



INSTITUTO POLITÉCNICO NACIONAL  
ESCUELA SUPERIOR DE CÓMPUTO

*”Tarea 10. Replicación de un servidor en la  
nube”*

Alumno:

Lara Cázares Jaime Arturo

Materia:

Desarrollo de sistemas distribuidos

Grupo:

4CM3

Profesor: Pineda Guerrero Carlos

# Índice general

1.	Descripción de la tarea . . . . .	3
2.	Capturas del procedimiento . . . . .	3
2.1.	Creación de maquinas virtuales . . . . .	3
2.2.	Abrir puerto en la máquina virtual 1 . . . . .	3
2.3.	Abrir puerto en la máquina virtual 2 . . . . .	4
2.4.	Conectar a la máquina virtual 1 con <i>Putty</i> . . . . .	4
2.5.	Instalar <i>JDK-8</i> en la máquina virtual 1 . . . . .	5
2.6.	Envío de los archivos <i>Servidor2.java</i> y <i>SimpleProxyServer.java</i> a la máquina virtual 1 . . . . .	5
2.7.	Editar <i>Servidor2.java</i> en la máquina virtual 1 . . . . .	6
2.8.	Compilar de los archivos <i>Servidor2.java</i> y <i>SimpleProxyServer.java</i> en la máquina virtual 1 . . . . .	7
2.9.	Conectar a la máquina virtual 2 con <i>Putty</i> . . . . .	8
2.10.	Instalar <i>JDK-8</i> en la máquina virtual 2 . . . . .	8
2.11.	Envío del archivo <i>Servidor2.java</i> a la máquina virtual 2 . . . . .	8
2.12.	Editar <i>Servidor2.java</i> en la máquina virtual 2 . . . . .	9
2.13.	Compilar el archivo <i>Servidor2.java</i> en la máquina virtual 2 . . . . .	10
2.14.	Ejecutar el archivo <i>Servidor2.java</i> en la máquina virtual 2 . . . . .	10
2.15.	Ejecutar el archivo <i>Servidor2.java</i> en la máquina virtual 1 . . . . .	11
2.16.	Ejecutar el archivo <i>SimpleProxyServer.java</i> en la máquina virtual 2 . . . . .	11
2.17.	Cliente en windows . . . . .	12
2.17.1.	Editar el programa <i>Servidor2.java</i> para que se conecte a la máquina virtual 1 . . . . .	12
2.17.2.	Compilar el programa <i>Servidor2.java</i> . . . . .	12
2.17.3.	Ejecutar el programa <i>Servidor2.java</i> . . . . .	13
3.	Conclusiones . . . . .	14

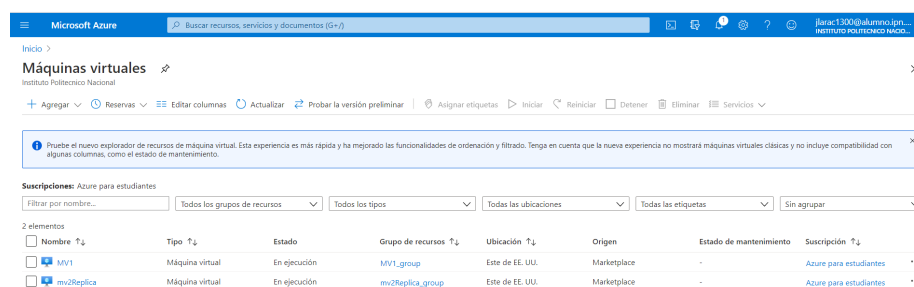
## 1. Descripción de la tarea

En esta tarea se realizará la replicación de un servidor en la nube, para ello se necesitan crear 2 máquinas virtuales en la plataforma *Azure* para lograr que a través de un servidor proxy las dos máquinas virtuales funciones de igual forma (en este caso recibir y contestar a un cliente) y se mantenga el servicio por si alguna llegase a fallar.

## 2. Capturas del procedimiento

### 2.1. Creación de maquinas virtuales

En esta práctica se necesita utilizar 2 maquinas virtuales sobre la plataforma de **Azure** con Ubuntu 18, 1 GB de RAM y disco HDD estándar. La Figura 1 muestra estas máquinas virtuales.



Nombre ↑↓	Tipo ↑↓	Estado	Grupo de recursos ↑↓	Ubicación ↑↓	Origen	Estado de mantenimiento	Suscripción ↑↓
MV1	Máquina virtual	En ejecución	MV1_group	Este de EE. UU.	Marketplace	-	Azure para estudiantes
mv2Replica	Máquina virtual	En ejecución	mv2Replica_group	Este de EE. UU.	Marketplace	-	Azure para estudiantes

Figura 1: Máquinas virtuales.

### 2.2. Abrir puerto en la máquina virtual 1

Ahora es necesario agregar la "regla de puerto de entrada" en la máquina virtual 1, para esta práctica se utiliza el puerto **50,000** con protocolo de comunicación **TCP**. En la Figura 2 se observa este puerto abierto.

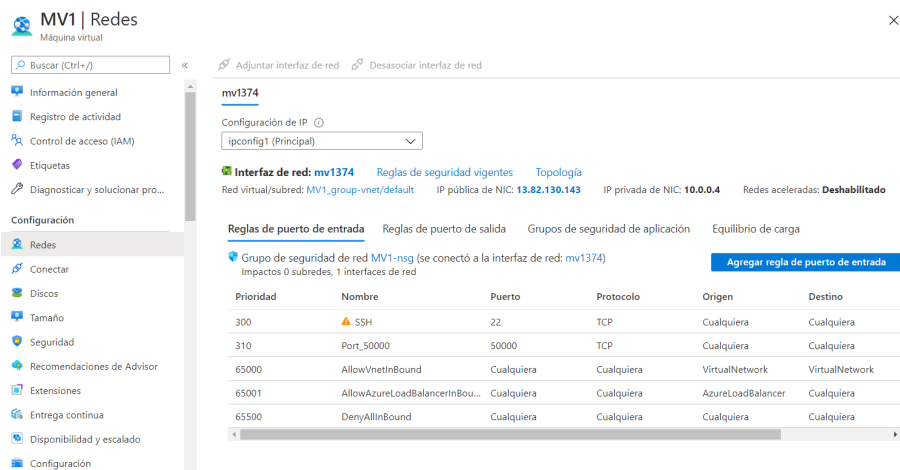


Figura 2: Puerto 50,000 (TCP) en máquina virtual 1.

### 2.3. Abrir puerto en la máquina virtual 2

De igual forma que en la máquina virtual 1, se debe agregar la "regla de puerto de entrada" en la máquina virtual 2, se utiliza también el puerto **50,000** con protocolo de comunicación **TCP**. En la Figura 3 se observa este puerto abierto.

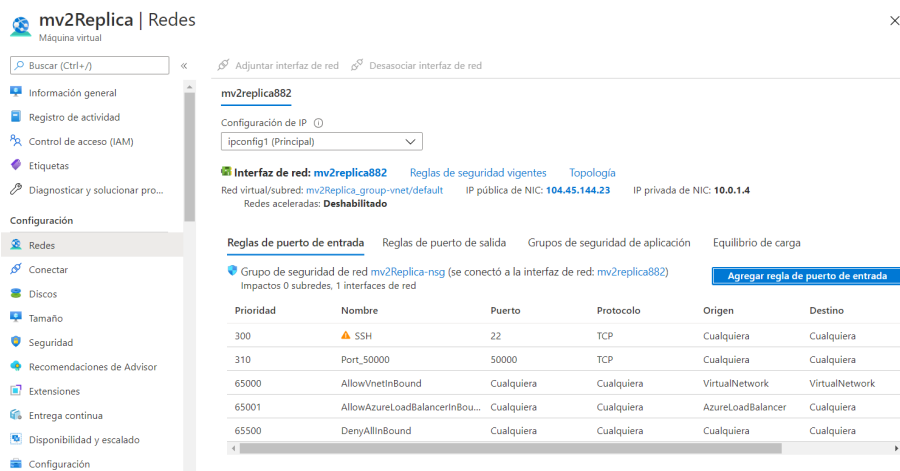
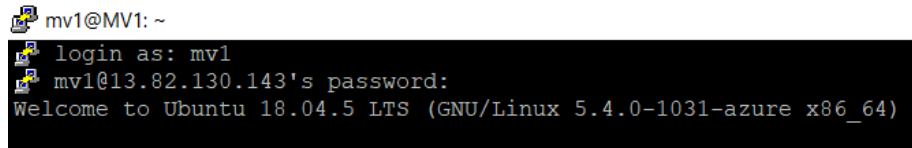


Figura 3: Puerto 50,000 (TCP) en máquina virtual 2.

### 2.4. Conectar a la máquina virtual 1 con Putty

Ahora es tiempo de conectarse a la máquina virtual 1, para ello se utiliza el programa *Putty* el cual requiere la dirección IP pública de la máquina virtual,

nombre y contraseña. La Figura 4 muestra la terminal de *Putty* ya conectada a la máquina virtual 1.



```
mv1@MV1: ~  
login as: mv1  
mv1@13.82.130.143's password:  
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1031-azure x86_64)
```

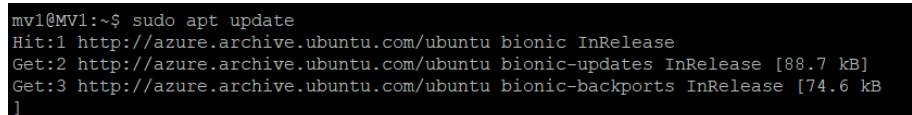
Figura 4: Conexión a la máquina virtual 1 utilizando *Putty*.

## 2.5. Instalar *JDK-8* en la máquina virtual 1

Una vez conectados es necesario instalar *JDK-8* para poder compilar y ejecutar programas de *Java*, para ello se utilizan los siguientes comandos:

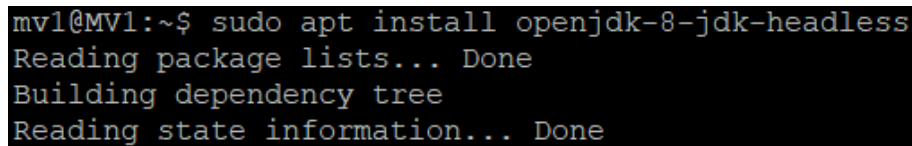
```
sudo apt update  
sudo apt install openjdk-8-jdk-headless
```

La Figura 5 muestra la actualización (update), mientras que la Figura 6 la instalación de *JDK-8*.



```
mv1@MV1:~$ sudo apt update  
Hit:1 http://azure.archive.ubuntu.com/ubuntu bionic InRelease  
Get:2 http://azure.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]  
Get:3 http://azure.archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]  
]
```

Figura 5: Update a la máquina virtual 1.



```
mv1@MV1:~$ sudo apt install openjdk-8-jdk-headless  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done
```

Figura 6: Instalación de *JDK-8* en la máquina virtual 1.

## 2.6. Envío de los archivos *Servidor2.java* y *SimpleProxyServer.java* a la máquina virtual 1

Ahora se deben enviar los archivos *Servidor2.java* y *SimpleProxyServer.java* a la máquina virtual 1, para esto se utiliza el programa *"psftp.exe"* que esta incluido en la suit de instalación de *Putty*. La Figura 7 refleja el procedimiento el cual consiste en establecer la conexión con el comando *open* seguido de *"nombre@IP-publica"*, justo después se solicitará la contraseña para establecer la conexión. Con el comando *put* seguido del nombre del archivo, se envía el

archivo a la máquina virtual (hay que asegurarse de estar posicionado en el directorio en que está el archivo a enviar).

```
psftp> open mv1@13.82.130.143
Using username "mv1".
mv1@13.82.130.143's password:
Remote working directory is /home/mv1
psftp> !ls
LICENCE      Servidor2.java.txt      plink.exe    putty-64bit-0.74-installer.msi  puttygen.exe
README.txt   SimpleProxyServer.java  pscp.exe    putty.chm                       website.url
Servidor2.java  pageant.exe            psftp.exe   putty.exe
psftp> put Servidor2.java
local:Servidor2.java => remote:/home/mv1/Servidor2.java
psftp> put SimpleProxyServer.java
local:SimpleProxyServer.java => remote:/home/mv1/SimpleProxyServer.java
psftp>
```

Figura 7: Envío de los archivos *Servidor2.java* y *SimpleProxyServer.java* a la máquina virtual 1 utilizando "psftp.exe".

## 2.7. Editar *Servidor2.java* en la máquina virtual 1

Es necesario editar el método "main" del archivo *Servidor2.java*, la modificación es la siguiente línea de código:

```
ServerSocket servidor = new ServerSocket(50001);
```

Se utiliza el editor "vi" para hacer el cambio del archivo en la máquina virtual 1, esto se observa en la Figura 8.

```

mv1@MV1:~$ vi Servidor2.java
    DataOutputStream salida = new DataOutputStream(conexion.getOutputStream());
    DataInputStream entrada = new DataInputStream(conexion.getInputStream());

    // recibe un entero de 32 bits
    int n = entrada.readInt();
    System.out.println(n);

    // recibe un numero punto flotante
    double x = entrada.readDouble();
    System.out.println(x);

    // recibe una cadena
    byte[] buffer = new byte[4];
    read(entrada,buffer,0,4);
    System.out.println(new String(buffer,"UTF-8"));

    // envia una cadena
    salida.write("HOLA".getBytes());

    // recibe 5 numeros punto flotante
    byte[] a = new byte[5*8];
    read(entrada,a,0,5*8);
    ByteBuffer b = ByteBuffer.wrap(a);
    for (int i = 0; i < 5; i++)
        System.out.println(b.getDouble());

    salida.close();
    entrada.close();
    conexion.close();
}
catch (Exception e)
{
    System.err.println(e.getMessage());
}
}

public static void main(String[] args) throws Exception
{
    ServerSocket servidor = new ServerSocket(50001);

    for (;;)
    {
        Socket conexion = servidor.accept();
        Worker w = new Worker(conexion);
        w.start();
    }
}

```

Figura 8: Modificación del archivo *Servidor2.java* en la máquina virtual 1.

## 2.8. Compilar de los archivos *Servidor2.java* y *SimpleProxyServer.java* en la máquina virtual 1

Es momento de compilar los archivos *Servidor2.java* y *SimpleProxyServer.java* en la máquina virtual 1, esto se presenta en la Figura 9.

```

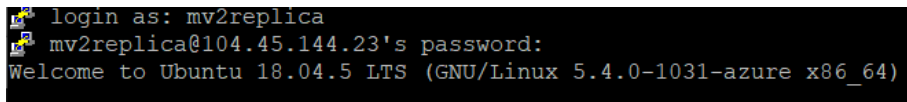
mv1@MV1:~$ javac Servidor2.java
mv1@MV1:~$ javac SimpleProxyServer.java
mv1@MV1:~$

```

Figura 9: Compilar los archivos *Servidor2.java* y *SimpleProxyServer.java* en la máquina virtual 1.

## 2.9. Conectar a la máquina virtual 2 con *Putty*

Ahora es tiempo de conectarse a la máquina virtual 2, de igual forma que con la máquina virtual 1 se utiliza el programa *Putty* el cual requiere la dirección IP pública de la máquina virtual, nombre y contraseña. La Figura 10 muestra la terminal de *Putty* ya conectada a la máquina virtual 2.



```
login as: mv2replica
mv2replica@104.45.144.23's password:
Welcome to Ubuntu 18.04.5 LTS (GNU/Linux 5.4.0-1031-azure x86_64)
```

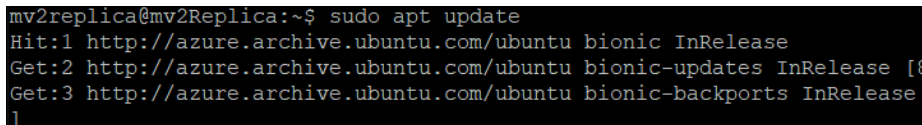
Figura 10: Conexión a la máquina virtual 2 utilizando *Putty*.

## 2.10. Instalar *JDK-8* en la máquina virtual 2

Una vez conectados es necesario instalar *JDK-8* para poder compilar y ejecutar programas de *Java*, para ello se utilizan los siguientes comandos:

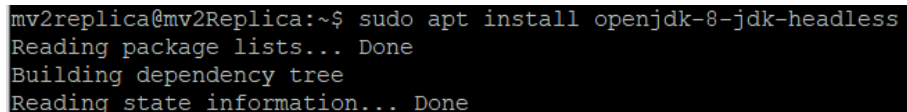
```
sudo apt update
sudo apt install openjdk-8-jdk-headless
```

La Figura 11 muestra la actualización (update), mientras que la Figura 12 la instalación de *JDK-8*.



```
mv2replica@mv2Replica:~$ sudo apt update
Hit:1 http://azure.archive.ubuntu.com/ubuntu bionic InRelease
Get:2 http://azure.archive.ubuntu.com/ubuntu bionic-updates InRelease [6
Get:3 http://azure.archive.ubuntu.com/ubuntu bionic-backports InRelease
]
```

Figura 11: Update a la máquina virtual 2.



```
mv2replica@mv2Replica:~$ sudo apt install openjdk-8-jdk-headless
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

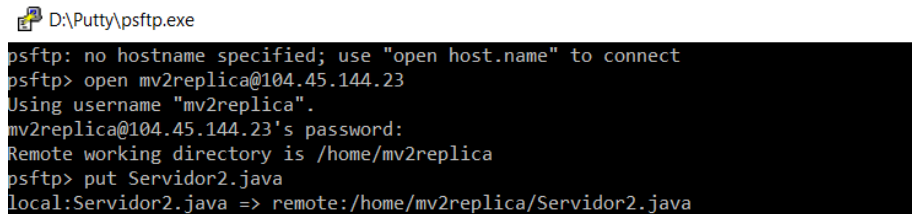
Figura 12: Instalación de *JDK-8* en la máquina virtual 2.

## 2.11. Envío del archivo *Servidor2.java* a la máquina virtual 2

Ahora se debe enviar el archivo *Servidor2.java* a la máquina virtual 3, al igual que en la máquina virtual 1 se utiliza el programa *"psftp.exe"* que esta incluido en la suit de instalación de *Putty*. La Figura 13 refleja el procedimiento el cual consiste en establecer la conexión con el comando *open* seguido de *"nombre@IP-publica"*, justo después se solicitará la contraseña para establecer la conexión.



Con el comando *put* seguido del nombre del archivo, se envía el archivo a la máquina virtual (hay que asegurarse de estar posicionado en el directorio en que está el archivo a enviar).



```
D:\Putty\psftp.exe
psftp: no hostname specified; use "open host.name" to connect
psftp> open mv2replica@104.45.144.23
Using username "mv2replica".
mv2replica@104.45.144.23's password:
Remote working directory is /home/mv2replica
psftp> put Servidor2.java
local:Servidor2.java => remote:/home/mv2replica/Servidor2.java
```

Figura 13: Envío del archivo *Servidor2.java* a la máquina virtual 2 utilizando *psftp.exe*.

## 2.12. Editar *Servidor2.java* en la máquina virtual 2

Es necesario asegurarse de tener establecido el puerto **50000** en el método *main* del archivo *Servidor2.java*, como se muestra en línea de código:

```
ServerSocket servidor = new ServerSocket(50000);
```

La Figura 14 muestra el archivo en la máquina virtual 2.

mv2replica@mv2Replica: ~

```
        salida.close();
        entrada.close();
        conexion.close();
    }
    catch (Exception e)
    {
        System.err.println(e.getMessage());
    }
}

public static void main(String[] args) throws Exception
{
    ServerSocket servidor = new ServerSocket(50000);

    for (;;)
    {
        Socket conexion = servidor.accept();
        Worker w = new Worker(conexion);
        w.start();
    }
}
}mv2replica@mv2Replica:~$
```

Figura 14: Archivo *Servidor2.java* en la máquina virtual 2.

### 2.13. Compilar el archivo *Servidor2.java* en la máquina virtual 2

Es momento de compilar el archivo *Servidor2.java* en la máquina virtual 2, esto se presenta en la Figura 15.

```
mv2replica@mv2Replica:~$ javac Servidor2.java
mv2replica@mv2Replica:~$
```

Figura 15: Compilar el archivo *Servidor2.java* en la máquina virtual 2.

### 2.14. Ejecutar el archivo *Servidor2.java* en la máquina virtual 2

Ahora se debe ejecutar el archivo *Servidor2.java* en la máquina virtual 2, esto se presenta en la Figura 16.

```
mv2replica@mv2Replica:~$ java Servidor2&
[1] 4251
mv2replica@mv2Replica:~$
```

Figura 16: Ejecución el archivo *Servidor2.java* en la máquina virtual 2.

### 2.15. Ejecutar el archivo *Servidor2.java* en la máquina virtual 1

Ahora se debe ejecutar el archivo *Servidor2.java* en la máquina virtual 1, esto se presenta en la Figura 17.

```
mv1@MV1:~$ java Servidor2&
[1] 4499
mv1@MV1:~$
```

Figura 17: Ejecución el archivo *Servidor2.java* en la máquina virtual 1.

### 2.16. Ejecutar el archivo *SimpleProxyServer.java* en la máquina virtual 2

Ahora se debe ejecutar el archivo *SimpleProxyServer.java* en la máquina virtual 1, esto se presenta en la Figura 18. La forma de utilizar el servidor proxy es pasando como argumento la dirección IP de la máquina virtual 2, seguido del puerto abierto de la máquina virtual 2 (en este caso 50000), después el puerto abierto de la máquina virtual 1 (en este caso 50000) y por último el puerto en la máquina virtual 1 donde el programa *Servidor2.java* recibe las peticiones (el puerto 50001 no se debe abrir en la máquina virtual 1, ya que el proxy y *Servidor2.java* se comunican mediante loopback). El comando de ejecución sería el siguiente siguiente:

```
java SimpleProxyServer IP-máquina-virtual-2 50000 50000 50001&
```

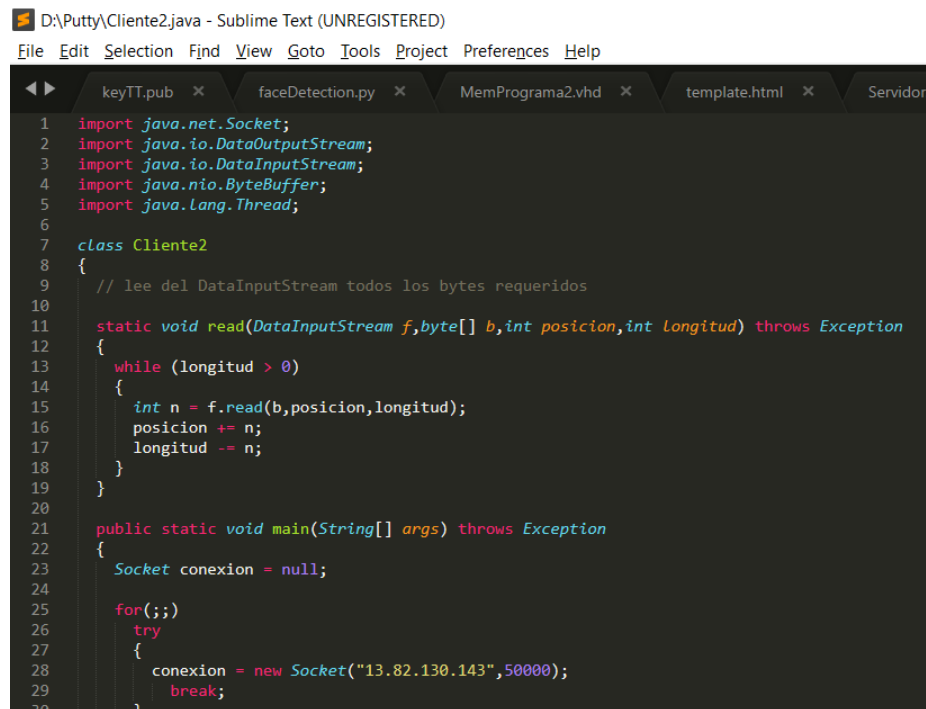
```
mv1@MV1:~$ java SimpleProxyServer 104.45.144.23 50000 50000 50001&
[2] 5220
mv1@MV1:~$ Starting proxy for 104.45.144.23:50000 on port 50000
```

Figura 18: Ejecución del servidor proxy en la máquina virtual 1.

## 2.17. Cliente en windows

### 2.17.1. Editar el programa *Servidor2.java* para que se conecte a la máquina virtual 1

Ahora se debe editar el *Servidor2.java* para que se conecte a la máquina virtual 1, modificando "localhost" por la dirección de IP pública de la máquina virtual 1. En la Figura 19 se muestra la modificación hecha.

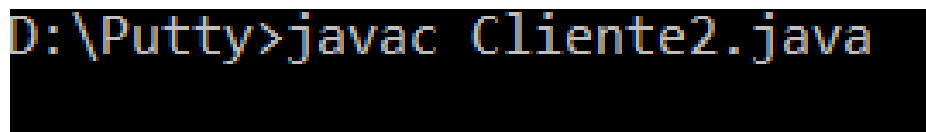
A screenshot of a code editor window titled "D:\Putty\Cliente2.java - Sublime Text (UNREGISTERED)". The editor shows the code for a Java class named "Cliente2". The code includes imports for "java.net.Socket", "java.io.DataOutputStream", "java.io.DataInputStream", "java.nio.ByteBuffer", and "java.lang.Thread". The class "Cliente2" has a static method "read" that takes a "DataInputStream f", a "byte[] b", an "int posicion", and an "int longitud" as parameters, and throws an "Exception". The "read" method contains a "while" loop that reads data from the stream until the "longitud" is reached. The "main" method is also shown, which creates a "Socket" connection to "13.82.130.143" on port "50000".

```
1 import java.net.Socket;
2 import java.io.DataOutputStream;
3 import java.io.DataInputStream;
4 import java.nio.ByteBuffer;
5 import java.lang.Thread;
6
7 class Cliente2
8 {
9     // lee del DataInputStream todos los bytes requeridos
10
11     static void read(DataInputStream f,byte[] b,int posicion,int longitud) throws Exception
12     {
13         while (longitud > 0)
14         {
15             int n = f.read(b,posicion,longitud);
16             posicion += n;
17             longitud -= n;
18         }
19     }
20
21     public static void main(String[] args) throws Exception
22     {
23         Socket conexion = null;
24
25         for(;;)
26         {
27             try
28             {
29                 conexion = new Socket("13.82.130.143",50000);
30                 break;
31             }
32         }
33     }
34 }
```

Figura 19: Edición de *Servidor2.java* para que se conecte a la máquina virtual 1.

### 2.17.2. Compilar el programa *Servidor2.java*

Una vez modificado es tiempo de compilar el archivo *Servidor2.java*. En la Figura 20 se muestra la compilación del archivo en la terminal de windows.

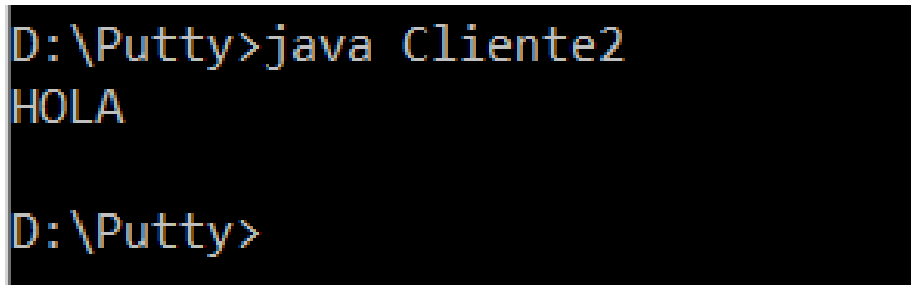
A screenshot of a Windows command prompt window. The prompt shows the command "javac Cliente2.java" being executed in the directory "D:\Putty".

```
D:\Putty>javac Cliente2.java
```

Figura 20: Compilación de *Servidor2.java* en windows.

### 2.17.3. Ejecutar el programa *Servidor2.java*

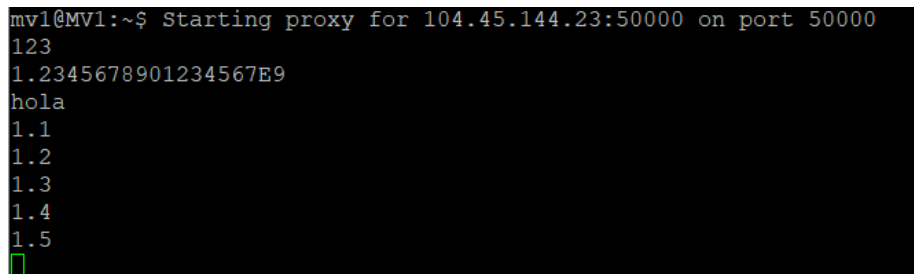
Ahora se debe ejecutar el archivo *Servidor2.java*. En la Figura 21 se muestra la ejecución del archivo en la terminal de windows donde tenemos la respuesta del servidor y se muestra solamente un "HOLA".

A screenshot of a Windows terminal window with a black background and yellow text. The prompt 'D:\Putty>' is followed by the command 'java Cliente2'. The output 'HOLA' is displayed on the next line. The prompt 'D:\Putty>' appears again on the third line.

```
D:\Putty>java Cliente2
HOLA
D:\Putty>
```

Figura 21: Ejecución de *Servidor2.java* en windows. Respuesta del servidor.

En la Figura 22 vemos la ejecución en la máquina virtual 1 donde el *Servidor2* imprime los datos enviados por el cliente, de igual forma la réplica en la máquina virtual 2 imprime los mismos resultados gracias a que el *Sevidor Proxy*, esto último se observa en la Figura 23.

A screenshot of a terminal window in a virtual machine. The prompt is 'mv1@MV1:~\$'. The first line shows a command to start a proxy: 'Starting proxy for 104.45.144.23:50000 on port 50000'. The following lines show a list of data: '123', '1.2345678901234567E9', 'hola', and a list of numbers '1.1' through '1.5'. A green cursor is visible at the end of the last line.

```
mv1@MV1:~$ Starting proxy for 104.45.144.23:50000 on port 50000
123
1.2345678901234567E9
hola
1.1
1.2
1.3
1.4
1.5
█
```

Figura 22: Ejecución en la máquina virtual 1.

```
mv2replica@mv2Replica:~$ 123
1.2345678901234567E9
hola
1.1
1.2
1.3
1.4
1.5
```

Figura 23: Ejecución en la máquina virtual 2.

### 3. Conclusiones

La realización de esta práctica destaca los requerimientos no funcionales de confiabilidad y rendimiento pues con la replicación se puede asegurar mantener el servicio ofrecido a los clientes en caso de que nuestro servidor principal falle.

Es importante destacar como la replicación evita problemas mayores como incumplimiento de funciones y brinda robustez al sistema distribuido. Muchas veces hay fallos cometidos por errores humanos y en otras cosas ocasionados por agentes externos, en cualquier panorama es importante tener respaldos del sistema e ir incrementando estos respaldos a medida que se requiera.