



Instituto Politécnico Nacional



Escuela Superior de Cómputo

Desarrollo de Sistemas Distribuidos

Tarea 5. Multiplicación de matrices utilizando objetos distribuidos

López Sánchez Kevin Ian

Grupo: 4CV12

Profesor: Pineda Guerrero Carlos

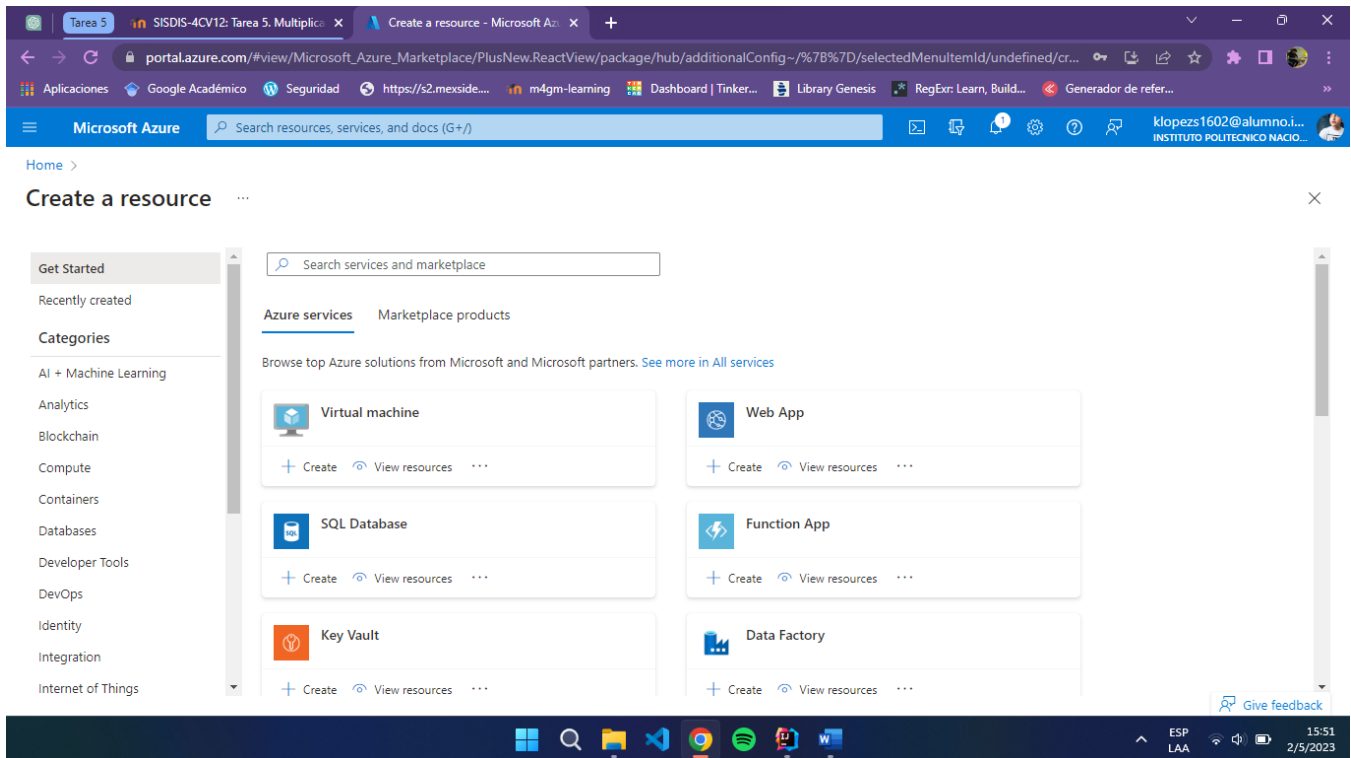
02 de mayo de 2023

Desarrollo

Creación de máquina virtual de Ubuntu Server en Azure

Para realizar esta práctica es necesario crear tres 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”.



2. Seleccionamos el grupo de recursos o creamos uno nuevo, escribimos el nombre de nuestra máquina virtual y seleccionamos la opción de Ubuntu Server 20.04 LTS, elegimos 1 Gb para la memoria RAM de nuestra máquina.

Microsoft Azure portal showing the "Create a virtual machine" wizard. The "Basics" tab is selected. The "Project details" section shows the subscription "Azure for Students" and the resource group "(New) ESCOM". The "Instance details" section is partially visible, showing the "Review + create" button and navigation links.

Microsoft Azure portal showing the "Create a virtual machine" wizard. The "Instance details" section is expanded, showing the following configuration:

- Virtual machine name: T5-2020630205-0
- Region: (US) East US
- Availability options: No infrastructure redundancy required
- Security type: Standard
- Image: Ubuntu Server 20.04 LTS - x64 Gen2
- VM architecture: x64
- Run with Azure Spot discount: ☐
- Size: Standard_B1s - 1 vcpu, 1 GiB memory (\$7.59/month)

The "Review + create" button is visible at the bottom of the wizard.

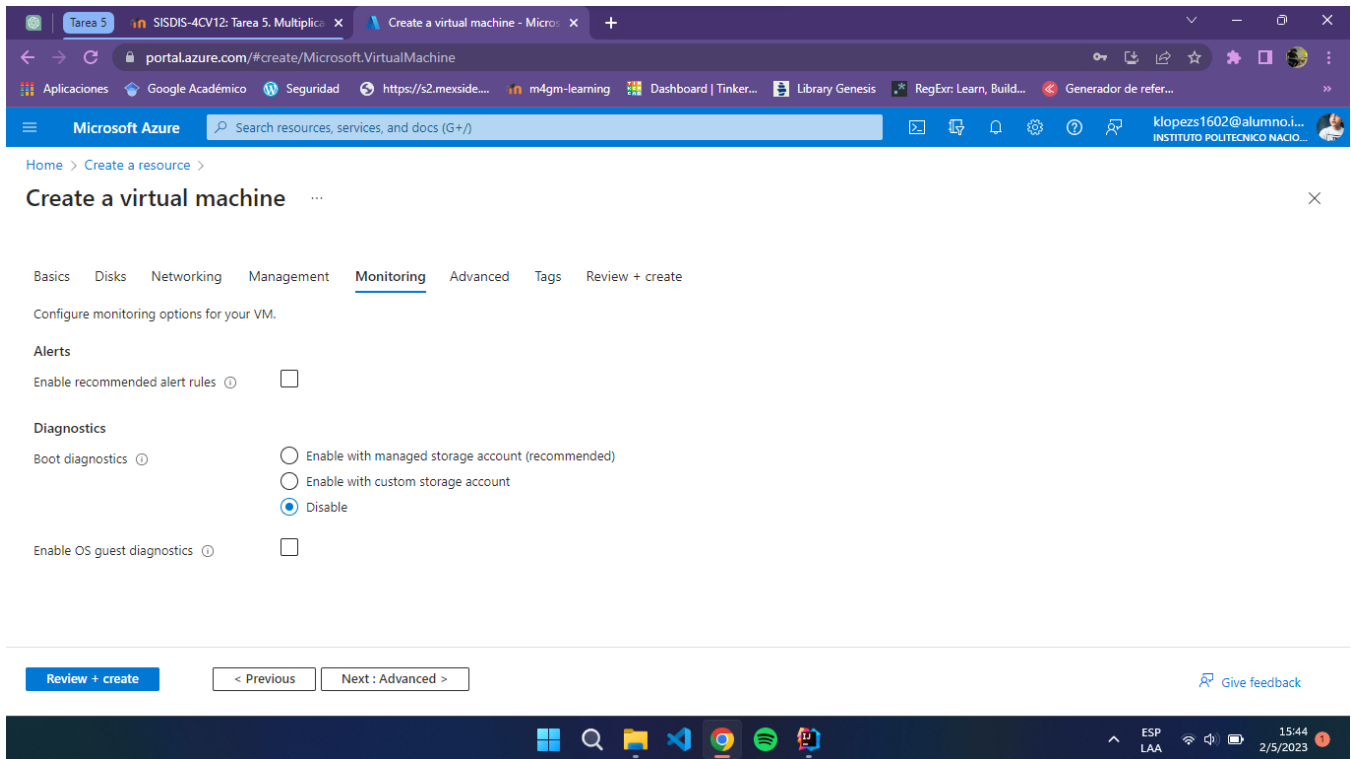
3. Ingresamos el usuario y la contraseña para ingresar posteriormente. Seleccionamos el puerto de entrada por el que accederemos a la máquina virtual, en este caso el de SSH.

Microsoft Azure portal showing the "Create a virtual machine" wizard. The "Administrator account" section is active, showing authentication type (Password selected), username (kubuntu), and password fields. The "Inbound port rules" section shows "Allow selected ports" selected, with "SSH (22)" selected in the dropdown. The "Review + create" button is visible.

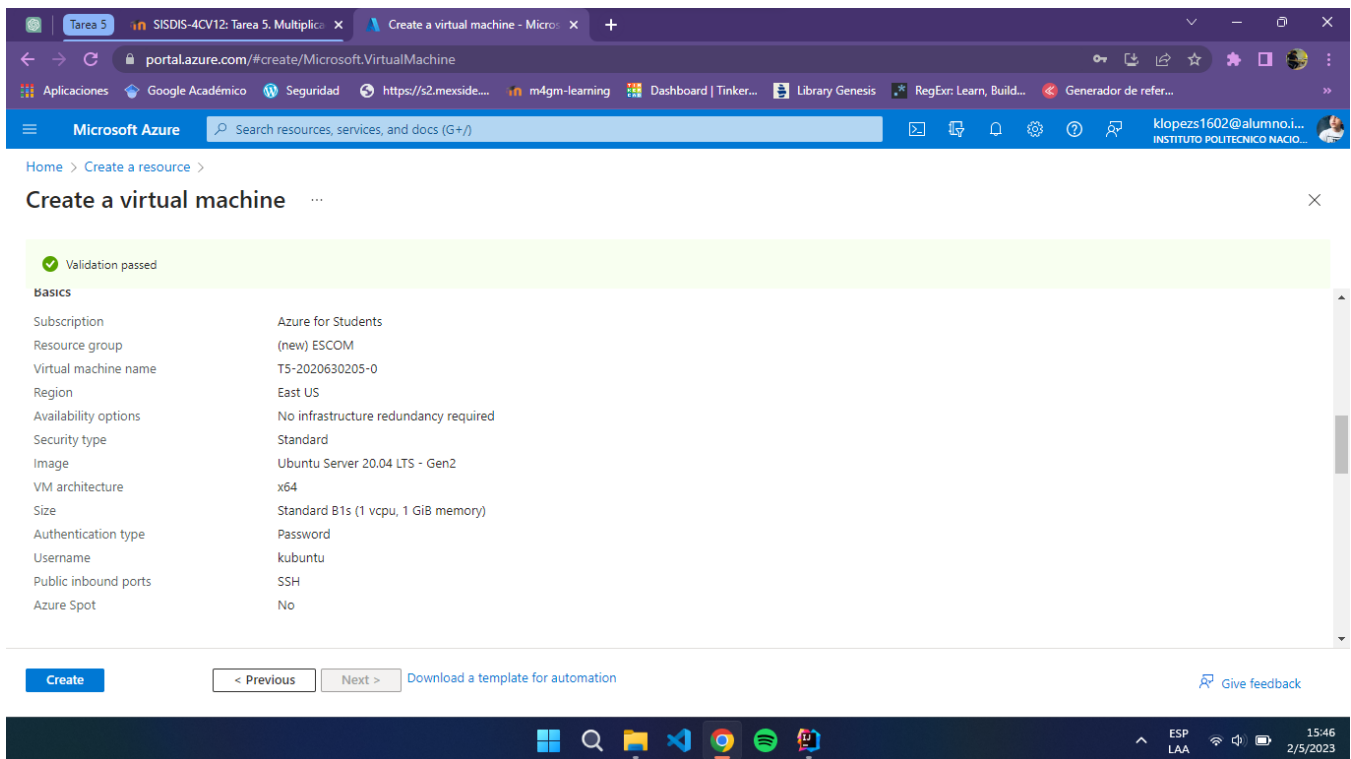
4. Seleccionamos como almacenamiento un HDD estándar

Microsoft Azure portal showing the "Create a virtual machine" wizard, specifically the "Disks" tab. The "OS disk" section shows "Default size (30 GiB)" and "Standard HDD (locally-redundant storage)" selected. A message indicates that encryption at host is not registered for the selected subscription. The "Next: Networking" button is visible.

5. Desactivamos el diagnóstico de inicio.



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



7. Esperamos a que nuestra máquina termine de crearse.

Microsoft Azure portal showing the deployment details for 'CreateVm-canonical.0001-com-ubuntu-server-focal-2-20230502153828'. The deployment is in progress.

Deployment name: CreateVm-canonical.0001-com-ubuntu-server-f... Start time: 5/2/2023, 3:46:44 PM
Subscription: Azure for Students Correlation ID: 7cf21d6b-6d49-4e73-b923-315cae
Resource group: ESCOM

Deployment details

Resource	Type	Status	Operation details
T5-2020630205-0-vnet	Microsoft.Network/virtual...	Created	Operation details
T5-2020630205-0-ip	Microsoft.Network/public...	OK	Operation details
T5-2020630205-0-nsg	Microsoft.Network/networ...	OK	Operation details

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 >](#)

8. Una vez creada, probamos la conexión.

Microsoft Azure portal showing the deployment details for 'CreateVm-canonical.0001-com-ubuntu-server-focal-2-20230502153828'. The deployment is complete.

Deployment name: CreateVm-canonical.0001-com-ubuntu-server-f... Start time: 5/2/2023, 3:46:44 PM
Subscription: Azure for Students Correlation ID: 7cf21d6b-6d49-4e73-b923-315cae
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](#)

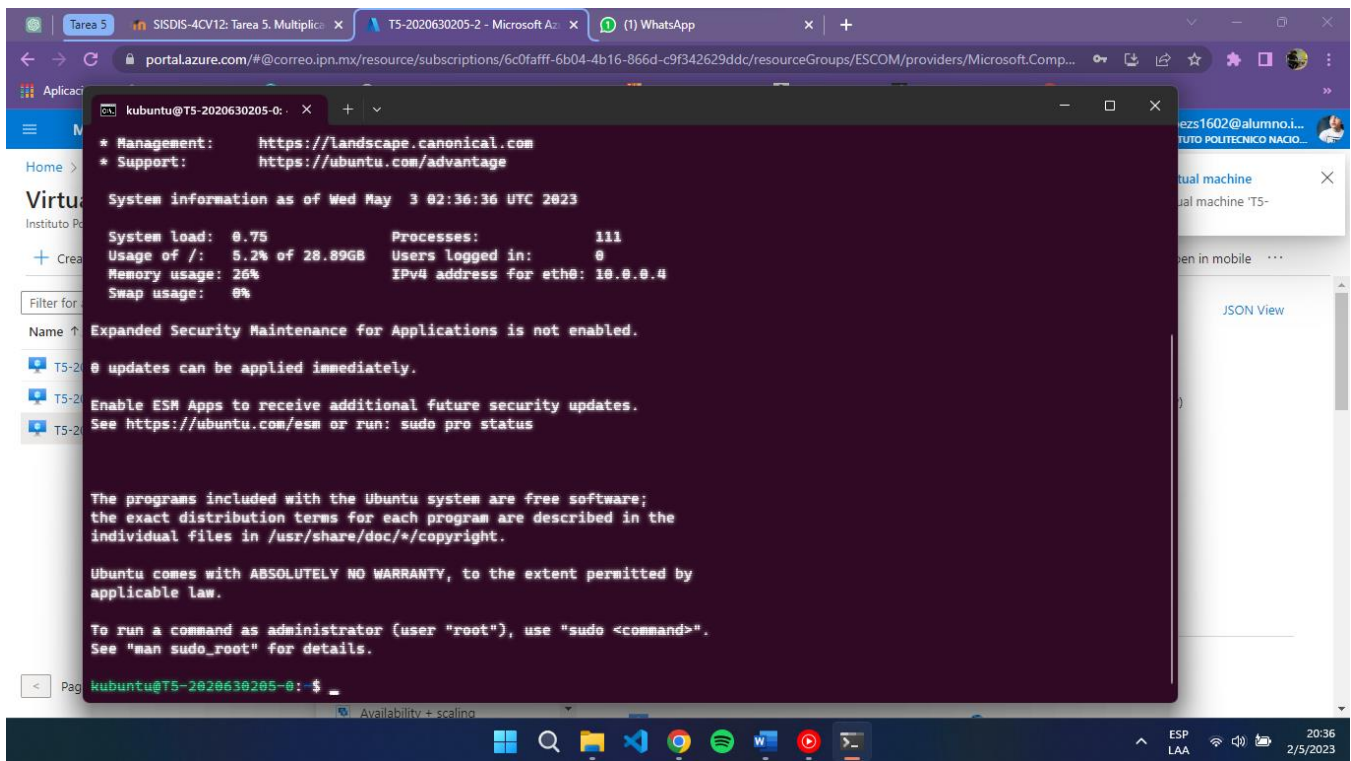
Give feedback
[Tell us about your experience with deployment](#)

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

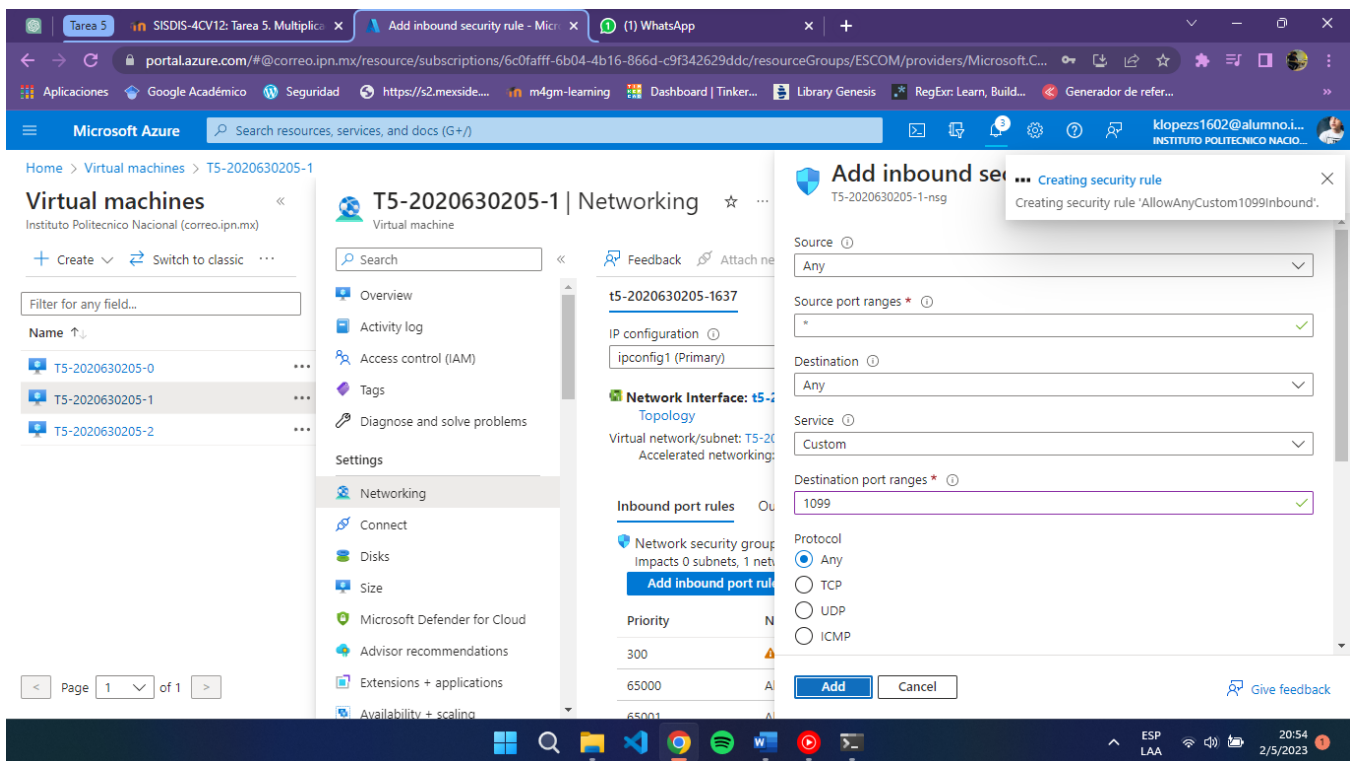
Microsoft Defender for Cloud
Secure your apps and infrastructure
[Go to Microsoft Defender for Cloud >](#)

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

Work with an expert



- Para realizar la implementación debemos crear una regla de puerto de entrada para abrir el puerto 1099 por donde iniciaremos RMRegistry.



Conversación con Chat GPT

Yo:

Necesito crear una implementación en java utilizando threads y RMI en una arquitectura cliente servidor de multiplicación de matrices rectangulares en donde la matriz A tiene dimension $M \times N$, B $N \times M$ y $C = A \times B$. Utilizando el siguiente programa como base, se deben dividir las matrices A y B en 9 partes iguales, por lo tanto la matriz C se divide en 81 partes. Se deben ejecutar 3 instancias del servidor, cada instancia recibe la tercera parte de la matriz A y Bt del cliente para calcular la tercera parte de los elementos de la matriz C mediante el método `multiplicarmatrices()` y enviarlo de nuevo al cliente, el cliente debe recibir cada parte de manera paralela por threads y al ultimo calculará el checksum que será la suma de todos los elementos de la matriz C.

```
public class MultMatrixRMI {

    public static void main(String[] args) {

        if (args.length != 2) {

            System.out.println("Debe ingresar 2 argumentos: N y M");

            System.exit(1);

        }

        int N = Integer.parseInt(args[0]);

        int M = Integer.parseInt(args[1]);

        // Initialize matrices A and B

        float[][] A = new float[N][M];

        float[][] B = new float[M][N];

        for (int i = 0; i < N; i++) {

            for (int j = 0; j < M; j++) {
```



```

        A[i][j] = 2 * i + 3 * j;
    }
}
for (int i = 0; i < M; i++) {
    for (int j = 0; j < N; j++) {
        B[i][j] = 3 * i - 2 * j;
    }
}

//Transpose Matrix B in Bt
float[][] Bt = new float[N][M];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        Bt[i][j] = B[j][i];
    }
}

// Multiply matrices A and Bt
float[][] C = new float[N][N];
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        for (int k = 0; k < M; k++) {
            C[i][j] += A[i][k] * Bt[j][k];
        }
    }
}

```

```

// Calculate checksum of matrix C

double checksum = 0.0;

for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        checksum += C[i][j];
    }
}

// Print result matrix C

/* for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        System.out.print(C[i][j] + " ");
    }
    System.out.println();
}*/

// Print checksum of matrix C

System.out.println("Checksum: " + checksum);

}

}

```

ChatGPT:

Para implementar la solución utilizando RMI y threads, primero necesitas crear una interfaz remota, luego implementarla en el servidor y finalmente modificar el cliente para que se comunique con el servidor. Aquí tienes un ejemplo de cómo hacerlo:

Crear la interfaz remota MatrixOperations:

```
import java.rmi.Remote;
```

```
import java.rmi.RemoteException;
```

```
public interface MatrixOperations extends Remote {  
    float[][] multiplyMatrices(float[][] A, float[][] Bt) throws RemoteException;  
}
```

Implementar la interfaz en el servidor MatrixServer:

```
import java.rmi.RemoteException;
```

```
import java.rmi.server.UnicastRemoteObject;
```

```
public class MatrixServer extends UnicastRemoteObject implements MatrixOperations {
```

```
    protected MatrixServer() throws RemoteException {  
        super();  
    }
```

```
    @Override
```

```
    public float[][] multiplyMatrices(float[][] A, float[][] Bt) throws RemoteException {  
        int N = A.length;  
        int M = Bt[0].length;  
        float[][] C = new float[N][M];  
  
        for (int i = 0; i < N; i++) {
```

```

        for (int j = 0; j < N; j++) {
            for (int k = 0; k < M; k++) {
                C[i][j] += A[i][k] * Bt[j][k];
            }
        }
    }
}

```

```

    return C;
}
}

```

Crear el ServerMain para ejecutar el servidor:

```
import java.rmi.registry LocateRegistry;
```

```
import java.rmi.registry.Registry;
```

```

public class ServerMain {

    public static void main(String[] args) {

        try {

            MatrixServer server = new MatrixServer();

            Registry registry = LocateRegistry.createRegistry(1099);

            registry.bind("MatrixServer", server);

            System.out.println("Servidor de matrices listo");

        } catch (Exception e) {

            System.err.println("Error en el servidor: " + e.toString());

            e.printStackTrace();
        }
    }
}

```

```
    }  
}  
}
```

Modificar el cliente MultMatrixRMI para que utilice RMI y threads:

```
import java.rmi.registry.LocateRegistry;
```

```
import java.rmi.registry.Registry;
```

```
public class MultMatrixRMI {
```

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

```
        if (args.length != 2) {
```

```
            System.out.println("Debe ingresar 2 argumentos: N y M");
```

```
            System.exit(1);
```

```
        }
```

```
        int N = Integer.parseInt(args[0]);
```

```
        int M = Integer.parseInt(args[1]);
```

```
        // Initialize matrices A and B
```

```
        // ... (mismo código que antes)
```

```
        // Transpose Matrix B in Bt
```

```
        // ... (mismo código que antes)
```

```

// Divide matrices A and Bt into 9 equal parts
// ... (implementar la lógica de división)

// Create threads to call remote method multiplyMatrices()
// ... (crear threads y llamar al método remoto)

// Calculate checksum of matrix C
// ... (mismo código que antes)

// Print checksum of matrix C
System.out.println("Checksum: " + checksum);
}
}

```

Para dividir las matrices y crear los threads, puedes seguir este ejemplo:

```

// Divide matrices A and Bt into 9 equal parts
int parts = 3;
float[][][] subMatricesA = divideMatrix(A, parts);
float[][][] subMatricesBt = divideMatrix(Bt, parts);

// Create threads to call remote method multiplyMatrices()
Thread[] threads = new Thread[parts * parts];
float[][][] subMatricesC = new float[parts * parts][][];

for (int i = 0; i < parts; i++) {

```

```

for (int j = 0; j < parts; j++) {
    int index = i * parts + j;
    threads[index] = new Thread(() -> {
        try {
            Registry registry = LocateRegistry.getRegistry("localhost");
            MatrixOperations server = (MatrixOperations) registry.lookup("MatrixServer");
            subMatricesC[index] = server.multiplyMatrices(subMatricesA[i], subMatricesBt[j]);
        } catch (Exception e) {
            System.err.println("Error en el cliente: " + e.toString());
            e.printStackTrace();
        }
    });
    threads[index].start();
}
}

```

```

// Join threads and merge subMatricesC into matrix C
float[][] C = new float[N][N];
for (int i = 0; i < parts * parts; i++) {
    try {
        threads[i].join();
        mergeSubMatrix(C, subMatricesC[i], i / parts, i % parts);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

```
}  
}
```

Implementar los métodos divideMatrix y mergeSubMatrix:

```
private static float[][][] divideMatrix(float[][] matrix, int parts) {  
    int rows = matrix.length;  
    int cols = matrix[0].length;  
    int subRows = rows / parts;  
    int subCols = cols / parts;  
    float[][][] subMatrices = new float[parts * parts][][];  
  
    for (int i = 0; i < parts; i++) {  
        for (int j = 0; j < parts; j++) {  
            int index = i * parts + j;  
            subMatrices[index] = new float[subRows][subCols];  
            for (int x = 0; x < subRows; x++) {  
                for (int y = 0; y < subCols; y++) {  
                    subMatrices[index][x][y] = matrix[i * subRows + x][j * subCols + y];  
                }  
            }  
        }  
    }  
  
    return subMatrices;  
}
```



```

private static void mergeSubMatrix(float[][] matrix, float[][] subMatrix, int rowOffset, int
colOffset) {

    int subRows = subMatrix.length;

    int subCols = subMatrix[0].length;

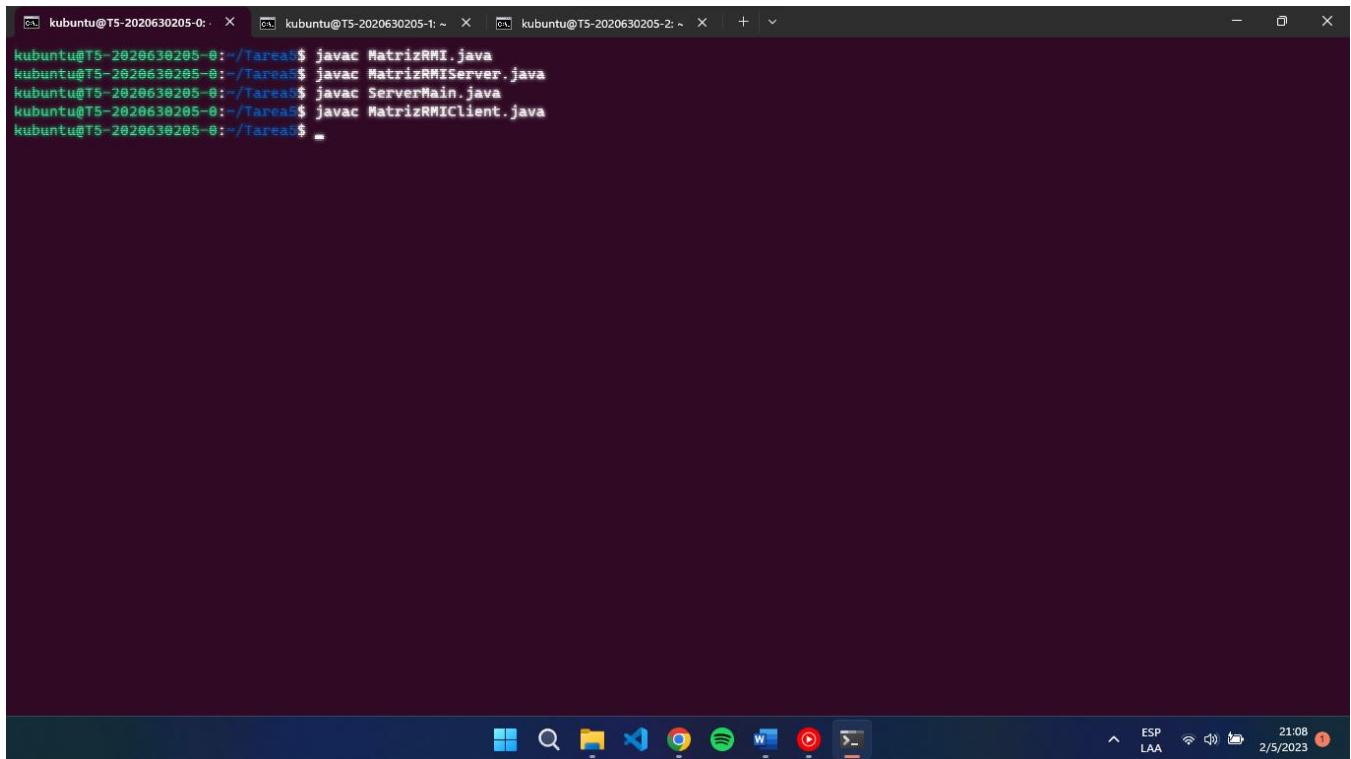
    for (int i = 0; i < subRows; i++) {
        for (int j = 0; j < subCols; j++) {
            matrix[rowOffset * subRows + i][colOffset * subCols + j] = subMatrix[i][j];
        }
    }
}

```

Con estos cambios, tendrás una implementación en Java utilizando threads y RMI en una arquitectura cliente-servidor para la multiplicación de matrices rectangulares. No olvides compilar y ejecutar el servidor antes de ejecutar el cliente.

Implementación

1. Primero debemos compilar los programas que se van a ejecutar en cada máquina virtual de Ubuntu Server, en el caso del nodo 0, compilamos la interfaz, el servidor con su método principal y el cliente.

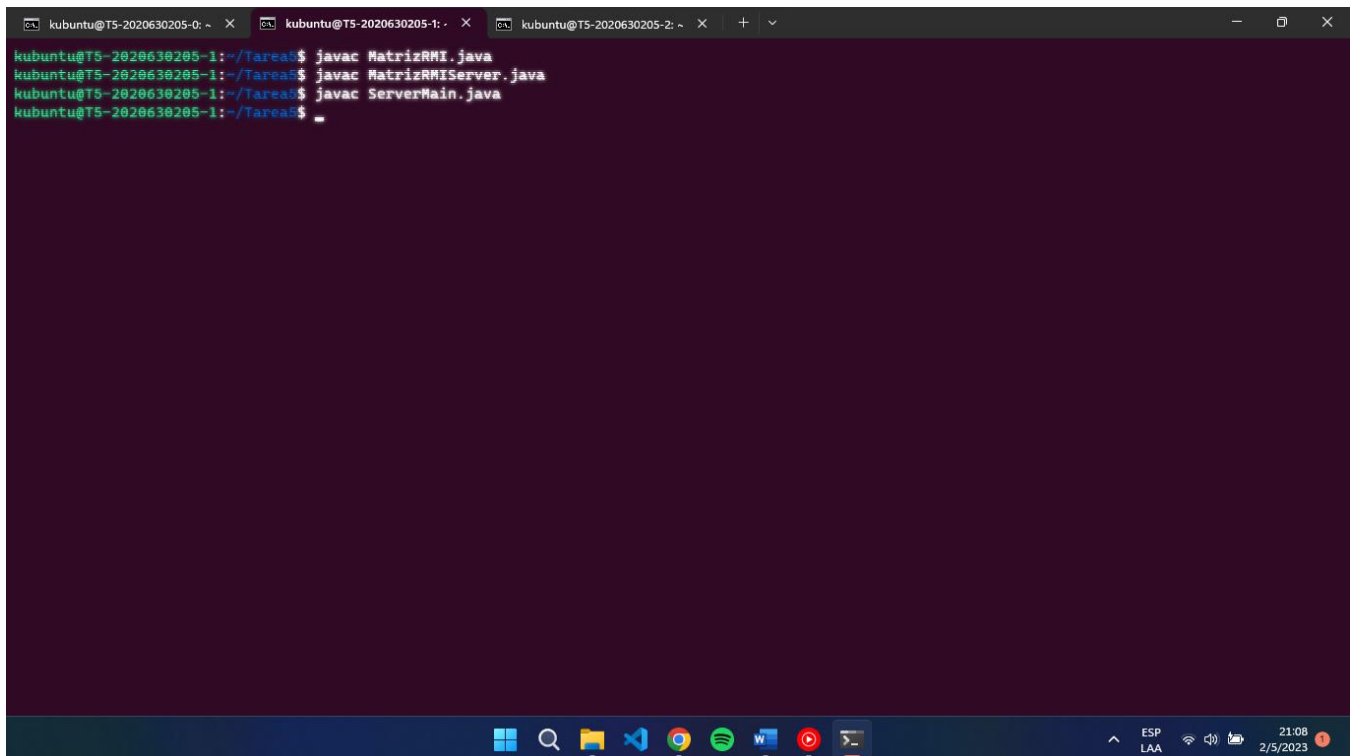


A terminal window with three tabs. The first tab is active and shows the following commands and output:

```
kubuntu@T5-2020630205-0: ~/Tarea5$ javac MatrizRMI.java
kubuntu@T5-2020630205-0: ~/Tarea5$ javac MatrizRMIServer.java
kubuntu@T5-2020630205-0: ~/Tarea5$ javac ServerMain.java
kubuntu@T5-2020630205-0: ~/Tarea5$ javac MatrizRMIClient.java
kubuntu@T5-2020630205-0: ~/Tarea5$ _
```

The terminal has a dark purple background and a Windows taskbar at the bottom with icons for File Explorer, VS Code, Chrome, Spotify, and others. The system tray shows the time as 21:08 on 2/5/2023.

2. En el nodo 1 solo compilamos la interfaz, la clase del servidor y su método principal

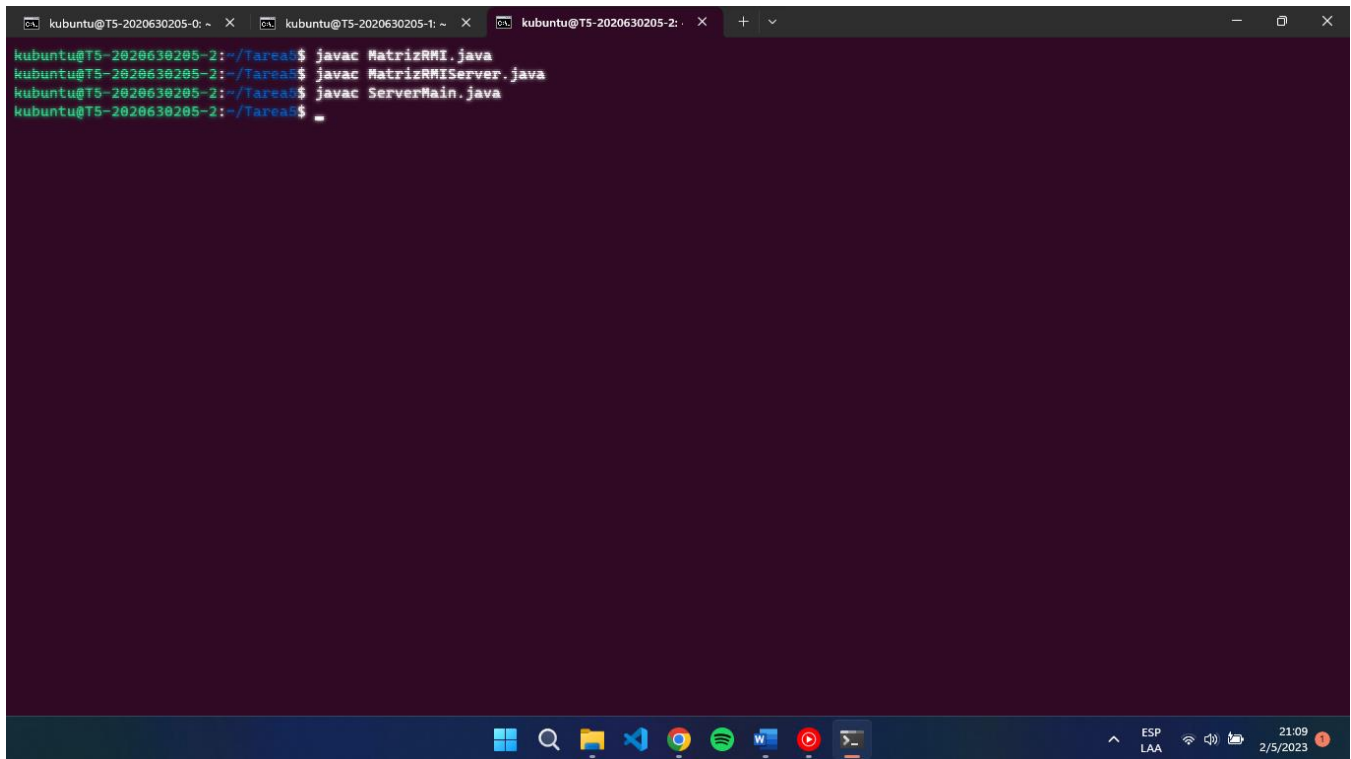


A terminal window with three tabs. The second tab is active and shows the following commands and output:

```
kubuntu@T5-2020630205-1: ~/Tarea5$ javac MatrizRMI.java
kubuntu@T5-2020630205-1: ~/Tarea5$ javac MatrizRMIServer.java
kubuntu@T5-2020630205-1: ~/Tarea5$ javac ServerMain.java
kubuntu@T5-2020630205-1: ~/Tarea5$ _
```

The terminal has a dark purple background and a Windows taskbar at the bottom with icons for File Explorer, VS Code, Chrome, Spotify, and others. The system tray shows the time as 21:08 on 2/5/2023.

3. En el nodo 2 solo compilamos la interfaz, la clase del servidor y su método principal

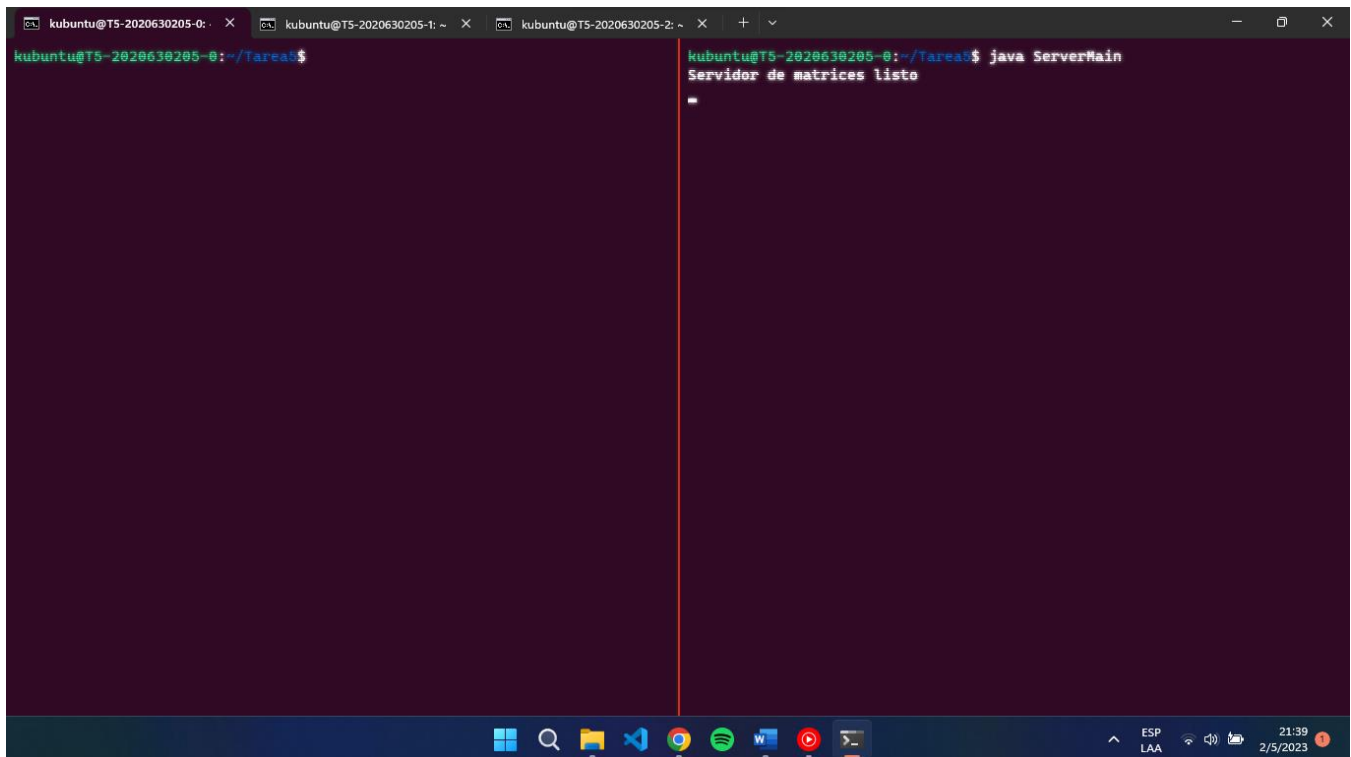


A terminal window with three tabs. The active tab shows the following commands and output:

```
kubuntu@T5-2020630205-2: ~/Tarea5$ javac MatrizRMI.java
kubuntu@T5-2020630205-2: ~/Tarea5$ javac MatrizRMIServer.java
kubuntu@T5-2020630205-2: ~/Tarea5$ javac ServerMain.java
kubuntu@T5-2020630205-2: ~/Tarea5$ _
```

The terminal is running on a Linux system with a dark theme. The taskbar at the bottom shows various application icons and system status information including the time 21:09 and date 2/5/2023.

4. Ejecutamos los servidores en todos los nodos a los que se va a conectar el cliente que correrá en el nodo 0.



A terminal window with three tabs. The active tab shows the following command and output:

```
kubuntu@T5-2020630205-0: ~/Tarea5$ java ServerMain
Servidor de matrices listo
=
```

The terminal is running on a Linux system with a dark theme. The taskbar at the bottom shows various application icons and system status information including the time 21:39 and date 2/5/2023.

```
kubuntu@T5-2020630205-0: ~$  
kubuntu@T5-2020630205-1: ~$ java ServerMain  
Servidor de matrices listo  
=
```

```
kubuntu@T5-2020630205-0: ~$  
kubuntu@T5-2020630205-1: ~$ java ServerMain  
Servidor de matrices listo  
=
```

5. Por último sólo queda ejecutar el cliente en el nodo 0 para que inicialice las matrices A y Bt, las envíe a cada nodo en un hilo, las reciba y al final sólo calcule el checksum.

Con $N=9$ y $M=4$

```
kubuntu@T5-2020630205-0: ~$ java MatrizRMIClient 9 4
126.0 90.0 54.0 18.0 -18.0 -54.0 -90.0 -126.0 -162.0
162.0 110.0 58.0 6.0 -46.0 -98.0 -150.0 -202.0 -254.0
198.0 130.0 62.0 -6.0 -74.0 -142.0 -210.0 -278.0 -346.0
234.0 150.0 66.0 -18.0 -102.0 -186.0 -270.0 -354.0 -438.0
270.0 170.0 70.0 -30.0 -130.0 -230.0 -330.0 -430.0 -530.0
306.0 190.0 74.0 -42.0 -158.0 -274.0 -390.0 -506.0 -622.0
342.0 210.0 78.0 -54.0 -186.0 -318.0 -450.0 -582.0 -714.0
378.0 230.0 82.0 -66.0 -214.0 -362.0 -510.0 -658.0 -806.0
414.0 250.0 86.0 -78.0 -242.0 -406.0 -570.0 -734.0 -898.0
Checksum: -10530.0
kubuntu@T5-2020630205-0: ~$
```

```
kubuntu@T5-2020630205-0: ~$ java ServerMain
Servidor de matrices listo
```

Con $N=900$ y $M=400$

```
kubuntu@T5-2020630205-0: ~$ java MatrizRMIClient 900 400
Checksum: -1.06919889581881E14
kubuntu@T5-2020630205-0: ~$
```

```
kubuntu@T5-2020630205-0: ~$ java ServerMain
Servidor de matrices listo
```

Conclusiones

A lo largo del desarrollo de la práctica, hemos podido entender y aplicar los conceptos clave de la programación distribuida, como la serialización de objetos, la gestión de excepciones, el paso de parámetros por referencia y la creación y administración de threads en paralelo.

En cuanto a la utilización de Java RMI como tecnología para desarrollar objetos distribuidos, hemos podido apreciar su potencial para la construcción de sistemas escalables y tolerantes a fallos. Al utilizar esta tecnología, hemos podido crear una aplicación distribuida que ha permitido a los clientes acceder a los servicios del servidor de manera transparente, sin tener que preocuparse por los detalles de la implementación del servidor.

Finalmente, el despliegue del sistema en distintas máquinas virtuales de Ubuntu Server en Azure ha sido una experiencia muy interesante, en la que hemos podido aprender a administrar y configurar las máquinas virtuales, y a establecer la comunicación entre ellas mediante el uso de las direcciones IP privadas y los puertos especificados en el cliente.

El desarrollo de un sistema distribuido con objetos distribuidos utilizando Java RMI y su ejecución en distintas máquinas virtuales de Ubuntu Server en Azure ha sido una experiencia muy valiosa en la que hemos aprendido y aplicado numerosos conceptos clave de la programación distribuida y de la administración de sistemas en la nube.