Instituto Politécnico Nacional
Escuela Superior de Cómputo

Multiplicación de matrices utilizando objetos distribuidos

Tarea 5

Alumno: Francisco Nicolas Sánchez García
Asignatura Desarrollo de Sistemas Distribuidos
Profesor: Carlos Pineda Guerrero
Grupo 4CV12

ÍNDICE

INTRODUCCIÓN

El uso de objetos distribuidos fue ampliamente usados gracias a implementaciones como las RMI en Java que permitían ejecutar código remoto conectándose a un servidor que lo tuviese. Cuenta con grandes ventajas comparándolo con otro tipo de implementaciones, por ejemplo, para realizar esto anteriormente mediante sockets, primero se tiene que empaquetar las matrices del lado del cliente, desempaquetarlas en el servidor y luego volver a empaquetar el resultado para que finalmente en el cliente se desempaquete y muestre el resultado.

DESARROLLO

A continuación, se muestra el código fuente del programa para el cliente, para el servidor, la interfaz y lo del RMI.

Código fuente de Matriz.java

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

public class Matriz {
    static int n = 9;
    static double a[][] = new double[n][n];
    static double b[][] = new double[n][n];
    static double c[][] = new double[n][n];

    public static void main(String[] args) throws Exception {

        inicializarMatrices();
        if (n == 9) {
            System.out.println("a");
            imprimirMatriz(a);
            System.out.println("b");
            imprimirMatriz(b);
        }
        transponerMatriz(b);

        double[][] a1 = separa_matriz(a, 0);
        double[][] a2 = separa_matriz(a, n / 3);
        double[][] a3 = separa_matriz(a, (2 * n) / 3);

        double[][] b1 = separa_matriz(b, 0);
        double[][] b2 = separa_matriz(b, n / 3);
        double[][] b3 = separa_matriz(b, (2 * n) / 3);

        // parte de vms
        String url_1 = "rmi://10.1.0.5/prueba";
        String url_2 = "rmi://10.1.0.6/prueba";
        String url_3 = "rmi://10.1.0.7/prueba";
        InterfaceMatriz r1 = (InterfaceMatriz) Naming.lookup(url_1);
```

```
        InterfaceMatriz r2 = (InterfaceMatriz) Naming.lookup(url_2);
        InterfaceMatriz r3 = (InterfaceMatriz) Naming.lookup(url_3);

        double[][] c1 = r1.multiplicarMatrices(a1, b1, n);
        double[][] c2 = r1.multiplicarMatrices(a1, b2, n);
        double[][] c3 = r1.multiplicarMatrices(a1, b3, n);
        double[][] c4 = r2.multiplicarMatrices(a2, b1, n);
        double[][] c5 = r2.multiplicarMatrices(a2, b2, n);
        double[][] c6 = r2.multiplicarMatrices(a2, b3, n);
        double[][] c7 = r3.multiplicarMatrices(a3, b1, n);
        double[][] c8 = r3.multiplicarMatrices(a3, b2, n);
        double[][] c9 = r3.multiplicarMatrices(a3, b3, n);

        acomoda_matriz(c, c1, 0, 0);
        acomoda_matriz(c, c2, 0, n / 3);
        acomoda_matriz(c, c3, 0, (2 * n) / 3);
        acomoda_matriz(c, c4, n / 3, 0);
        acomoda_matriz(c, c5, n / 3, n / 3);
        acomoda_matriz(c, c6, n / 3, (2 * n) / 3);

        acomoda_matriz(c, c7, (2 * n) / 3, 0);
        acomoda_matriz(c, c8, (2 * n) / 3, n / 3);
        acomoda_matriz(c, c9, (2 * n) / 3, (2 * n) / 3);

        if (n == 9) {
            System.out.println("matriz c");
            imprimirMatriz(c);
        }

        System.out.println(obtenerChecksum(c));
    }

    private static double obtenerChecksum(double[][] matriz) {
        double res = 0;
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                res += matriz[i][j];
            }
        }
        return res;
    }

    private static void imprimirMatriz(double[][] matriz) {
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                System.out.print(matriz[i][j] + "\t");
            }
            System.out.println("");
        }
    }

    private static void transponerMatriz(double[][] b2) {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < i; j++) {
                double x = b2[i][j];
                b2[i][j] = b2[j][i];
                b2[j][i] = x;
            }
    }

    private static void inicializarMatrices() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                a[i][j] = 4 * i + j;
                b[i][j] = 4 * i - j;
            }
        }
    }

    static double[][] separa_matriz(double[][] a, int renglon_inicial) {
        double[][] m = new double[n / 2][n];
```

```
        for (int i = 0; i < n / 3; i++) {
            for (int j = 0; j < n; j++) {
                m[i][j] = a[i + renglon_inicial][j];
            }
        }
        return m;
    }

    static void acomoda_matriz(double[][] C, double[][] c, int renglon, int columna) {
        for (int i = 0; i < n / 3; i++) {
            for (int j = 0; j < n / 3; j++) {
                C[i + renglon][j + columna] = c[i][j];
            }
        }
    }
}
```

Código fuente de MatrizRMI.java

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

public class MatrizRMI extends UnicastRemoteObject implements InterfaceMatriz {
    public MatrizRMI() throws RemoteException {
        super();
    }

    public double[][] multiplicarMatrices(double[][] m1, double[][] m2, int n) throws
RemoteException {
        double res[][] = new double[m1.length][m2.length];
        for (int i = 0; i < m1.length; i++) {
            for (int j = 0; j < m2.length; j++) {
                for (int k = 0; k < n; k++) {
                    res[i][j] += m1[i][k] * m2[j][k];
                }
            }
        }
        return res;
    }
}
```

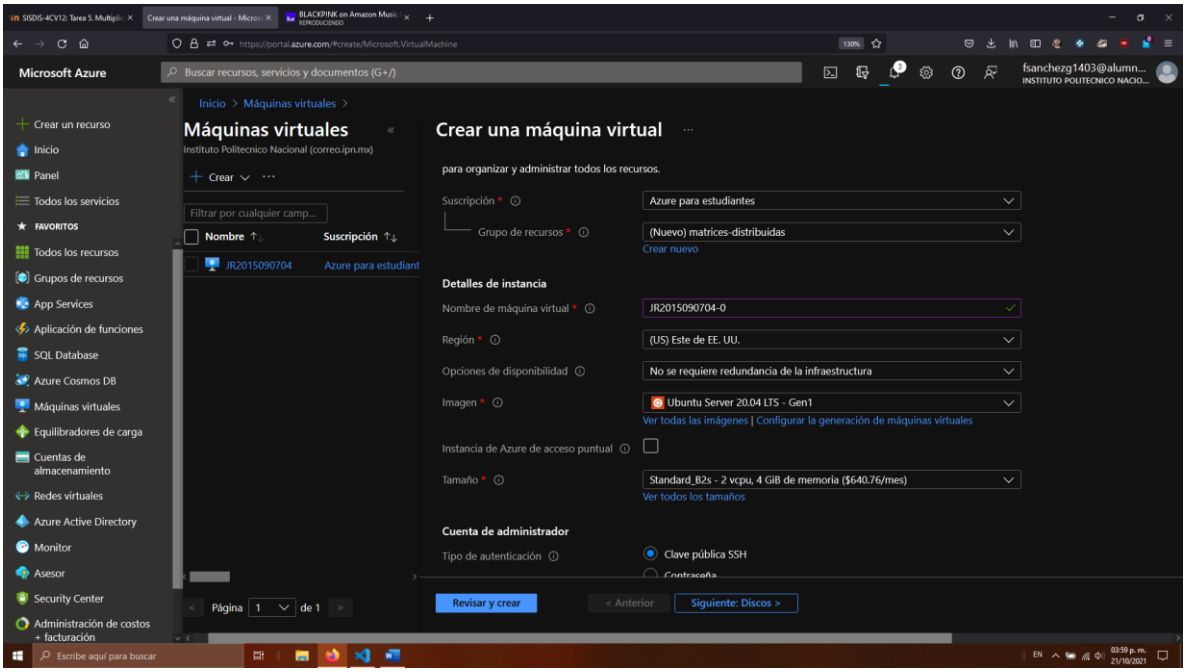
Código fuente de InterfaceMatriz.java

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

public interface InterfaceMatriz extends Remote {
    public double[][] multiplicarMatrices(double[][] m1, double[][] m2, int n) throws
RemoteException;
}
```

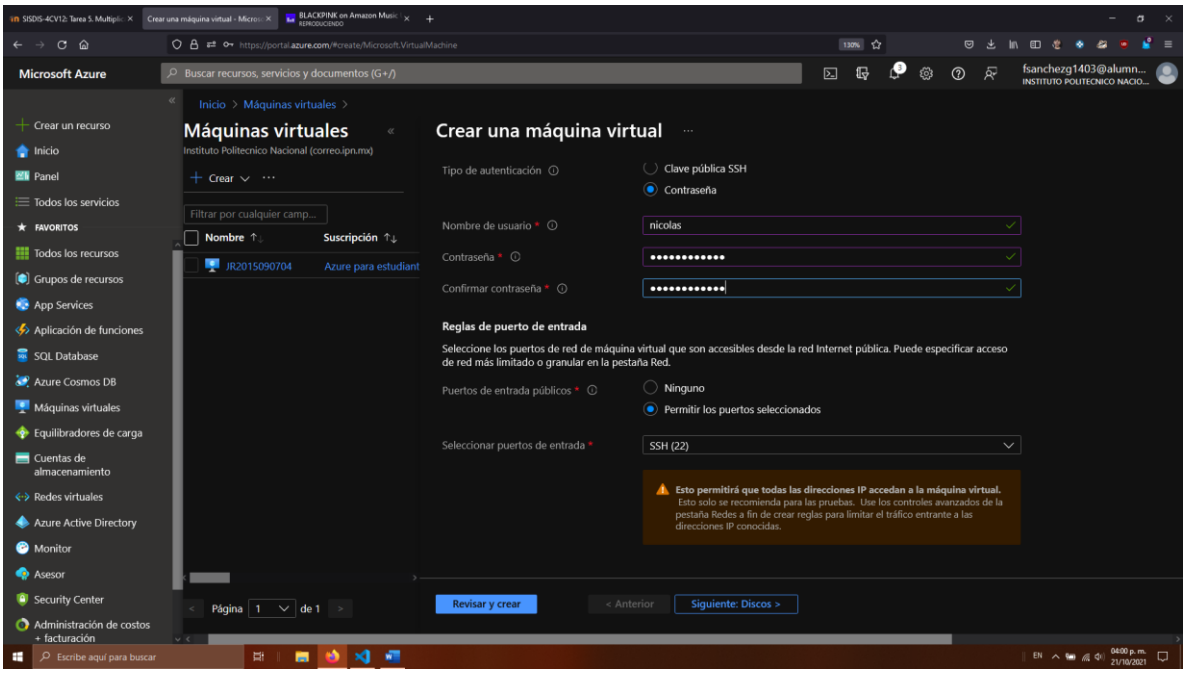
Código fuente de Servidor.java

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

public class Servidor {
    public static void main(String[] args) throws Exception, RemoteException {
        // puerto 1099 creo
        String url = "rmi://localhost/prueba";
        MatrizRMI obj = new MatrizRMI();

        // registrar instancia en rmi registry
        Naming.rebind(url, obj);
    }
}
```

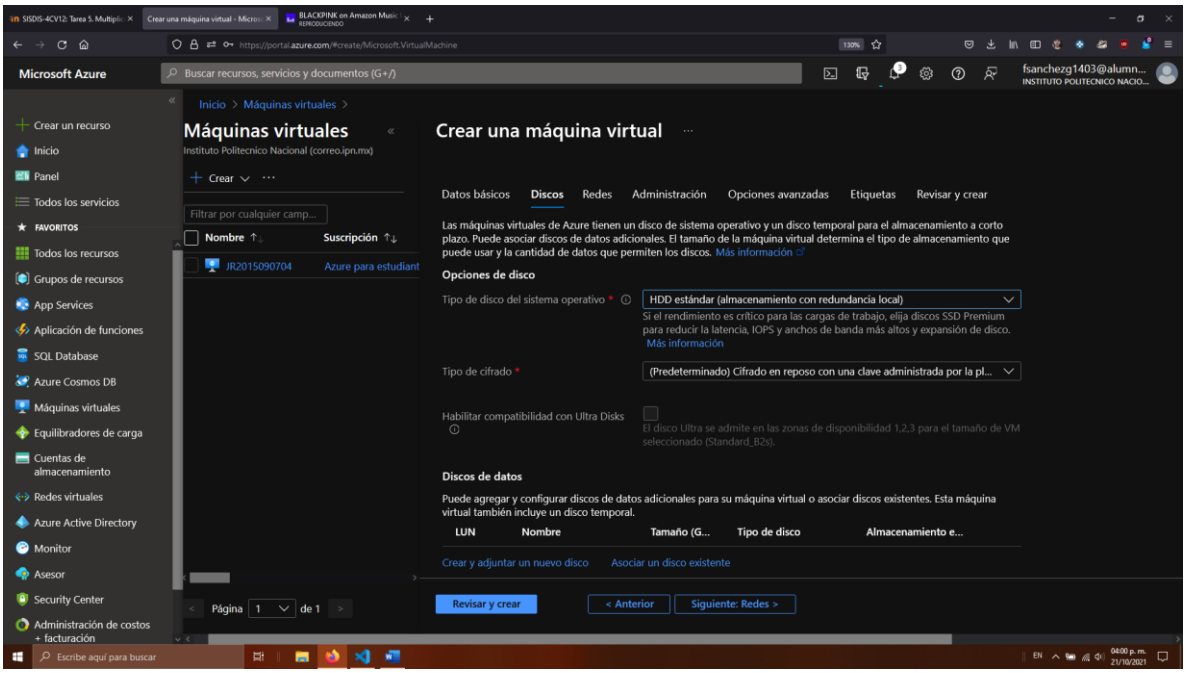
Creación de máquina virtual



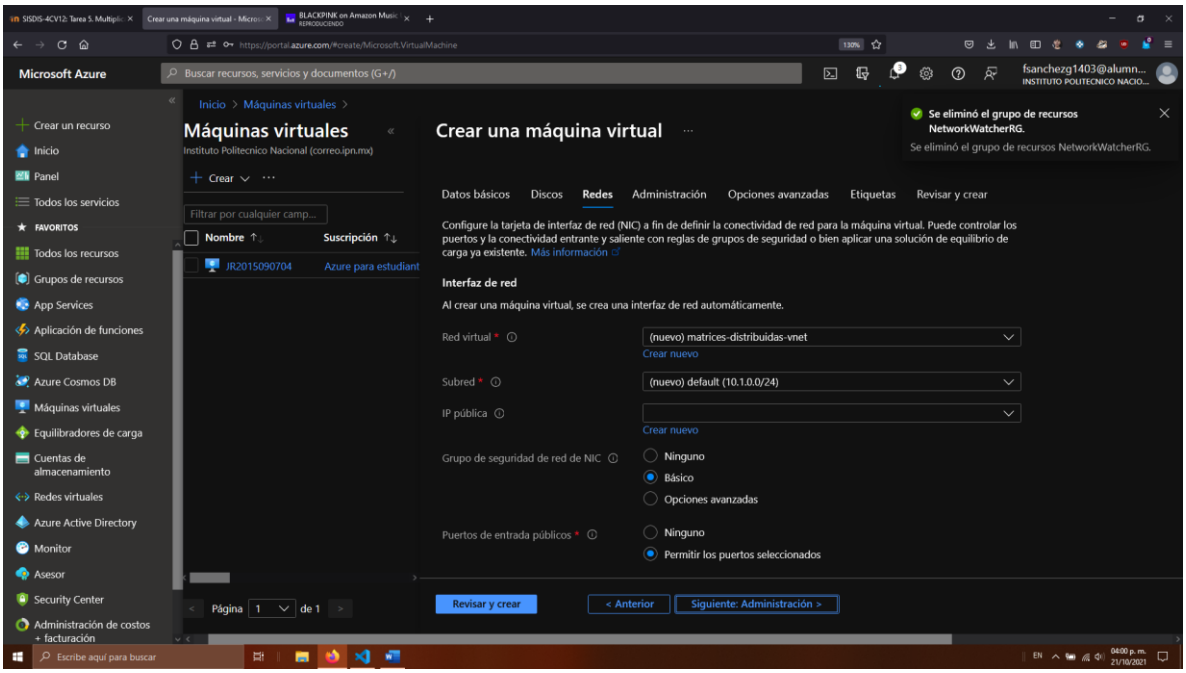
Para crear la máquina virtual se crea un nuevo grupo de recursos para la VM llamado “matrices-distribuidas”, se agrega el nombre de la máquina siguiendo la nomenclatura de la tarea (JR2015090704-0 para el primer nodo). Se selecciona la imagen de Ubuntu Server 20.04 LTS - Gen1. Se selecciona el tamaño B2s.



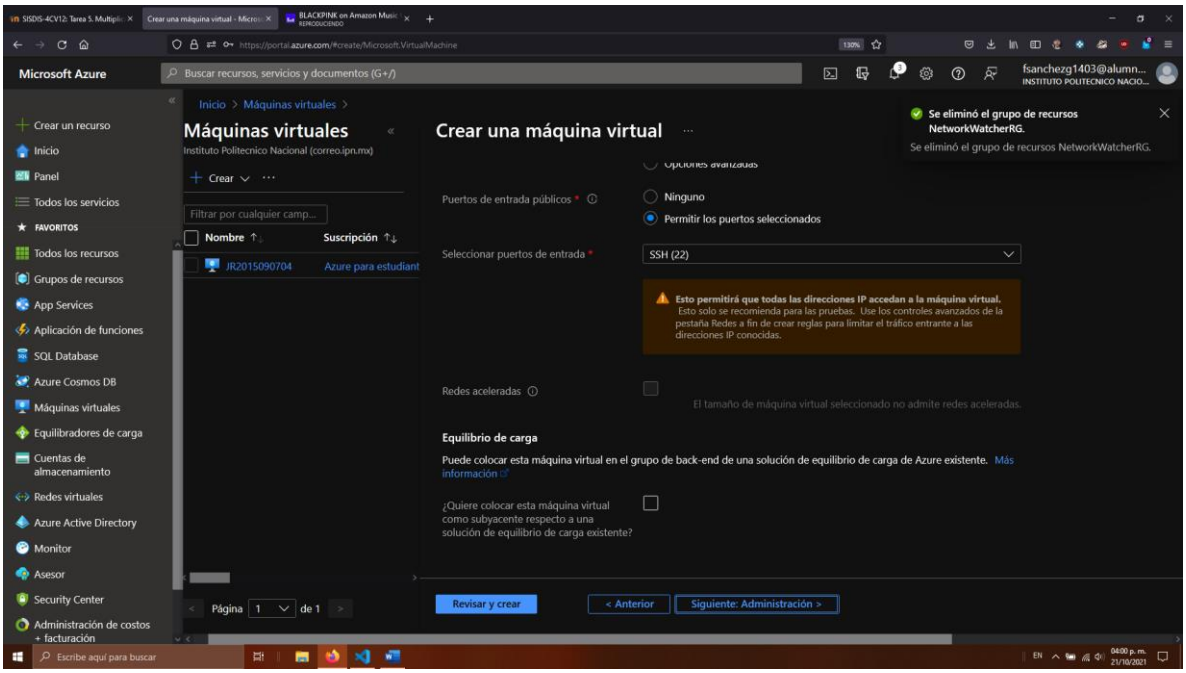
Agregué usuario y contraseña y se deja el puerto de entrada como SSH para realizar la posterior conexión y pasamos al apartado de “Discos”.



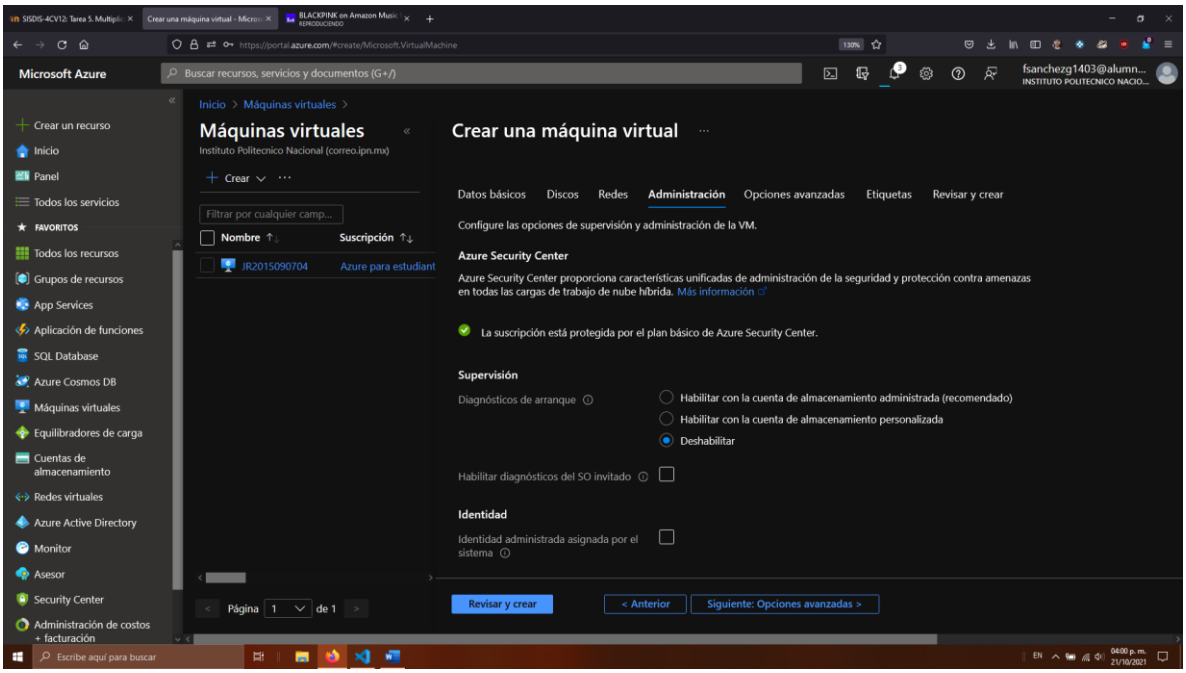
Cambiamos el tipo de disco del sistema operativo a un “HDD estándar”. No se modifica nada de “Opciones avanzadas” y pasamos al apartado de “Redes”.



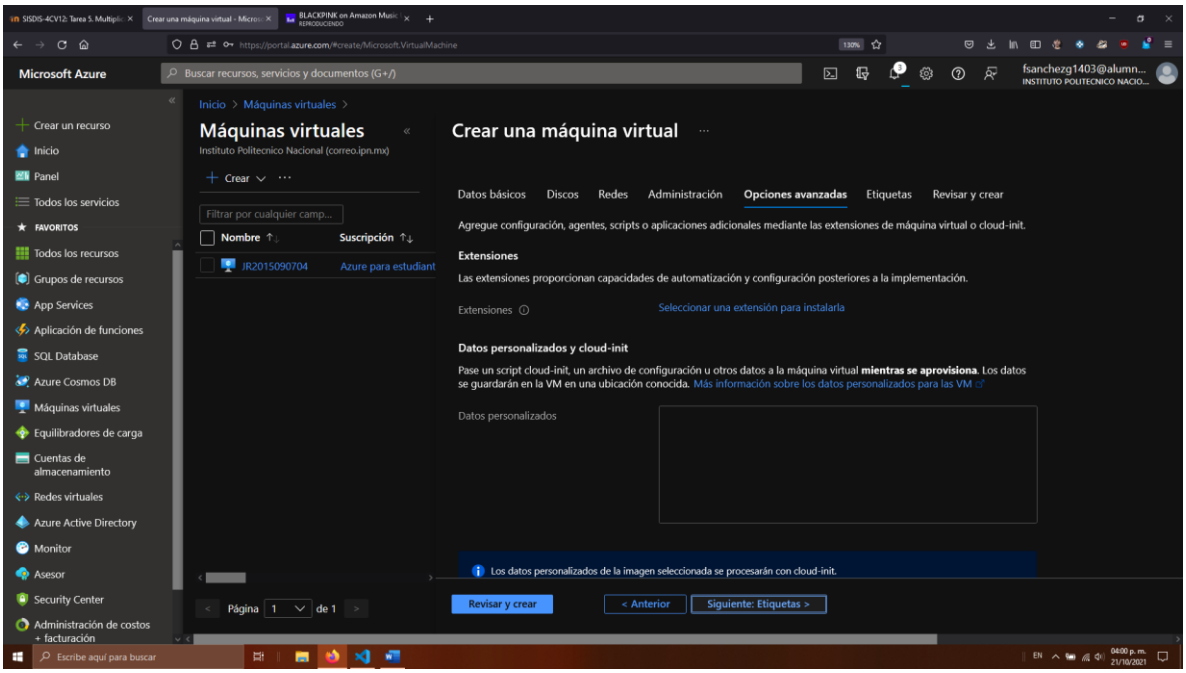
Dejamos los valores generados por defecto en el apartado de “Redes”.



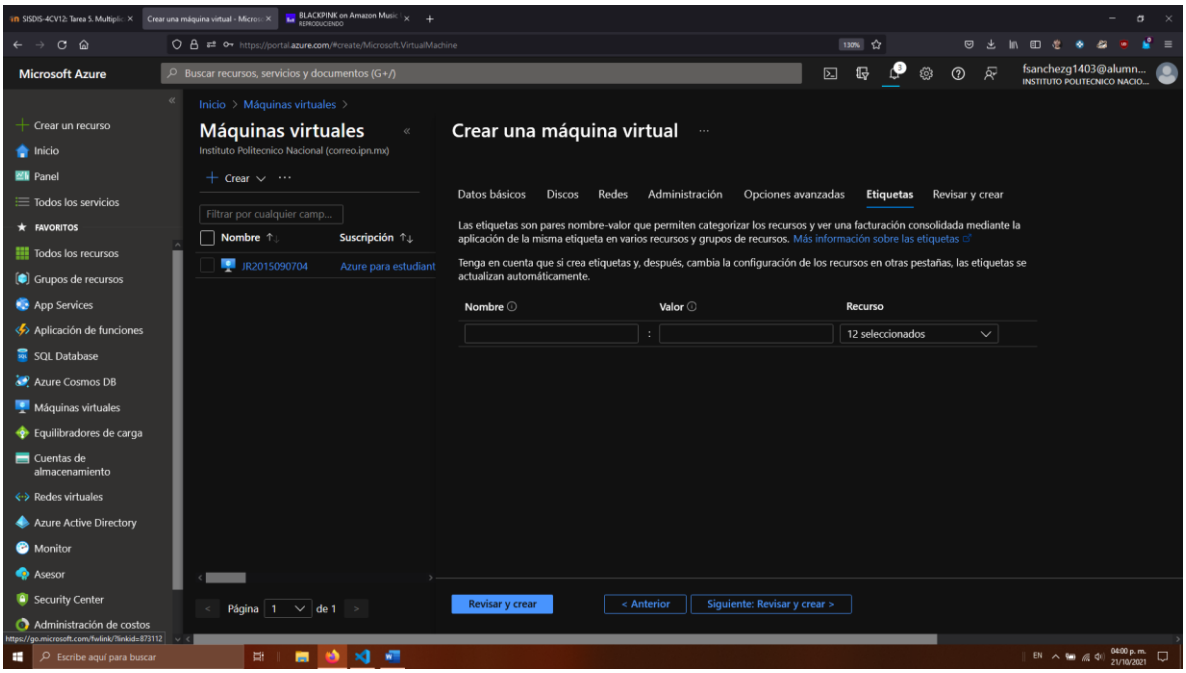
Pasamos al apartado de “Administración”.



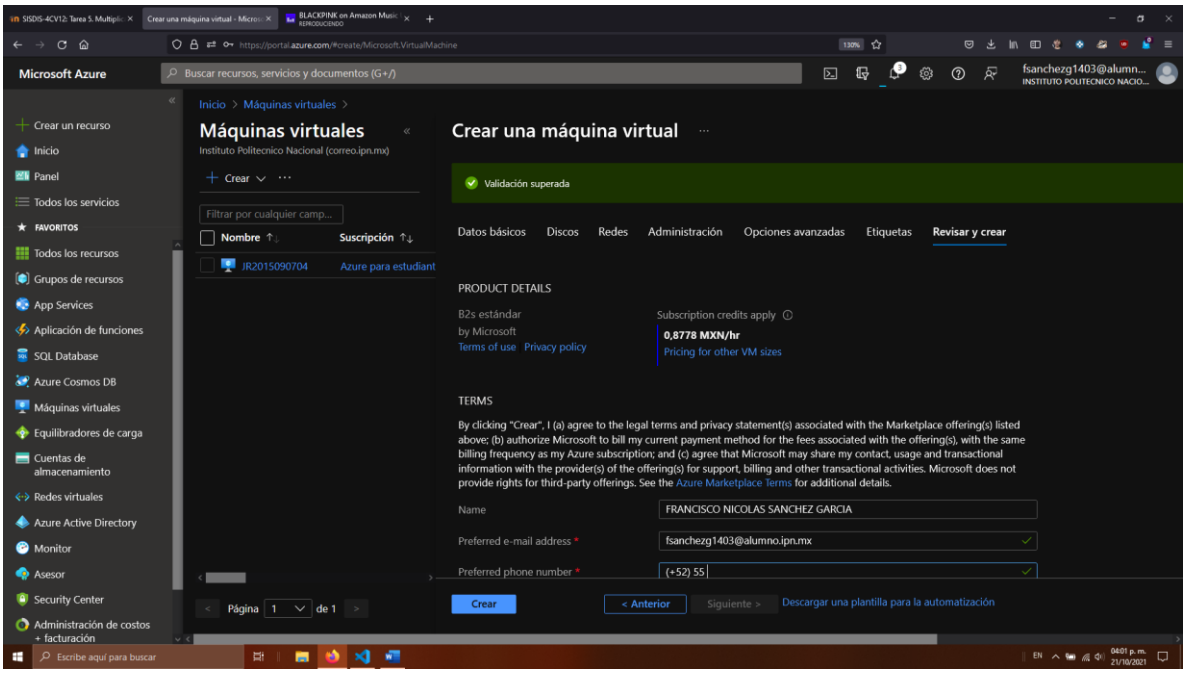
Se deshabilita el diagnóstico de arranque de la máquina virtual. No se modifica ninguna otra opción y pasamos al apartado de “Opciones avanzadas”.



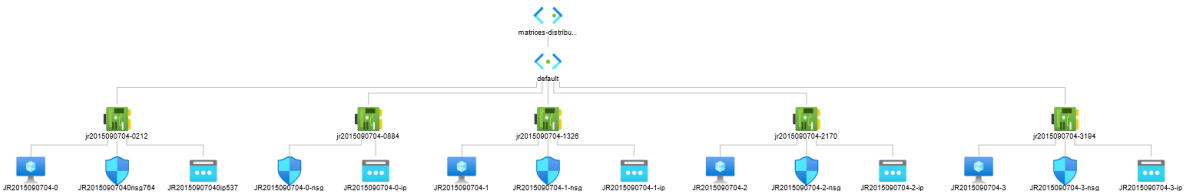
No se modifica nada del apartado de “Opciones avanzadas”. Pasamos al apartado de “Etiquetas”.



No se agrega ninguna etiqueta. Pasamos al apartado de “Revisar y crear”.

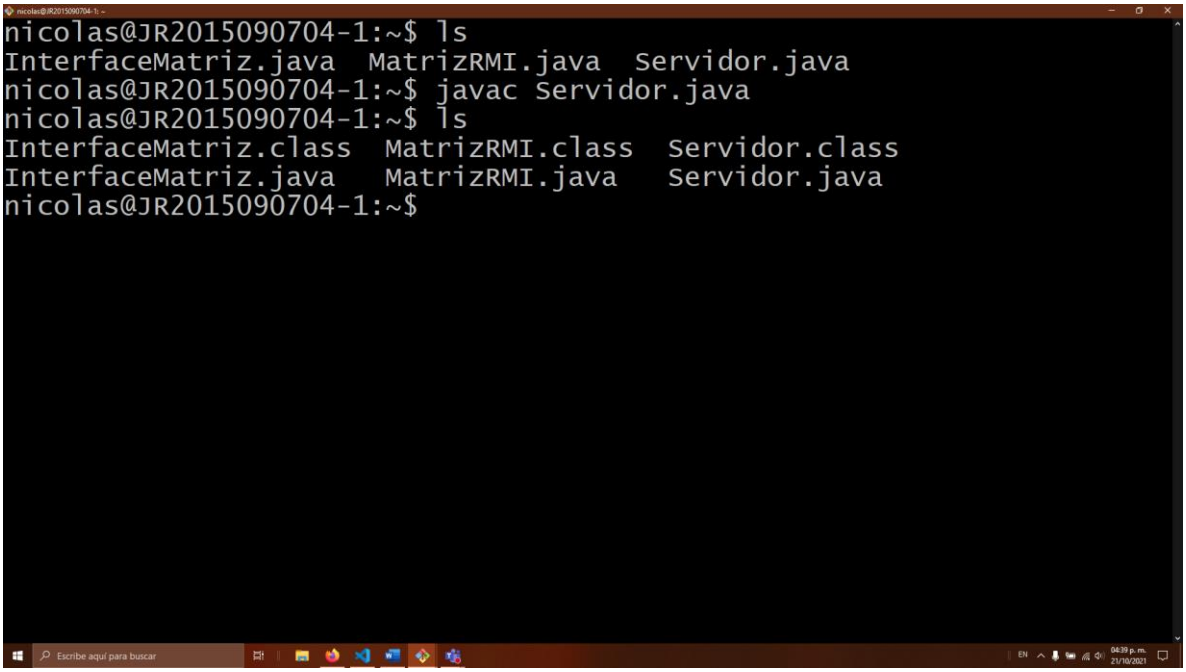


Una vez que la validación ha sido superada, creamos la máquina virtual para el nodo 0.

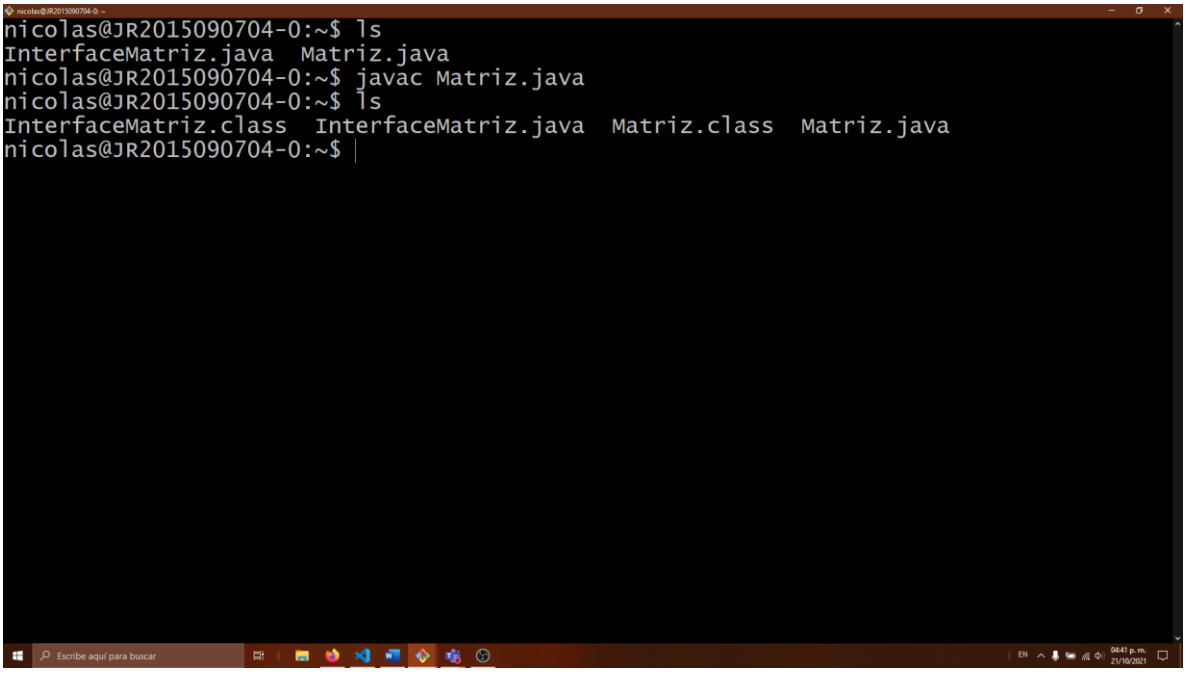


La topología generada en la red virtual al ingresar los cuatro nodos es la siguiente, al capturar la primera máquina virtual tuve un error con una máquina virtual y al borrarla no se eliminaron los recursos de red asociados a la misma, sin embargo, no afectó en la ejecución del programa.

Compilación del programa

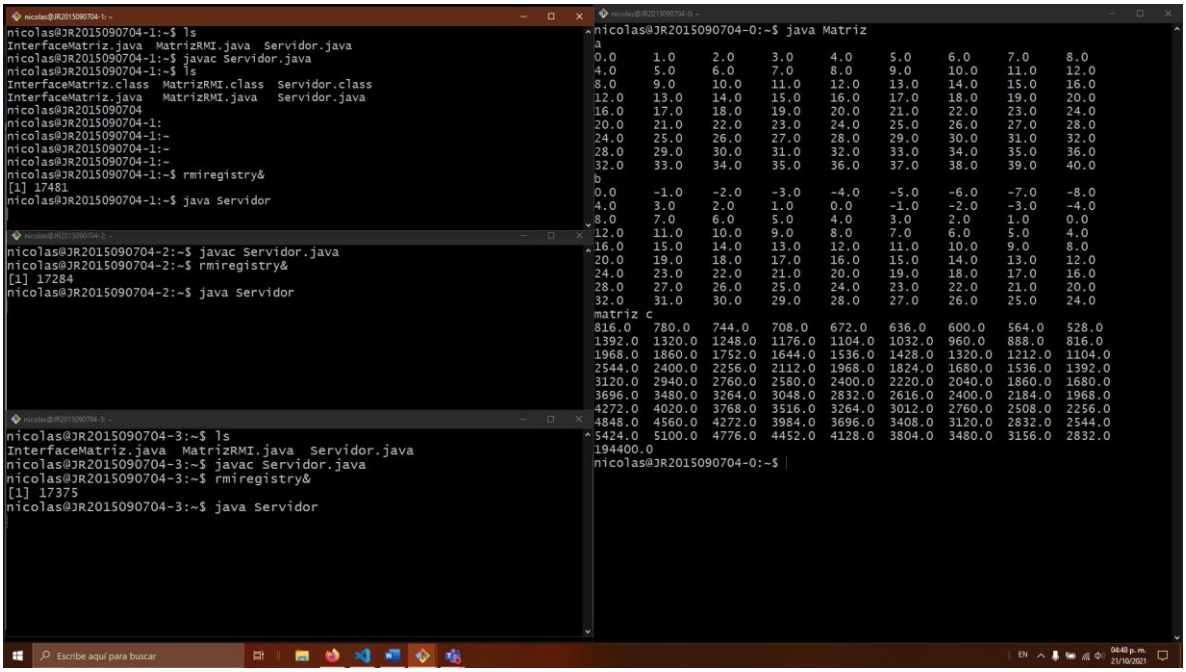


Compilación del programa del Servidor en la máquina virtual del nodo 1.

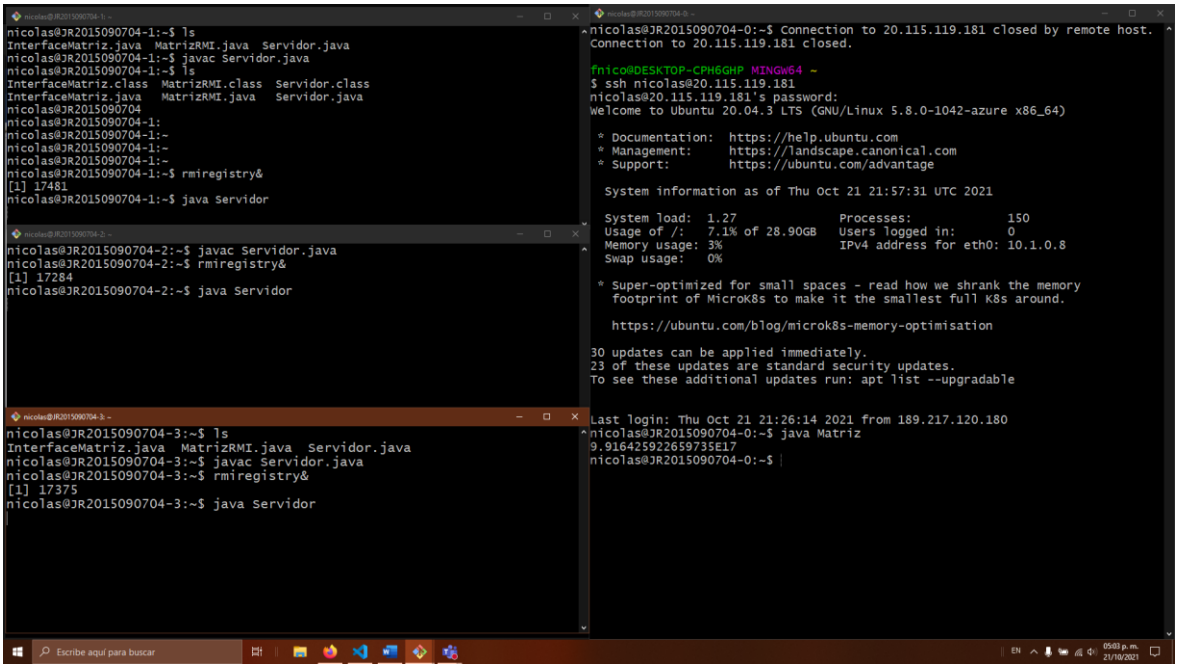


Compilación del programa cliente llamado en este caso Matriz.java dentro del nodo 0.

Ejecución del programa



Para n = 9. Del lado izquierdo los tres nodos que actúan como servidores, donde primero se ejecutó el rmiregistry y luego el programa del servidor, del lado derecho la impresión de la matriz a, b y c, finalmente el checksum con un valor de 194400.0.



Para n = 3000. Al intentar realizar esta ejecución el programa requirió más recursos de memoria RAM, por lo que cambié el tamaño de la máquina virtual del nodo 0 a uno con 8 GB de RAM.

Del lado izquierdo los tres nodos servidores y del lado derecho la conexión a la máquina virtual después del cambio de tamaño y el checksum de 9.916425922659735E17.

CONCLUSIONES

A diferencia de la práctica en la cual se empaquetaron las matrices y se desempaquetaban en el lado del servidor, esta resultó mucho más simple por el hecho de que RMI me permitió ejecutar los métodos remotos sin tener que preocuparme por como enviaba y recibía datos por medio de sockets. Además, en esta práctica por primera vez tuve que aumentar el tamaño de la máquina virtual del cliente después de que la hubiese creado. También, revisando el tiempo de ejecución de la matriz de n = 3000 creo que fue algo tardado comparado con el de sockets aun teniendo en cuenta que con los sockets el tamaño fue de 1500 (una cuarta parte de las operaciones de la de n = 3000).