



## Tarea 3

### Portada

**Materia:** Diseño de sistemas distribuidos

**Profesor(a):** Pineda Guerrero Carlos

**Fecha:** @March 29, 2023

Nombre : **Rivera Barron Isaac Alejandro**

Numero de cuenta: **2020630578**



---

### Índice

[Portada](#)

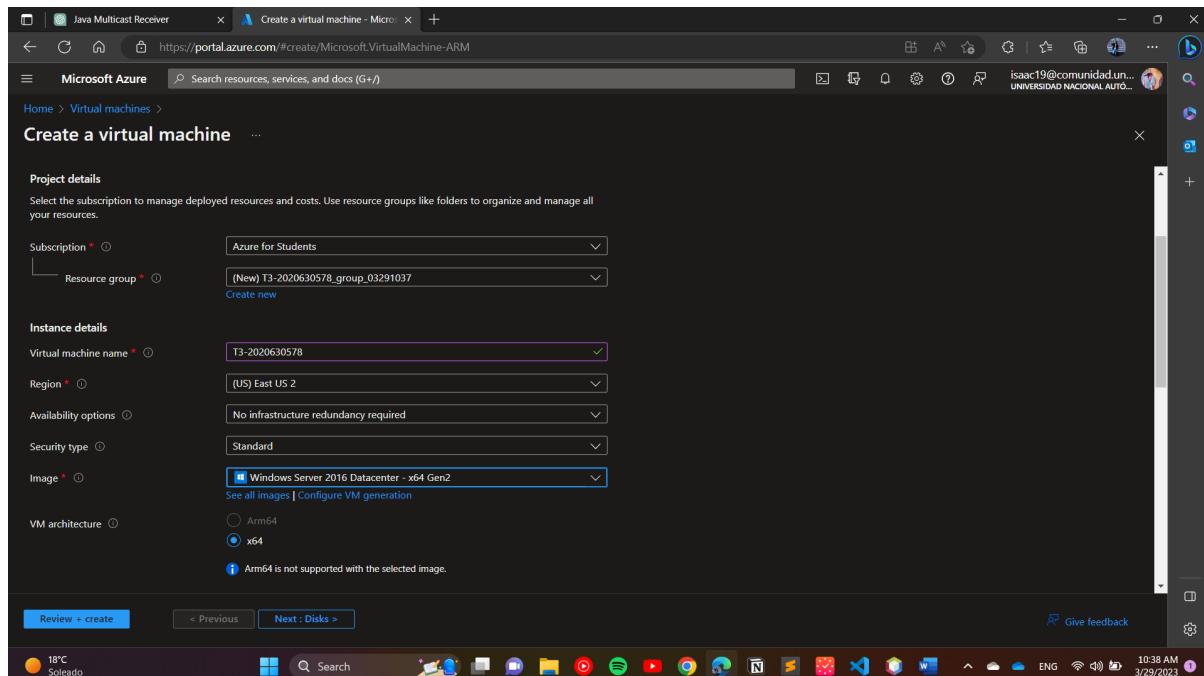
[Índice](#)

[Procesos](#)

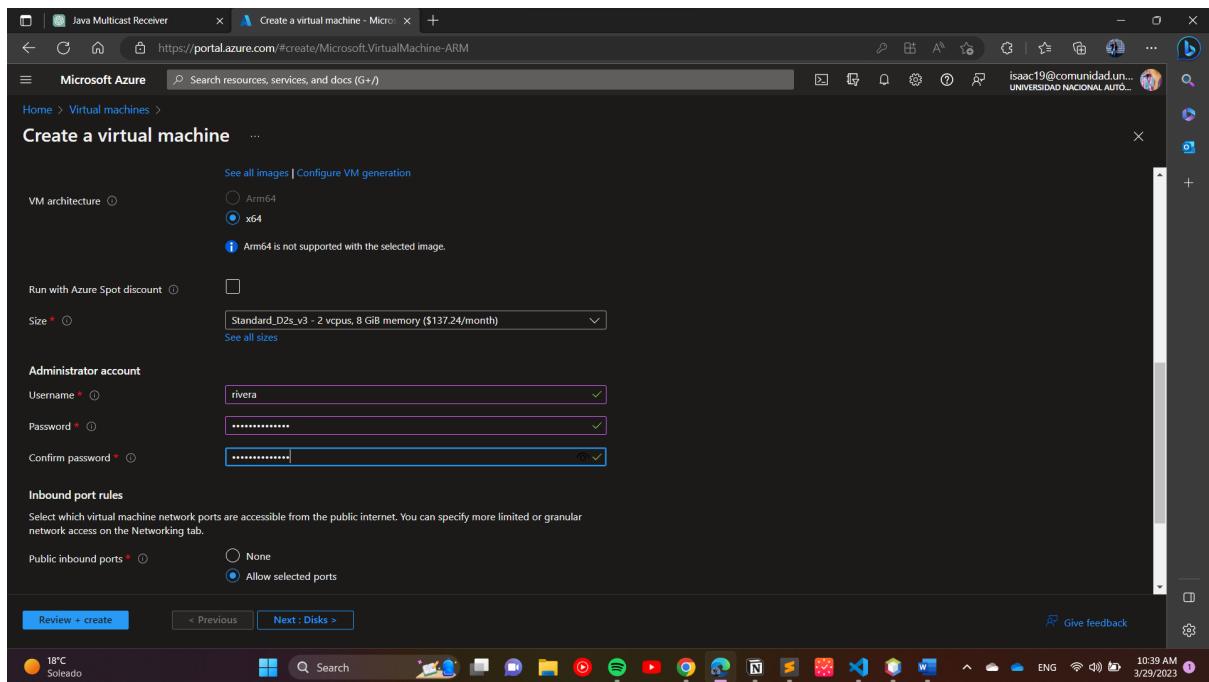
[Creación de la maquina virtual](#)  
[Configuracion de puertos](#)  
[Conexión a a la maquina virtual](#)  
[Instalación de java en la maquina virtual](#)  
Desarrollo  
[Modelado del sistema con chat GPT](#)  
Resultados  
[Compilacion & Ejecucion](#)  
[Ejecución de la conversación](#)  
Conclusiones

## Procesos

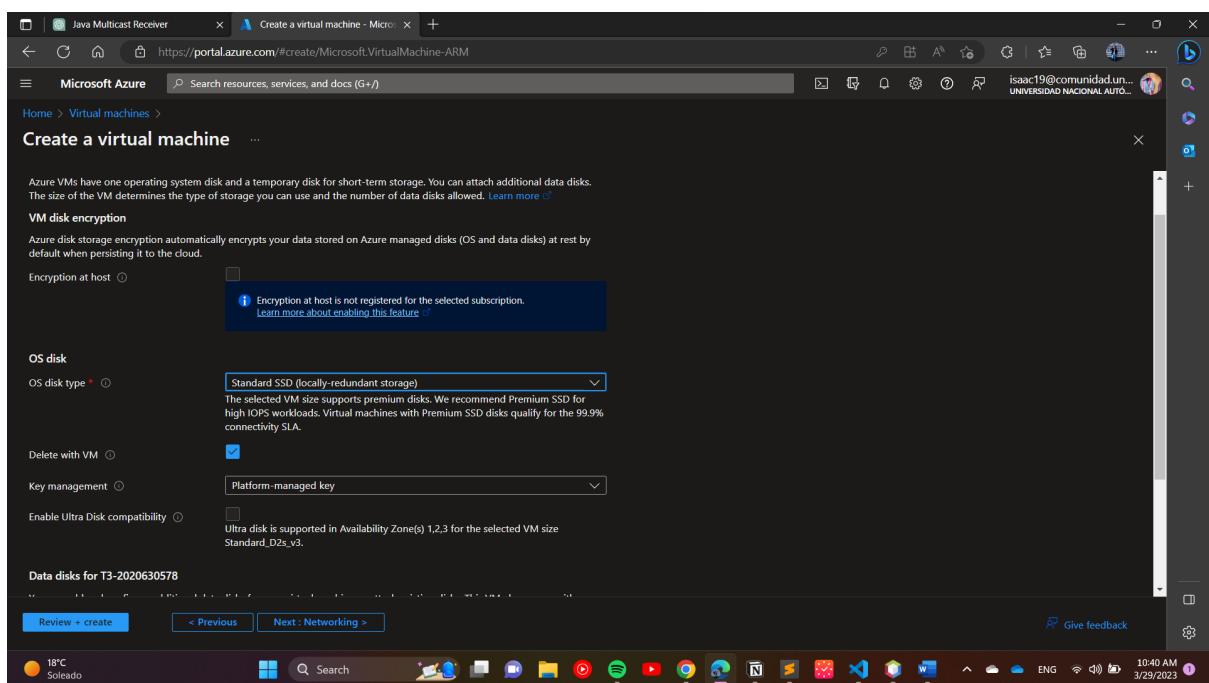
### Creación de la maquina virtual



Debemos elegir ahora un sistema operativo windows



Asignamos las licencias para ingresar a nuestra maquina virtual



En el apartado de discos elegimos un SSD standard

Validation passed

Basics Disks Networking Management Monitoring Advanced Tags **Review + create**

Cost given below is an estimate and not the final price. Please use [Pricing calculator](#) for all your pricing needs.

Price

1 X Standard D2s v3 by Microsoft **0.1880 USD/hr**

Subscription credits apply

[Terms of use](#) [Privacy policy](#)

Pricing for other VM sizes

TERMS

By clicking "Create", I (a) agree to the legal terms and privacy statement(s) associated with the Marketplace offering(s) listed above; (b) authorize Microsoft to bill my current payment method for the fees associated with the offering(s), with the same billing frequency as my Azure subscription; and (c) agree that Microsoft may share my contact, usage and transactional information with the provider(s) of the offering(s) for support, billing and other transactional activities. Microsoft does not provide rights for third-party offerings. See the [Azure Marketplace Terms](#) for additional details.

Create < Previous Next > Download a template for automation Give feedback

18°C Soleado 10:42 AM 3/29/2023

Ahora procedemos a crear nuestra maquina virtual

| Name          | Type            | Subscription       | Resource group      | Location  | Status  | Operating system | Size            | Public IP address |
|---------------|-----------------|--------------------|---------------------|-----------|---------|------------------|-----------------|-------------------|
| T3-2020630578 | Virtual machine | Azure for Students | T3-2020630578_group | West US 2 | Running | Windows          | Standard_D2s_v3 | 20.171.50.70      |

< Previous Page 1 of 1 Next > Give feedback

18°C Soleado 10:43 AM 3/29/2023

Aqui podemos ver nuestra maquina creada la cual podemos empezar a usar

## Configuracion de puertos

Virtual machines

T3-2020630578 | Networking

IP configuration: Ipconfig1 (Primary)

Network Interface: T3-2020630578-vnet/default

Effective security rules Troubleshoot VM connection issues Topology

Accelerated networking: Enabled

Inbound port rules Outbound port rules Application security groups Load balancing

Network security group T3-2020630578-nsg (attached to network interface: T3-2020630578-763)

| Priority | Name                          | Port | Protocol | Source            | Destination    |
|----------|-------------------------------|------|----------|-------------------|----------------|
| 300      | RDP                           | 3389 | TCP      | Any               | Any            |
| 65000    | AllowVnetInBound              | Any  | Any      | VirtualNetwork    | VirtualNetwork |
| 65001    | AllowAzureLoadBalancerInBound | Any  | Any      | AzureLoadBalancer | Any            |
| 65500    | DenyAllInBound                | Any  | Any      | Any               | Any            |

Add inbound port rule

Ahora en este apartado agregaremos una regla en el puerto 5000 en la cual abriremos el puerto 5000

Add inbound security rule

Source: Any

Source port ranges: \*

Destination: Any

Service: Custom

Destination port ranges: 5000:8080

Protocol: UDP

Action: Allow

Priority: 310

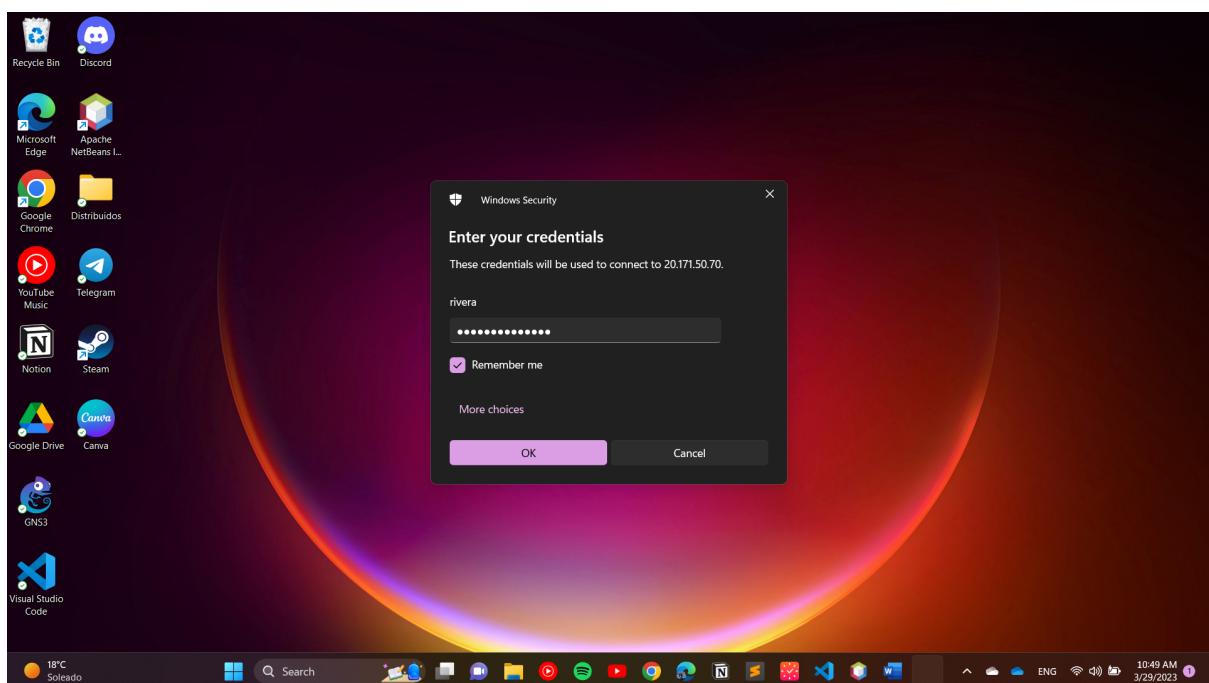
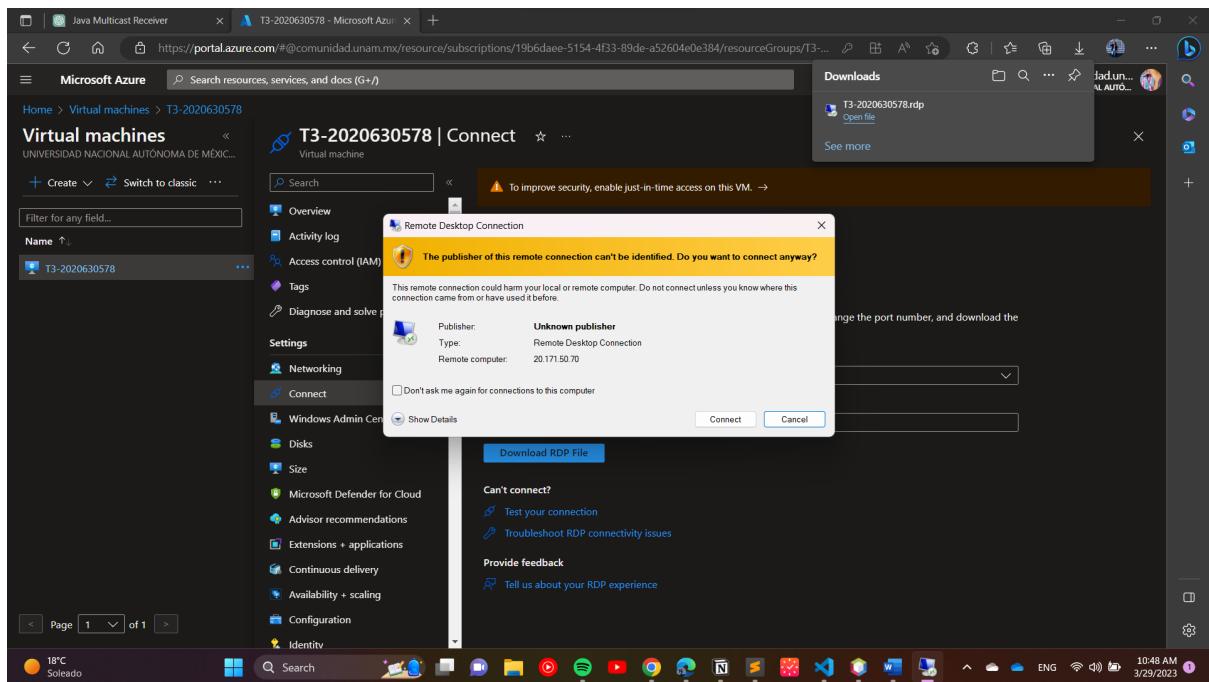
Procedemos a crear y aplicar nuestra regla

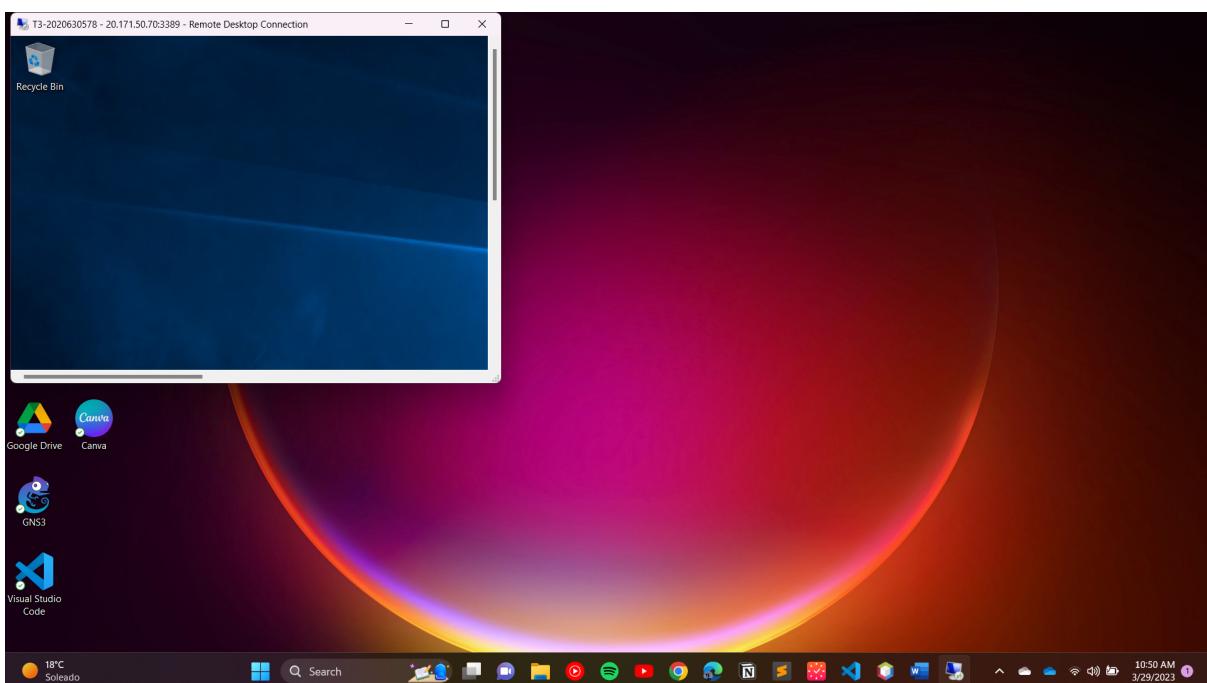
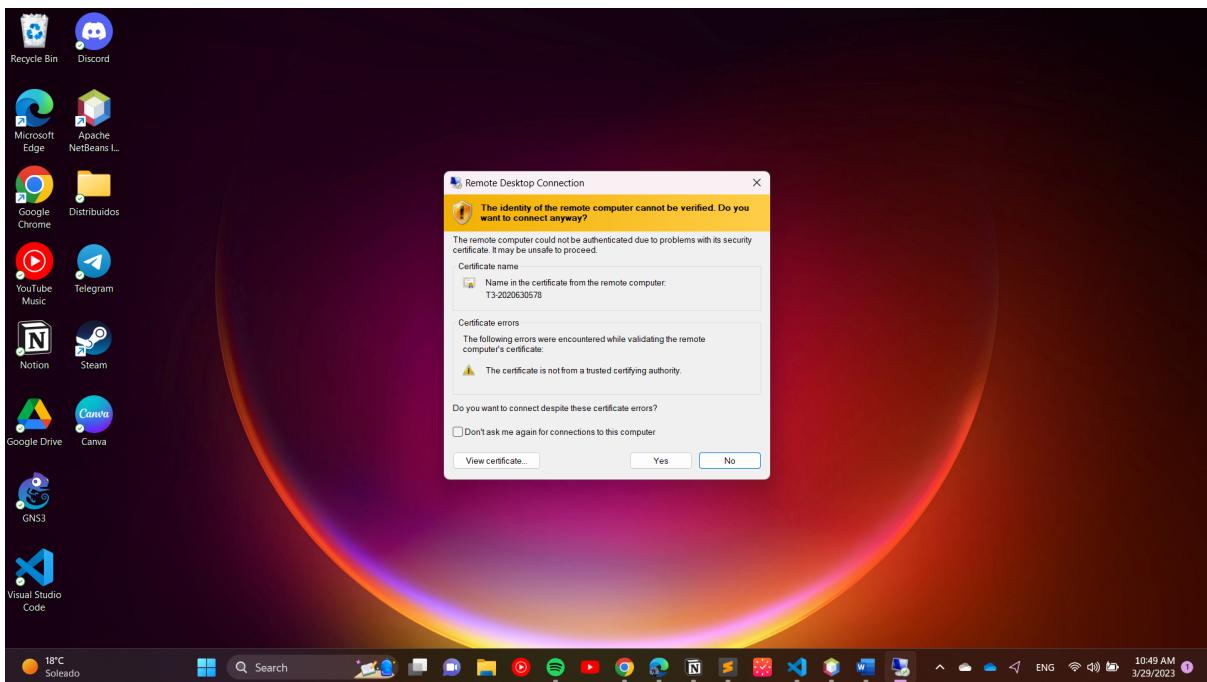
The screenshot shows the Microsoft Azure portal interface. On the left, the navigation pane is visible with 'Virtual machines' selected. In the center, a detailed view of a virtual machine named 'T3-2020630578' is shown under the 'Networking' tab. A modal window titled 'Add inbound security rule' is open on the right, allowing the configuration of a new network rule. The 'Source' dropdown is set to 'Any'. The 'Source port ranges' field contains a single asterisk (\*). The 'Destination' dropdown is also set to 'Any'. The 'Service' dropdown is set to 'Custom' and has the value '5000:8080'. The 'Protocol' dropdown is set to 'UDP'. The 'Action' dropdown is set to 'Allow'. The 'Priority' dropdown is set to '310'. Below the main configuration area, there is a section for 'Need help?' with links to 'Understand Azure load balancing' and buttons for 'Add' and 'Cancel'.

## Conexión a la maquina virtual

This screenshot shows the 'Connect' blade for the same virtual machine 'T3-2020630578'. The blade includes sections for 'RDP', 'SSH', and 'Bastion'. The 'RDP' section is active and displays a message encouraging the user to enable just-in-time access. It lists three prerequisites: 'Checking network security group for inbound access from your client's IP address.', 'The VM's network interface has a Public IP address.', and 'The VM is running.' Below this, it provides instructions to connect via RDP by selecting an IP address and port number. The 'IP address' dropdown is set to 'Public IP address (20.171.50.70)' and the 'Port number' input field is set to '3389'. A 'Download RDP File' button is available. At the bottom, there are troubleshooting links for 'Test your connection' and 'Troubleshoot RDP connectivity issues'.

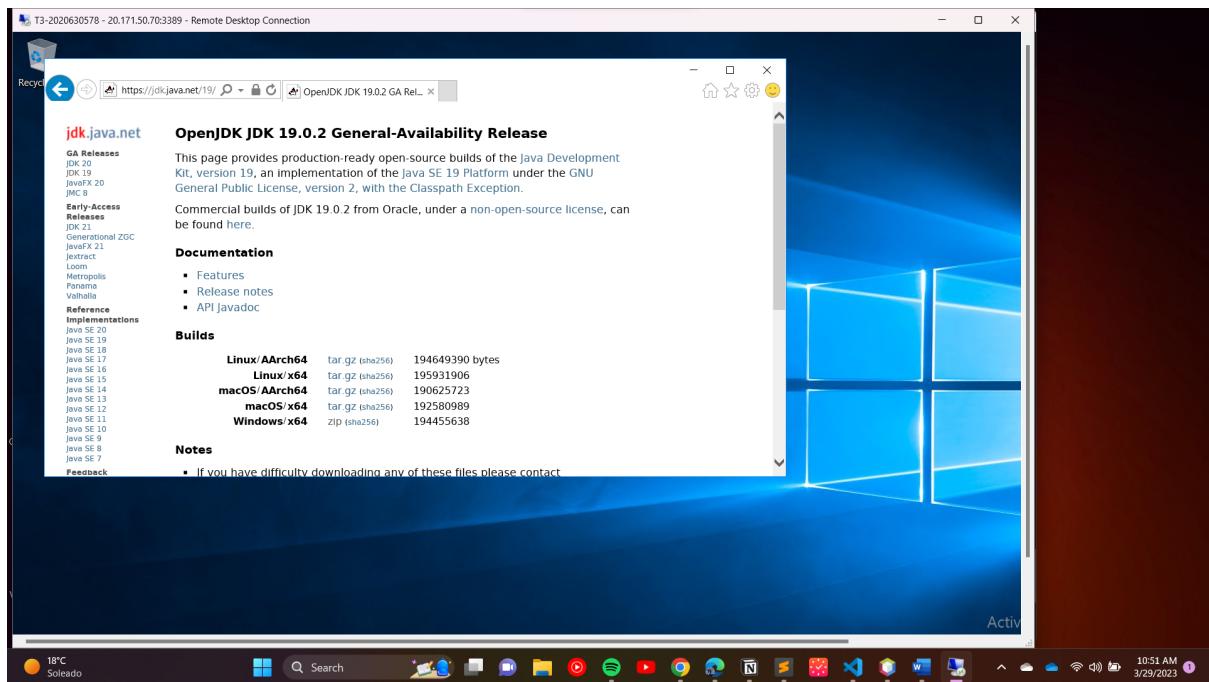
Ahora procedemos a conectarnos a la maquina virtual , descargamos el archivo RDP y lo abrimos



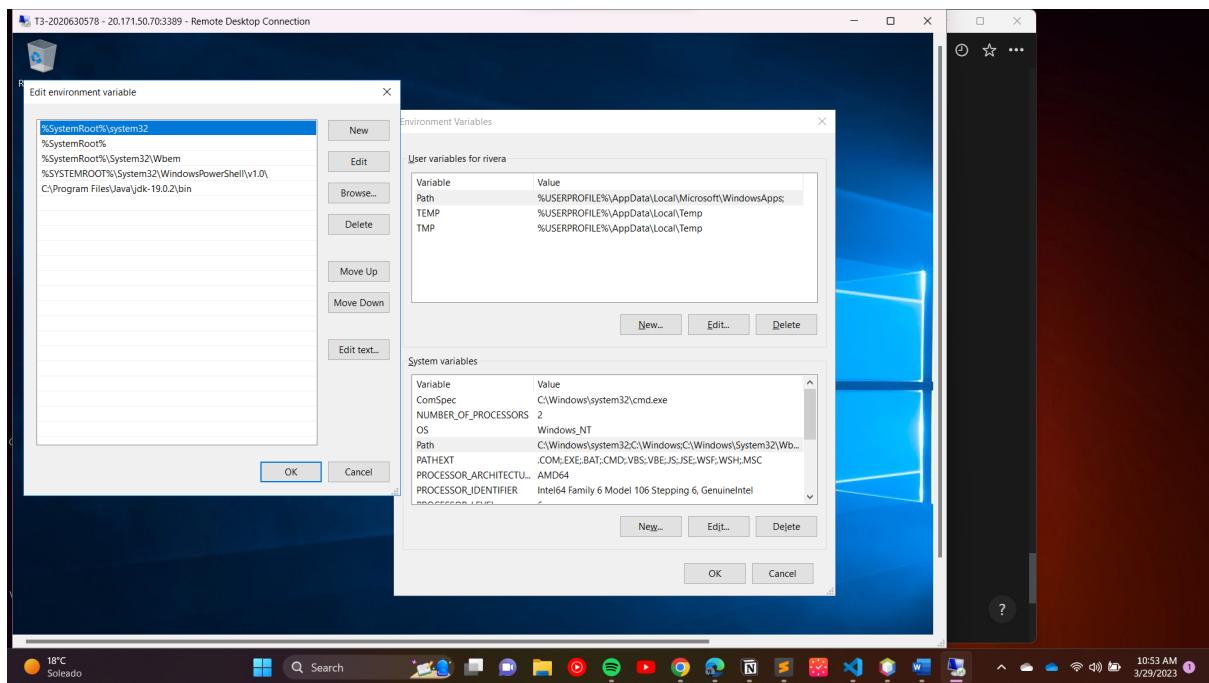


Listo tendremos abierto nuestro entorno

## Instalación de java en la maquina virtual



Descargaremos y descomprimimos la versión de openjdk 8 , el directorio se pondra en algun directorio de preferencia pero debera establecerse como el directorio en el cual se estableceran las variables de entorno



Ahora se configuran las variables de entorno de la siguiente manera.

Ahora esta nuestra maquina virtual lista para trabajar , solo deberemos abrir un power shell y trabajar directamente ahí.

## Desarrollo

### Modelado del sistema con chat GPT

El sistema se realizo en 2 partes , para despues implementar los tres programas en una sola clase para su ejecucion el chat fue el siguiente:

Vamos a desarrollar un sistema en java

#### **PARTE I**

Crea una clase estatica la cual herede de Thread, en esta misma clase , el programa en un ciclo infinito recibira los

mensajes enviados (a través de sockets multicast) al grupo 239.0.0.0 a través del puerto 5000 y se desplegará en la pantalla.

- Claro, podemos comenzar a trabajar en ello. Primero, vamos a crear la clase estática que herede de Thread y que sea capaz de recibir los mensajes enviados a través de sockets multicast al grupo 239.0.0.0 en el puerto 5000:

```
javaCopy code
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;

public class MulticastReceiver extends Thread {

    @Override
    public void run() {
        try {
            // Creamos un socket multicast en el puerto 5000
            MulticastSocket socket = new MulticastSocket(5000);
            // Nos unimos al grupo 239.0.0.0
            InetAddress group = InetAddress.getByName("239.0.0.0");
            socket.joinGroup(group);

            // Ciclo infinito para recibir los mensajes
            while (true) {
                // Creamos un paquete para recibir los datos
                byte[] buffer = new byte[1000];
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
                socket.receive(packet);
                // Convertimos los datos recibidos a un string
                String message = new String(packet.getData(), 0, packet.getLength());
                // Mostramos el mensaje en la pantalla
                System.out.println(message);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Ahora, podemos crear una clase principal que instancie y ejecute la clase MulticastReceiver:

```
javaCopy code
public class Main {

    public static void main(String[] args) {
        // Creamos una instancia de MulticastReceiver y la iniciamos
        MulticastReceiver receiver = new MulticastReceiver();
        receiver.start();
    }
}
```

De esta manera, al ejecutar el programa, la clase MulticastReceiver comenzará a recibir los mensajes enviados al grupo 239.0.0.0 en el puerto 5000 y los mostrará en la pantalla de manera continua.

## PARTE II

Crea dos métodos; El primer método tendrá que enviar y el otro recibir los mensajes de los sockets:

El método de enviarMensaje recibirá como parámetros el buffer de bytes(byte[]), la ip (String), y el puerto(int). Este método no retornará nada.

El método de recibirMensaje recibirá como parámetro: el socket multicast(MulticastSocket), la longitud del mensaje (int). Este método retornará el mensaje (byte[])

- Por supuesto, podemos crear los dos métodos solicitados en la clase `MulticastReceiver`. Para el método `enviarMensaje`, podríamos agregar el siguiente código:

```
javaCopy code
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public static void enviarMensaje(byte[] mensaje, String ip, int puerto) {
```

```

try {
    // Crear socket y paquete para enviar datos
    DatagramSocket socket = new DatagramSocket();
    InetAddress address = InetAddress.getByName(ip);
    DatagramPacket packet = new DatagramPacket(mensaje, mensaje.length, address, puerto);

    // Enviar paquete
    socket.send(packet);

    // Cerrar socket
    socket.close();
} catch (Exception e) {
    e.printStackTrace();
}
}

```

Este método recibe como parámetros el buffer de bytes que se desea enviar, la dirección IP a la cual se enviará el mensaje y el puerto por el cual se realizará la comunicación. Dentro del método, se crea un socket y un paquete para enviar los datos, y se envía el paquete a la dirección y puerto especificados. Finalmente, se cierra el socket.

Para el método `recibirMensaje`, podemos agregar lo siguiente:

```

javaCopy code
public static byte[] recibirMensaje(MulticastSocket socket, int longitud) {
    try {
        // Crear paquete para recibir datos
        byte[] buffer = new byte[longitud];
        DatagramPacket packet = new DatagramPacket(buffer, buffer.length);

        // Recibir paquete
        socket.receive(packet);

        // Devolver mensaje recibido
        return packet.getData();
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
}

```

Este método recibe como parámetros el socket multicast y la longitud del mensaje que se espera recibir. Dentro del método, se crea un paquete para recibir los datos y se llama al método `receive()` del socket para recibir el paquete. Luego, se devuelve el buffer de bytes del paquete recibido. En caso de que ocurra algún error, se muestra la traza de la excepción y se devuelve `null`.

Es importante mencionar que ambos métodos deben ser estáticos, ya que no necesitan ser invocados en una instancia particular de la clase `MulticastReceiver`.

En base a estos códigos se desarrolló el programa implementado con algunos cambios ya que el código generado por Chat GPT presentó algunas inconsistencias

## Código fuente

```

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.InetSocketAddress;
import java.net.MulticastSocket;
import java.net.NetworkInterface;
import java.util.Scanner;

public class ProgramaPrincipal {
    static class MyThread extends Thread {
        public void run() {
            while(true) {
                try {
                    MulticastSocket socket = new MulticastSocket(5000);
                    InetAddress group = InetAddress.getByName("239.0.0.0");
                    NetworkInterface ni = NetworkInterface.getByName("Ethernet"); // <--- el nombre de la interfaz de red

```

```

        socket.joinGroup(new InetSocketAddress(group, 5000), ni);
        byte[] buf = recibirMensaje(socket, 1024); // <-- Aquí se queda esperando, los parametros son el socket
byte[] name = recibirNombre(socket,1024);
        System.out.print(new String(name,"ISO-8859-1") +"\b ---> " + new String(buf, "ISO-8859-1"));
    } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
    }
}
}

public static void enviarMensaje(byte[] buffer, String ip, int puerto) throws IOException {
try {
    DatagramSocket socket = new DatagramSocket();
    InetAddress address = InetAddress.getByName(ip);
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, puerto);
    socket.send(packet);
    socket.close();
} catch (IOException e) {
    System.out.println("Error al enviar el mensaje: " + e.getMessage());
}
}

public static byte[] recibirMensaje(MulticastSocket socket, int longitudMensaje) throws IOException {
byte[] buffer = new byte[longitudMensaje];
try {
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
    socket.receive(packet);
    return packet.getData();
} catch (IOException e) {
    System.out.println("Error al recibir el mensaje: " + e.getMessage());
}
return null;
}

}

//-----
public static byte[] recibirNombre(MulticastSocket socket,int longitudMensaje) throws IOException {
byte[] buffer = new byte[longitudMensaje];
try {
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length);
    socket.receive(packet);
    return packet.getData();
} catch (IOException e) {
    System.out.println("Error al recibir el mensaje: " + e.getMessage());
}
return null;
}

}

public static void enviarNombre(byte[] buffer, String ip, int puerto) throws IOException {
try {
    DatagramSocket socket = new DatagramSocket();
    InetAddress address = InetAddress.getByName(ip);
    DatagramPacket packet = new DatagramPacket(buffer, buffer.length, address, puerto);
    socket.send(packet);
    socket.close();
} catch (IOException e) {
    System.out.println("Error al enviar el mensaje: " + e.getMessage());
}
}

//-----

public static void main(String[] args) throws IOException {
MyThread hilo = new MyThread();
hilo.start();

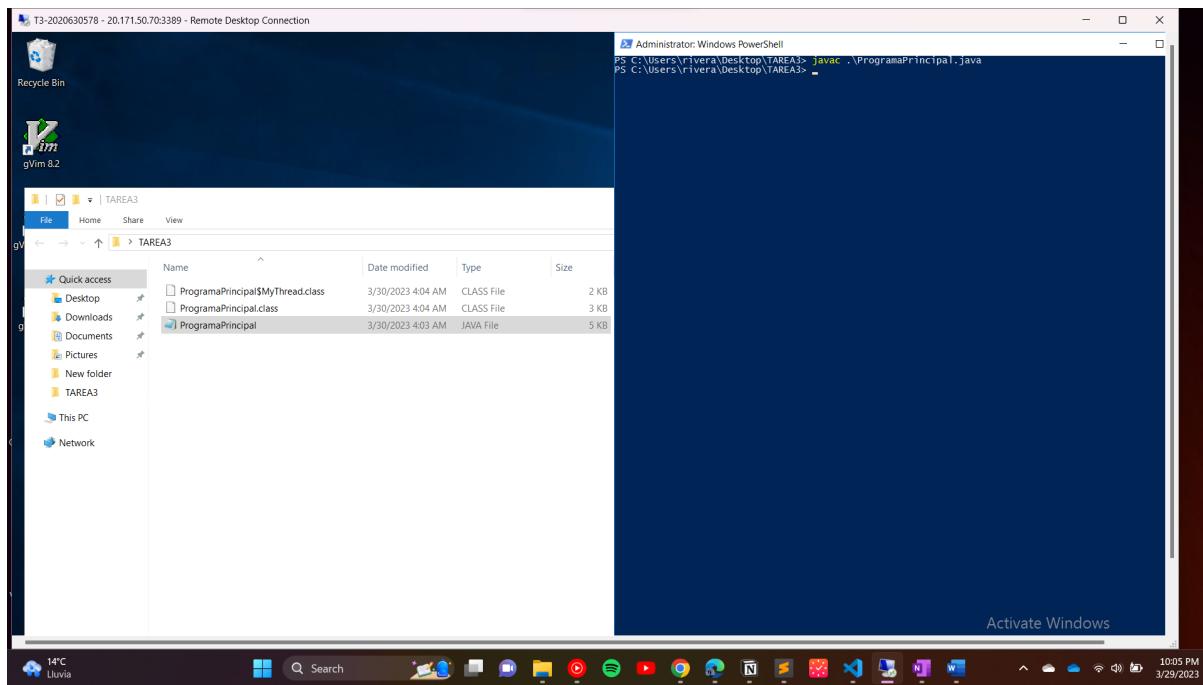
Scanner scanner = new Scanner(System.in);
String user = args[0];
String ip = "239.0.0.0";
int puerto = 5000;
System.out.println("Bienvenido " + user);
System.out.println("Ingresa tu mensaje");
while(true){
    String mensaje = scanner.nextLine();
    byte[] buffer = mensaje.getBytes();
byte[] name = user.getBytes();
    enviarMensaje(buffer, ip, puerto);
    enviarNombre(name,ip, puerto);
}
}
}

```

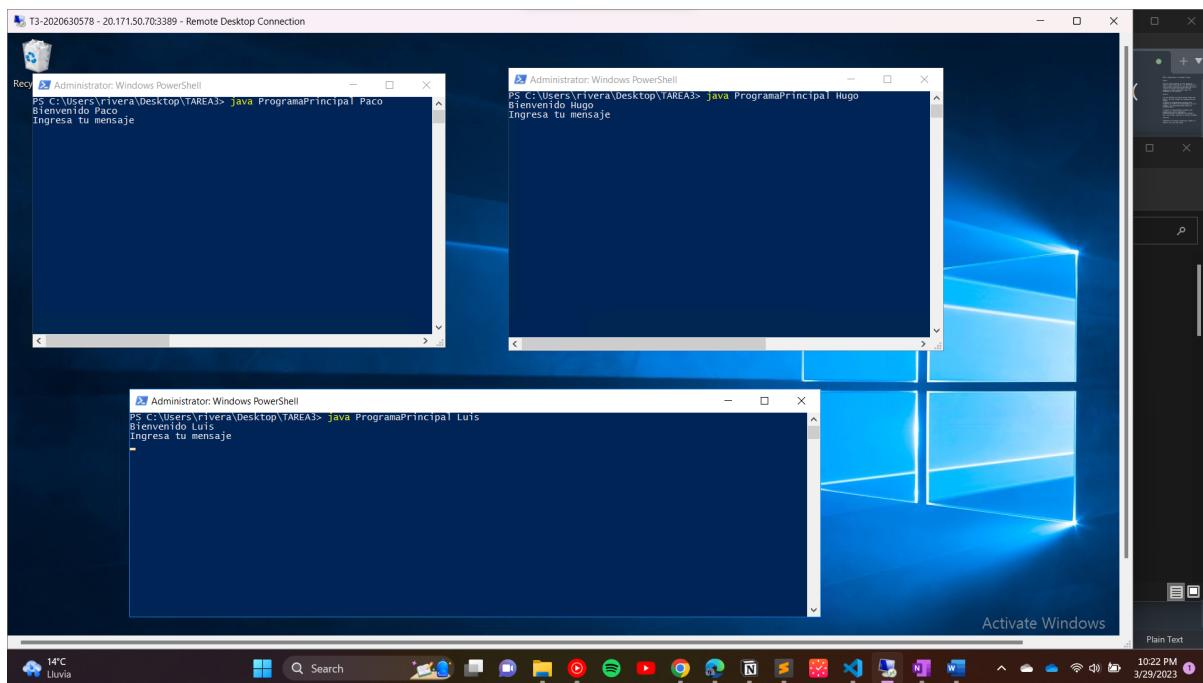
## Resultados

### Compilacion & Ejecucion

Ahora procedemos a compilar nuestro Código fuente en nuestra maquina virtual como se muestra a continuación.

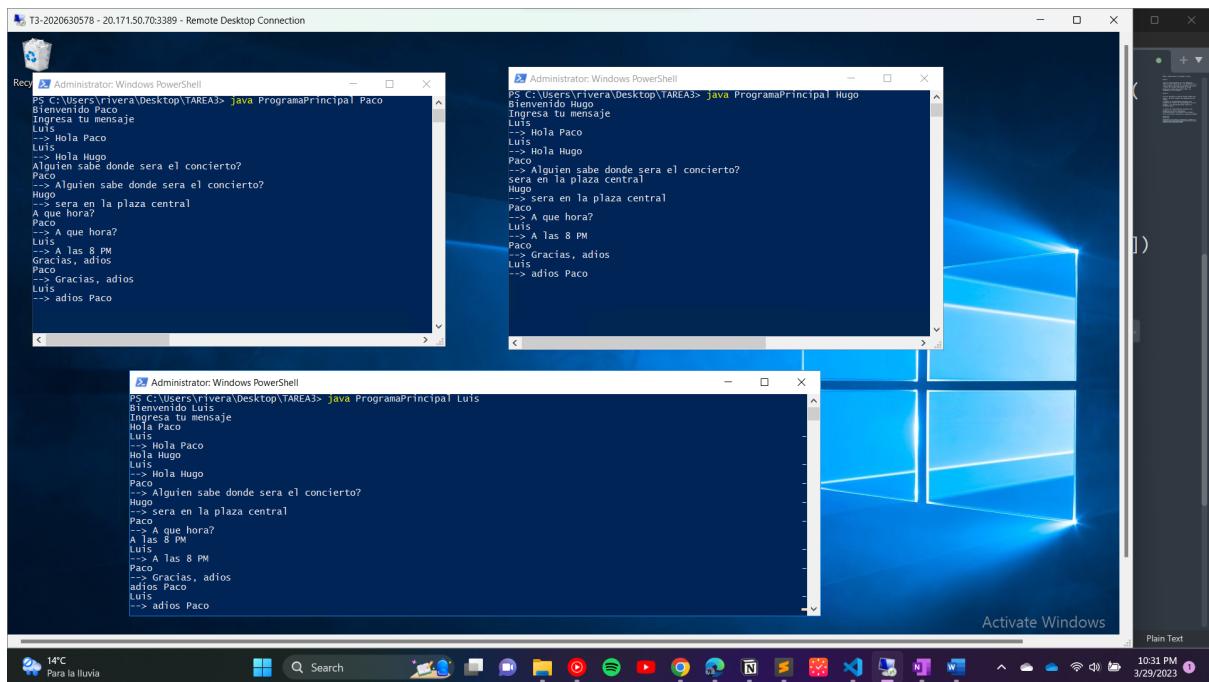


Ahora se mostrara como se ejecuta el programa en 3 ventanas distintas en la maquina virtual.



### Ejecución de la conversación

A continuación se escribirá la conversación indicada en la asignación:



Se adjunta la conversación como evidencia del correcto funcionamiento del programa

## Conclusiones

Después de desarrollar y probar el sistema en Java que permite enviar y recibir mensajes a través de sockets multicast, se puede concluir que se trata de una tecnología útil para implementar aplicaciones distribuidas que requieren una comunicación en tiempo real entre múltiples nodos.

La implementación del sistema resultó relativamente sencilla, gracias a las clases y métodos provistos por Java para manejar sockets y multicast. En particular, se utilizaron las clases `MulticastSocket` y `DatagramPacket` para recibir y enviar mensajes, respectivamente.

Una ventaja importante de utilizar multicast es que permite la transmisión de mensajes a múltiples destinatarios simultáneamente, lo que lo hace adecuado para aplicaciones que requieren una comunicación en grupo, como chats o videoconferencias. Además, se puede utilizar la misma dirección IP multicast para diferentes grupos de trabajo, lo que facilita la configuración y administración del sistema.

En general, se puede afirmar que el uso de sockets multicast en Java es una solución eficiente y práctica para implementar sistemas distribuidos que requieren una comunicación en tiempo real entre múltiples nodos. Este sistema puede ser mejorado y adaptado para aplicaciones específicas, y se espera que sea de utilidad en futuros proyectos de programación distribuida.