



INSTITUTO POLITÉCNICO NACIONAL  
ESCUELA SUPERIOR DE CÓMPUTO

## Tarea 5. Multiplicación de matrices utilizando objetos distribuidos

Unidad de aprendizaje: Desarrollo de Sistemas Distribuidos

Grupo: 4CV11

*Alumno:*  
Sanchez Mendez Edmundo Josue

*Profesor:*  
Pineda Guerrero Carlos

17 de octubre de 2021

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Desarrollo</b>	<b>2</b>
2.1. Compilación del código . . . . .	7
2.2. Ejecución del programa . . . . .	9
2.2.1. N=9 . . . . .	9
2.2.2. N=3000 . . . . .	10
<b>3. Conclusiones</b>	<b>13</b>

## 1. Introducción

Se deberá desarrollar un sistema que calcule el producto de dos matrices cuadradas utilizando Java RMI, tal como fue explicado en clase. Se deberán ejecutar dos casos con las siguientes restricciones:

- $N=9$ , se deberá desplegar las matrices A, B y C y el checksum de la matriz C.
- $N=3000$ , deberá desplegar el checksum de la matriz C.

Recordar que el checksum de la matriz es la suma de los valores de esta. Los requisitos son los siguientes:

- Los elementos de las matrices A, B y C serán de tipo double y el checksum será de tipo double.
- Se deberá inicializar las matrices A y B de la siguiente manera:
  - $A[i][j] = i + 4 * j$
  - $B[i][j] = i - 4 * j$
- Se deberá dividir las matrices A y B en tres partes, por tanto la matriz C estará dividida en 9 partes.
- El cliente RMI ejecutará en una máquina virtual con Ubuntu en Azure (nodo 0). El servidor RMI ejecutará en tres máquinas virtuales (nodo 1, nodo 2 y nodo 3) con Ubuntu en Azure. El programa rmiregistry ejecutará en cada nodo donde ejecute el servidor RMI. El nodo 1 calculará los productos C1, C2 y C3, el nodo 2 calculará los productos C4, C5 y C6, y el nodo 3 calculará los productos C7, C8 y C9.

## 2. Desarrollo

Para el desarrollo de la practica nos basamos en el código dado en clase sobre el calculo de las matrices el cual calcula la multiplicación de las matrices pero las divide en dos partes, un objetivo de la practica es poder dividir las matrices en 3 partes, esto provoca que el resultado sea compuesto por 9 matrices de tamaño  $N/3$  cada una, ademas trae como consecuencia que solo podamos multiplicar matrices de un tamaño múltiplo de 3 para tener un correcto funcionamiento. Después el objetivo final es llevar acabo la multiplicación de las matrices mediante el uso de Java RMI para esto primeramente deberemos crear 4 maquinas virtuales con Java instalado y el código del programa, mencionar que para los nodos 1, 2 y 3 solo necesitamos los archivos ServidorRMI.java, InterfaceRMI.java y Clase.java también en estos nodos debemos iniciar el comando rmiregistry para que la practica funcione, para el nodo 0 solo necesitamos las clases ClienteRMI.java y InterfaceRMI.java.

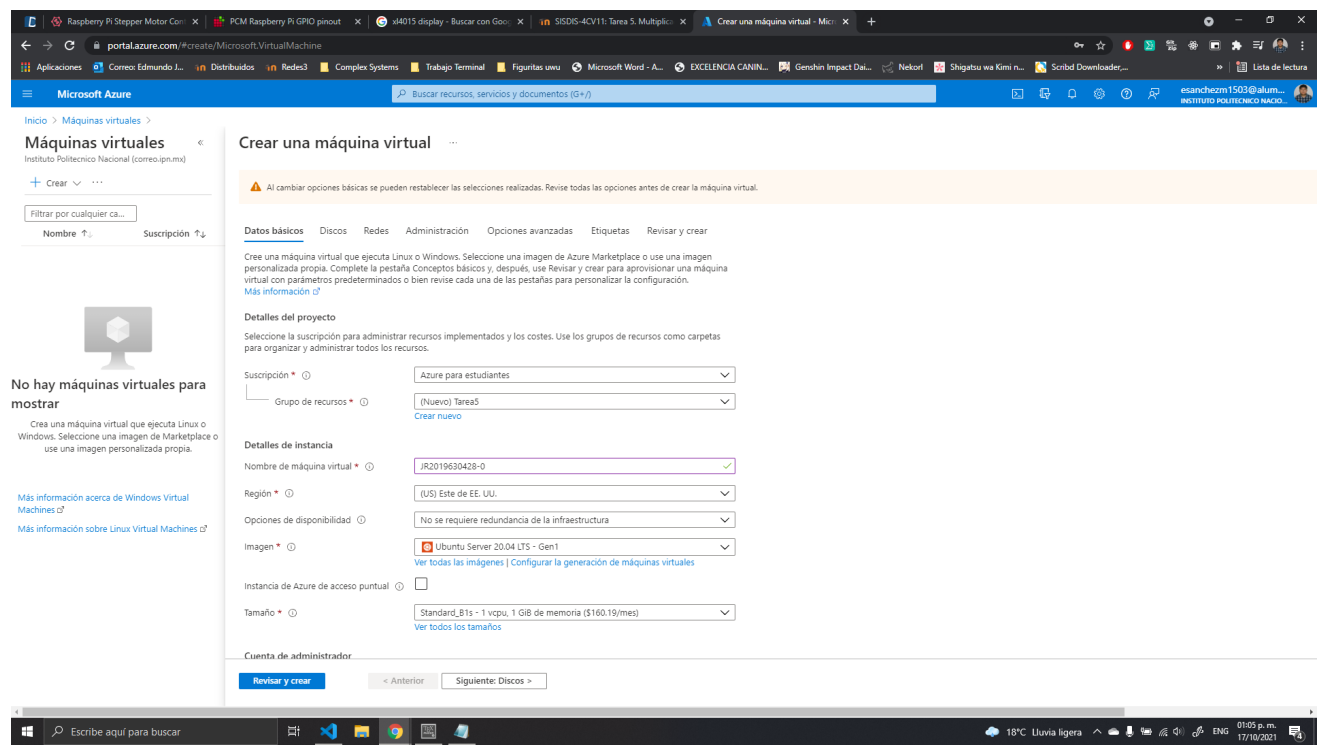


Figura 1: Datos básicos para el nodo 0.

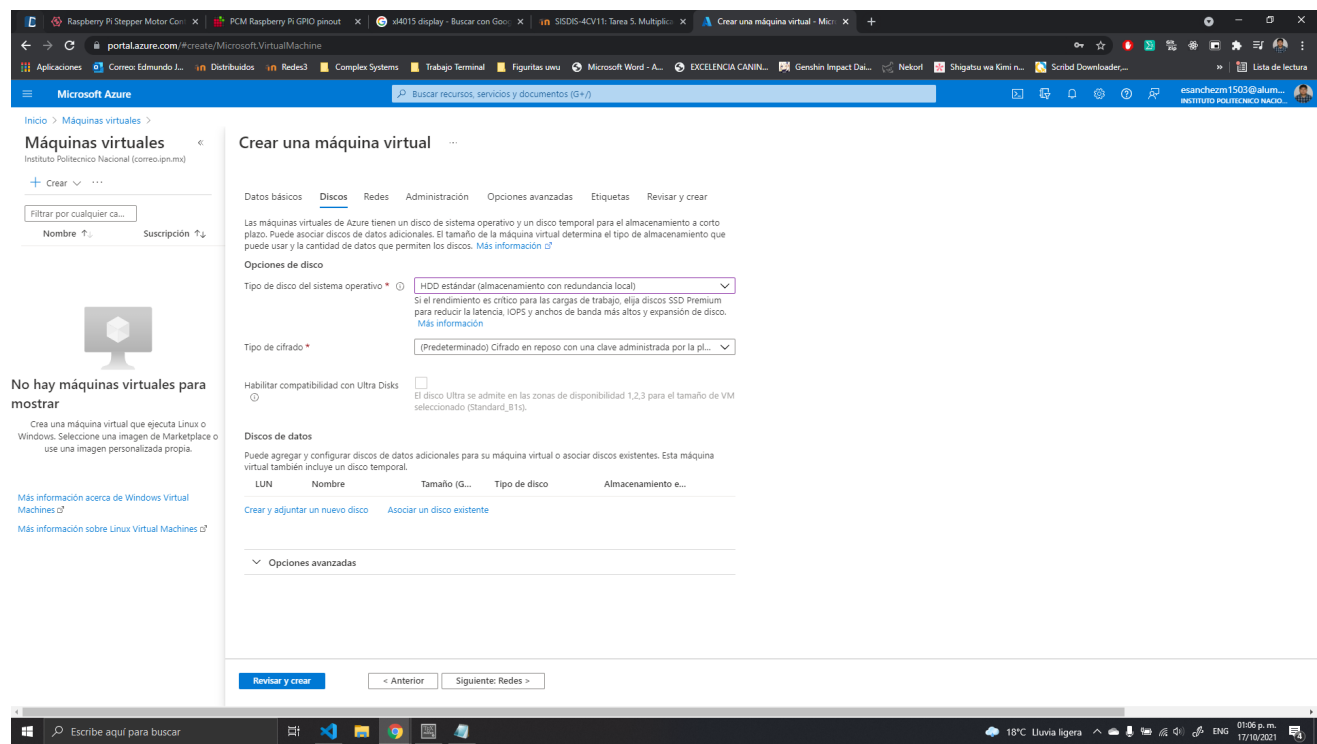


Figura 2: Configuración del tipo de disco para el nodo 0.

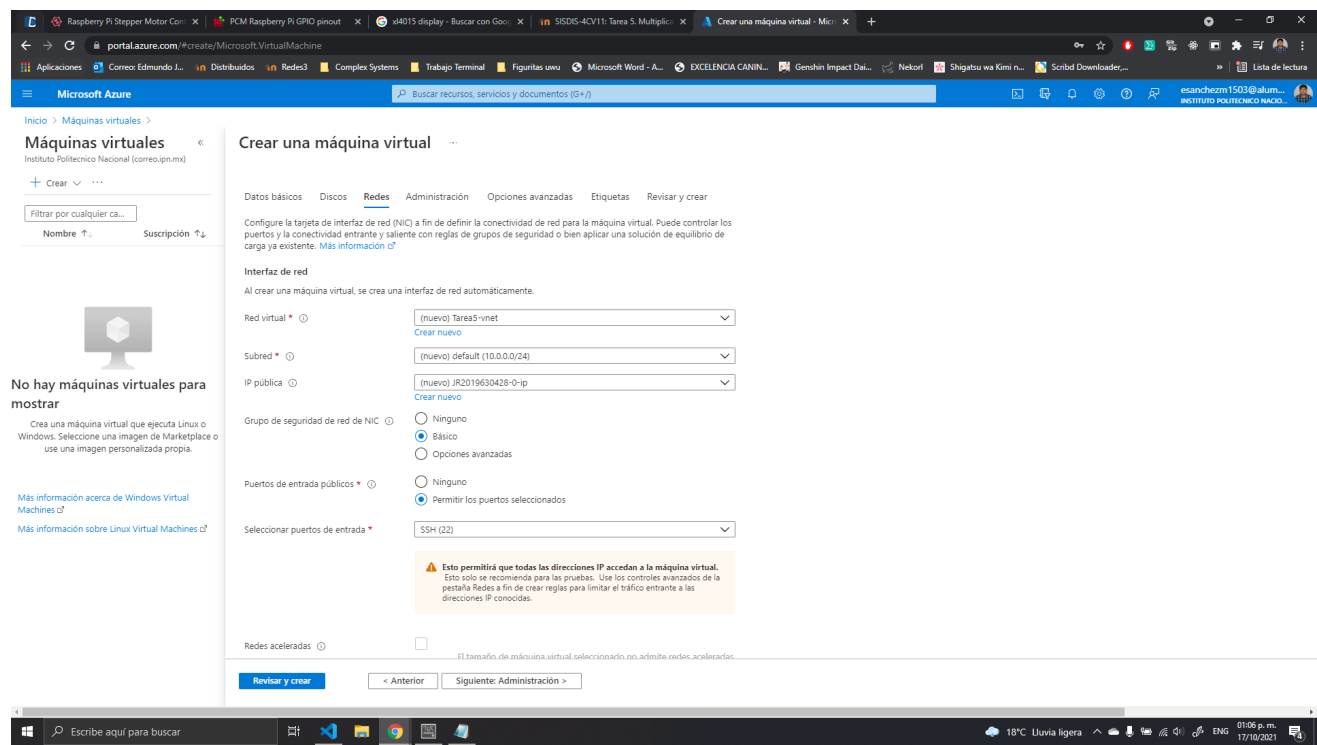


Figura 3: Información sobre la redes para el nodo 0.

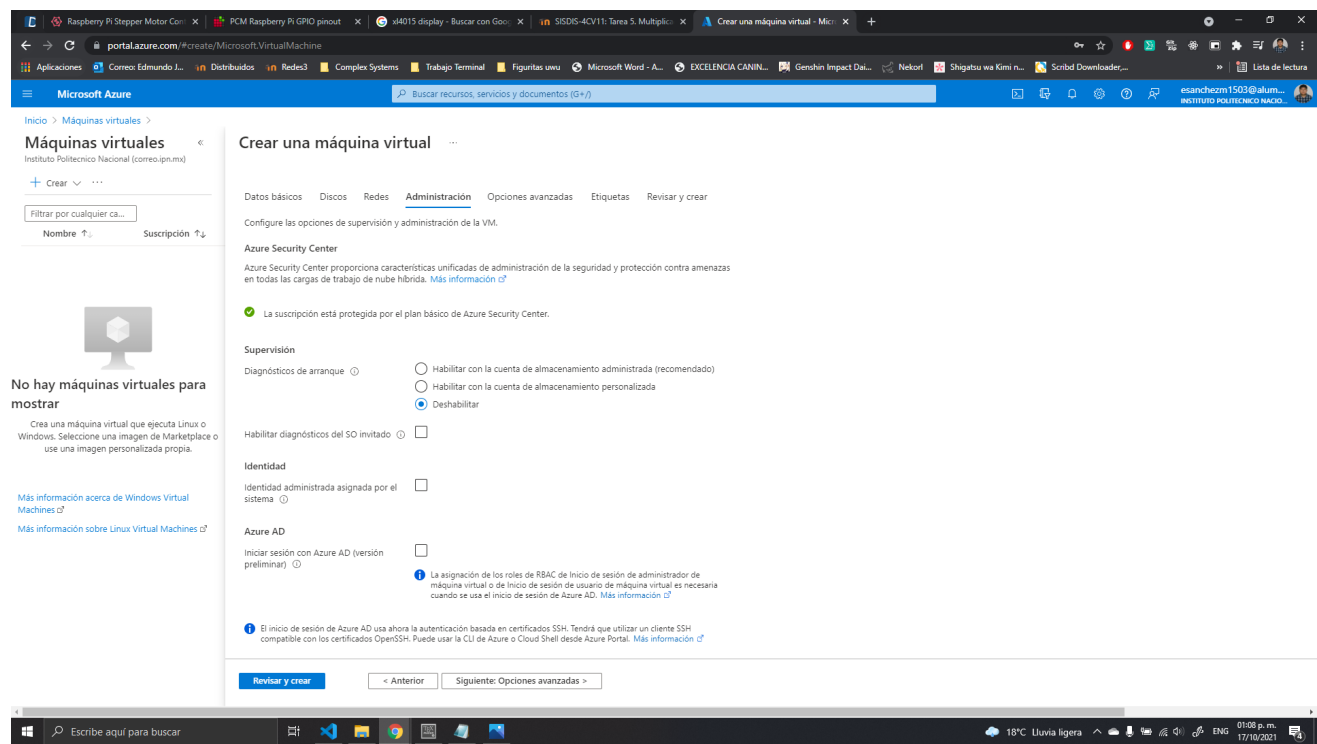


Figura 4: Configuración de la administración para el nodo 0.

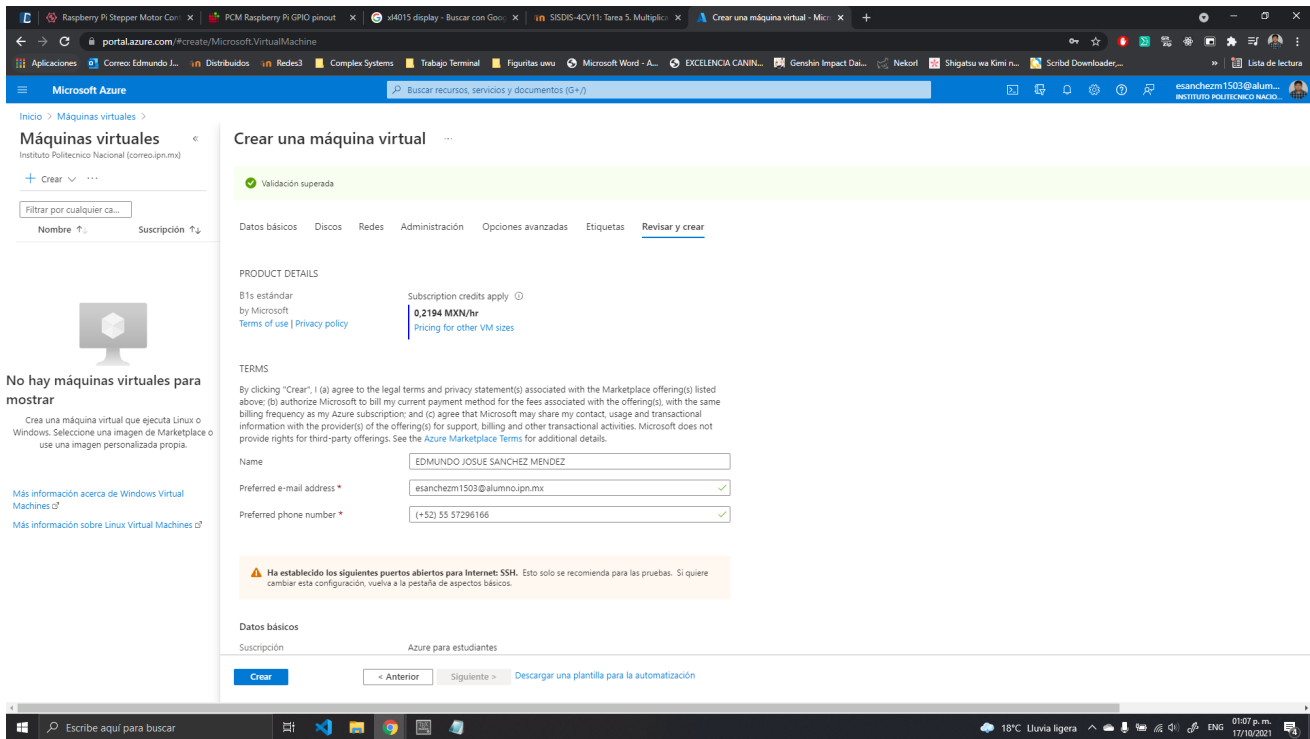


Figura 5: Creación de la maquina virtual para el nodo 0.

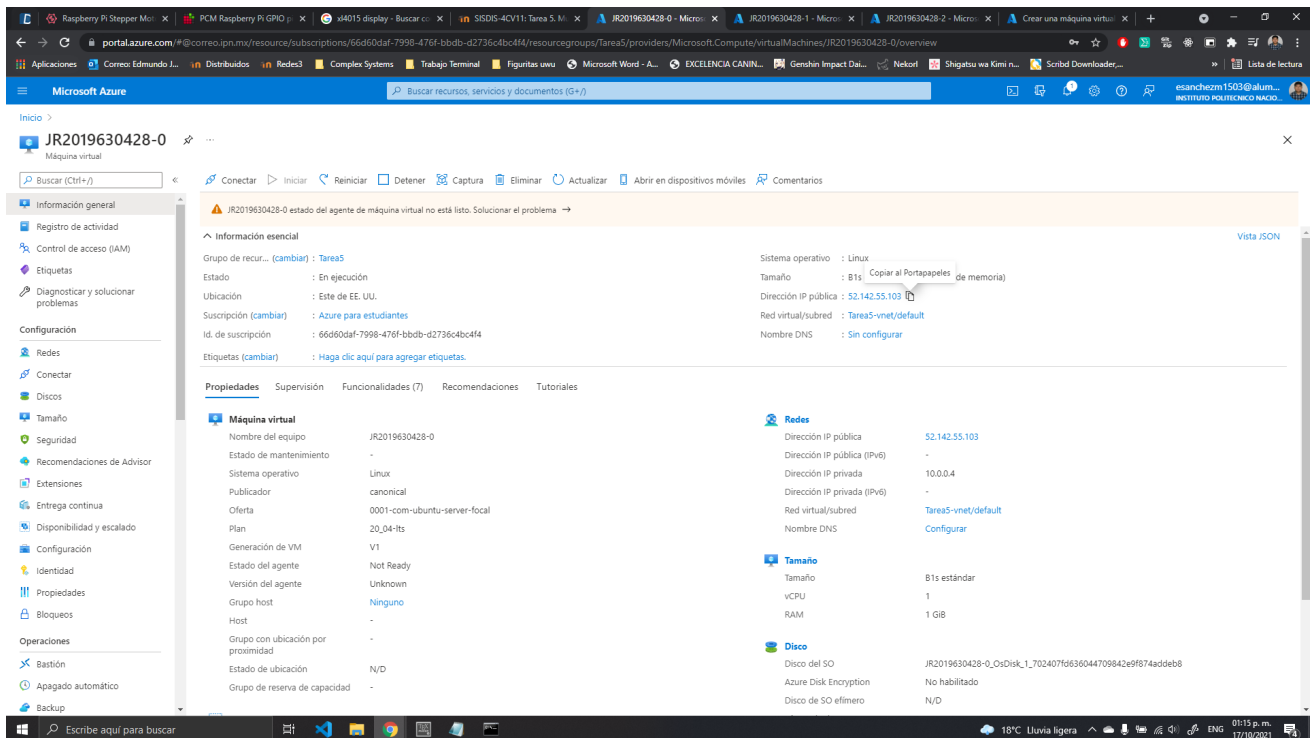


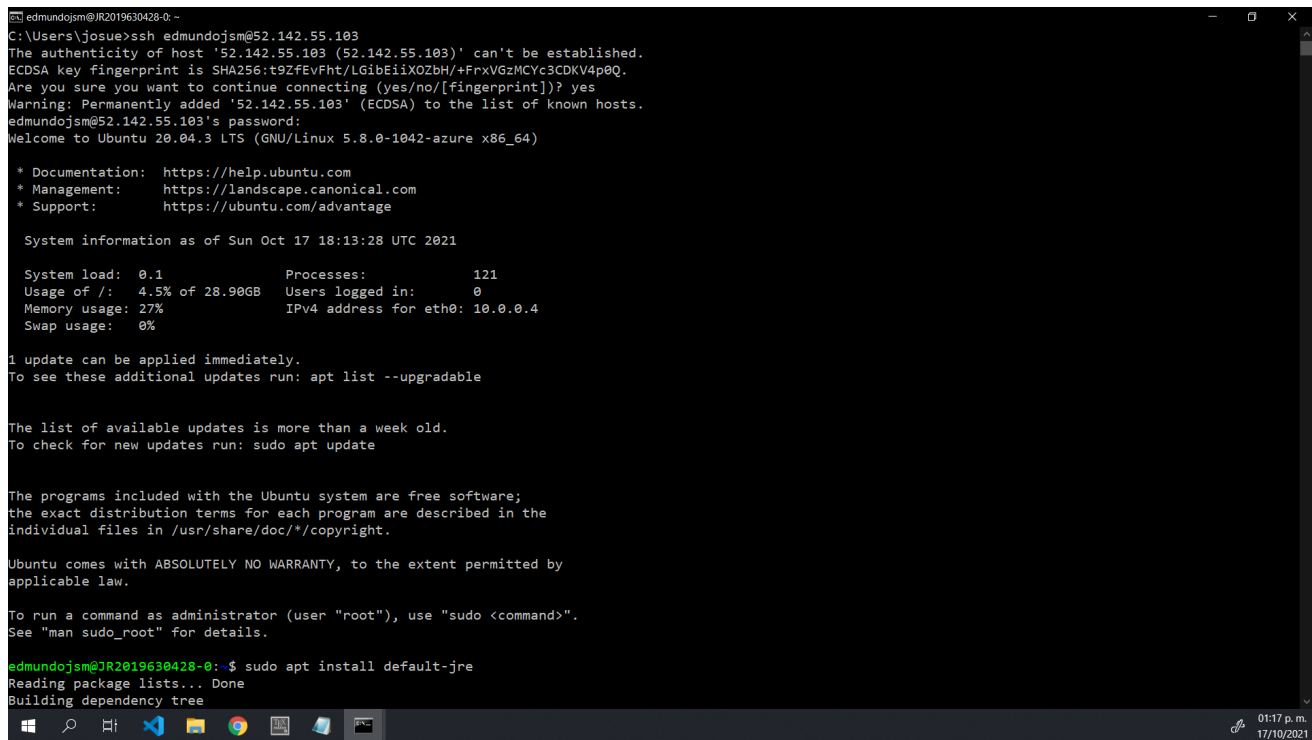
Figura 6: Panel de control de la maquina virtual para el nodo 0.

Las imágenes de la 1 a la 6 se tuvieron que repetir tres veces mas, esto debido al tipo de topologia a implementar, mencionar que en mi caso al momento de querer crear una quinta maquina virtual con las características dadas en la tarea no se me dejo, ya que tengo un limite de 4 maquinas al usar standar.

B1, por lo que tuve que crear una maquina virtual con otras características, intente aumentar el limite que tengo pero no me fue posible ya que me solicitan crear un token con el soporte técnico, sin embargo, al usar una maquina con otras características aparentemente no me genero un gran costo como pensaba que seria.

El cliente RMI ejecutará en una máquina virtual con Ubuntu en Azure (nodo 0). El servidor RMI ejecutará en tres máquinas virtuales (nodo 1, nodo 2 y nodo 3) con Ubuntu en Azure. El programa rmiregistry ejecutará en cada nodo donde ejecute el servidor RMI. El nodo 1 calculará los productos C1, C2 y C3, el nodo 2 calculará los productos C4, C5 y C6, y el nodo 3 calculará los productos C7, C8 y C9, antes de probar nuestro programa necesitamos hacer uso de las IP privadas de los nodos 1, 2 y 3, esto implica crear 3 lookup para cada nodo, esto para poder distribuir el calculo de las matrices ya que el nodo 1 calculará los productos C1, C2 y C3, el nodo 2 calculará los productos C4, C5 y C6, y el nodo 3 calculará los productos C7, C8 y C9.

Una vez creadas las maquinas virtuales se procedió a hacer lo que se muestra en las figuras de 7 y 8 por las 4 maquinas que tenemos, usamos como ejemplo el nodo 0



```
C:\Users\josue>ssh edmundojm@52.142.55.103
The authenticity of host '52.142.55.103 (52.142.55.103)' can't be established.
ECDSA key fingerprint is SHA256:t9ZfEvFht/LGibEiiXOZbH/+FrXVGzMCYc3CDKV4p0Q.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.142.55.103' (ECDSA) to the list of known hosts.
edmundojm@52.142.55.103's password:
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.8.0-1042-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:   https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

System information as of Sun Oct 17 18:13:28 UTC 2021

System load: 0.1          Processes:            121
Usage of /:  4.5% of 28.9GB Users logged in:          0
Memory usage: 27%        IPv4 address for eth0: 10.0.0.4
Swap usage:  0%

1 update can be applied immediately.
To see these additional updates run: apt list --upgradable

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

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

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

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

edmundojm@JR2019630428-0:~$ sudo apt install default-jre
Reading package lists... Done
Building dependency tree
```

Figura 7: Conexión vía ssh a la maquina virtual que hará como nodo 0.

```

edmundojm@JR2019630428-0: ~
Reading package lists... Done
Building dependency tree
Reading state information... Done
14 packages can be upgraded. Run 'apt list --upgradable' to see them.
edmundojm@JR2019630428-0:~$ sudo apt install default-jre
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  at-spi2-core ca-certificates-java default-jre-headless fontconfig-config fonts-dejavu-core fonts-dejavu-extra java-common libatk-bridge2.0-0
  libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0 libavahi-client3 libavahi-common-data libavahi-common3 libcups2
  libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libfontconfig1 libfontenc1 libgl1 libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0
  libglx0 libgraphite2-3 libharfbuzz0b libice6 libjpeg-turbo8 libjpeg8 liblcms2-2 libl1vm12 libpciaccess0 libpcsclite1 libsensors-config libsensors5 libsm6
  libvulkan1 libwayland-client0 libx11-xcb1 libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-randr0 libxcb-shape0 libxcb-shm0
  libxcb-sync1 libxcb-xf86vm0 libxcomposite1 libxcursor1 libxdamage1 libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxpm4 libxrandr2 libxrender1 libxshmfence1 libxt6
  libxtst6 libxxf86dga1 libxxf86vm1 mesa-vulkan-drivers openjdk-11-jre openjdk-11-jre-headless x11-common x11-utils
Suggested packages:
  cups-common liblcms2-utils pscd lm-sensors libnss-mdns fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei | fonts-wqy-zenhei fonts-indic
  mesa-utils
The following NEW packages will be installed:
  at-spi2-core ca-certificates-java default-jre-headless fontconfig-config fonts-dejavu-core fonts-dejavu-extra java-common libatk-bridge2.0-0
  libatk-wrapper-java libatk-wrapper-java-jni libatk1.0-0 libatk1.0-data libatspi2.0-0 libavahi-client3 libavahi-common-data libavahi-common3 libcups2
  libdrm-amdgpu1 libdrm-intel1 libdrm-nouveau2 libdrm-radeon1 libfontconfig1 libfontenc1 libgl1 libgl1-mesa-dri libglapi-mesa libglvnd0 libglx-mesa0
  libglx0 libgraphite2-3 libharfbuzz0b libice6 libjpeg-turbo8 libjpeg8 liblcms2-2 libl1vm12 libpciaccess0 libpcsclite1 libsensors-config libsensors5 libsm6
  libvulkan1 libwayland-client0 libx11-xcb1 libxaw7 libxcb-dri2-0 libxcb-dri3-0 libxcb-glx0 libxcb-present0 libxcb-randr0 libxcb-shape0 libxcb-shm0
  libxcb-sync1 libxcb-xf86vm0 libxcomposite1 libxcursor1 libxdamage1 libxfixes3 libxft2 libxi6 libxinerama1 libxkbfile1 libxmu6 libxpm4 libxrandr2 libxrender1 libxshmfence1 libxt6
  libxtst6 libxxf86dga1 libxxf86vm1 mesa-vulkan-drivers openjdk-11-jre openjdk-11-jre-headless x11-common x11-utils
0 upgraded, 77 newly installed, 0 to remove and 14 not upgraded.
Need to get 15.4 MB/77.7 MB of archives.
After this operation, 648 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 http://azure.archive.ubuntu.com/ubuntu focal-updates/main amd64 libglapi-mesa amd64 21.0.3-0ubuntu0.3~20.04.3 [26.8 kB]
Get:2 http://azure.archive.ubuntu.com/ubuntu focal-updates/main amd64 libgl1-mesa-dri amd64 21.0.3-0ubuntu0.3~20.04.3 [10.5 MB]
Get:3 http://azure.archive.ubuntu.com/ubuntu focal-updates/main amd64 libglx-mesa0 amd64 21.0.3-0ubuntu0.3~20.04.3 [137 kB]
Get:4 http://azure.archive.ubuntu.com/ubuntu focal-updates/main amd64 mesa-vulkan-drivers amd64 21.0.3-0ubuntu0.3~20.04.3 [4729 kB]
Fetched 15.4 MB in 0s (53.1 MB/s)
Extracting templates from packages: 100%
Selecting previously unselected package libatspi2.0-0:amd64.
Reading database ... 85%

```

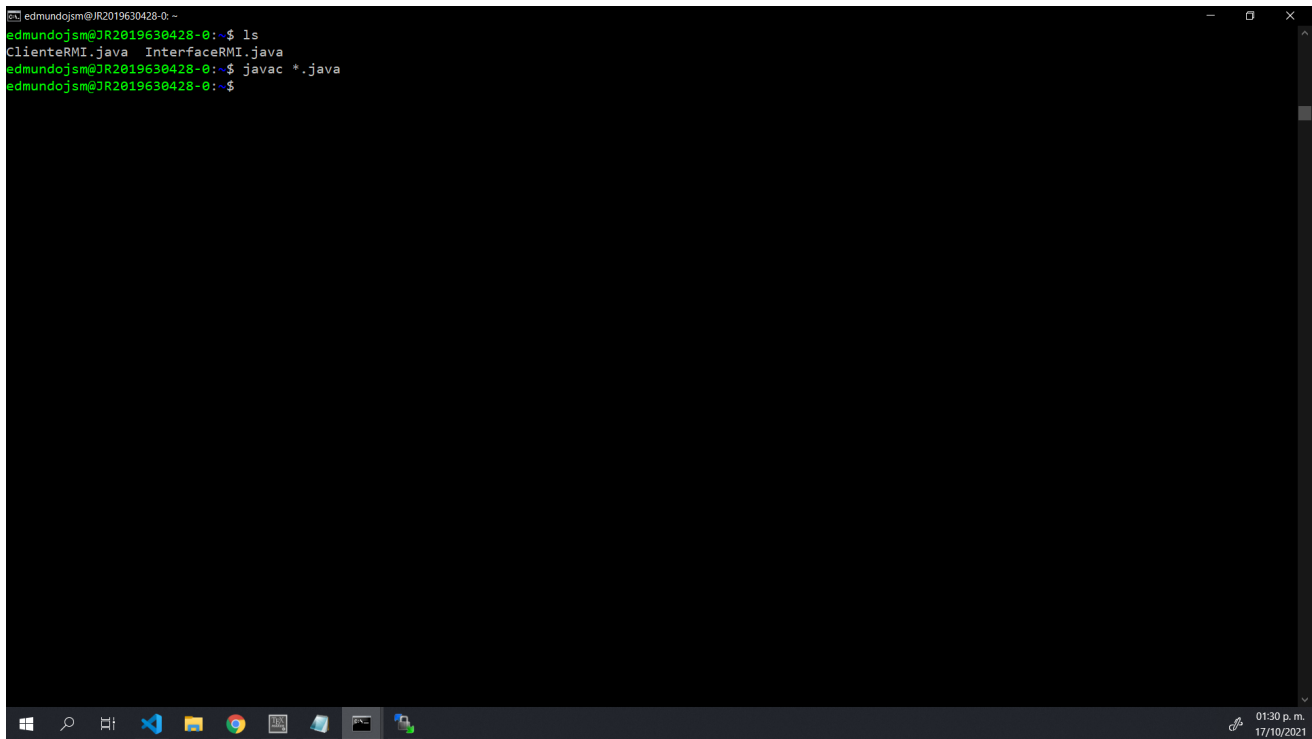
Figura 8: Instalación del jdk en el nodo 0.

Una vez configurado cada maquina virtual se procedió a enviar los archivos por medio de WinSCP el cual solo es necesario arrastrar los archivos a usar a la maquina virtual y eso es todo, una vez copiado los archivos correspondientes a cada nodo procedemos a la compilación del código.

## 2.1. Compilación del código

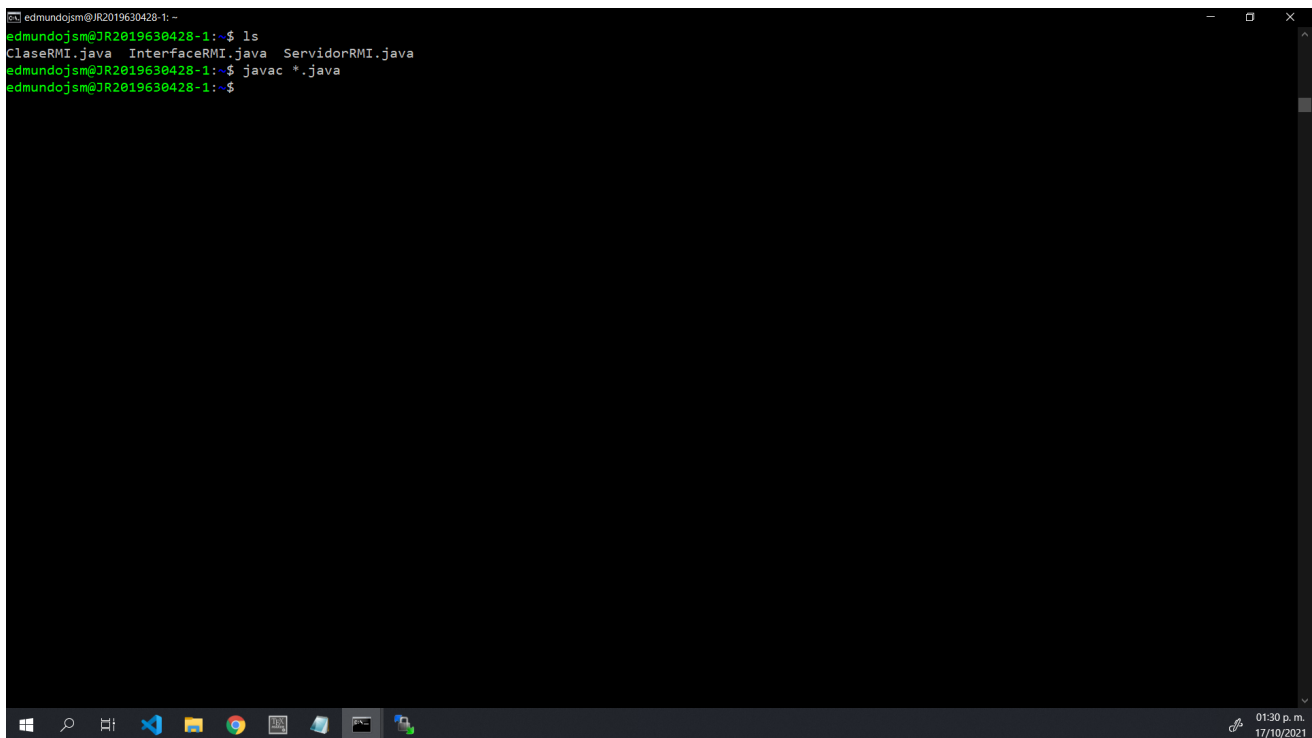
En la figura 9 podemos ver como la compilación del archivo ClienteRMI.java e InterfaceRMI.java en el nodo 0, ademas de que en la figura 10 podemos ver la compilación de los archivos InterfaceRMI.java, ServidorRMI.java y ClaseRMI.java en el nodo 1 esta compilación se hace de manera exitosa y sin ningún error alguno, esto desde la terminal de Ubuntu de nuestras maquinas virtuales, recordar que lo que se hizo en el nodo 1 se repite en los nodos 2 y 3.





```
edmundojm@JR2019630428-0: ~$ ls
ClienteRMI.java  InterfaceRMI.java
edmundojm@JR2019630428-0: ~$ javac *.java
edmundojm@JR2019630428-0: ~$
```

Figura 9: Compilación del código ClienteRMI.java e InterfaceRMI.java en el nodo 0.

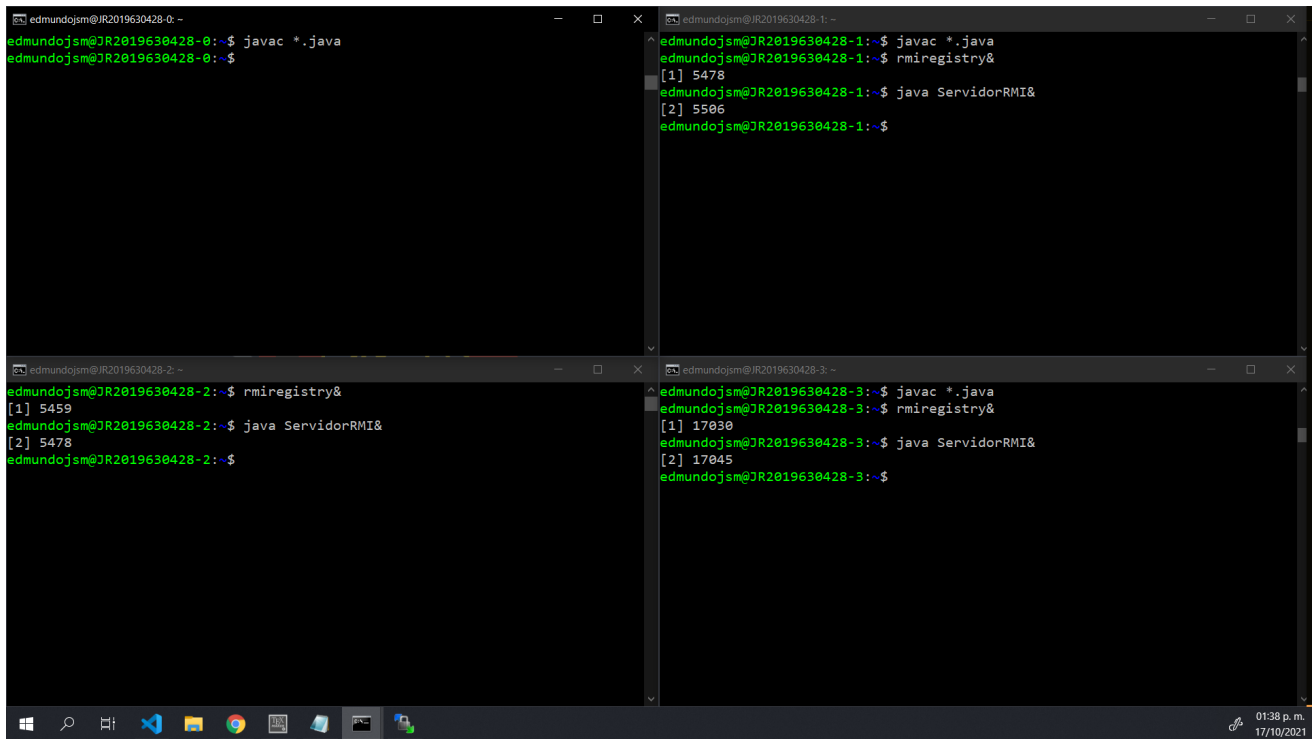


```
edmundojm@JR2019630428-1: ~$ ls
ClaseRMI.java  InterfaceRMI.java  ServidorRMI.java
edmundojm@JR2019630428-1: ~$ javac *.java
edmundojm@JR2019630428-1: ~$
```

Figura 10: Compilación de los archivos InterfaceRMI.java, ServidorRMI.java y ClaseRMI.java en el nodo 1.

## 2.2. Ejecución del programa

Antes de ir con las pruebas necesarias para la entrega de esta tarea en la imagen 11 podemos ver la ejecución de los servidores RMI y rmiregistry en las terminales de los nodos 1, 2 y 3.



```
edmundojasm@JR2019630428-0:~$ javac *.java
edmundojasm@JR2019630428-0:~$

edmundojasm@JR2019630428-1:~$ javac *.java
edmundojasm@JR2019630428-1:~$ rmiregistry&
[1] 5478
edmundojasm@JR2019630428-1:~$ java ServidorRMI&
[2] 5506
edmundojasm@JR2019630428-1:~$

edmundojasm@JR2019630428-2:~$ rmiregistry&
[1] 5459
edmundojasm@JR2019630428-2:~$ java ServidorRMI&
[2] 5478
edmundojasm@JR2019630428-2:~$

edmundojasm@JR2019630428-3:~$ javac *.java
edmundojasm@JR2019630428-3:~$ rmiregistry&
[1] 17030
edmundojasm@JR2019630428-3:~$ java ServidorRMI&
[2] 17045
edmundojasm@JR2019630428-3:~$
```

Figura 11: Ejecución del programa ServidorRMI en los nodos 1, 2 y 3.

### 2.2.1. N=9

Una vez compilado nuestro programa en cada maquina virtual se procede a la ejecución del cliente RMI y como vemos en la figura 12 podemos ver como se nos imprime el valor de la matriz A, la matriz B antes de ser transpuesta, la matriz C y el valor de checksum de la matriz c.

```

edmundojm@JR2019630428-0:~$ java ClienteRMI
Valor Matriz A
0.0  4.0  8.0  12.0  16.0  20.0  24.0  28.0  32.0
1.0  5.0  9.0  13.0  17.0  21.0  25.0  29.0  33.0
2.0  6.0  10.0  14.0  18.0  22.0  26.0  30.0  34.0
3.0  7.0  11.0  15.0  19.0  23.0  27.0  31.0  35.0
4.0  8.0  12.0  16.0  20.0  24.0  28.0  32.0  36.0
5.0  9.0  13.0  17.0  21.0  25.0  29.0  33.0  37.0
6.0  10.0  14.0  18.0  22.0  26.0  30.0  34.0  38.0
7.0  11.0  15.0  19.0  23.0  27.0  31.0  35.0  39.0
8.0  12.0  16.0  20.0  24.0  28.0  32.0  36.0  40.0

Valor Matriz B
0.0  -4.0  -8.0  -12.0  -16.0  -20.0  -24.0  -28.0  -32.0
1.0  -3.0  -7.0  -11.0  -15.0  -19.0  -23.0  -27.0  -31.0
2.0  -2.0  -6.0  -10.0  -14.0  -18.0  -22.0  -26.0  -30.0
3.0  -1.0  -5.0  -9.0  -13.0  -17.0  -21.0  -25.0  -29.0
4.0  0.0  -4.0  -8.0  -12.0  -16.0  -20.0  -24.0  -28.0
5.0  1.0  -3.0  -7.0  -11.0  -15.0  -19.0  -23.0  -27.0
6.0  2.0  -2.0  -6.0  -10.0  -14.0  -18.0  -22.0  -26.0
7.0  3.0  -1.0  -5.0  -9.0  -13.0  -17.0  -21.0  -25.0
8.0  4.0  0.0  -4.0  -8.0  -12.0  -16.0  -20.0  -24.0

Valor Matriz C
816.0  240.0  -336.0  -912.0  -1488.0  -2064.0  -2640.0  -3216.0  -3792.0
852.0  240.0  -372.0  -984.0  -1596.0  -2208.0  -2820.0  -3432.0  -4044.0
888.0  240.0  -408.0  -1056.0  -1704.0  -2352.0  -3000.0  -3648.0  -4296.0
924.0  240.0  -444.0  -1128.0  -1812.0  -2496.0  -3180.0  -3864.0  -4548.0
960.0  240.0  -480.0  -1200.0  -1920.0  -2640.0  -3360.0  -4080.0  -4800.0
996.0  240.0  -516.0  -1272.0  -2028.0  -2784.0  -3540.0  -4296.0  -5052.0
1032.0  240.0  -552.0  -1344.0  -2136.0  -2928.0  -3720.0  -4512.0  -5304.0
1068.0  240.0  -588.0  -1416.0  -2244.0  -3072.0  -3900.0  -4728.0  -5556.0
1104.0  240.0  -624.0  -1488.0  -2352.0  -3216.0  -4080.0  -4944.0  -5808.0

Checksum: -155520.0
edmundojm@JR2019630428-0:~$

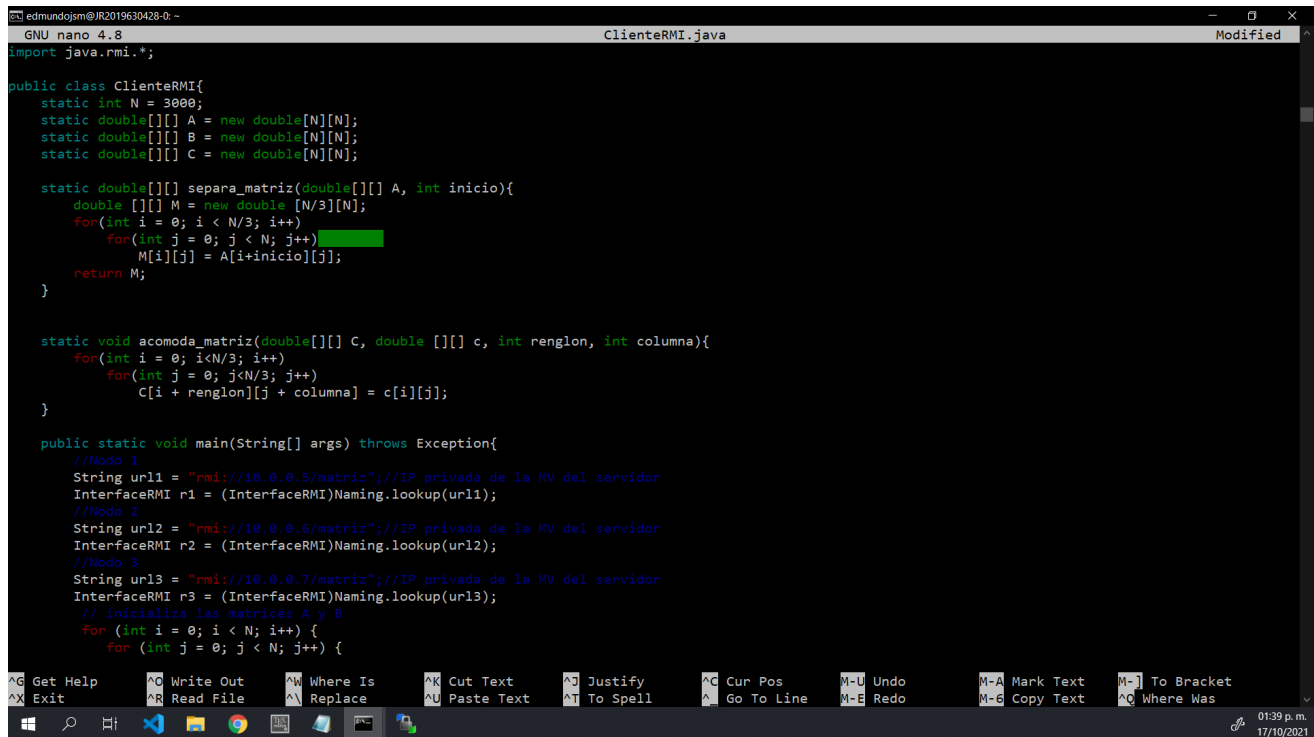
```

Figura 12: Valor de checksum para  $N=9$  y valores de las matrices A, B y c.

Ahora veamos el siguiente caso con  $N=3000$ .

### 2.2.2. $N=3000$

Para este caso se tuvo que modificar el programa asignando 3000 esto por medio del uso del comando nano ClienteRMI.java en el nodo 0 para poder cambiar el valor de N como vemos en la figura 13



```
GNU nano 4.8 ClienteRMI.java
import java.rmi.*;

public class ClienteRMI{
    static int N = 3000;
    static double[][] A = new double[N][N];
    static double[][] B = new double[N][N];
    static double[][] C = new double[N][N];

    static double[][] separa_matriz(double[][] A, int inicio){
        double [][] M = new double [N/3][N];
        for(int i = 0; i < N/3; i++){
            for(int j = 0; j < N; j++){
                M[i][j] = A[i+inicio][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];
            }
        }
    }

    public static void main(String[] args) throws Exception{
        //Nodo 1
        String url1 = "rmi://10.0.0.5/matriz"; //IP privada de la MV del servidor
        InterfaceRMI r1 = (InterfaceRMI)Naming.lookup(url1);
        //Nodo 2
        String url2 = "rmi://10.0.0.6/matriz"; //IP privada de la MV del servidor
        InterfaceRMI r2 = (InterfaceRMI)Naming.lookup(url2);
        //Nodo 3
        String url3 = "rmi://10.0.0.7/matriz"; //IP privada de la MV del servidor
        InterfaceRMI r3 = (InterfaceRMI)Naming.lookup(url3);
        // Inicializa las matrices A y B
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {

```

Figura 13: Modificación del valor de N de 9 a 3000.

Antes de ver el resultado de valor del checksum calculado sucedió algo interesante y es que como vemos en la figura 14 nos salio un error el cual nos indica que no tenemos suficiente cantidad de memoria para poder separar la matriz, para poder solucionar esto, la única solución era aumentar la memoria de la maquina virtual como podemos ver en la figura 15.

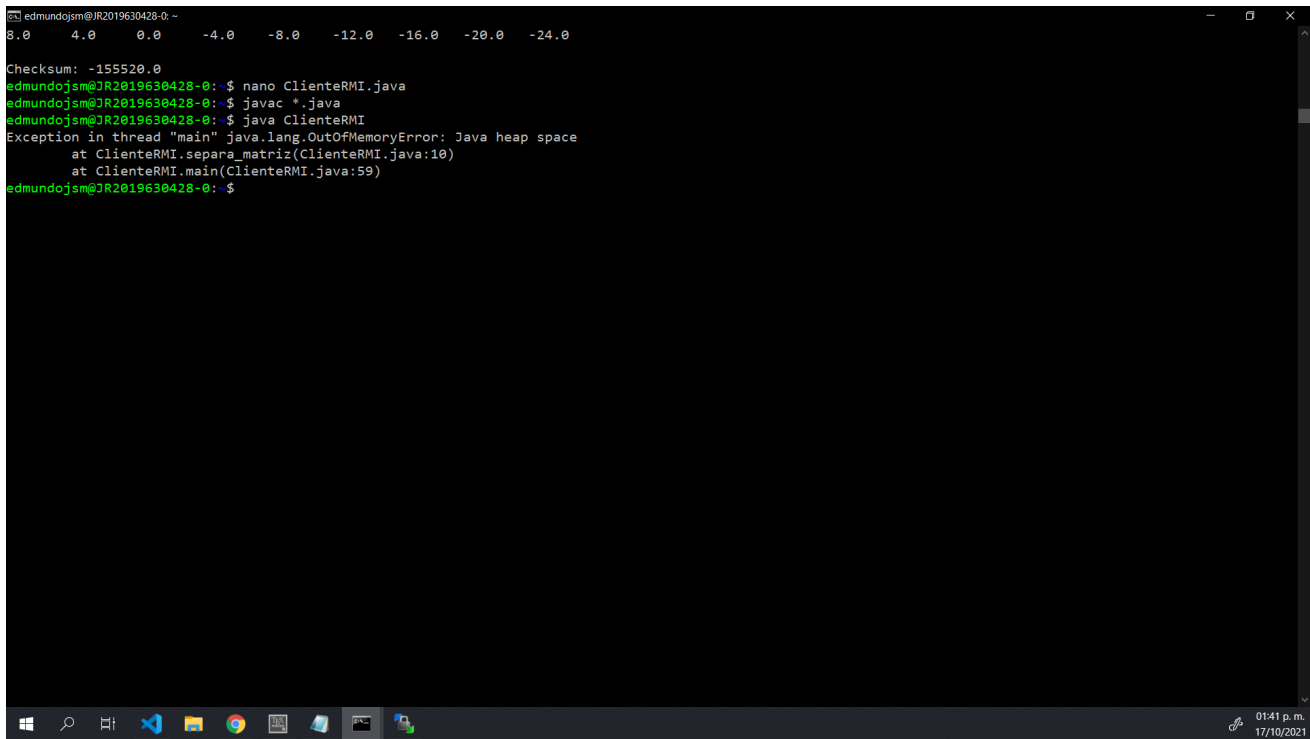


Figura 14: Error al ejecutar el programa.

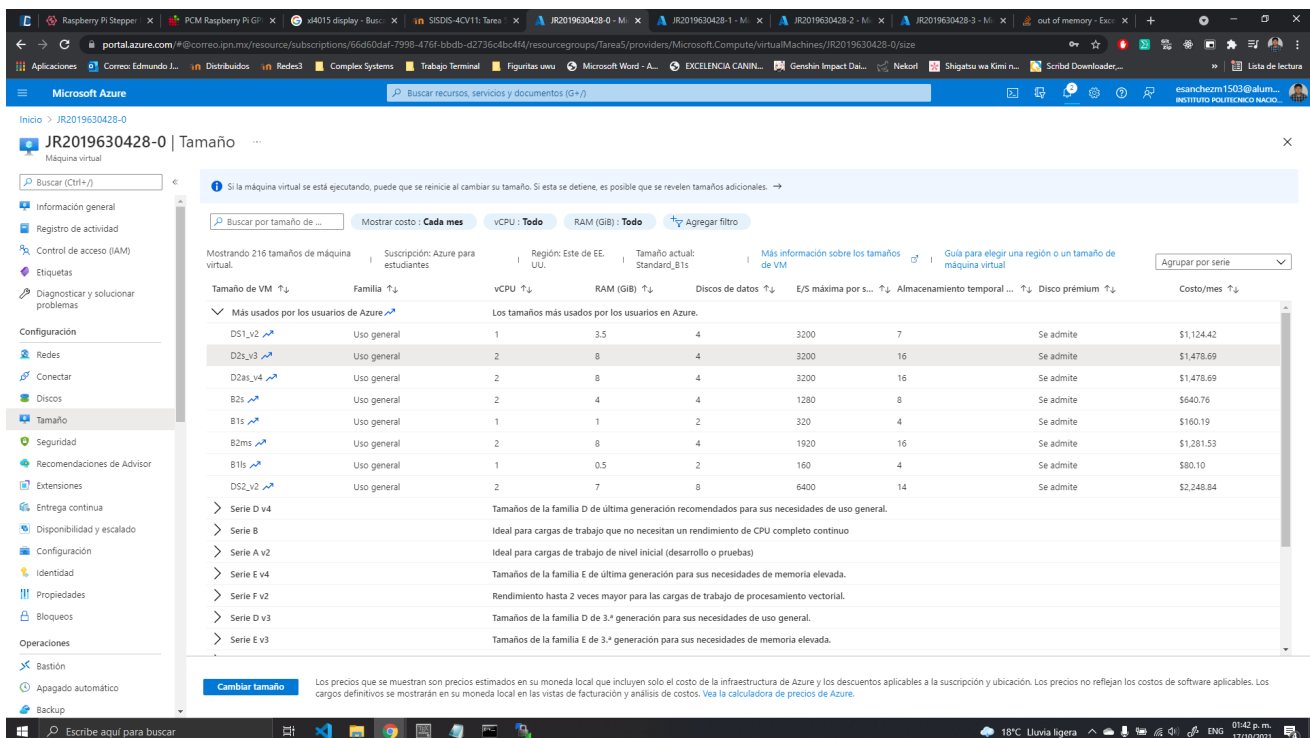
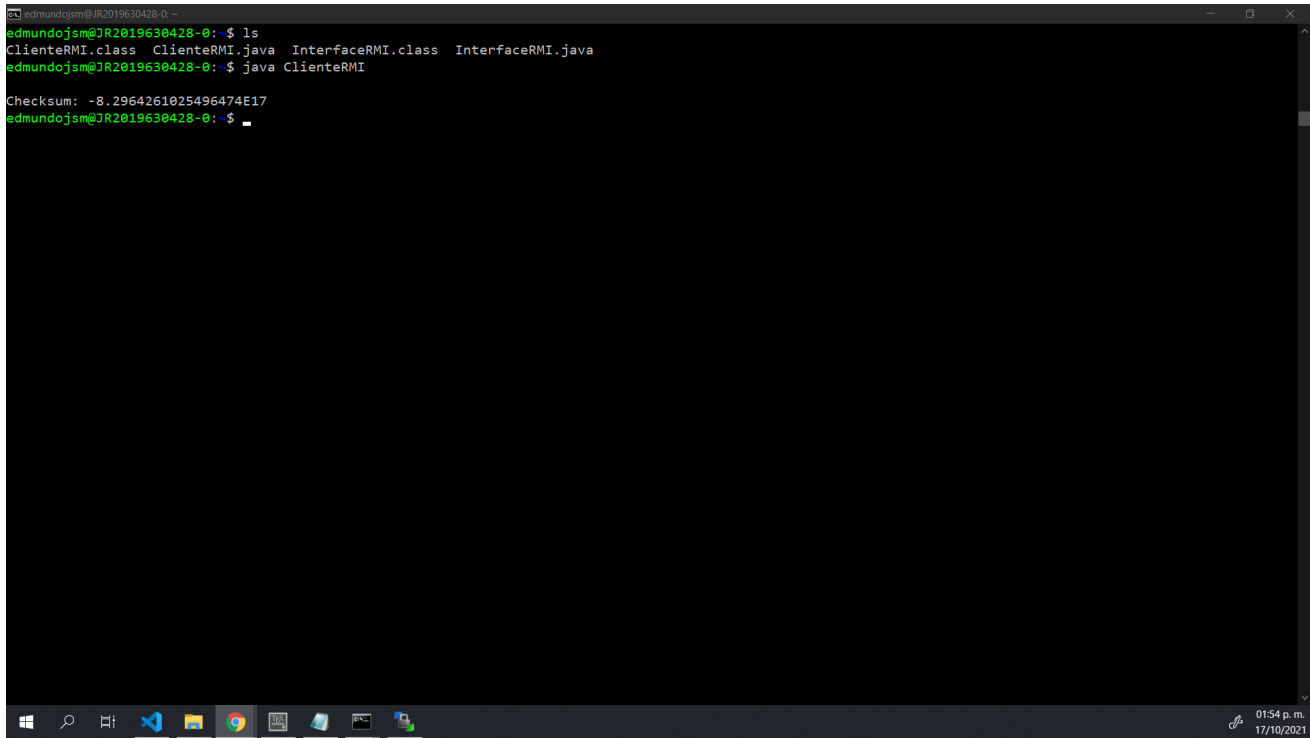


Figura 15: Cambiando el tamaño de la maquina virtual del nodo 0.

Si vemos la figura 6 en donde podemos ver la cantidad de memoria RAM que poseía el nodo 0 es de 1 RAM, al hacer la actualización a 8 de RAM ya no tenemos ese problema, sin embargo puede que nos estemos preguntando, entonces porque en los servidores no salio ningún error y es que al dividir

las matrices en 3 nos facilita el manejo del tamaño de 3000 elementos y esto provoca que las maquinas restantes no sufran de este error, sin embargo, es evidente que la ejecución sera lenta.



```
edmundojm@JR2019630428-0:~$ ls
ClienteRMI.class  ClienteRMI.java  InterfaceRMI.class  InterfaceRMI.java
edmundojm@JR2019630428-0:~$ java ClienteRMI

Checksum: -8.2964261025496474E17
edmundojm@JR2019630428-0:~$
```

Figura 16: Valor de checksum para N=3000.

Como vemos solo se nos despliega el valor del checksum de la matriz C, que es como se nos pide en la tarea.

### 3. Conclusiones

En esta practica implementamos un programa similar a la de la tarea 3, pero en esta ocasión utilizamos Java RMI. Es mucho mas sencillo hacerlo de esta forma ya que el envío de los pedazos de las matrices no nos es relevante ya que ahora invocamos métodos de manera remota. Para hacer las pruebas creamos 4 maquinas virtuales de Ubuntu Server, en Microsoft Azure. En este caso la implementación de las maquinas es muy sencilla y la conexión a través de ssh facilita mucho las cosas, ademas de que usando WinSCP nos facilita aun mas el paso de los archivos ya que es una forma muy visual el paso de los archivos de la maquina local con las maquinas virtuales. Finalmente me pareció interesante esta forma de usar el computo distribuido, ya que nos da una facilidad y agilidad al implementar este tipo de programas.