];

                if ("GET".equals(action)) {
                    // Manejar la petición GET
                    sendFile(fileName, salida);
                } else if ("PUT".equals(action)) {
                    // Manejar la petición PUT
                    receiveFile(fileName, entrada);
                } else {
                    salida.writeUTF("ERROR: Acción no válida.");
                }
            } else {
```


Servidor multithread para peticiones
chat.openai.com/c/6331a5d5-8703-4508-9230-356da62af6c1
Nuevo chat
Hoy
Servidor multithread p
Últimos 7 Días
La Historia del Rutelio
Calculadora huella carbono.
Últimos 30 Días
Calcular Producto de Matrices
Programa Cliente TCP Java
Generación de fragmentos de
Refactorización del envío arch
Servidor TCP multithread en Ja
septiembre
Comando Minecraft Educatio
Actualizar a Plus
Jose Eduardo Olay Silis
clientSocket.close();
} catch (IOException e) {
e.printStackTrace();
}
}
private void sendFile(String fileName, DataOutputStream output) throws I
// Aquí debes implementar la lógica para leer el archivo del disco l
// y enviarlo al cliente. Debes manejar los casos de éxito y error.
// Si se lee con éxito, envía "OK", la longitud del archivo y el con
// En caso de error, envía "ERROR".
}
private void receiveFile(String fileName, DataInputStream input) throws
// Aquí debes implementar la lógica para recibir un archivo del clie
// guardarlo en el disco local. Debes manejar los casos de éxito y e
// Si se guarda con éxito, envía "OK". En caso de error, envía "ERRO
}
}
Este código crea un servidor multithread que escucha peticiones GET y PUT. Debes
implementar la lógica para manejar estas peticiones de acuerdo a tus necesidad
Volver a generar
Envía un mensaje
Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version
9°C
Mayorm. soleado
Búsqueda
06:59 a. m.
18/10/2023


Servidor multithread para peticiones
chat.openai.com/c/6331a5d5-8703-4508-9230-356da62af6c1
Nuevo chat
Default (GPT-3.5)
completa los métodos sendFile y recieveFile porfavor
Claro, a continuación te proporciono una implementación básica de los métodos `sendFile` y `receiveFile`. Ten en cuenta que esta implementación asume que los archivos se almacenan y se envían como secuencias de bytes. Debes personalizarla según tus necesidades, como manejar rutas de archivo y directorios específicos. Además, esta implementación no maneja adecuadamente errores o excepciones específicas. Debes agregar un manejo más robusto de errores según tus requerimientos.
java
Copy code
import java.io.*;
import java.net.*;
import javax.net.ssl.SSLServerSocketFactory;
public class SecureMultiThreadServer {
// ... (código previo)
private void sendFile(String fileName, DataOutputStream outpu
File file = new File(fileName);
Volver a generar
Envía un mensaje
Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version
9°C
Mayorm. soleado
Búsqueda
07:00 a. m.
18/10/2023





Una vez obtenido el ServidorSSL de manera similar ocupando el ClienteSSL de la plataforma le pedí al chat que lo adaptara para que implementara el cliente PUT generando el código de la siguiente manera:

Curso: Desarrollo de Sistemas Distribu | SISDIS-4CV12: Tarea extra. Transferen | [Resuelta] java | Error de Java Keytool | Servidor multithread para petici

chat.openai.com/c/6331a5d5-8703-4508-9230-356da62af6c1

Nuevo chat

Hoy

Servidor multithread p

Últimos 7 Días

La Historia del Rutelio

Calculadora huella carbono.

Últimos 30 Días

Calcular Producto de Matrices

Programa Cliente TCP Java

Generación de fragmentos de

Refactorización del envío arch

Servidor TCP multithread en J

septiembre

Comando Minecraft Educatio

Actualizar a Plus

Jose Eduardo Olay Silis

Default (GPT-3.5)

Ahora tengo este código del cliente ssl el cual forma parte en conjunto del servidor ssl que te proporcione en un principio (este cliente le envía un double al servidor):

```
SSLSocketFactory cliente = (SSLSocketFactory) SSLSocketFactory.getDefault();

Socket conexion = cliente.createSocket("localhost",50000);

DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());

DataInputStream entrada = new DataInputStream(conexion.getInputStream());

salida.writeDouble(123456789.123456789);

Thread.sleep(1000);

conexion.close();
```

Necesito que basandote en este programa lo modifiques para crear un cliente PUT para el ServidorMultithread que create, el cual deberá recibir como parámetros en la línea de comandos la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a enviar.

Volver a generar

Entonces el cliente leerá el archivo del disco local, si puede leer el archivo se conectará al

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

9°C Mayorm. soleado | Búsqueda | 07:02 a. m. 18/10/2023

Curso: Desarrollo de Sistemas Distribu | SISDIS-4CV12: Tarea extra. Transferen | [Resuelta] java | Error de Java Keytool | Servidor multithread para petici

chat.openai.com/c/6331a5d5-8703-4508-9230-356da62af6c1

Nuevo chat

Hoy

Servidor multithread p

Últimos 7 Días

La Historia del Rutelio

Calculadora huella carbono.

Últimos 30 Días

Calcular Producto de Matrices

Programa Cliente TCP Java

Generación de fragmentos de

Refactorización del envío arch

Servidor TCP multithread en J

septiembre

Comando Minecraft Educatio

Actualizar a Plus

Jose Eduardo Olay Silis

Default (GPT-3.5)

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.

Aquí tienes un código de cliente que puedes utilizar para enviar una solicitud PUT al servidor SSL Multithread que mencionaste. Asegúrate de pasar la dirección IP del servidor, el puerto y el nombre del archivo como argumentos de línea de comandos.

java | Copy code

```java
import java.io.*;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import java.util.Scanner;

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

Volver a generar

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

9°C Mayorm. soleado | Búsqueda | 07:02 a. m. 18/10/2023

```
public class SecureClientPUT {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Uso: java SecureClientPUT <IP del servidor>
            return;
        }

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

        try {
            SSLSocketFactory clientFactory = (SSLSocketFactory) SSLSocketFac
            SSLSocket connection = (SSLSocket) clientFactory.createSocket(se

            DataOutputStream output = new DataOutputStream(connection.getOut
            DataInputStream input = new DataInputStream(connection.getInputS

            File file = new File(fileName);

            if (file.exists() && file.isFile()) {
                FileInputStream fis = new FileInputStream(file);
                int fileSize = (int) file.length();

                output.writeUTF("PUT " + file.getName());
                output.writeInt(fileSize);
```

```
                int bytesRead;

                while ((bytesRead = fis.read(buffer)) != -1) {
                    output.write(buffer, 0, bytesRead);
                }

                fis.close();

                String response = input.readUTF();
                if ("OK".equals(response)) {
                    System.out.println("El archivo fue enviado al servidor c
                } else {
                    System.out.println("Error: El servidor no pudo escribir
                }
            } else {
                System.out.println("Error: No se puede leer el archivo del d
            }

            connection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

El código anteriormente proporcionado por el Chat funcionaba de manera correcta, el único detalle es que una vez que el servidor recibía el archivo y lo escribía en el disco debía mandar un ok al cliente y una vez recibido ese ok por parte del cliente ya mostraba un mensaje que el archivo fue recibido de manera correcta; dicha implementación el Chat la había hecho de manera errónea ya que solamente enviaba el ok pero el cliente no mostraba ningún mensaje de que el archivo se haya recibido correctamente por lo que le pedí al chat que hiciera esas modificaciones tanto en el servidor como en el cliente PUT de la siguiente forma:

Curso: Desarrollo de Sistemas Distribu | SISDIS-4CV12: Tarea extra. Transferen | [Resuelta] java | Error de Java Keytool | Servidor multithread para peticio

chat.openai.com/c/6331a5d5-8703-4508-9230-356da62af6c1

Nuevo chat

Hoy

Servidor multithread p

Últimos 7 Días

La Historia del Rutelio

Calculadora huella carbono.

Últimos 30 Días

Calcular Producto de Matrices

Programa Cliente TCP Java

Generación de fragmentos de

Refactorización del envío arch

Servidor TCP multithread en Ja

septiembre

Comando Minecraft Educatio

Actualizar a Plus

Jose Eduardo Olay Silis

En el código SecureMultiThreadServer que me proporcionaste en este método:

```
private void receiveFile(String fileName, DataInputStream input) throws IOException {
    File file = new File(fileName);
    FileOutputStream fos = new FileOutputStream(file);

    int fileSize = input.readInt();
    byte[] buffer = new byte[1024];

    int bytesRead;
    int totalBytesRead = 0;

    while (totalBytesRead < fileSize) {
      bytesRead = input.read(buffer, 0, Math.min(buffer.length, fileSize - totalBytesRead));
      if (bytesRead == -1) {
        break;
      }
      fos.write(buffer, 0, bytesRead);
      totalBytesRead += bytesRead;
    }

    fos.close();

    if (totalBytesRead == fileSize) {
      input.readUTF(); // Leer "OK" del cliente (confirmación)
      input.readUTF(); // Leer una línea en blanco
```

Volver a generar

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

9°C Mayorm. soleado | Búsqueda | 07:08 a. m. 18/10/2023

Curso: Desarrollo de Sistemas Distribu | SISDIS-4CV12: Tarea extra. Transferen | [Resuelta] java | Error de Java Keytool | Servidor multithread para peticio

chat.openai.com/c/6331a5d5-8703-4508-9230-356da62af6c1

Nuevo chat

Hoy

Servidor multithread p

Últimos 7 Días

La Historia del Rutelio

Calculadora huella carbono.

Últimos 30 Días

Calcular Producto de Matrices

Programa Cliente TCP Java

Generación de fragmentos de

Refactorización del envío arch

Servidor TCP multithread en Ja

septiembre

Comando Minecraft Educatio

Actualizar a Plus

Jose Eduardo Olay Silis

```
    if (totalBytesRead == fileSize) {
      input.readUTF(); // Leer "OK" del cliente (confirmación)
      input.readUTF(); // Leer una línea en blanco
      System.out.println("Archivo recibido y guardado con éxito: " + fileName);
    } else {
      file.delete(); // Eliminar el archivo si la transferencia no se completó
      System.out.println("Error al recibir el archivo: " + fileName);
    }
```

Queria ver si podías modificar la última parte, ya que lo que requiero es que si el archivo se pudo escribir en el disco, el servidor envie al cliente PUT OK, sino que envie ERROR.

Para lograr que el servidor envíe "PUT OK" o "PUT ERROR" al cliente en función de si el archivo se pudo escribir en el disco o no, puedes realizar la siguiente modificación en el método `receiveFile` del servidor:

java | Copy code

```
private void receiveFile(String fileName, DataInputStream input, DataOutputS
    File file = new File(fileName);
    FileOutputStream fos = new FileOutputStream(file);

    int fileSize = input.readInt();
    byte[] buffer = new byte[1024];
```

Volver a generar

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

9°C Mayorm. soleado | Búsqueda | 07:08 a. m. 18/10/2023

```java
    int bytesRead;
    int totalBytesRead = 0;

    while (totalBytesRead < fileSize) {
        bytesRead = input.read(buffer, 0, Math.min(buffer.length, fileSize -
        if (bytesRead == -1) {
            break;
        }
        fos.write(buffer, 0, bytesRead);
        totalBytesRead += bytesRead;
    }

    fos.close();

    if (totalBytesRead == fileSize) {
        output.writeUTF("PUT OK");
        System.out.println("Archivo recibido y guardado con éxito: " + fileN
    } else {
        file.delete(); // Eliminar el archivo si la transferencia no se comp
        output.writeUTF("PUT ERROR");
        System.out.println("Error al recibir el archivo: " + fileName);
    }
}
```

Volver a generar

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

Ahora puedes agregar en el código de ClientePUT que cuando reciba el OK por parte del servidor el cliente despliegue un mensaje indicando que el archivo fue recibido por el servidor con éxito, de otra manera despliegue un mensaje de error indicando que el servidor no pudo escribir el archivo en el disco local.

Por supuesto, puedes agregar código en el cliente para mostrar mensajes de éxito o error en función de la respuesta del servidor. Aquí tienes la modificación en el cliente:

```java
// ... (código previo)

String response = input.readUTF();
if ("PUT OK".equals(response)) {
    System.out.println("El archivo fue recibido por el servidor con éxito.")
} else if ("PUT ERROR".equals(response)) {
    System.out.println("Error: El servidor no pudo escribir el archivo en el
} else {
    System.out.println("Respuesta desconocida del servidor: " + response);
}

connection.close();
```

Volver a generar

Envía un mensaje

Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT September 25 Version

Y ya una vez realizados esos cambios le pedí al Chat que me implementara el cliente GET de la siguiente forma:

Ahora de acuerdo al código que me proporcionaste de SecureMultiThreadServer y de acuerdo al cliente SSL que te porporcione el cual es este:
SSLSocketFactory cliente = (SSLSocketFactory) SSLSocketFactory.getDefault();

Socket conexion = cliente.createSocket("localhost",50000);

DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());

DataInputStream entrada = new DataInputStream(conexion.getInputStream());

Me puedes realizar la implementación del clienteGET el cual deberá recibir como parámetros la IP del servidor, el puerto que escucha el servidor y el nombre del archivo a recibir.

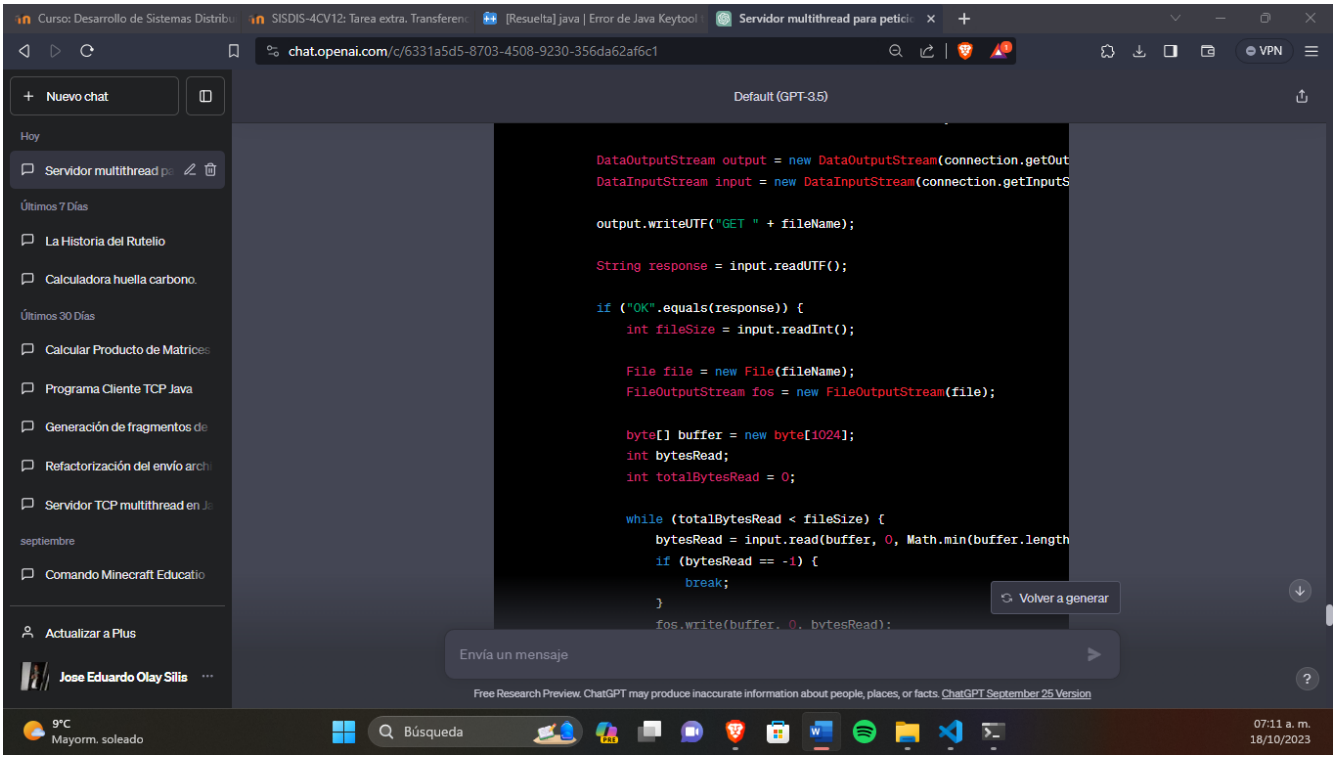
Este cliente se conectará al servidor y enviará la petición GET y el nombre del archivo.

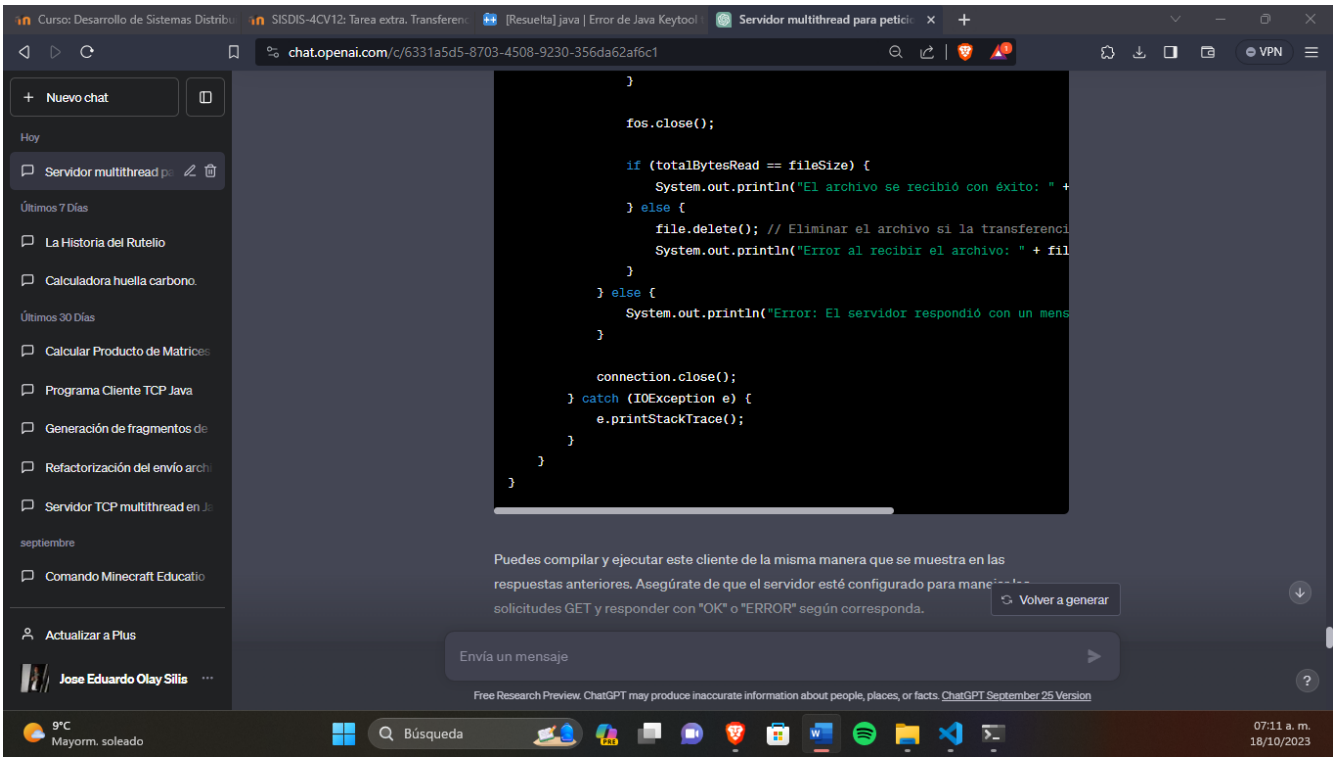
El cliente deberá esperar la respuesta del servidor, si el servidor responde OK el cliente deberá recibir la longitud del archivo y el contenido del archivo. Entonces el cliente deberá escribir el archivo en el disco local, si puede escribir el archivo deberá desplegar un mensaje indicando que el archivo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

Aquí tienes un cliente GET que se conecta al servidor y recibe un archivo en func

Aquí tienes un cliente GET que se conecta al servidor y recibe un archivo en función de la petición GET y el nombre del archivo proporcionados como argumentos de línea de comandos:

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

public class SecureClientGET {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Uso: java SecureClientGET <IP del servidor>
            return;
        }

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

        try {
            SSLSocketFactory clientFactory = (SSLSocketFactory) S
            SSLSocket connection = (SSLSocket) clientFactory.creat
```





Ya una vez teniendo los 3 archivos cambie el nombre de las clases para que se pudieran ejecutar de una forma más sencilla y posteriormente se agregó manualmente las propiedades en los programas tanto de los clientes como del servidor utilizando el método setProperty() de la siguiente manera:

```
System.setProperty("javax.net.ssl.trustStore","keystore_cliente.jks");
System.setProperty("javax.net.ssl.trustStorePassword","123456");
```

```
System.setProperty("javax.net.ssl.keyStore","keystore_servidor.jks");
System.setProperty("javax.net.ssl.keyStorePassword","1234567");
```

El código de los archivos quedaron de la siguiente manera:

ServidorMultithread.java

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

public class ServidorMultithread {

    public static void main(String[] args) throws Exception {
        System.setProperty("javax.net.ssl.keyStore","keystore_servidor.jks");
        System.setProperty("javax.net.ssl.keyStorePassword","1234567");
        SSLServerSocketFactory socket_factory = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
        ServerSocket socket_servidor = socket_factory.createServerSocket(8080);
        System.out.println("Servidor listo para recibir peticiones...");

        while (true) {
            Socket conexion = socket_servidor.accept();
            Thread clientThread = new ClientHandler(conexion);
            clientThread.start();
        }
    }
}

class ClientHandler extends Thread {
    private Socket clientSocket;

    public ClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            DataOutputStream salida = new
DataOutputStream(clientSocket.getOutputStream());
            DataInputStream entrada = new
DataInputStream(clientSocket.getInputStream());

            String request = entrada.readUTF();
            String[] requestParts = request.split(" ");

            if (requestParts.length >= 2) {
```

```java
                String action = requestParts[0];
                String fileName = requestParts[1];

                if ("GET".equals(action)) {
                    // Manejar la petición GET
                    sendFile(fileName, salida);
                } else if ("PUT".equals(action)) {
                    // Manejar la petición PUT
                    receiveFile(fileName, entrada,salida);
                } else {
                    salida.writeUTF("ERROR: Acción no válida.");
                }
            } else {
                salida.writeUTF("ERROR: Formato de solicitud incorrecto.");
            }

            clientSocket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private void sendFile(String fileName, DataOutputStream output) throws
IOException {
        File file = new File(fileName);
        if (file.exists() && file.isFile()) {
            FileInputStream fis = new FileInputStream(file);
            int fileSize = (int) file.length();

            output.writeUTF("OK");
            output.writeInt(fileSize);

            byte[] buffer = new byte[1024];
            int bytesRead;

            while ((bytesRead = fis.read(buffer)) != -1) {
                output.write(buffer, 0, bytesRead);
            }

            fis.close();
        } else {
            output.writeUTF("ERROR: El archivo no existe.");
        }
    }

    private void receiveFile(String fileName, DataInputStream input,DataOutputStream
output) throws IOException {
```

```
        File file = new File(fileName);
        FileOutputStream fos = new FileOutputStream(file);

        int fileSize = input.readInt();
        byte[] buffer = new byte[1024];

        int bytesRead;
        int totalBytesRead = 0;

        while (totalBytesRead < fileSize) {
            bytesRead = input.read(buffer, 0, Math.min(buffer.length, fileSize -
totalBytesRead));
            if (bytesRead == -1) {
                break;
            }
            fos.write(buffer, 0, bytesRead);
            totalBytesRead += bytesRead;
        }
                if (totalBytesRead == fileSize) {
                output.writeUTF("OK");
                System.out.println("Archivo recibido y guardado con éxito: " +
fileName);
            } else {
                file.delete(); // Eliminar el archivo si la transferencia no se
completó
                output.writeUTF("ERROR");
                System.out.println("Error al recibir el archivo: " + fileName);
            }
        fos.close();

    }
}
```

ClientePUT.java

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

public class ClientePUT {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Uso: java SecureClientPUT <IP del servidor> <puerto>
<nombre del archivo>");
            return;
        }
```

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

        try {

            System.setProperty("javax.net.ssl.trustStore","keystore_cliente.jks");
            System.setProperty("javax.net.ssl.trustStorePassword","123456");
            SSLSocketFactory clientFactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
            SSLSocket connection = (SSLSocket) clientFactory.createSocket(serverIP,
serverPort);

            DataOutputStream output = new
DataOutputStream(connection.getOutputStream());
            DataInputStream input = new
DataInputStream(connection.getInputStream());

            File file = new File(fileName);

            if (file.exists() && file.isFile()) {
                FileInputStream fis = new FileInputStream(file);
                int fileSize = (int) file.length();

                output.writeUTF("PUT " + file.getName());
                output.writeInt(fileSize);

                byte[] buffer = new byte[1024];
                int bytesRead;

                while ((bytesRead = fis.read(buffer)) != -1) {
                    output.write(buffer, 0, bytesRead);
                }

                fis.close();

                String response = input.readUTF();
                if ("PUT OK".equals(response)) {
                    System.out.println("El archivo fue recibido por el servidor con
éxito.");
                } else if ("PUT ERROR".equals(response)) {
                    System.out.println("Error: El servidor no pudo escribir el
archivo en el disco local.");
                } else {
                    System.out.println("Respuesta del servidor: " + response);
                }
            } else {
```

```
                System.out.println("Error: No se puede leer el archivo del disco
local.");
            }

            connection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ClienteGET.java

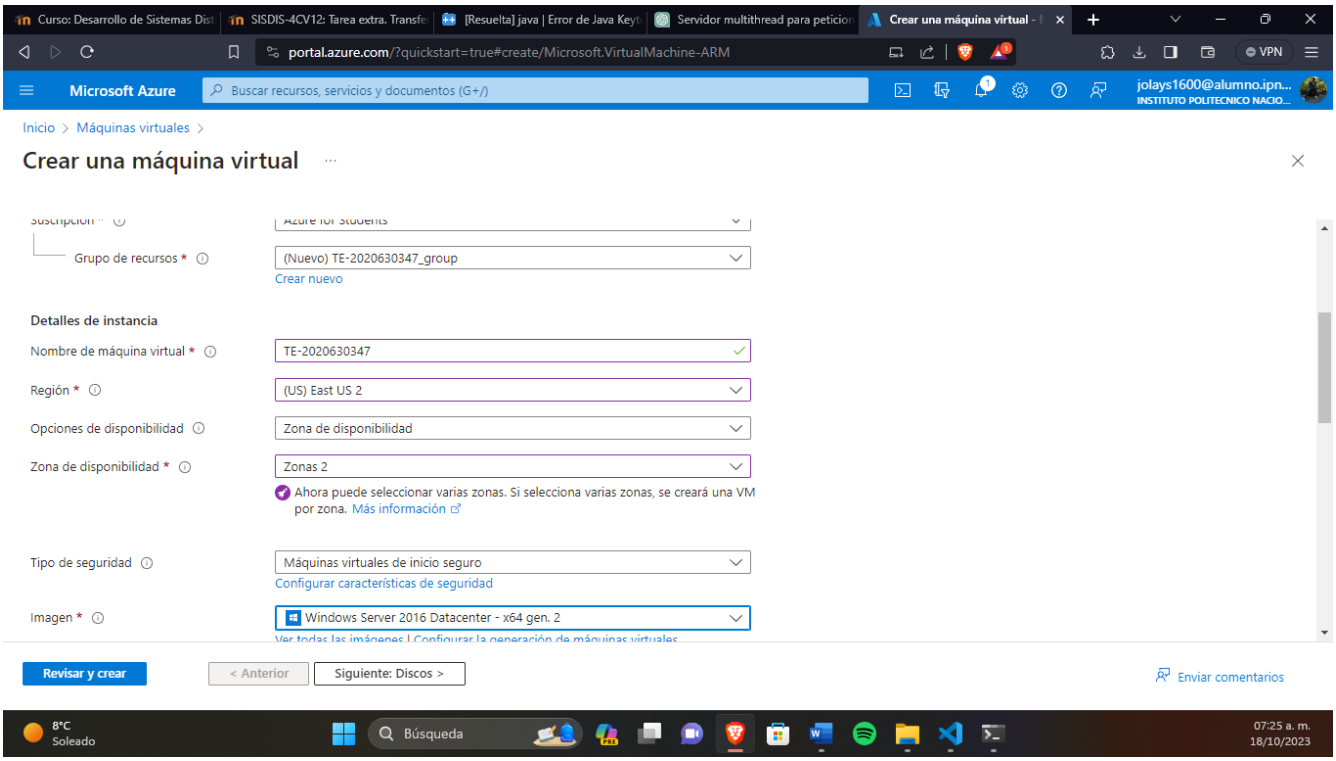
```
import java.io.*;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;

public class ClienteGET {
    public static void main(String[] args) {
        if (args.length != 3) {
            System.out.println("Uso: java SecureClientGET <IP del servidor> <puerto>
<nombre del archivo>");
            return;
        }

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

        try {
            System.setProperty("javax.net.ssl.trustStore","keystore_cliente.jks");
            System.setProperty("javax.net.ssl.trustStorePassword","123456");
            SSLSocketFactory clientFactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
            SSLSocket connection = (SSLSocket) clientFactory.createSocket(serverIP,
serverPort);

            DataOutputStream output = new
DataOutputStream(connection.getOutputStream());
            DataInputStream input = new
DataInputStream(connection.getInputStream());

            output.writeUTF("GET " + fileName);

            String response = input.readUTF();

            if ("OK".equals(response)) {
```

```
                int fileSize = input.readInt();

                File file = new File(fileName);
                FileOutputStream fos = new FileOutputStream(file);

                byte[] buffer = new byte[1024];
                int bytesRead;
                int totalBytesRead = 0;

                while (totalBytesRead < fileSize) {
                    bytesRead = input.read(buffer, 0, Math.min(buffer.length,
fileSize - totalBytesRead));
                    if (bytesRead == -1) {
                        break;
                    }
                    fos.write(buffer, 0, bytesRead);
                    totalBytesRead += bytesRead;
                }

                fos.close();

                if (totalBytesRead == fileSize) {
                    System.out.println("El archivo se recibió con éxito: " +
fileName);
                } else {
                    file.delete(); // Eliminar el archivo si la transferencia no se
completó
                    System.out.println("Error al recibir el archivo: " + fileName);
                }
            } else {
                System.out.println("Error: El servidor respondió con un mensaje de
error.");
            }

            connection.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

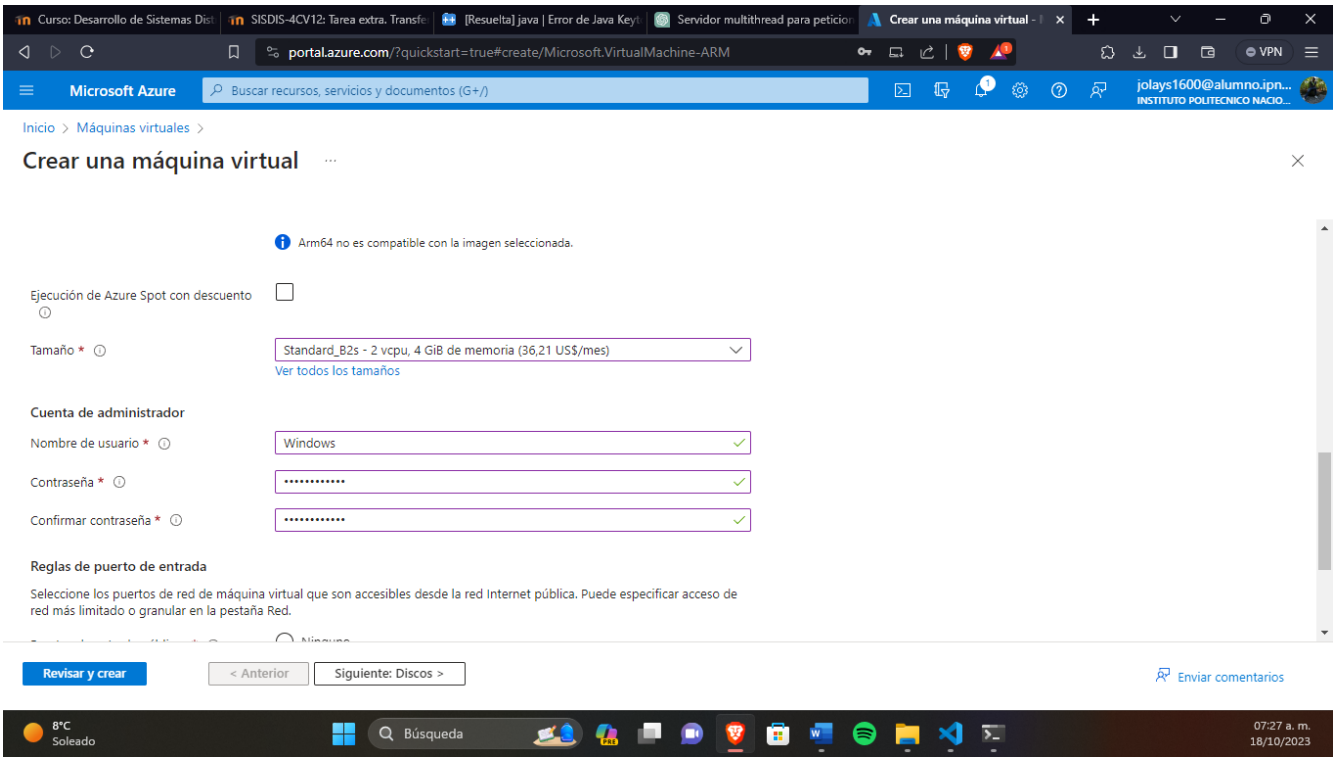
Fase 2

Una vez realizado el código tanto del cliente como de los servidores, toca la creación de la máquina virtual en Azure la cual se encargará de ejecutar el servidor por lo que comencé con la creación de la misma.

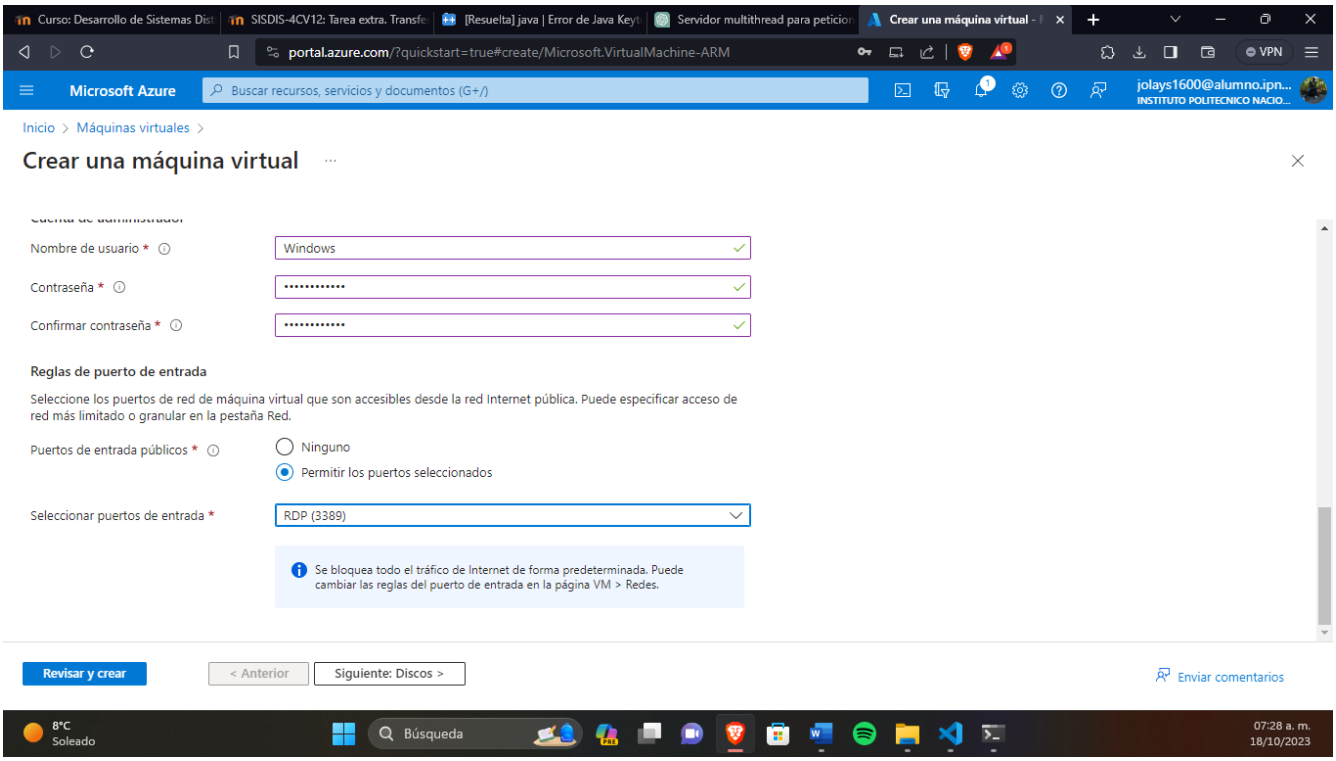
Comenzamos escribiendo el nombre, la zona y la Imagen de la misma que para esta tarea será de Windows.



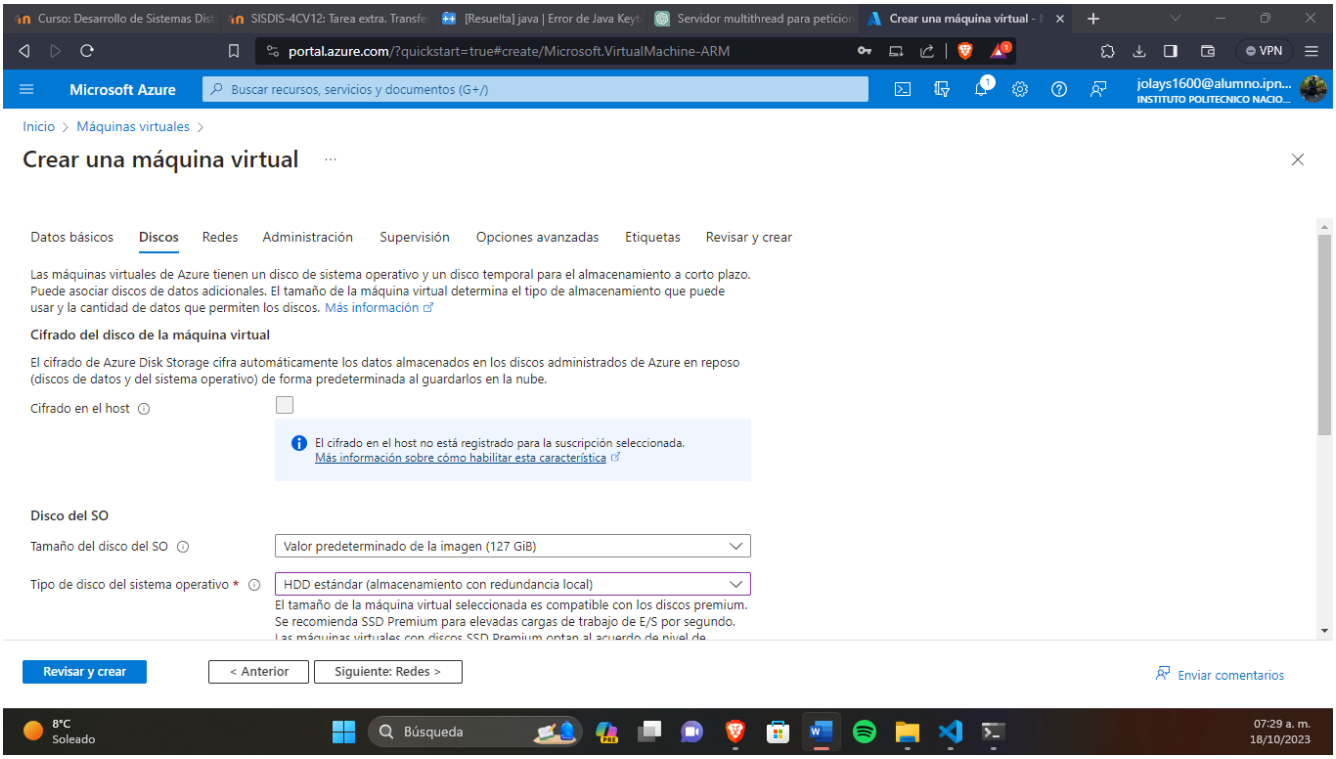
Ahora asignamos el tamaño de la máquina virtual para el cual seleccionamos el de 2GB y anotamos el usuario y la contraseña.



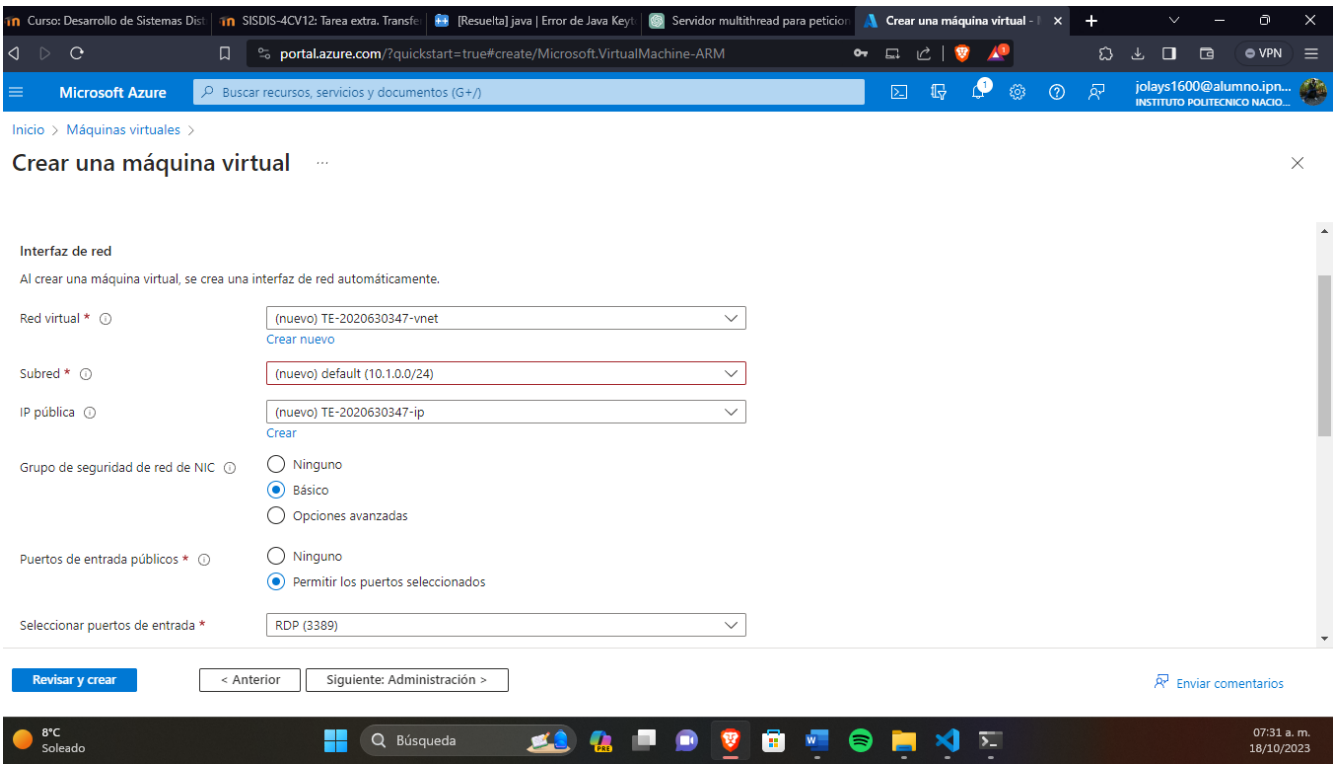
En la regla de puerto de entrada dejaremos abierto el puerto 3389 para utilizar el Remote Desktop Protocol.



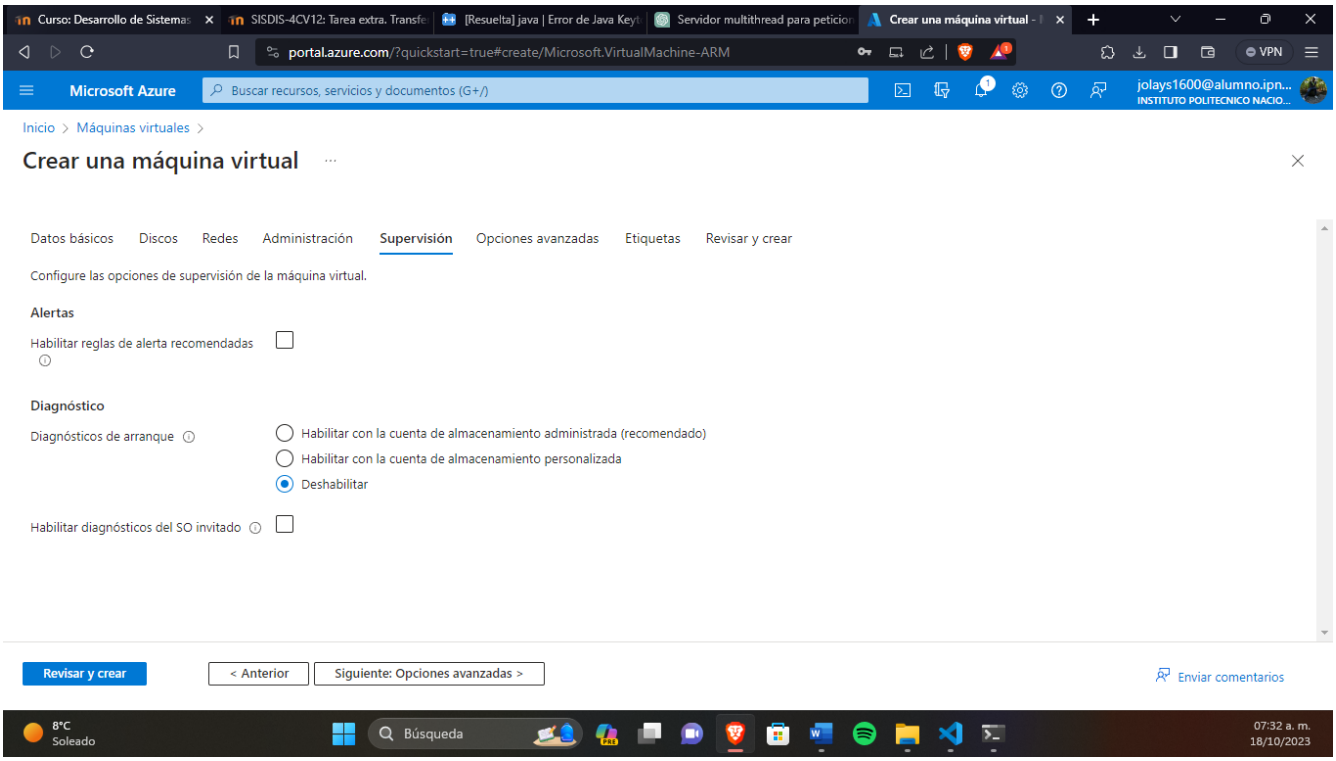
En la parte del tipo de disco del sistema operativo seleccionamos el HDD estándar.



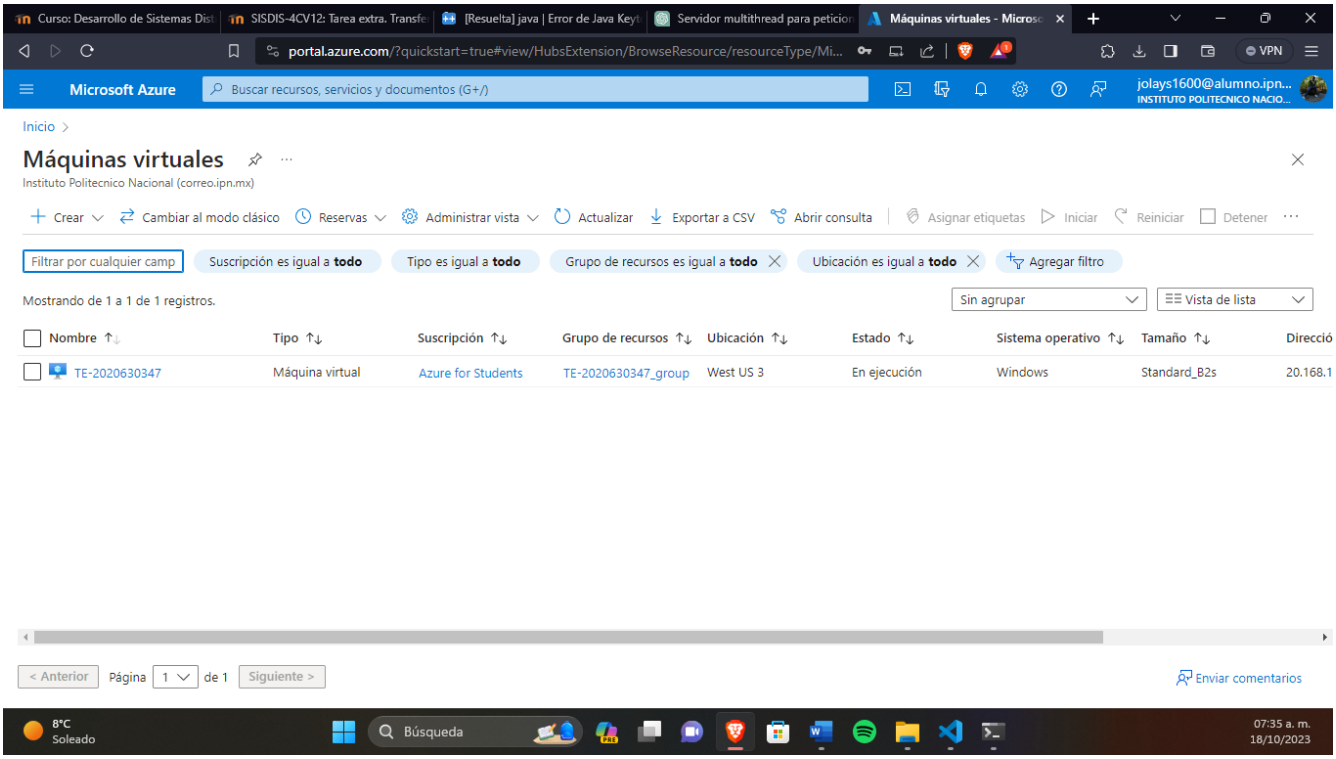
Observamos que se crea automáticamente al IP pública



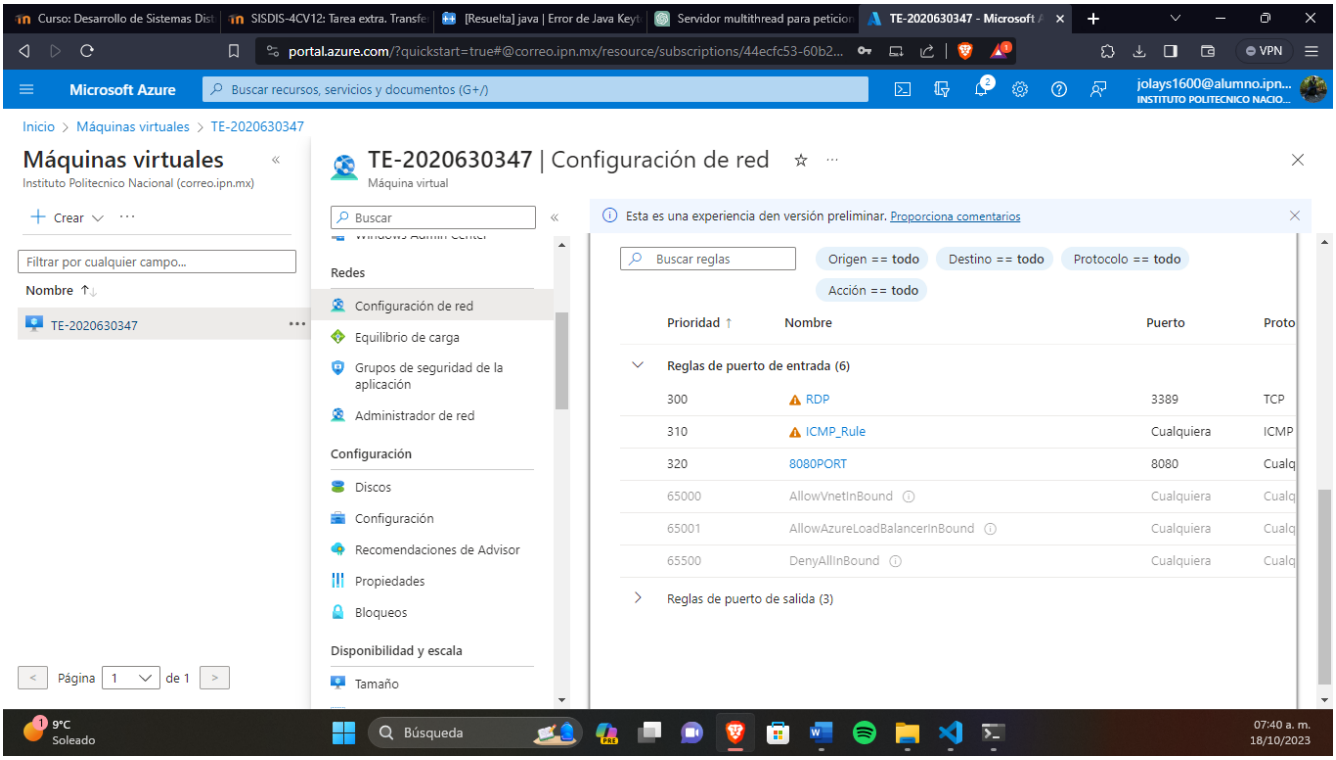
Deshabilitamos el diagnostico de arranque



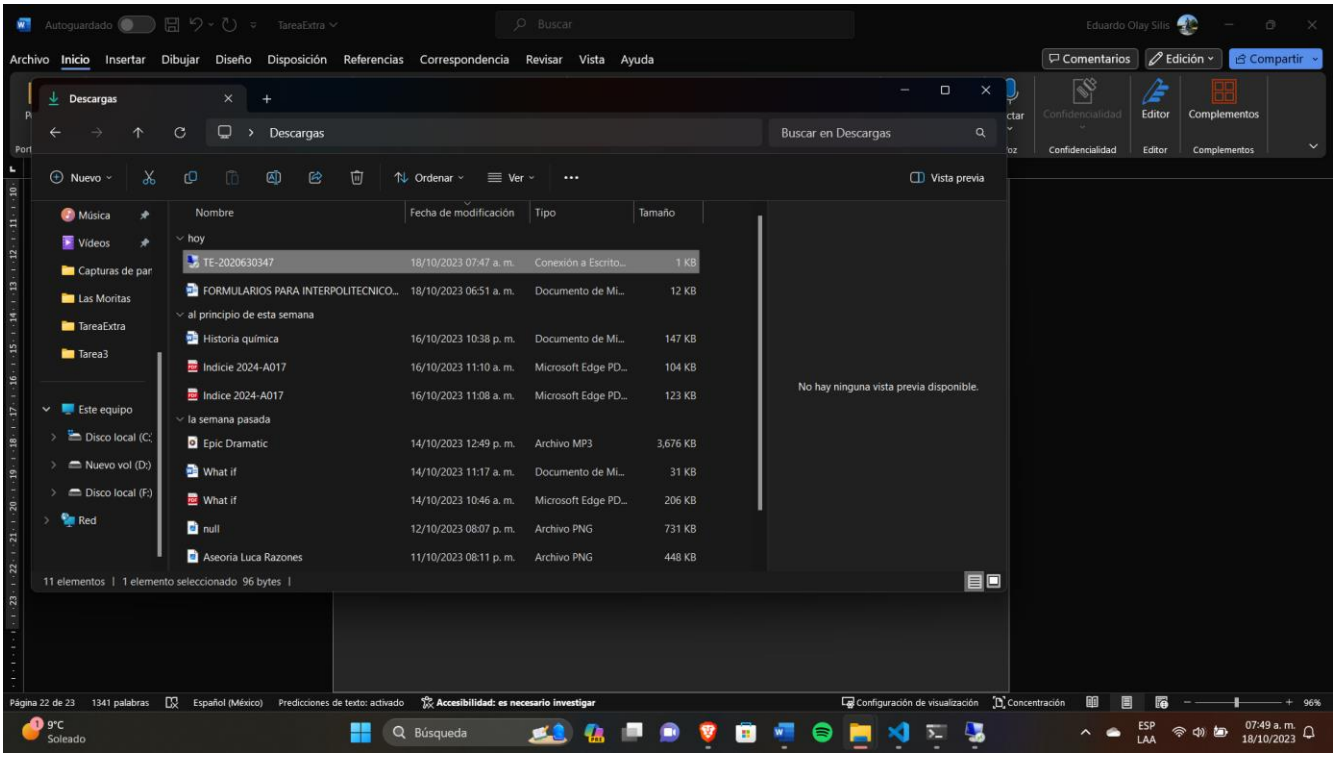
Y finalmente seleccionamos la opción revisar y crear, verificando que la máquina fue creada correctamente.



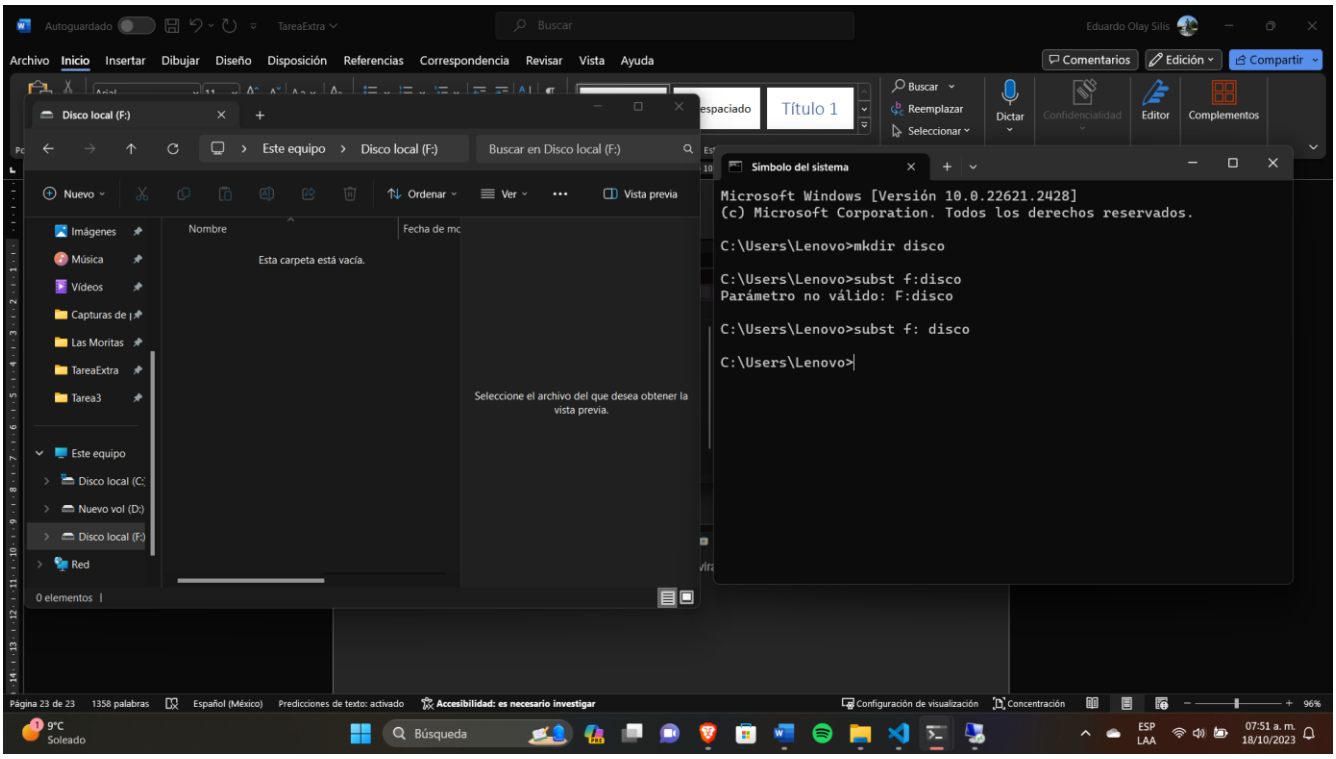
Una vez realizado eso le agregamos algunas reglas y condiciones. Para poder verificar la conexión mediante el ping creamos la regla ICMP y en el programa del servidor tenemos como puerto el 8080 por lo que de igual manera se les debe de agregar esa regla.



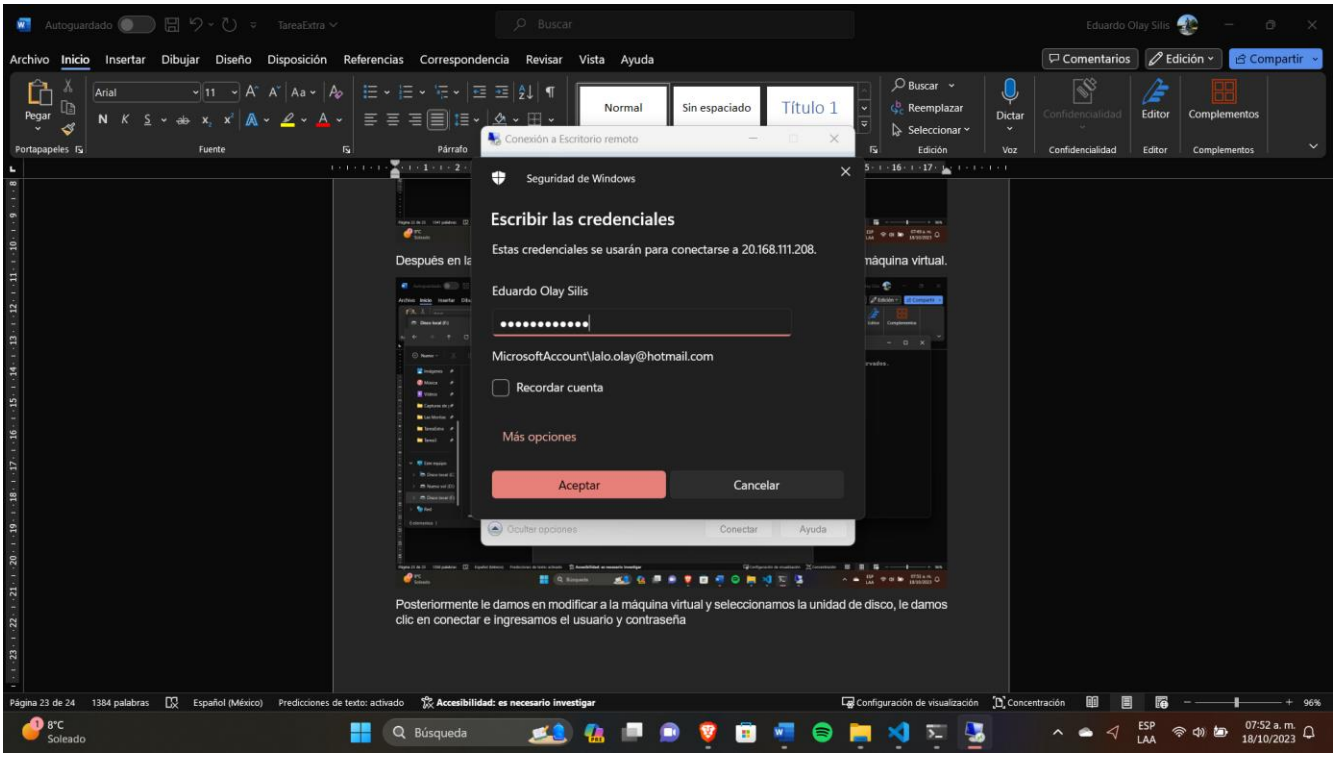
Una vez realizado ese paso descargamos el archivo de conexión rdp



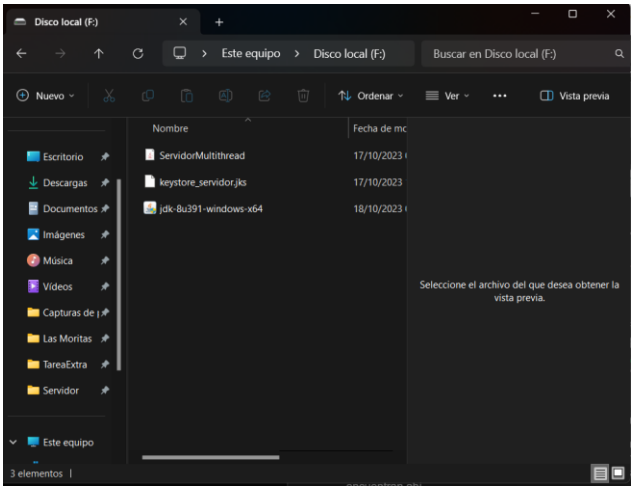
Después en la consola creamos un directorio el cual servirá como disco lógico para la máquina virtual.

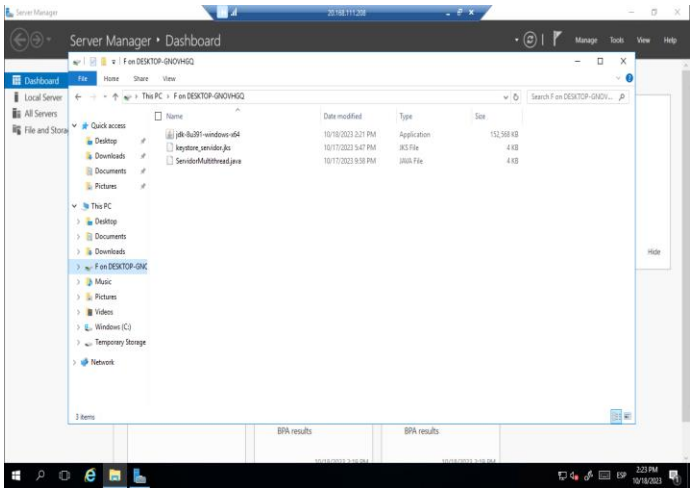


Posteriormente le damos en modificar a la máquina virtual y seleccionamos la unidad de disco, le damos clic en conectar e ingresamos el usuario y contraseña

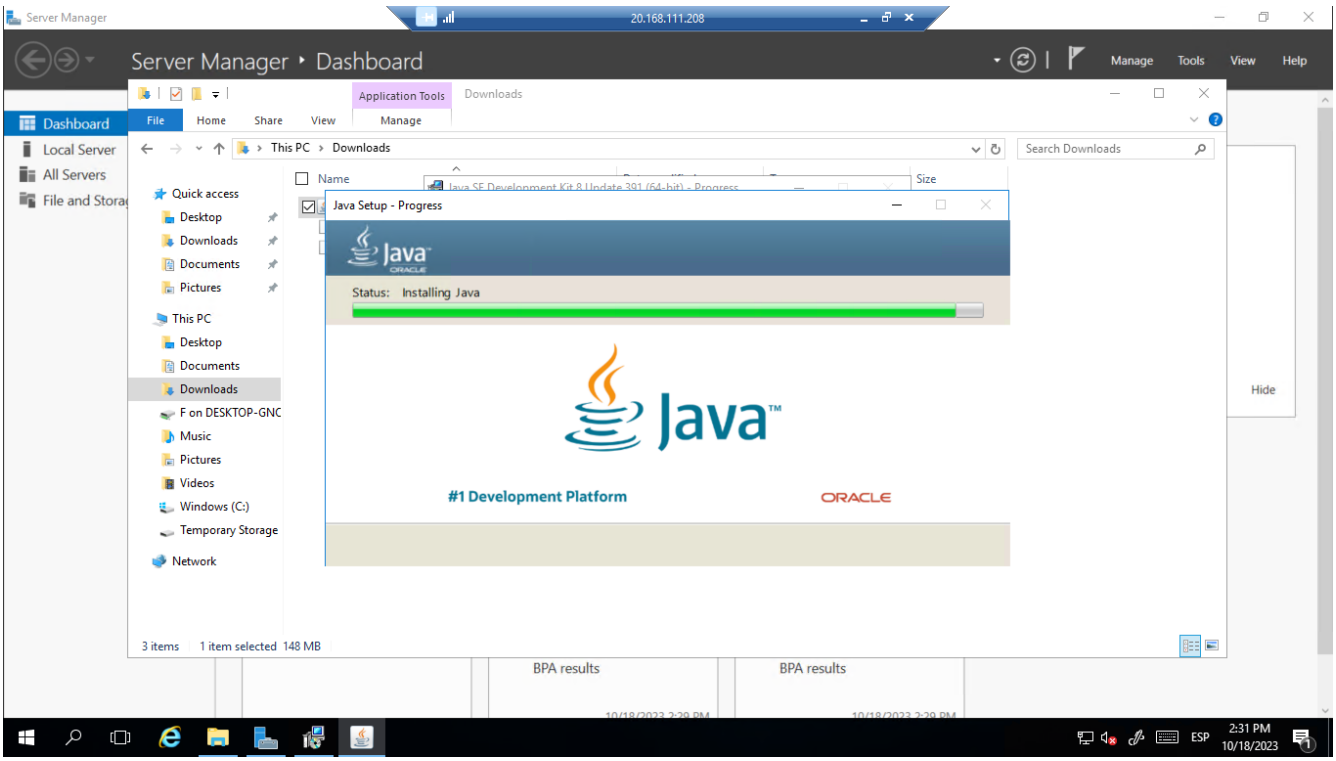


Posteriormente comenzaremos el proceso de instalación del jdk para poder compilar el programa. Nos metemos a la página de Oracle y descargamos el jdk. Una vez descargado lo pasaremos a la máquina virtual a través del disco lógico que creamos observando que efectivamente los archivos ya se encuentran ahí.

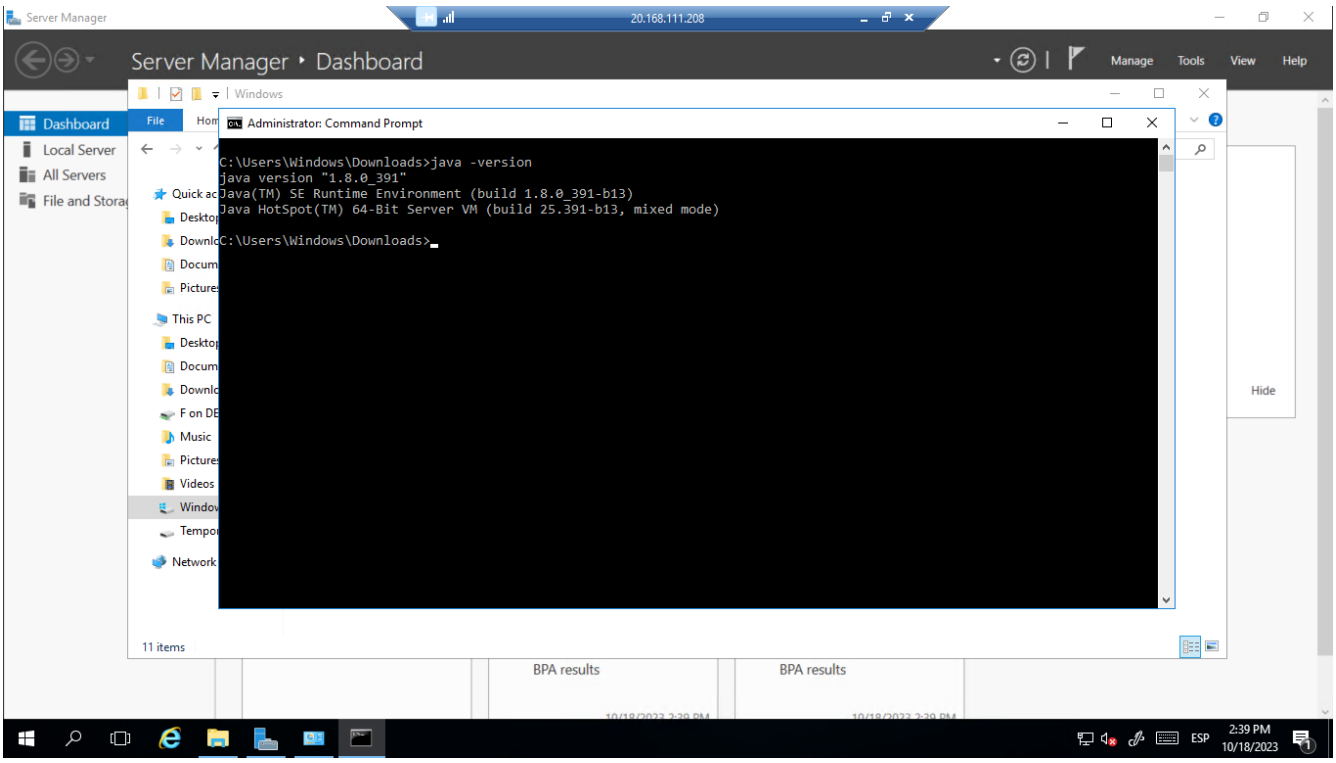




Ahora realizamos el proceso de instalación del jdk.

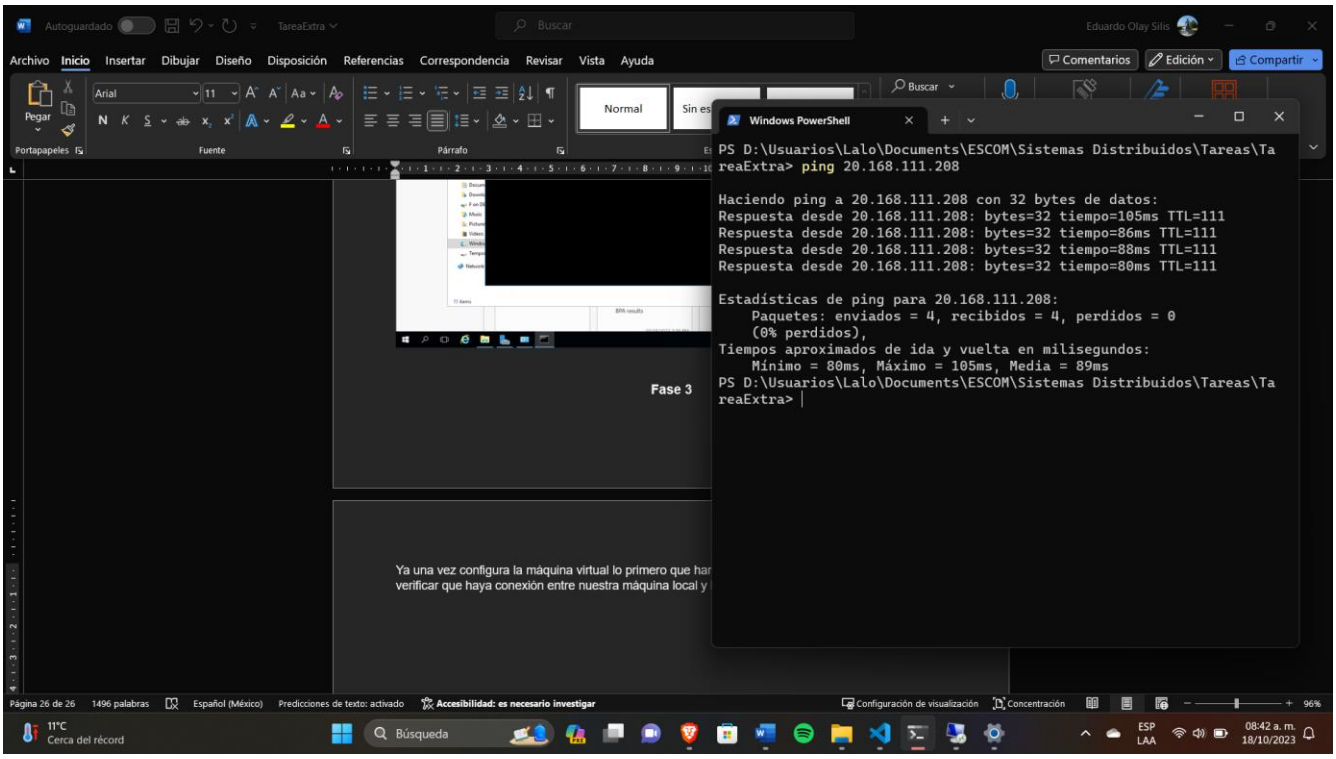


Una vez instalado, agregamos el jdk a las variables de entorno y abrimos una terminal para ver si se hizo la instalación correcta.

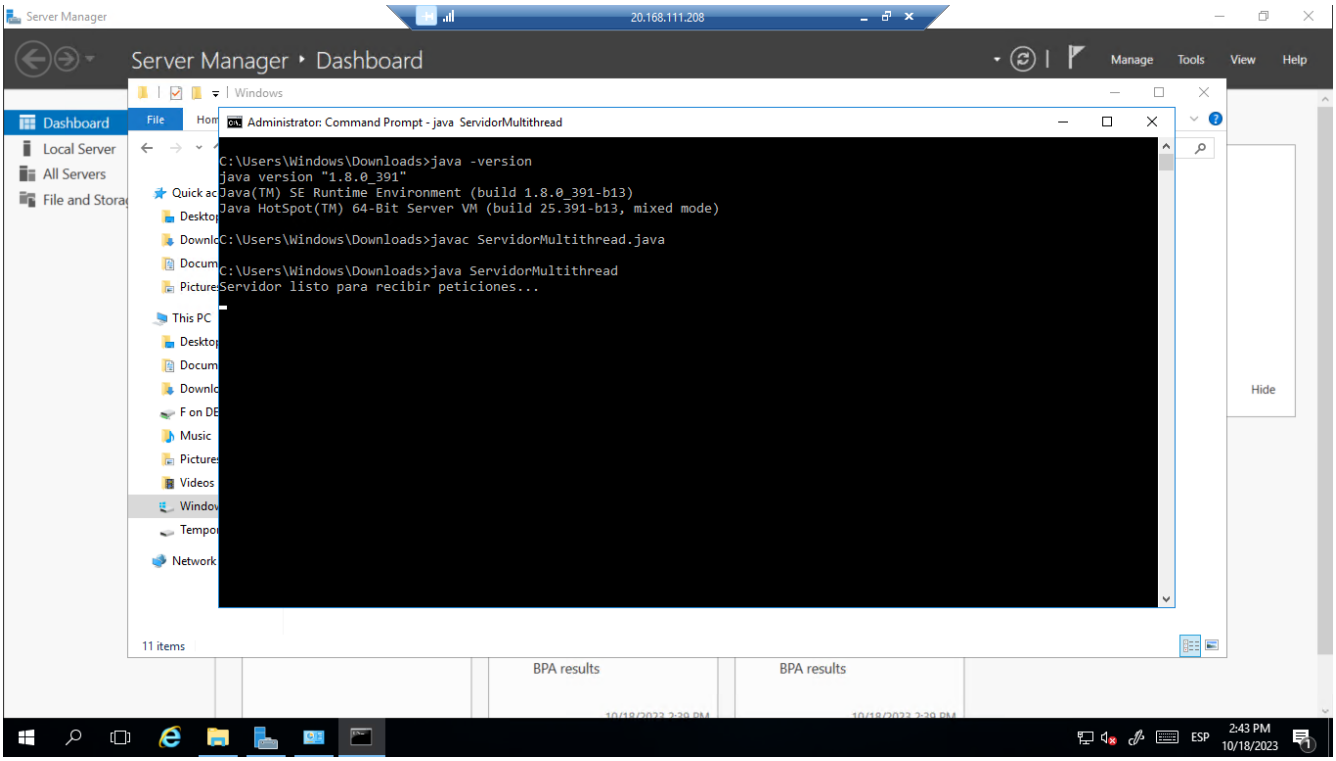


Fase 3

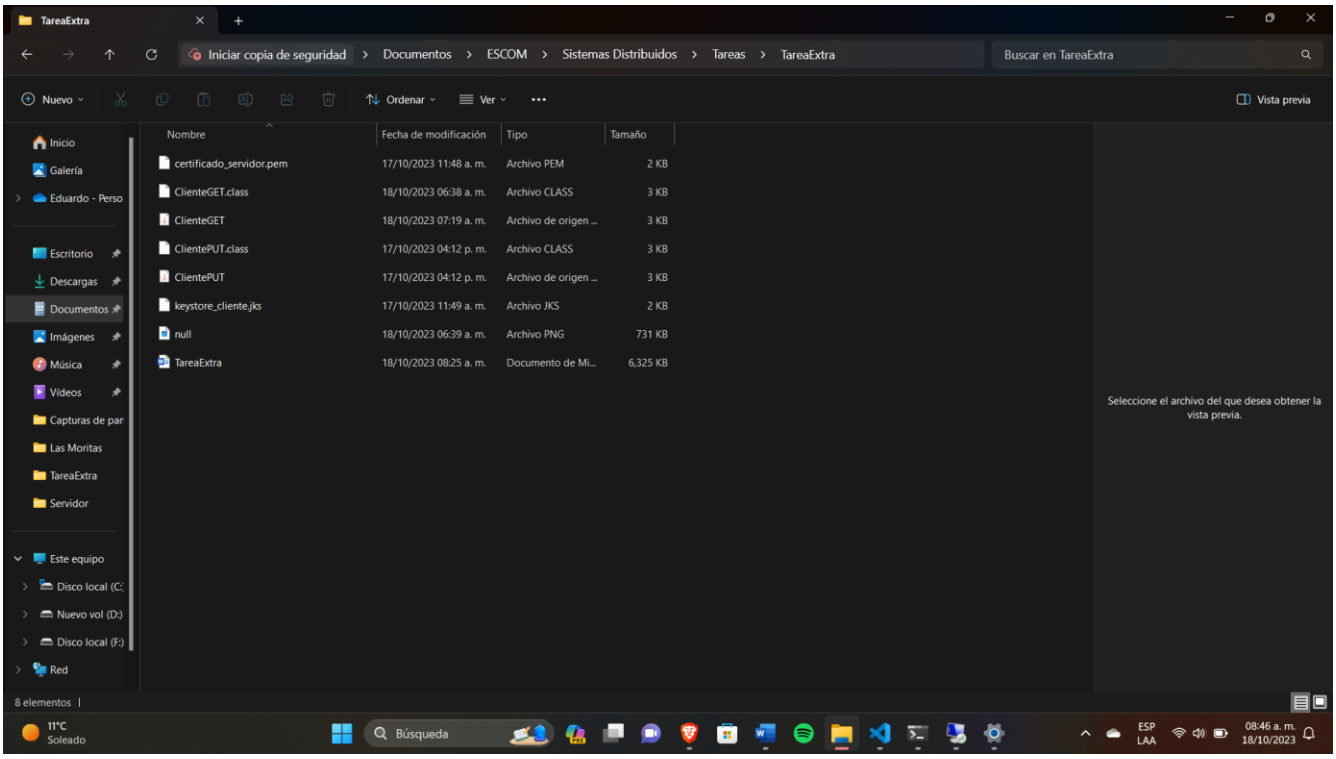
Ya una vez configura la máquina virtual lo primero que haremos será hacer un ping a la máquina para verificar que haya conexión entre nuestra máquina local y la virtual.



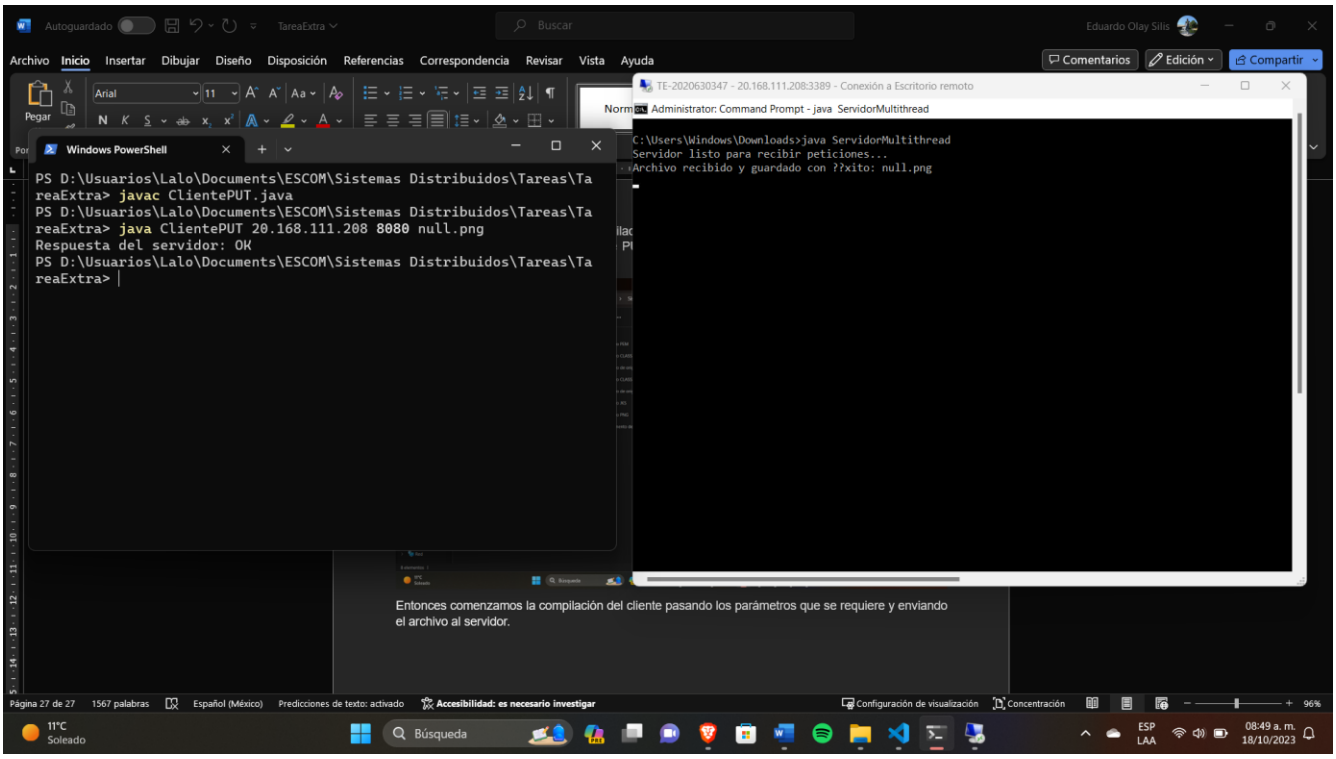
Una vez realizado eso comenzamos el proceso de compilación y ejecución del servidor de la máquina virtual.



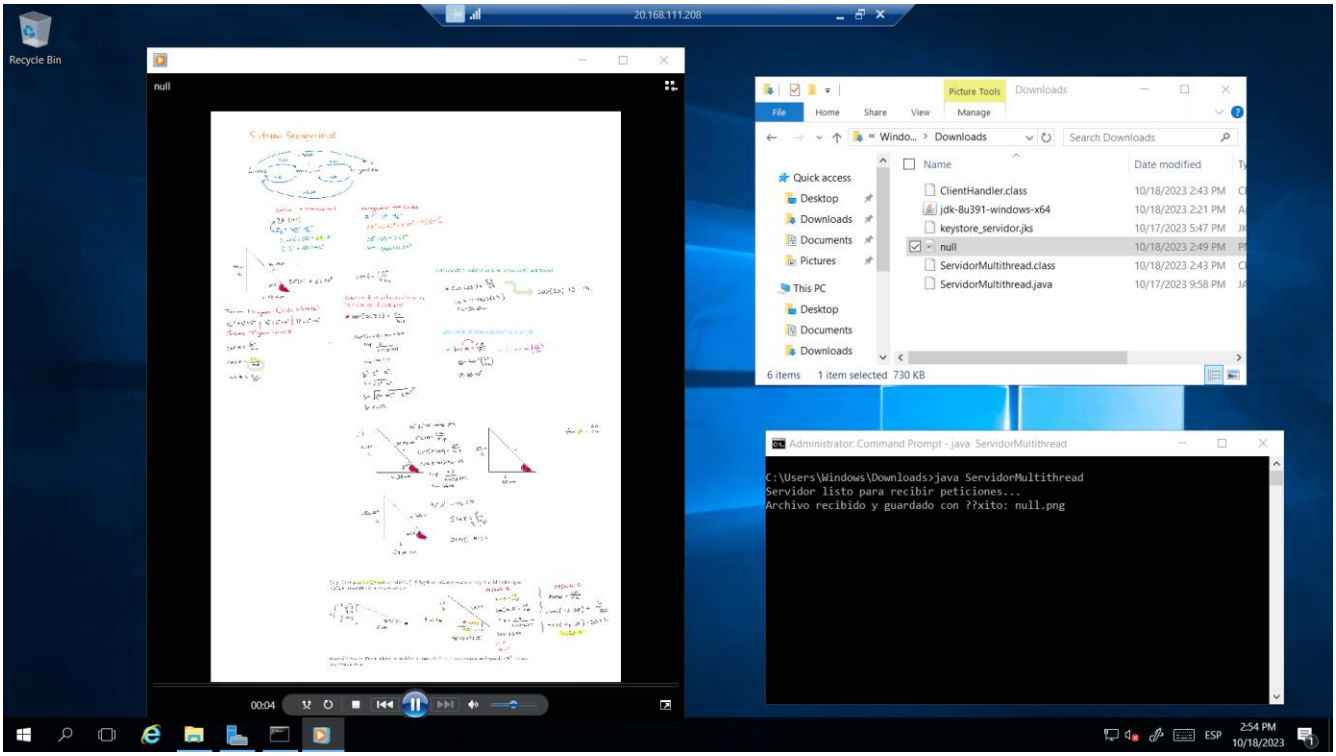
Una vez iniciado el servidor, comenzaremos la compilación del cliente PUT. Como se puede observar en la carpeta de los clientes tenemos tanto el Cliente PUT como el GET y el archivo que enviaremos el cual es null.png

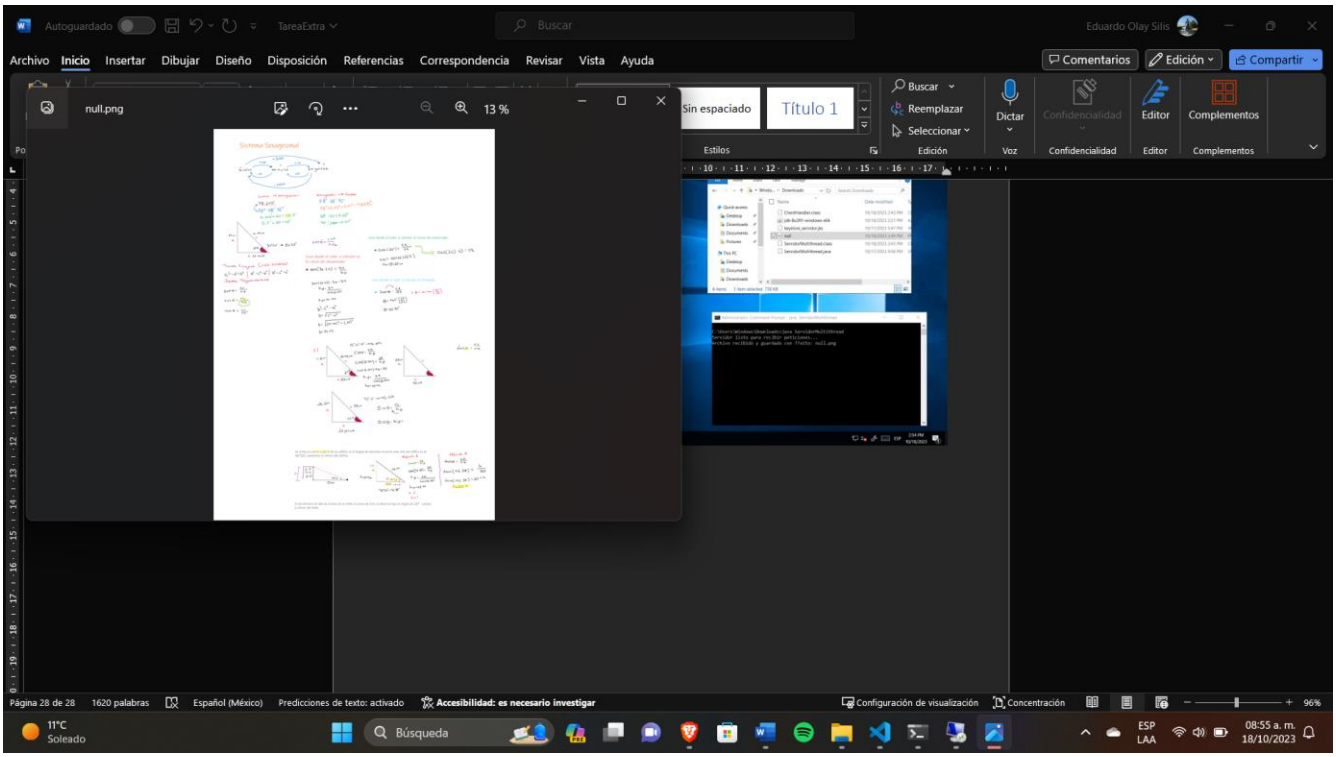


Entonces comenzamos la compilación del cliente pasando los parámetros que se requiere y enviando el archivo al servidor.

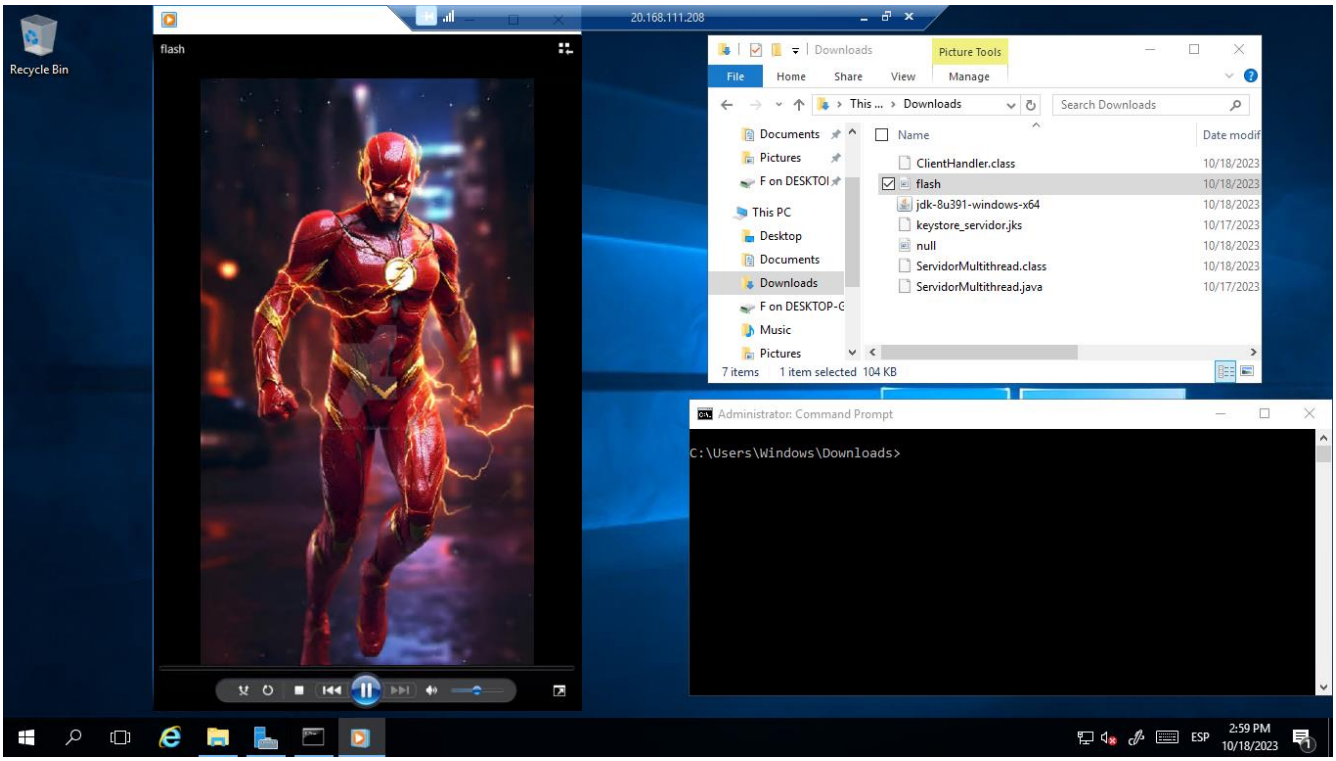


Como podemos ver el servidor muestra que se envio el archivo correctamente para verificar podemos observar que dentro de la carpeta donde se ejecuto el servidor ya se encuentra el archivo null.png y se abre sin ningún problema. De igual manera se muestra desde el escritorio local que la imagen es la misma.

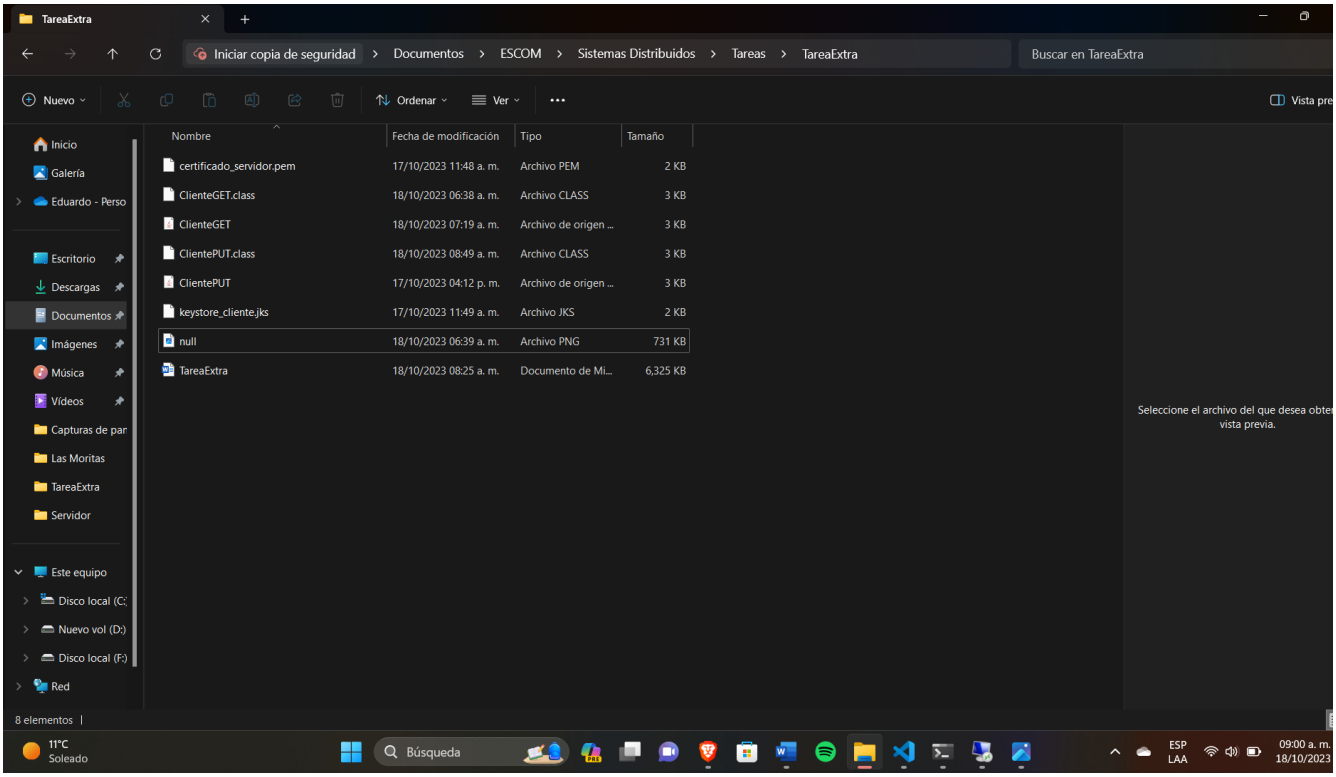




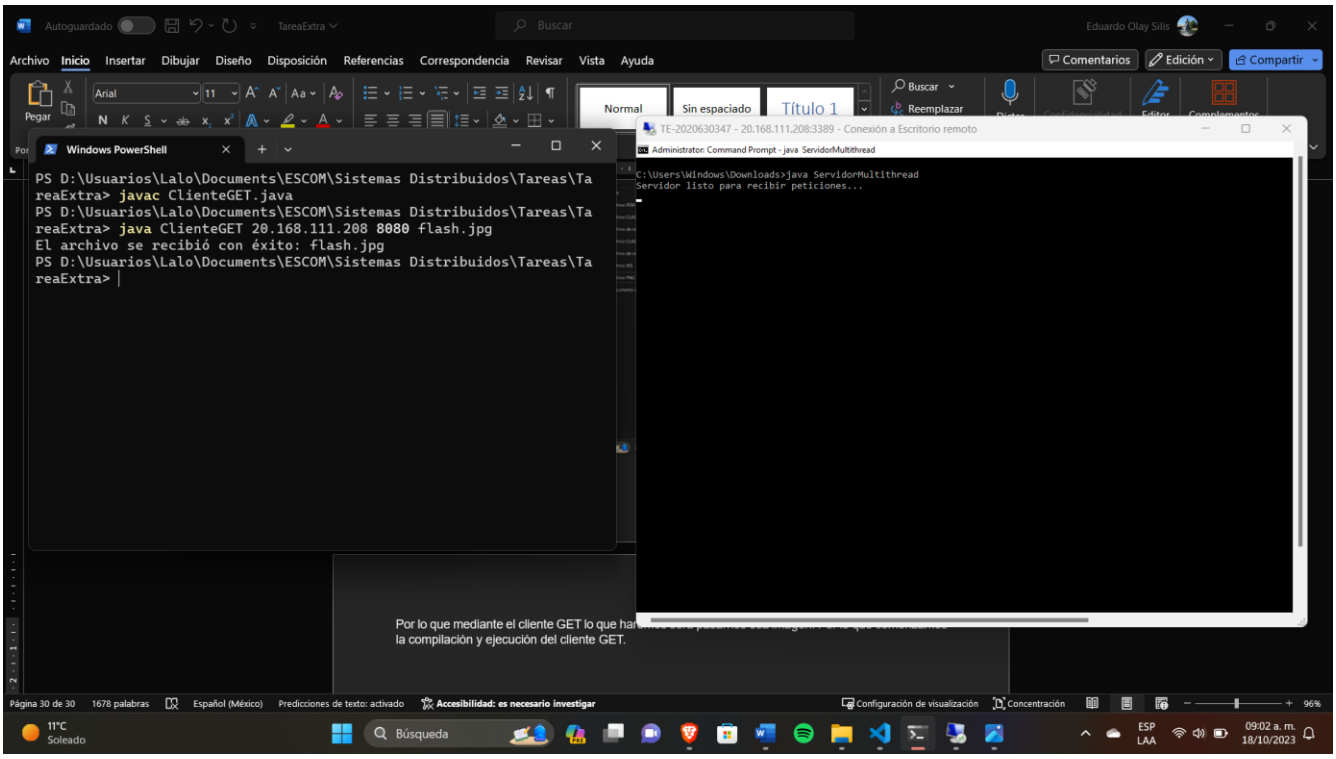
Ahora desde la máquina virtual descargamos una imagen con nombre flash.jpg la cual es la siguiente



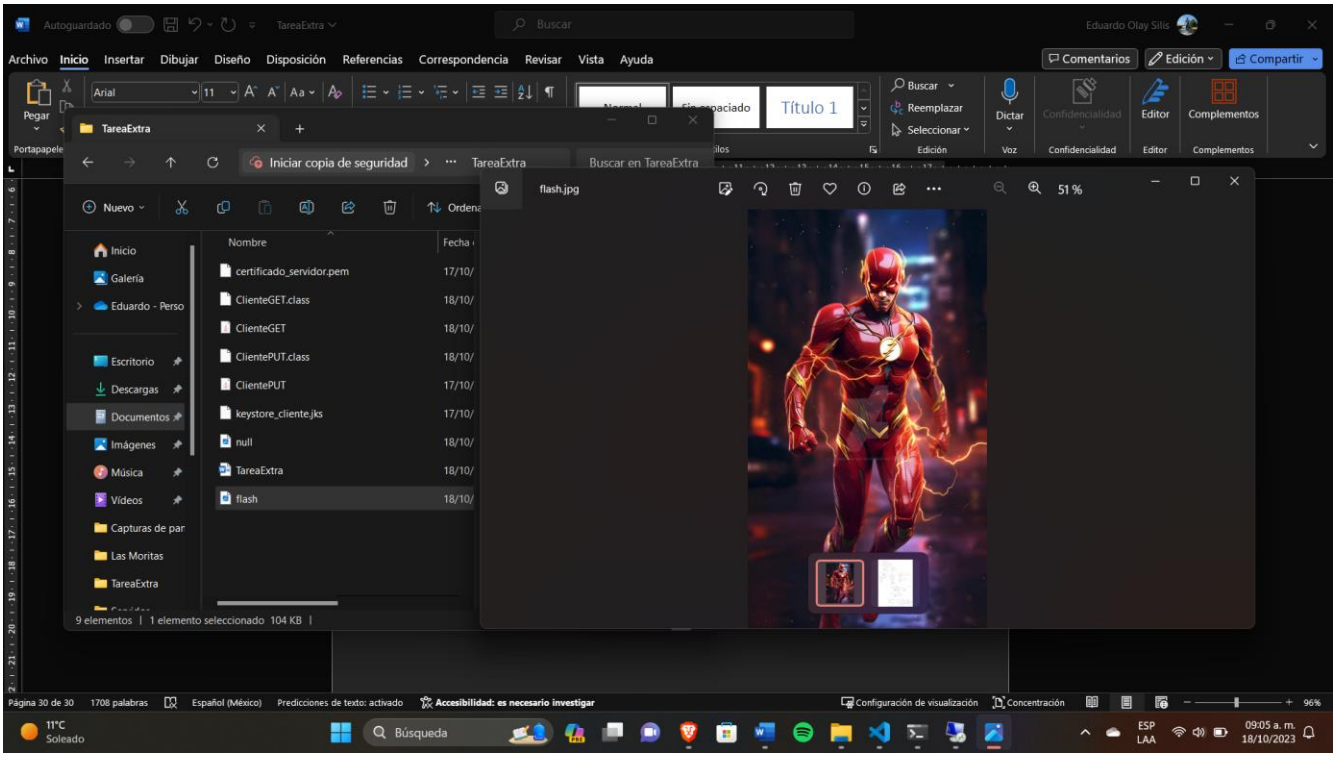
Y podemos observar que en nuestro directorio donde se localiza el cliente, dicha imagen no se encuentra



Por lo que mediante el cliente GET lo que haremos será hacer la transferencia de esa imagen de la máquina virtual a nuestra computadora local. Por lo que comenzamos la compilación y ejecución del cliente GET.



Y comprobando la carpeta del cliente podemos observar que efectivamente el archivo se transfirió y se abrió sin ningún problema, observando ser la misma imagen mostrada desde la máquina virtual unos pasos anteriores.



Conclusión

En el desarrollo de esta práctica, pude realizar la creación de un sistema cliente-servidor seguro para la transferencia de archivos, y adquirir una comprensión más profunda de los desafíos y consideraciones clave involucrados en la implementación de soluciones de comunicación segura. A lo largo de esta experiencia, pude observar la importancia de las conexiones SSL para garantizar la privacidad y la integridad de los datos durante la transmisión, así como la necesidad de manejar adecuadamente las operaciones de lectura y escritura de archivos para asegurar la fiabilidad de la transferencia. De igual manera pude demostrar cómo los clientes pueden solicitar archivos del servidor mediante solicitudes GET y cómo pueden enviar archivos al servidor mediante solicitudes PUT, y cómo el servidor responde de manera apropiada.

Además, subrayando la relevancia del manejo de errores en el desarrollo de aplicaciones seguras y robustas. La práctica ha proporcionado una valiosa oportunidad para explorar situaciones del mundo real en las que la comunicación segura y la transferencia de archivos son fundamentales, como en el intercambio de datos empresariales y en la distribución segura de contenido en línea.

En última instancia, esta práctica ha fortalecido mi comprensión de cómo implementar soluciones seguras y fiables en el ámbito de la transferencia de archivos. Estas habilidades son de gran importancia en un mundo digital en constante evolución y desempeñarán un papel esencial en el desarrollo de aplicaciones y servicios seguros en el futuro.