



# CLASE 4

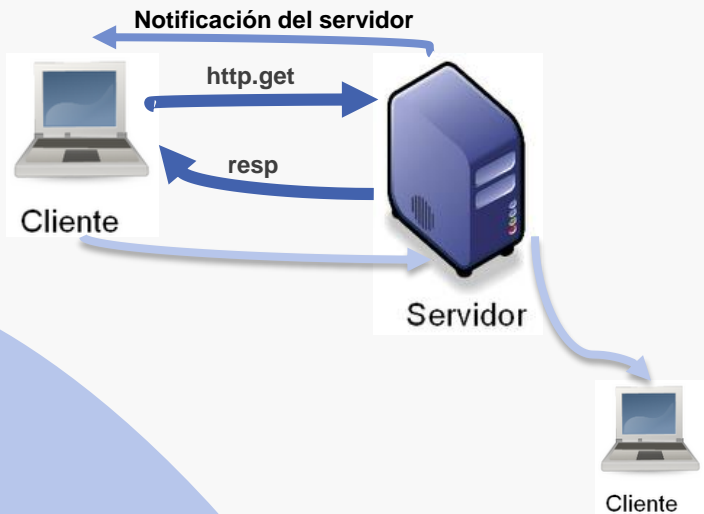
Capa de Aplicación y aplicaciones de red



**Aux. Boris Vargas**

# ¿Problemática?

## Normalmente con HTTP



## Utilizando Sockets



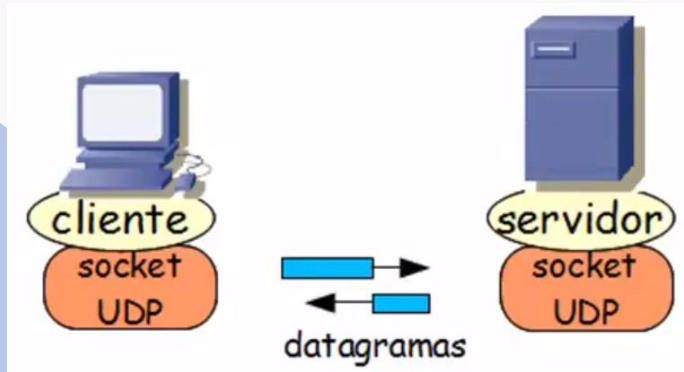
Imaginen que tenemos un cliente y un servidor, normalmente haremos una petición `http.get` y decir al servidor que devuelva información que el cliente esta pidiendo, donde el servidor le dará una respuesta.

Los **sockets** te permiten mantener una comunicación activa activa con la maquina cliente y servidor, es decir que el servidor será capaz de lanzar notificaciones a la maquina cliente.

# Tipos de sockets

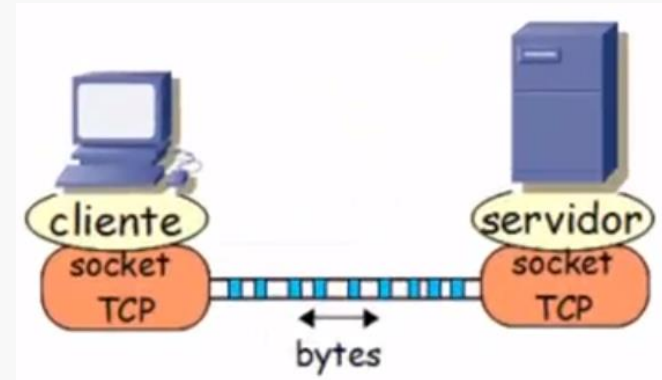
## Sockets UDP

- Las aplicaciones piden al S.O. una comunicación controlada por UDP:
  - Transferencia de bloques de datos
  - Sin conexión ni fiabilidad ni entrega ordenada
  - Permite difusiones
- También se denomina sockets de tipo **Datagram**



## Sockets TCP

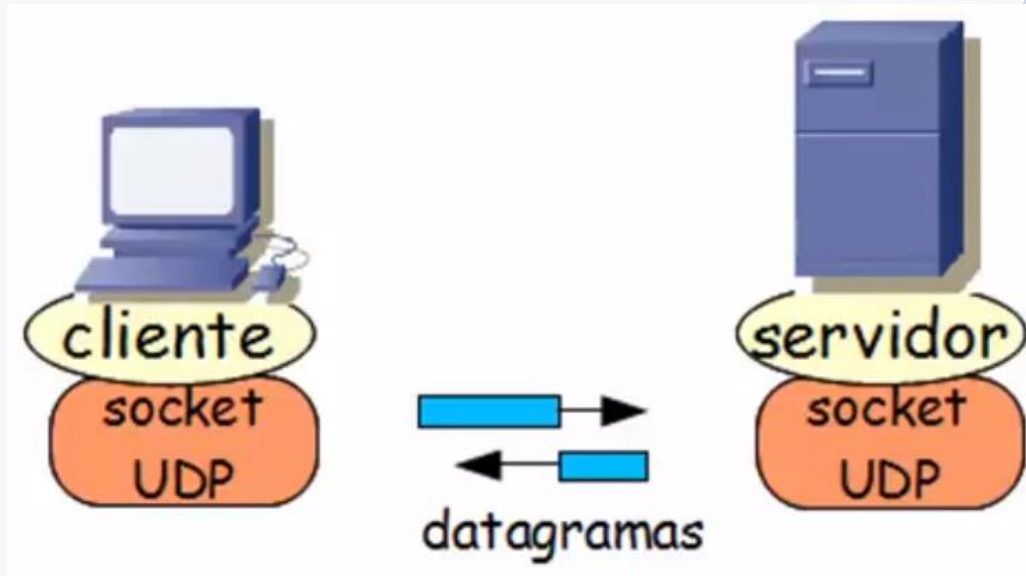
- Las aplicaciones piden al S.O. una comunicación controlada por TCP:
  - Orientada a la conexión
  - Comunicación fiable y ordenada
- También se denominan sockets de tipo **Stream**



# Identificación de los sockets UDP

Un socket UDP se identifica por dos valores:

- Dirección IP, numero de puerto



# Identificación de los sockets TCP

Pero ... un socket TCP **conectado** se identifica por **cuatro** valores

- Dirección IP y puerto local
- Dirección IP y puerto remoto



# Puertos y Servicios

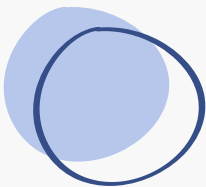
- Los puertos se identifican por un numero entero de 16 bits (rango de 0 a 65535)
- Cada paquete lleva un puerto de origen y uno de destino (como las direcciones IP)
- Los puertos 0 a 1023 están reservados para los servicios ‘bien conocidos’, por ejemplo:

Puerto 80 → servicio web (HTTP)

De esta forma los clientes web (o lo que sea) saben a que puerto han de dirigir sus peticiones

- Algunas aplicaciones usan TCP, otras UDP. Algunas usan ambos, dependiendo del tipo de operación.

Servicio	Puerto	TCP	UDP
FTP	21	X	
SSH	22	X	
Telnet	23	X	
SMTP	24	X	
DNS	53	X	X
HTTP	80	X	



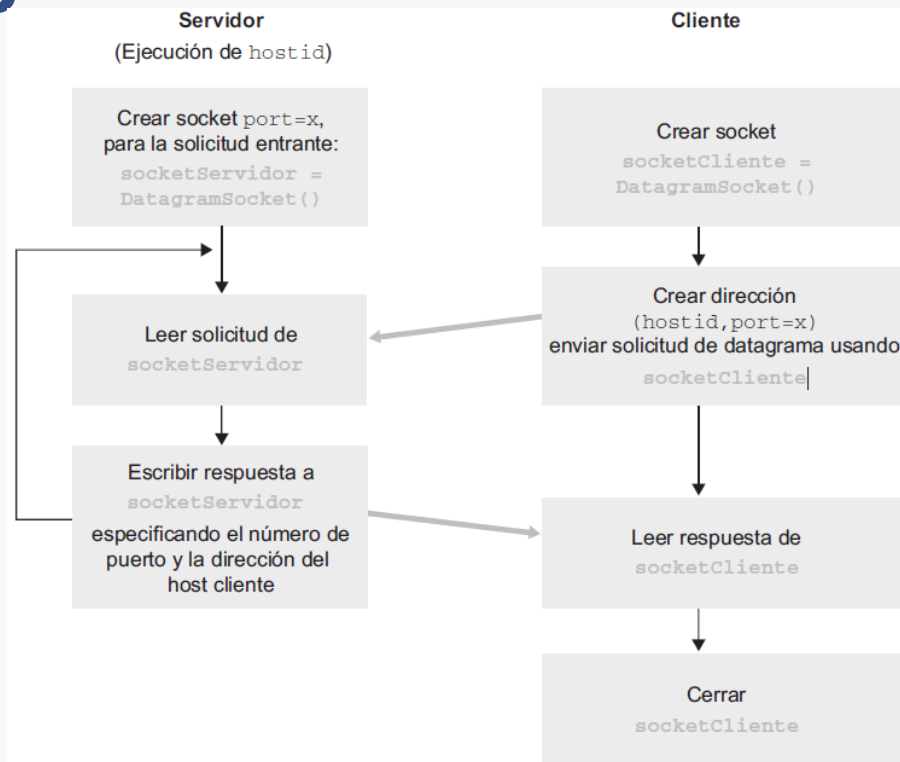
# Java



Sockets UDP y TCP



# Programación de sockets: Java



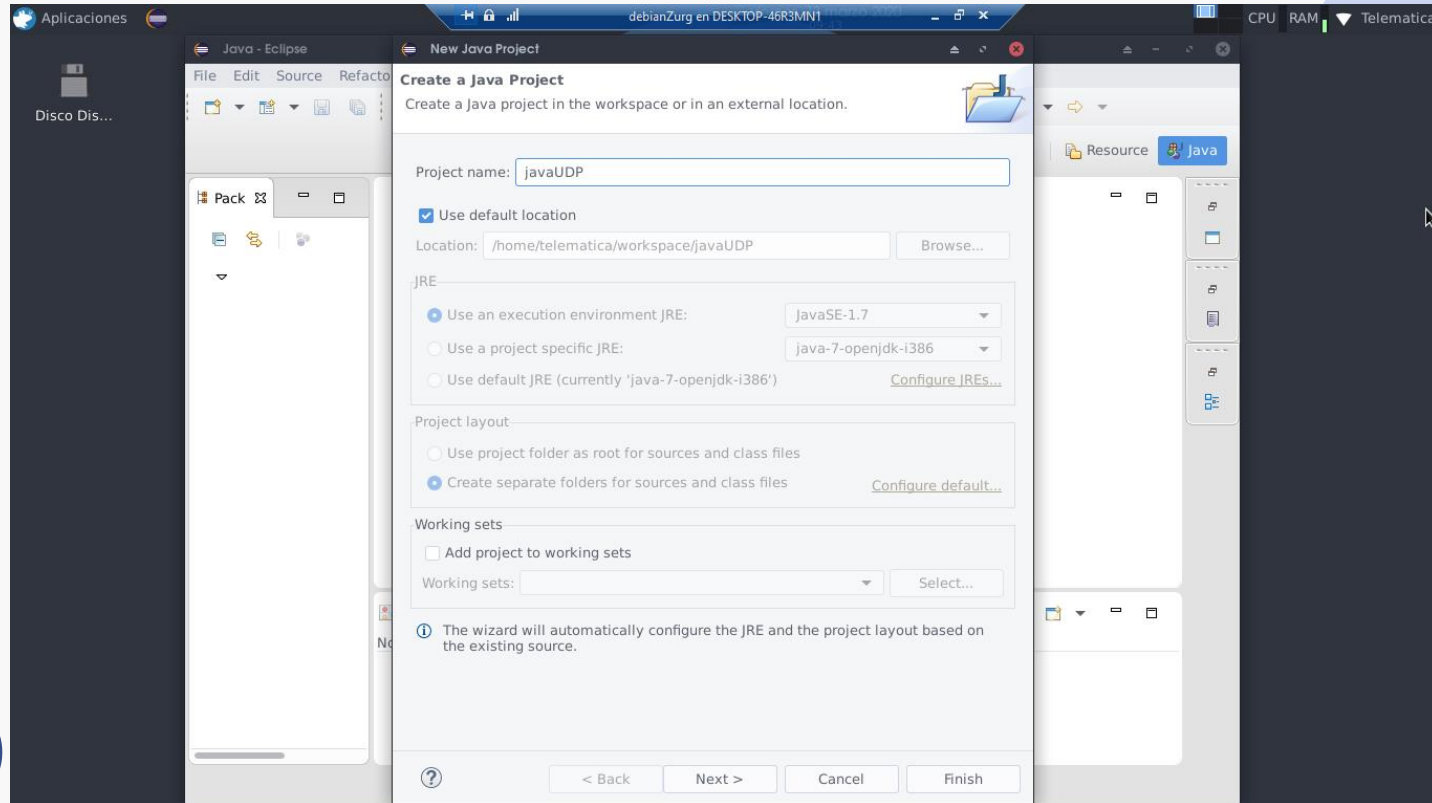
Aplicación cliente-servidor que utiliza servicios de transporte sin conexión (UDP)





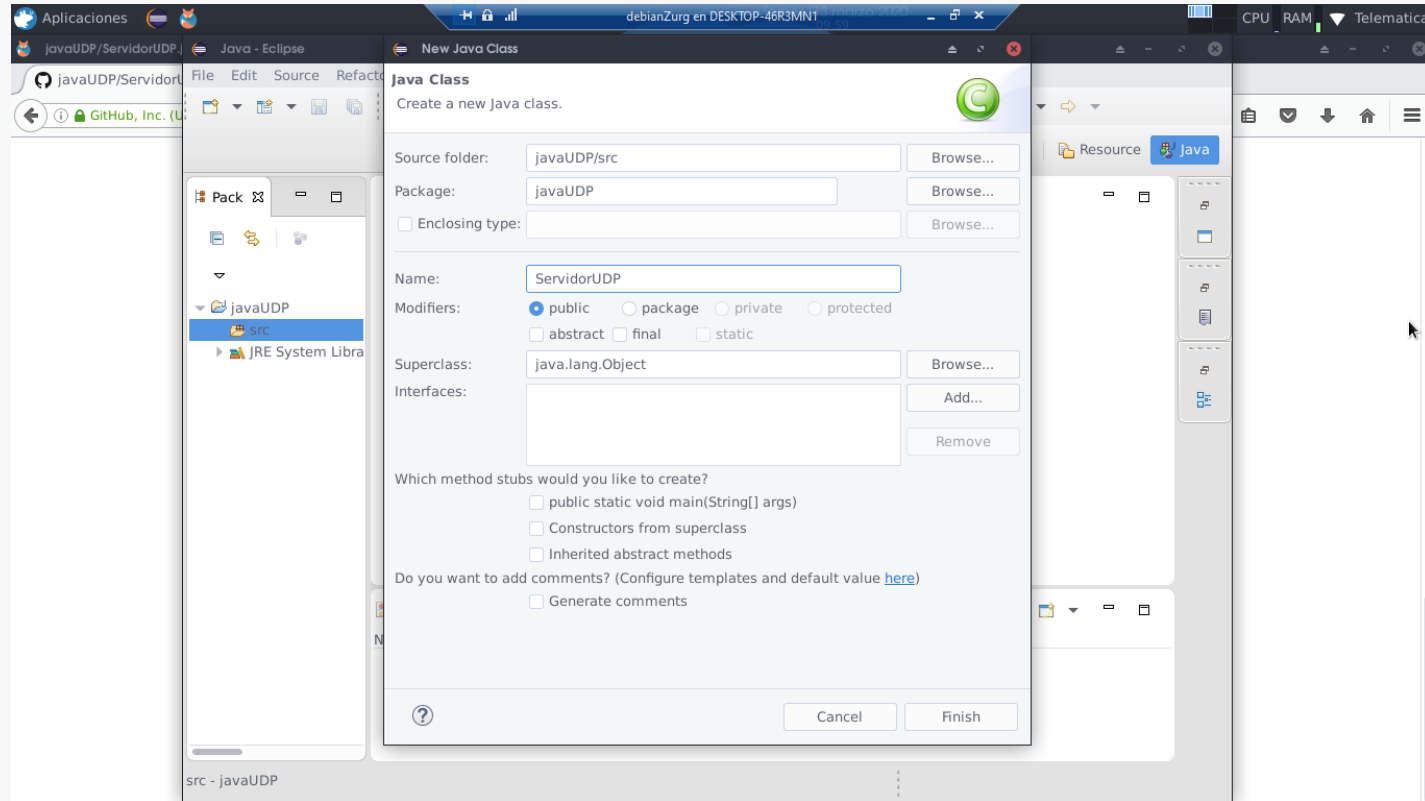
Para el ejemplo utilizaremos Eclipse para la programación de sockets con JAVA. Abrimos el programa Eclipse en nuestra maquina virtual.

- Creamos el proyecto: File -> New -> Java Project



Creamos las clases ServidorUDP y ClienteUDP en nuestro proyecto:

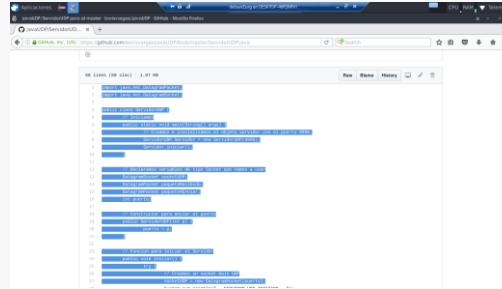
- Seleccionamos nuestro proyecto: New -> Class



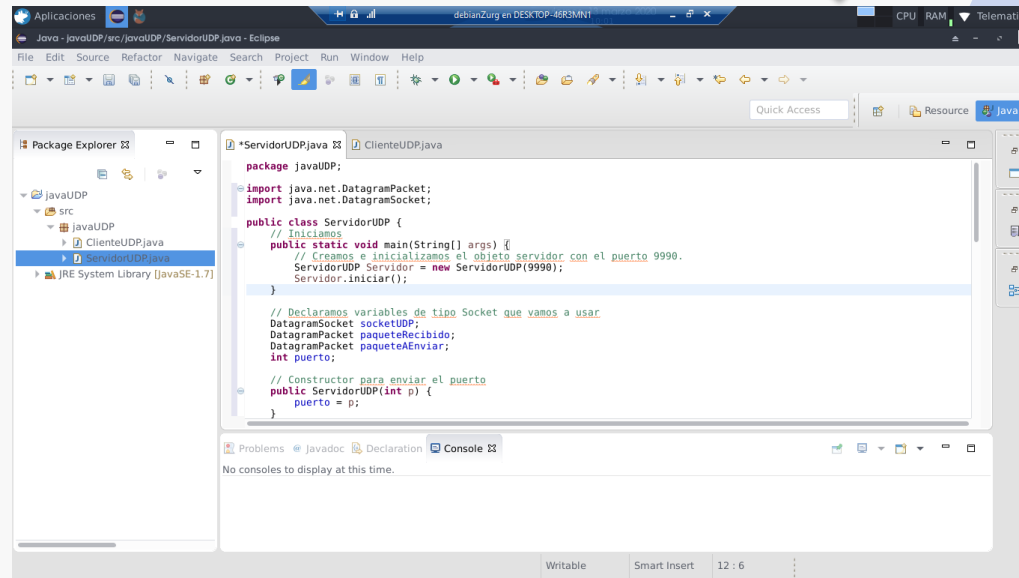
Hacemos de la misma manera para crear el ClienteUDP

# Copiamos el código dejado en el siguiente enlace [aquí](#) en las clases ServidorUDP y ClienteUDP

Recuerda copiarlo por debajo de **package javaUDP;** (si lo tiene)

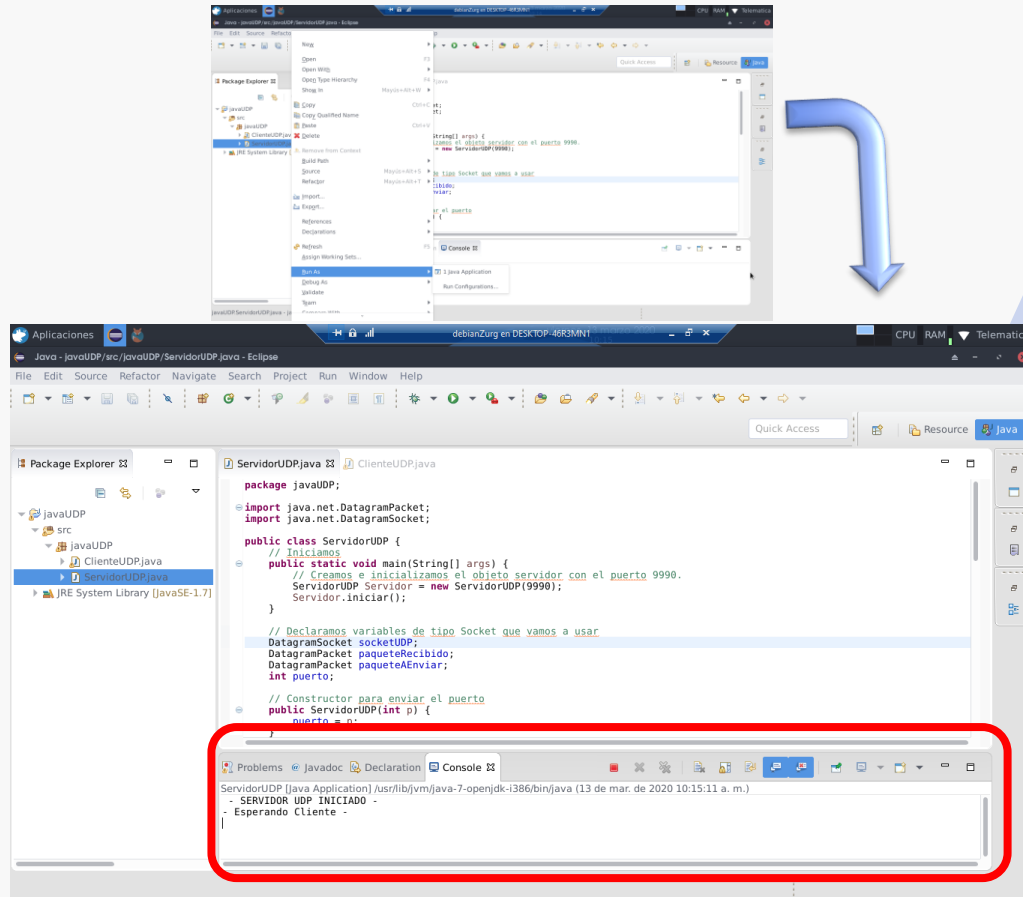


A screenshot of a code editor window showing Java code. The code is for a UDP server and client. The server code is in a class named 'ServidorUDP' and the client code is in a class named 'ClienteUDP'. The code is written in Java and includes package declarations, imports, and class definitions. A large blue arrow points from this screenshot towards the Eclipse IDE screenshot below.



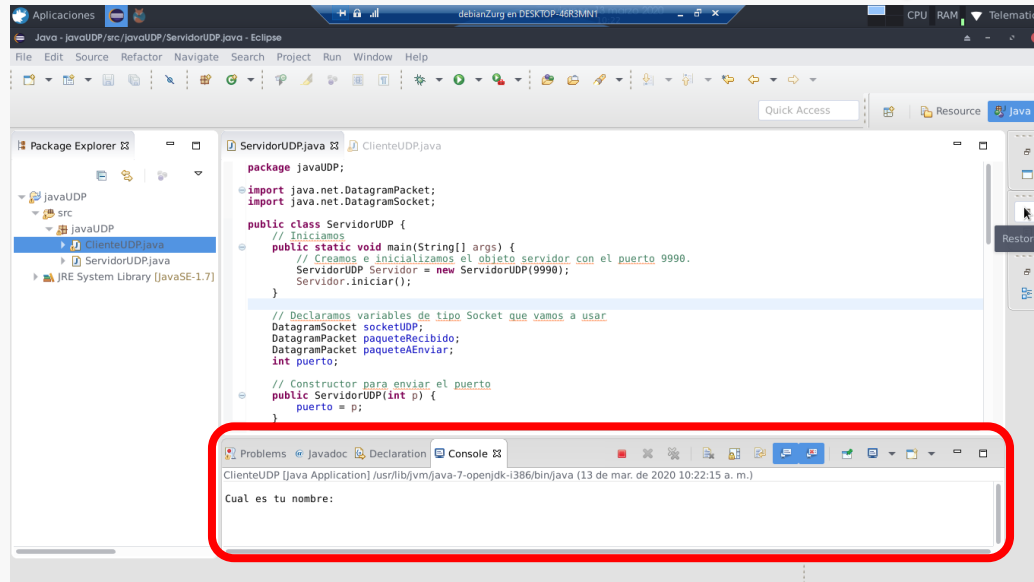
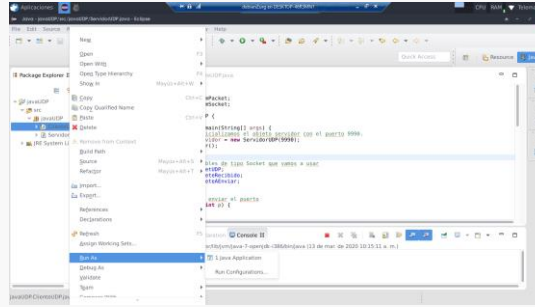
Una vez que tengamos el código en las clases ServidorUDP y ClienteUDP

- Iniciamos la clase **ServidorUDP**: Click Derecho ServidorUDP -> Run As -> Java Application



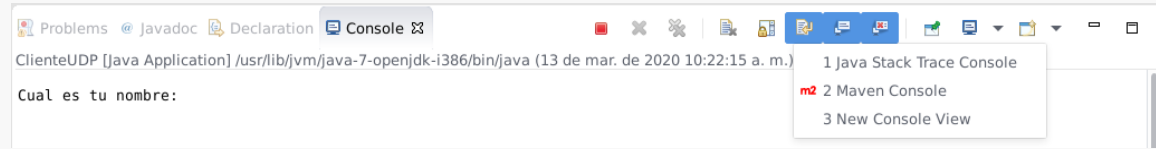
De la misma manera corremos la clase **ClienteUDP**

- Iniciamos la clase ClienteUDP: Click Drecho ClienteUDP -> Run As -> Java Application

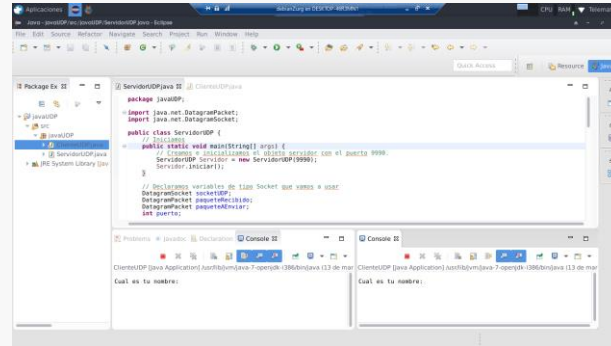



Ahora lo que haremos es crear dos ventanas de Consola.

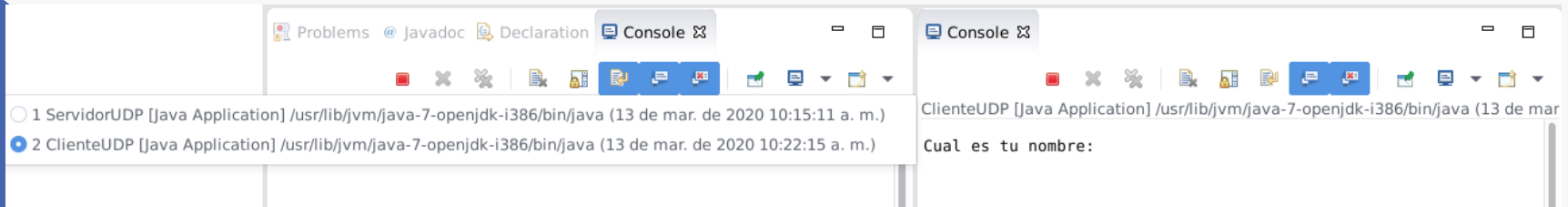
- Damos click a  -> New Console View



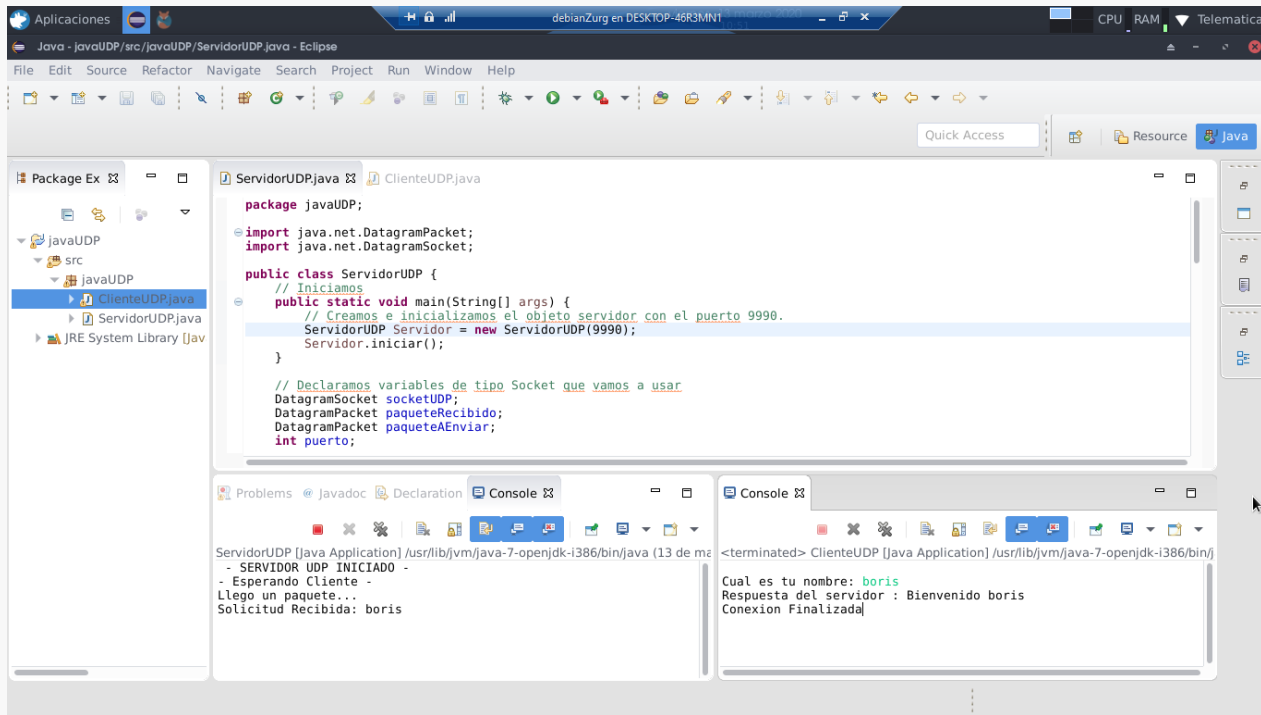
Vamos a dividir en dos ventanas las dos consolas, yo lo hare de la siguiente manera



La primera consola la cambiamos a servidor, damos click  de la primera consola y la cambiamos a *1 ServidorUDP [Java Application]....*



Una vez que tengamos dos ventanas de consola, vamos a experimentar con el código. Enviamos tu nombre por el cliente



```
package javaUDP;

import java.net.DatagramPacket;
import java.net.DatagramSocket;

public class ServidorUDP {
    // Iniciamos
    public static void main(String[] args) {
        // Creamos e inicializamos el objeto servidor con el puerto 9990.
        ServidorUDP servidor = new ServidorUDP(9990);
        servidor.iniciar();
    }

    // Declaramos variables de tipo Socket que vamos a usar
    DatagramSocket socketUDP;
    DatagramPacket paqueteRecibido;
    DatagramPacket paqueteAEnviar;
    int puerto;
}
```


ServidorUDP [Java Application] /usr/lib/jvm/java-7-openjdk-1386/bin/java (13 de marzo de 2014)

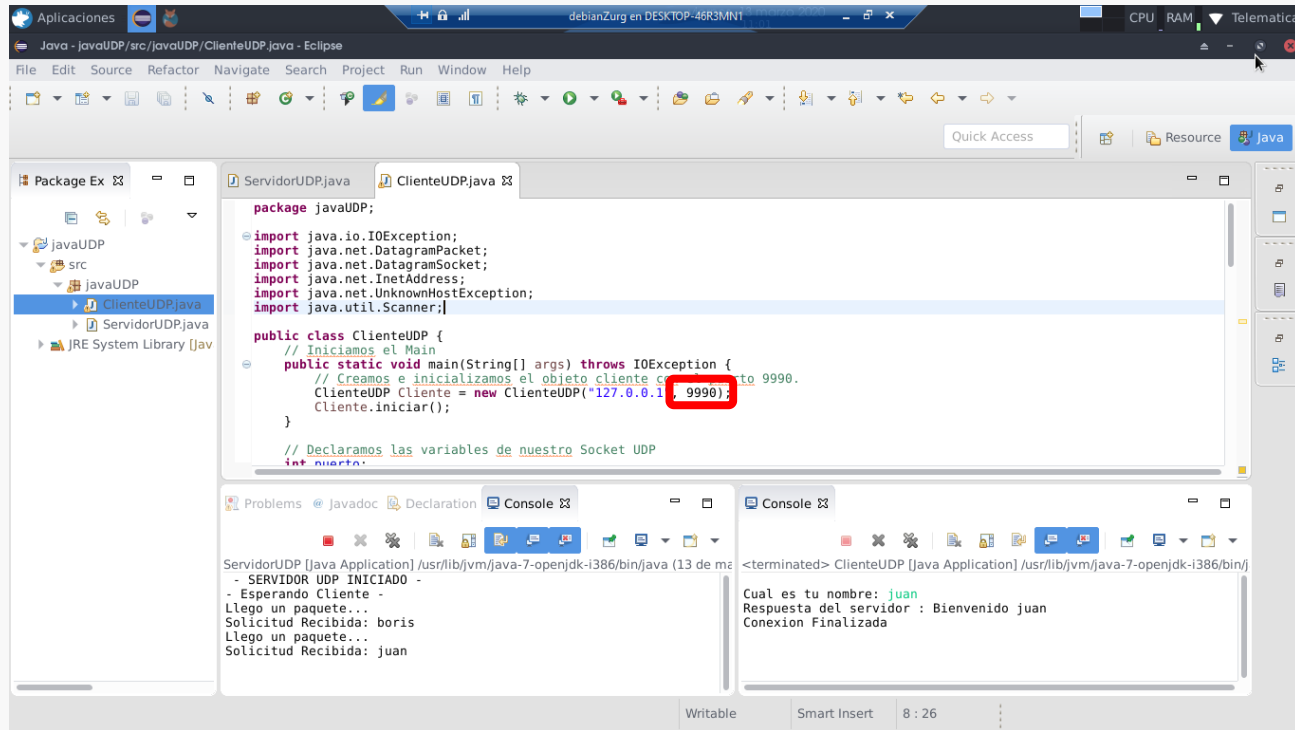
- SERVIDOR UDP INICIADO -  
- Esperando Cliente -  
Llego un paquete...  
Solicitud Recibida: boris

<terminated> ClienteUDP [Java Application] /usr/lib/jvm/java-7-openjdk-1386/bin/java (13 de marzo de 2014)

Cual es tu nombre: boris  
Respuesta del servidor : Bienvenido boris  
Conexion Finalizada

Enviamos una cadena **boris** la cual se empaqueta y se envía al servidor, después el servidor lo recibe y lo concatena a una cadena el paquete recibido, después el servidor envia **Bienvenido boris** al cliente, una vez recibido el cliente cierra conexión.

**NOTA:** Para hacer otras pruebas paramos el servidor dando click a , se debe cambiar el puerto en ClienteUDP y ServidorUDP, ya que no se cerro el puerto en el servidor, entonces estará ocupado. Ejemplo: 9991



```
package javaUDP;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Scanner;

public class ClienteUDP {
    // Iniciamos el Main
    public static void main(String[] args) throws IOException {
        // Creamos e inicializamos el objeto cliente con el puerto 9990.
        ClienteUDP cliente = new ClienteUDP("127.0.0.1", 9990);
        cliente.iniciar();
    }

    // Declaramos las variables de nuestro Socket UDP
    int puerto;
}
```

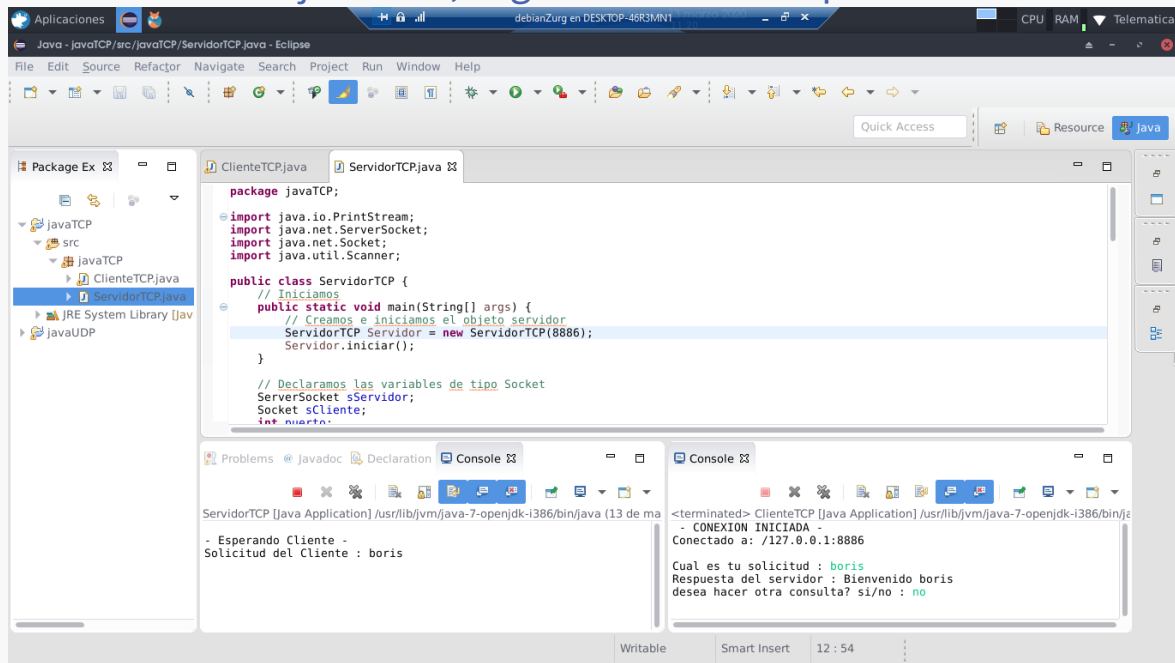
ServidorUDP [Java Application] /usr/lib/jvm/java-7-openjdk-1386/bin/java (13 de me...  
- SERVIDOR UDP INICIADO -  
- Esperando Cliente -  
Llego un paquete...  
Solicitud Recibida: boris  
Llego un paquete...  
Solicitud Recibida: juan

<terminated> ClienteUDP [Java Application] /usr/lib/jvm/java-7-openjdk-1386/bin/j...  
Cual es tu nombre: juan  
Respuesta del servidor : Bienvenido juan  
Conexion Finalizada

Recuerda que el servidor siempre estará corriendo, para hacer este otro ejemplo ejecutamos nuevamente cliente enviamos **juan** dando el siguiente resultado.



**Experimentar** de la misma manera con TCP, descargamos el código [aquí](#).  
De la misma manera como javaUDP, seguimos todos los pasos.



```
package javaTCP;

import java.io.InputStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.Scanner;

public class ServidorTCP {
    // Iniciamos
    public static void main(String[] args) {
        // Creamos e iniciamos el objeto servidor
        ServidorTCP servidor = new ServidorTCP(8886);
        servidor.iniciar();
    }

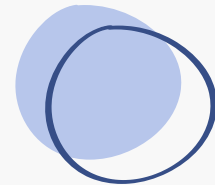
    // Declaramos las variables de tipo Socket
    ServerSocket sServidor;
    Socket sCliente;
    int puerto;
}
```

```
<terminated>- ClienteTCP [Java Application] /usr/lib/jvm/java-7-openjdk-1386/bin/java (13 de ma
- Esperando Cliente -
Solicitud del Cliente : boris
```

```
<terminated>- ClienteTCP [Java Application] /usr/lib/jvm/java-7-openjdk-1386/bin/j
- CONEXION INICIADA -
Conectado a: /127.0.0.1:8886
Cual es tu solicitud : boris
Respuesta del servidor : Bienvenido boris
desea hacer otra consulta? si/no : no
```

Enviamos una cadena **boris** la cual se empaqueta y se envía al servidor, después el servidor lo recibe y lo concatena a una cadena el paquete recibido, después el servidor envía **Bienvenido boris** al cliente, una vez recibido el cliente le pedirá si desea hacer otra consulta en mi caso **no** y se cierra conexión con el servidor.





# CODIGO



## Java

- UDP: <https://github.com/borisvargas/javaUDP>
- TCP: <https://github.com/borisvargas/javaTCP>

## Cheatsheet

- <https://drive.google.com/open?id=1hBj-o6lecaH2UjuXa-mrAUq21bk11kFL>



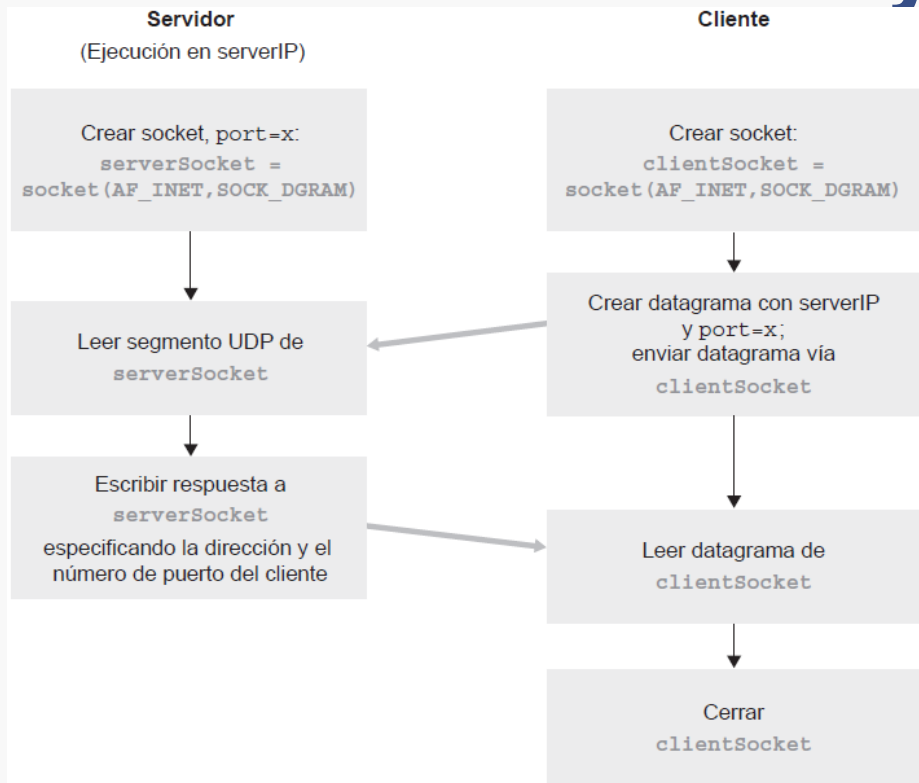
# Python



Sockets UDP y TCP



# Programación de sockets: Python



Aplicación cliente-servidor usando UDP.

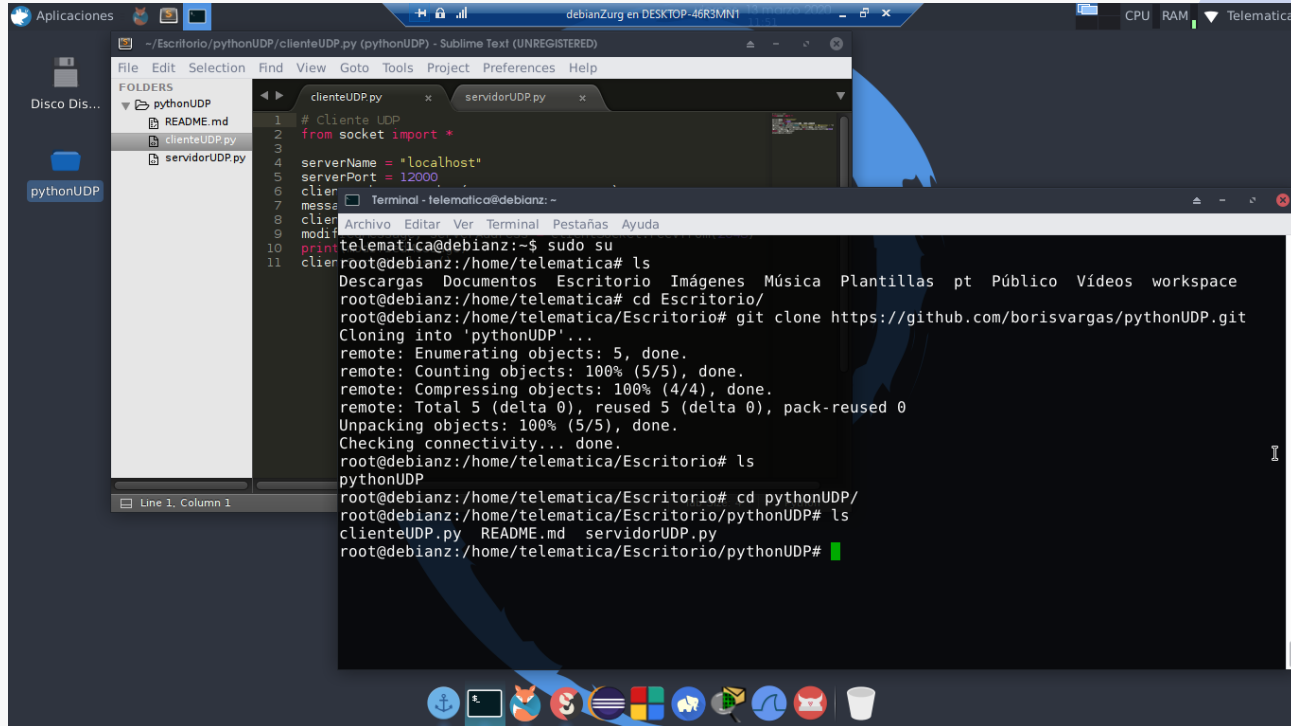


Para el ejemplo utilizaremos Sublime Text para editar y la terminal para la ejecución del código. Para no tener problemas la terminal debe estar en súper usuario.

Clonaremos el proyecto con la ayuda de git, en mi caso en Escritorio:

```
git clone https://github.com/borisvargas/pythonUDP.git
```

Abrimos la carpeta en Sublime Text



The screenshot displays a Linux desktop with a dark theme. In the background, a terminal window is open, showing the execution of several commands to clone a repository and navigate through the file system. In the foreground, the Sublime Text editor is open, displaying two Python files: `clienteUDP.py` and `servidorUDP.py`. The `clienteUDP.py` file contains code for a UDP client, including imports, variable assignments for server name and port, and a loop for sending messages. The terminal output shows the successful cloning of the repository and the current directory being `/home/telematica/Escritorio/pythonUDP`.

```
telematica@debianz:~$ sudo su
root@debianz:/home/telematica# ls
Descargas Documentos Escritorio Imágenes Música Plantillas pt Público Vídeos workspace
root@debianz:/home/telematica# cd Escritorio/
root@debianz:/home/telematica/Escritorio# git clone https://github.com/borisvargas/pythonUDP.git
Cloning into 'pythonUDP'...
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 5 (delta 0), pack-reused 0
Unpacking objects: 100% (5/5), done.
Checking connectivity... done.
root@debianz:/home/telematica/Escritorio# ls
pythonUDP
root@debianz:/home/telematica/Escritorio# cd pythonUDP/
root@debianz:/home/telematica/Escritorio/pythonUDP# ls
clienteUDP.py README.md servidorUDP.py
root@debianz:/home/telematica/Escritorio/pythonUDP#
```

Sublime Text (UNREGISTERED) interface shows the following files open:

- `clienteUDP.py`:

```
1 # Cliente UDP
2 from socket import *
3
4 serverName = "localhost"
5 serverPort = 12000
6
7 client = socket(AF_INET, SOCK_DGRAM)
8
9 while True:
10     message = input("Enter message: ")
11     client.sendto(message.encode(), (serverName, serverPort))
```
- `servidorUDP.py`:

```
1 # Servidor UDP
2 from socket import *
3
4 serverName = "localhost"
5 serverPort = 12000
6
7 server = socket(AF_INET, SOCK_DGRAM)
8 server.bind((serverName, serverPort))
9
10 while True:
11     (message, address) = server.recvfrom(1024)
12     print("Received from", address, ": ", message.decode())
```

Abrimos otra terminal (Archivo -> Abrir terminal) para ejecutar servidorUDP.py y clienteUDP.py, tendremos dos terminales, entonces ejecutamos primero el servidor.

Primera terminal:

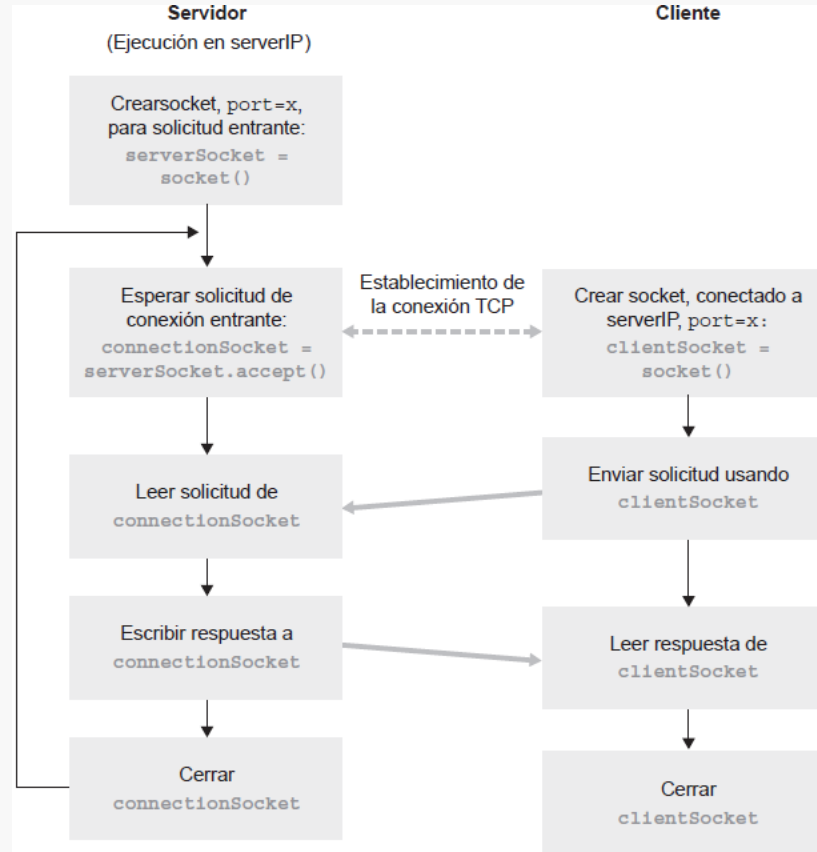
```
python3 servidorUDP.py
```

Segunda terminal:

```
python3 clienteUDP.py
```

```
~/Escritorio/pythonUDP/servidorUDP.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

Terminal - telematica@debianz: ~
Archivo Editar Ver Terminal Pestañas Ayuda
root@debianz:/home/telematica/Escritorio/pythonUDP# python3 servidorUDP.py
Servidor listo para recibir .... 12000
Se ha recibido : b'boris'
1 serverSocket = socket(AF_INET, SOCK_DGRAM)
2 serverSocket.bind(('localhost', serverPort))
3 print("Servidor listo para recibir ....")
4 while True:
5     message, clientAddress = serverSocket.recvfrom(2048)
6     print("Se ha recibido : ", message)
7     modifiedMessage = message.upper()
8     serverSocket.sendto(modifiedMessage, clientAddress)
9
python3 clienteUDP.py
Terminal - telematica@debianz: ~
Archivo Editar Ver Terminal Pestañas Ayuda
telematica@debianz:~$ sudo su
[sudo] password for telematica:
root@debianz:/home/telematica# cd Escritorio/
root@debianz:/home/telematica/Escritorio# cd pythonUDP
root@debianz:/home/telematica/Escritorio/pythonUDP# ls
clienteUDP.py README.md servidorUDP.py
root@debianz:/home/telematica/Escritorio/pythonUDP# python3 clienteUDP.py
Ingrese frase en minusculas : boris
b'BORIS'
root@debianz:/home/telematica/Escritorio/pythonUDP#
```

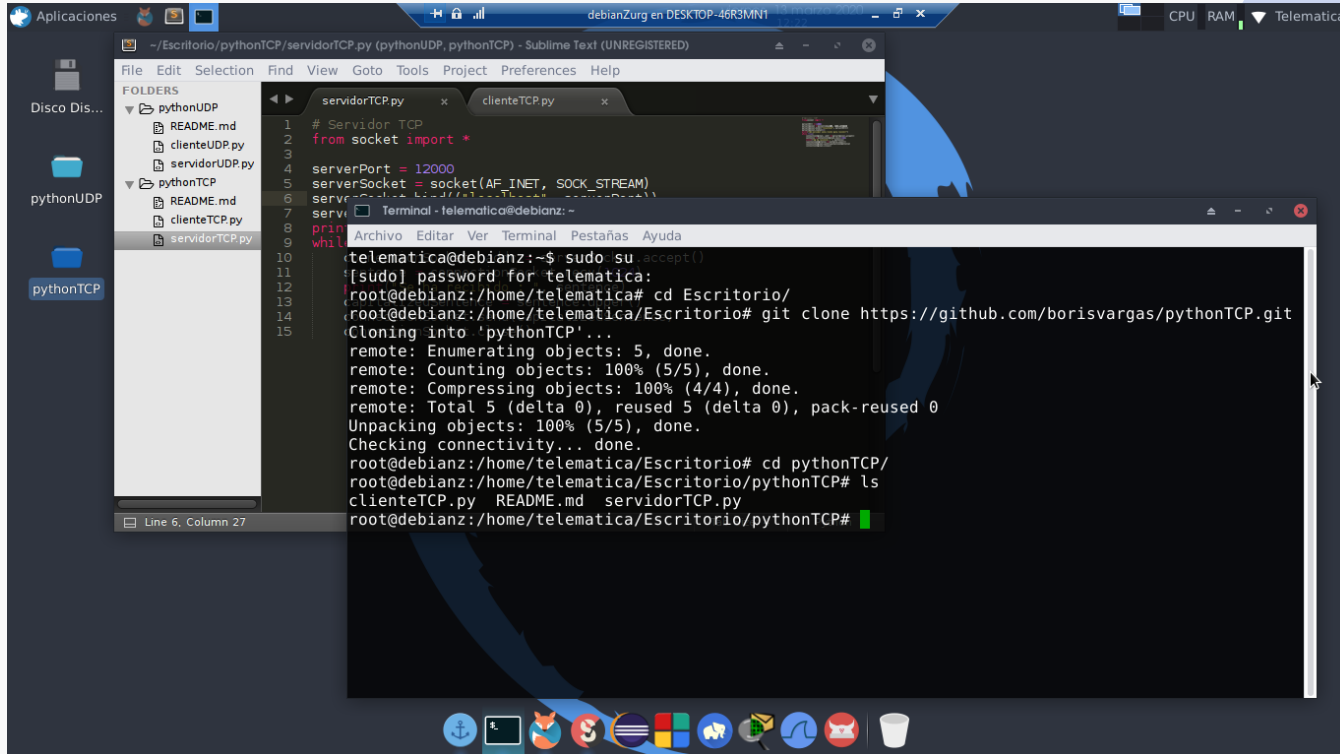


La aplicación cliente-servidor utilizando TCP.

Para este ejemplo para no tener problemas la terminal debe estar en súper usuario.  
Clonaremos el proyecto con la ayuda de git, en mi caso en Escritorio:

```
git clone https://github.com/borisvargas/pythonTCP.git
```

Abrimos la carpeta en Sublime Text





Abrimos otra terminal (Archivo -> Abrir terminal) para ejecutar servidor y cliente, tendremos dos terminales, entonces ejecutamos primero el servidor.

Primera terminal:

```
python3 servidorTCP.py
```

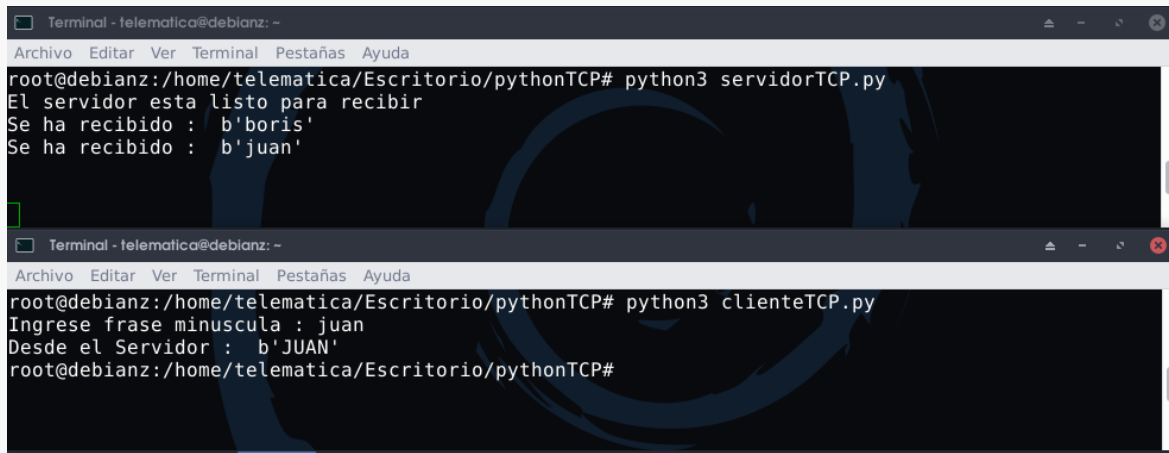
Segunda terminal:

```
python3 clienteTCP.py
```

```
Terminal - telematica@debianz: ~  
root@debianz:/home/telematica/Escritorio/pythonTCP# python3 servidorTCP.py  
El servidor esta listo para recibir  
Se ha recibido : b'boris'
```

```
Terminal - telematica@debianz: ~  
telematica@debianz:~$ sudo su  
[sudo] password for telematica:  
root@debianz:/home/telematica# cd Escritorio/  
root@debianz:/home/telematica/Escritorio# ls  
pythonTCP  pythonUDP  
root@debianz:/home/telematica/Escritorio# cd pythonTCP/  
root@debianz:/home/telematica/Escritorio/pythonTCP# ls  
clienteTCP.py  README.md  servidorTCP.py  
root@debianz:/home/telematica/Escritorio/pythonTCP# python3 clienteTCP.py  
Ingrese frase minuscula : boris  
Desde el Servidor : b'BORIS'  
root@debianz:/home/telematica/Escritorio/pythonTCP#
```

Desde el cliente enviamos una frase en minúsculas, el servidor lo recibe, entonces lo pasa a mayúsculas, después envía al cliente y cliente cierra conexión.



```
Terminal - telematica@debianz: ~
Archivo Editar Ver Terminal Pestañas Ayuda
root@debianz:/home/telematica/Escritorio/pythonTCP# python3 servidorTCP.py
El servidor esta listo para recibir
Se ha recibido : b'boris'
Se ha recibido : b'juan'

Terminal - telematica@debianz: ~
Archivo Editar Ver Terminal Pestañas Ayuda
root@debianz:/home/telematica/Escritorio/pythonTCP# python3 clienteTCP.py
Ingrese frase minuscula : juan
Desde el Servidor : b'JUAN'
root@debianz:/home/telematica/Escritorio/pythonTCP#
```

**NOTA:** Para cerrar el puerto ocupado hacemos lo siguiente, o simplemente puedes cambiar el puerto en tu código de python tanto en cliente como en servidor.

```
netstat -lp | grep [puerto]
kill [proceso]
```



```
Terminal - telematica@debianz: ~
Archivo Editar Ver Terminal Pestañas Ayuda
telematica@debianz:~$ sudo su
[sudo] password for telematica:
root@debianz:/home/telematica# netstat -lp | grep 12000
udp        0      0 localhost:12000    *:.*                3295/python3
root@debianz:/home/telematica# kill 3295
root@debianz:/home/telematica#
```



# CODIGO



## Python

- UDP: <https://github.com/borisvargas/pythonUDP>
- TCP: <https://github.com/borisvargas/pythonTCP>

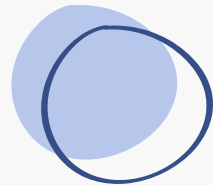
## Cheatsheet

- [https://drive.google.com/open?id=1jtQ2BFw-\\_CThNfLEQ8yjBH4AkTKA5vh4](https://drive.google.com/open?id=1jtQ2BFw-_CThNfLEQ8yjBH4AkTKA5vh4)

## Referencias

Ver Libro “Redes de computadoras, un enfoque descendente” 7ª Edición James F. Kurose , Keith W. Ross

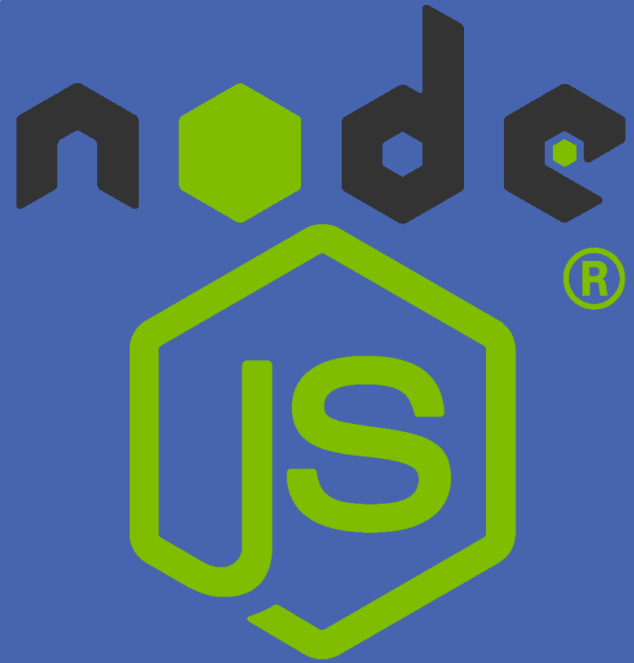
- Pagina 131, 2.7.1 Programación de sockets con UDP
- Pagina 135, 2.7.2 Programación de sockets con TCP



# JavaScript



Sockets UDP y TCP



# Programación de sockets: JavaScript

Instalamos Nodejs en nuestra maquina virtual

- `sudo apt-get install nodejs`
- `nodejs -versión`  
`v0.10.29`

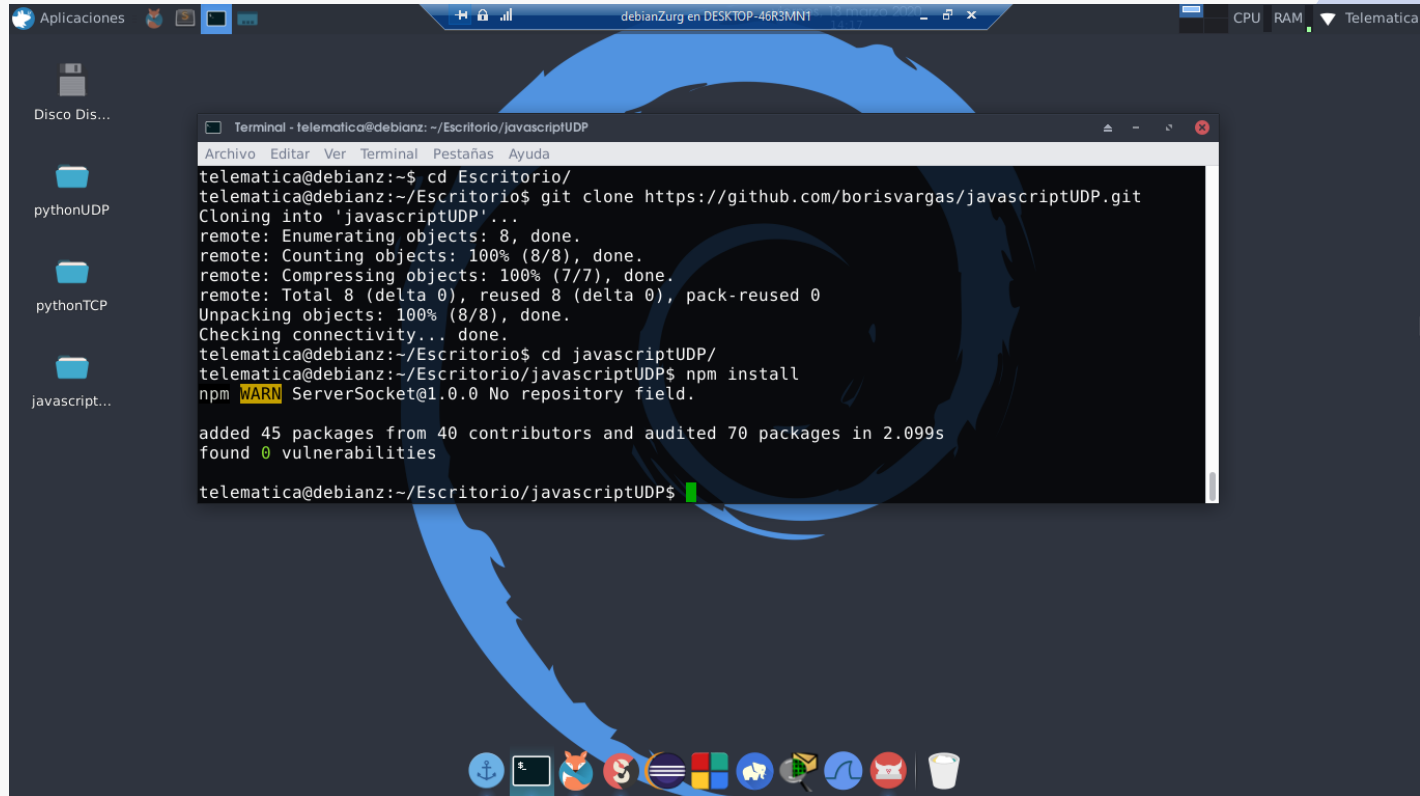
Debemos tener instalado la versión 8 o superior

- `sudo apt-get install curl`
- `cd ~`
- `curl -sL https://deb.nodesource.com/setup\_8.x -o config.sh`
- `sudo bash config.sh`
- `sudo apt-get install -y nodejs`
- `nodejs -versión`  
`v8.17.0`



Primeramente experimentaremos JavaScript con UDP, para eso observamos el código [aquí](#).  
Para este ejemplo vamos a clonar el ejemplo:

```
git clone https://github.com/borisvargas/javascriptUDP.git
```



The screenshot shows a Linux desktop environment with a dark theme. The top panel displays system information: 'debianZurg en DESKTOP-46R3MNT', 'CPU', 'RAM', and 'Telematica'. The left sidebar shows a file manager with folders named 'pythonUDP', 'pythonTCP', and 'javascript...'. The terminal window, titled 'Terminal - telematica@debianz: ~/Escritorio/javascriptUDP', shows the following commands and output:

```
telematica@debianz:~$ cd Escritorio/
telematica@debianz:~/Escritorio$ git clone https://github.com/borisvargas/javascriptUDP.git
Cloning into 'javascriptUDP'...
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 8 (delta 0), reused 8 (delta 0), pack-reused 0
Unpacking objects: 100% (8/8), done.
Checking connectivity... done.
telematica@debianz:~/Escritorio$ cd javascriptUDP/
telematica@debianz:~/Escritorio/javascriptUDP$ npm install
npm WARN ServerSocket@1.0.0 No repository field.

added 45 packages from 40 contributors and audited 70 packages in 2.099s
found 0 vulnerabilities

telematica@debianz:~/Escritorio/javascriptUDP$
```

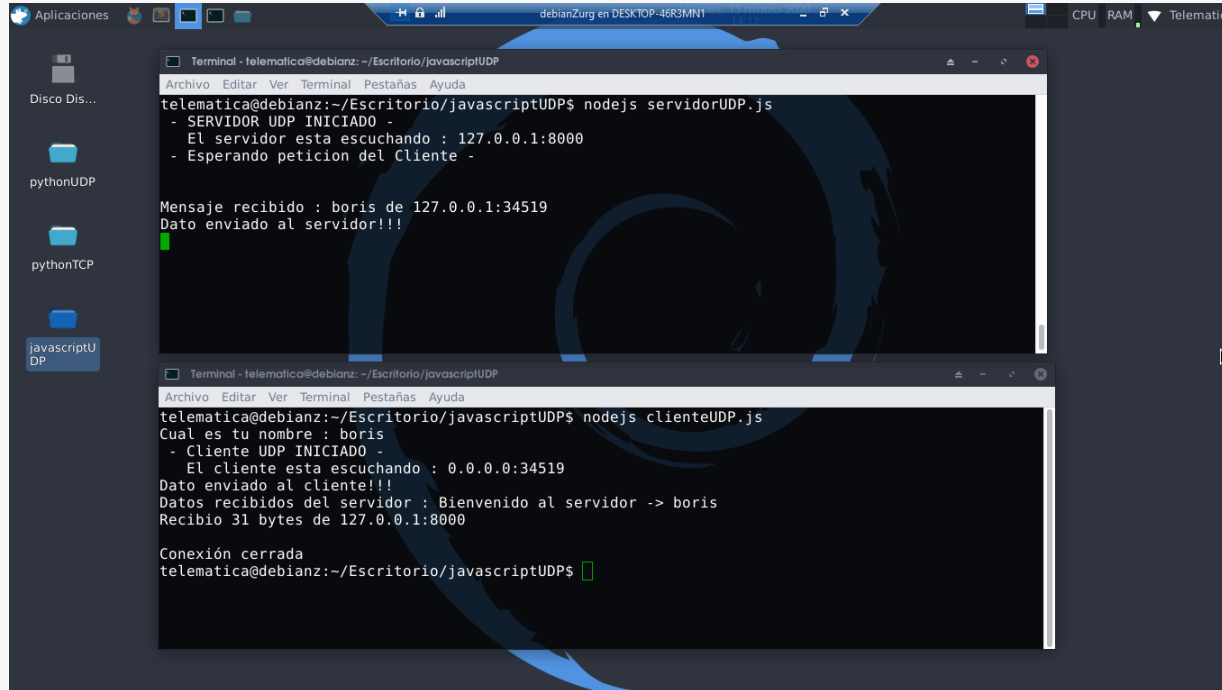
Abrimos otra terminal (Archivo -> Abrir terminal) para ejecutar servidor y cliente, tendremos dos terminales, entonces ejecutamos primero el servidor.

Primera terminal:

```
nodejs servidorUDP.js
```

Segunda terminal:

```
nodejs clienteUDP.js
```



The screenshot shows two terminal windows on a Linux desktop. The top terminal window is titled 'Terminal - telematica@debian: ~/Escritorio/javascriptUDP' and shows the execution of 'nodejs servidorUDP.js'. The output indicates the server is listening on 127.0.0.1:8000 and receives a message from 127.0.0.1:34519. The bottom terminal window is also titled 'Terminal - telematica@debian: ~/Escritorio/javascriptUDP' and shows the execution of 'nodejs clienteUDP.js'. The output shows the client sending a message to the server, receiving a response, and then closing the connection.

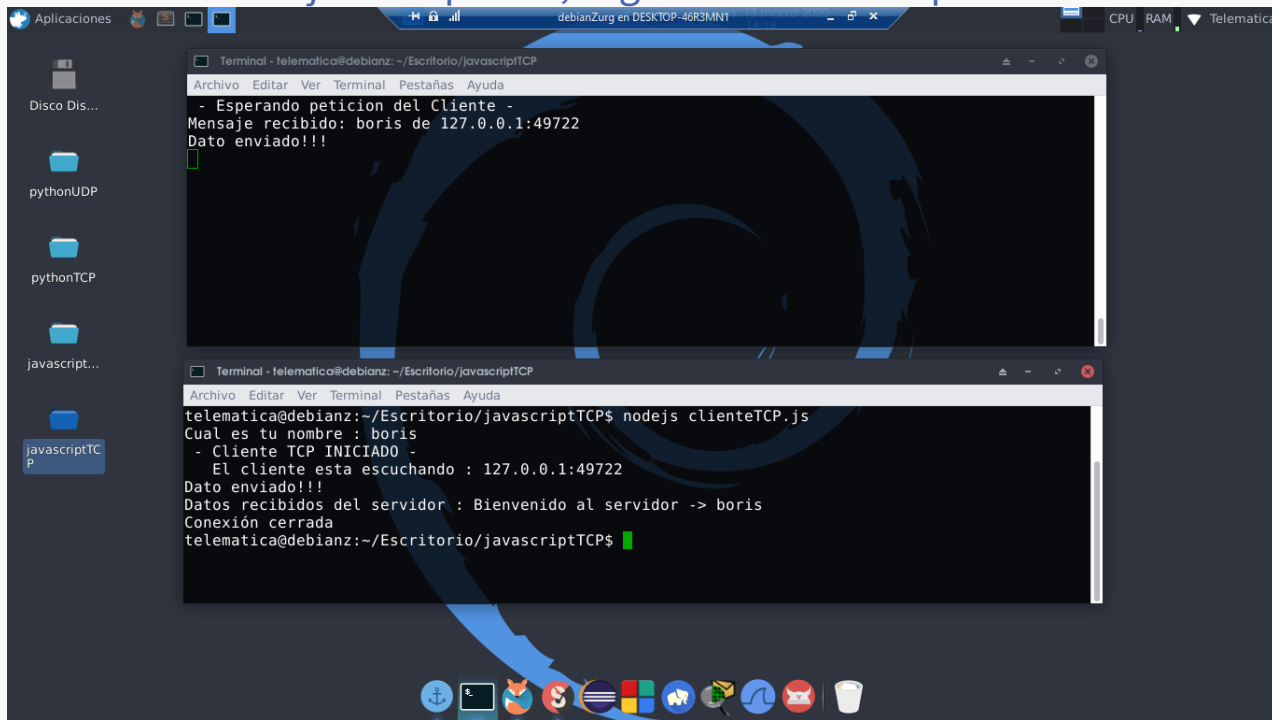
```
telematica@debian:~/Escritorio/javascriptUDP$ nodejs servidorUDP.js
- SERVIDOR UDP INICIADO -
El servidor esta escuchando : 127.0.0.1:8000
- Esperando peticion del Cliente -

Mensaje recibido : boris de 127.0.0.1:34519
Dato enviado al servidor!!!

telematica@debian:~/Escritorio/javascriptUDP$ nodejs clienteUDP.js
Cual es tu nombre : boris
- Cliente UDP INICIADO -
El cliente esta escuchando : 0.0.0.0:34519
Dato enviado al cliente!!!
Datos recibidos del servidor : Bienvenido al servidor -> boris
Recibio 31 bytes de 127.0.0.1:8000

Conexión cerrada
telematica@debian:~/Escritorio/javascriptUDP$
```

**Experimentar** de la misma manera con TCP, descargamos el código [aquí](#).  
De la misma manera como javascriptUDP, seguimos todos los pasos.



```
Terminal - telematica@debianz: ~/Escritorio/javascriptTCP
Archivo Editar Ver Terminal Pestañas Ayuda
- Esperando peticion del Cliente -
Mensaje recibido: boris de 127.0.0.1:49722
Dato enviado!!!

Terminal - telematica@debianz: ~/Escritorio/javascriptTCP
Archivo Editar Ver Terminal Pestañas Ayuda
telematica@debianz:~/Escritorio/javascriptTCP$ nodejs clienteTCP.js
Cual es tu nombre : boris
- Cliente TCP INICIADO -
El cliente esta escuchando : 127.0.0.1:49722
Dato enviado!!!
Datos recibidos del servidor : Bienvenido al servidor -> boris
Conexión cerrada
telematica@debianz:~/Escritorio/javascriptTCP$
```

Enviamos una cadena **boris** la cual se empaqueta y se envía al servidor, después el servidor lo recibe y lo concatena a una cadena el paquete recibido, después el servidor envía **Bienvenido al servidor -> boris** al cliente y se cierra conexión con el servidor.



# CODIGO

## JavaScript

- UDP: <https://github.com/borisvargas/javascriptUDP>
- TCP: <https://github.com/borisvargas/javascriptTCP>

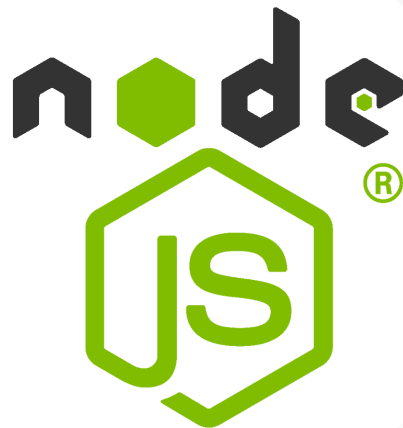
## Cheatsheet

- [https://drive.google.com/open?id=15P1L\\_O9JQgGOhhr7QVAvY90eyVeZZwUD](https://drive.google.com/open?id=15P1L_O9JQgGOhhr7QVAvY90eyVeZZwUD)

## Referencias

[https://github.com/rodrigoms2004/ServerSocketTCP\\_UDP](https://github.com/rodrigoms2004/ServerSocketTCP_UDP)

- UDP: <https://nodejs.org/api/dgram.html>
- TCP: <https://nodejs.org/api/net.html>





# EJERCICIOS

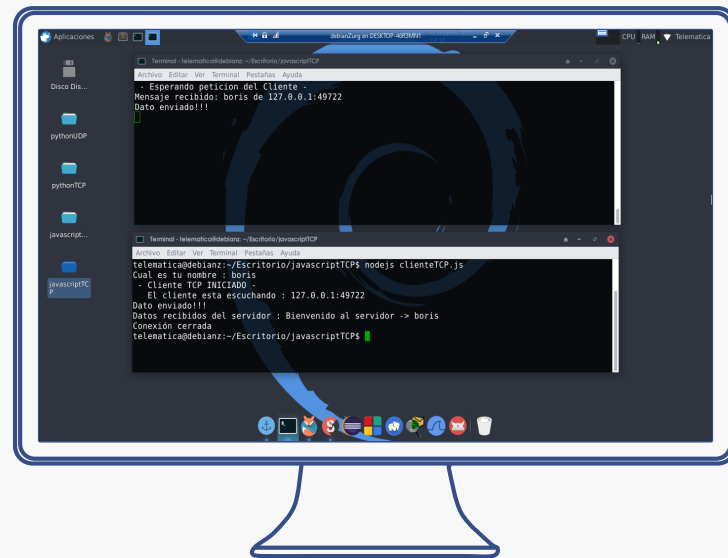


Puntos extras

# Capturas de pantalla

Enviar capturas de pantalla de los ejemplos que hicimos en clase (Envía cadena: Su-Nombre):

- javaUDP y javaTCP
- pythonUDP y pythonTCP
- javascriptUDP y javascriptTCP



Puntos extras





# EJERCICIO 1

Usando NodeJS desarrolle un servidor que procese peticiones GET y POST enviadas desde un formularios HTML. EL servidor deberá extraer los valores de los campos y los retornara al cliente en una pagina HTML.

Cuando reciba un GET, el servidor procesara el querystring para recuperar los datos del formulario; mientras que para peticiones POST deberá procesar el cuerpo del mensaje para recuperar los datos. Los formularios HTML deberán tener al menos dos campos (Ejemplo: nombre y email) que serán llenados por el usuario y un botón “enviar” para remitir los datos al servidor. Para el desarrollo de su programa deberá usar los módulos: http, url y querystring.





# EJERCICIO 2

Se desea un programa Java que use sockets para inspeccionar los puertos de una maquina local y reportar aquellos puertos que están en uso por algún servidor TCP. Considere todo el rango de puertos.

Ejemplo de la salida buscada es:

Hay un servidor en el puerto 80

Hay un servidor en el puerto 135





# EJERCICIO 3

Usando NodeJS desarrolle un servidor que retorne la fecha y hora. El servidor debe escuchar conexiones TCP en el puerto indicado por el primer argumento del programa. La información de la hora deberá estar en formato 24hrs y tanto el mes, el día como la hora y minuto deben tener un 0 para ocupar 2 espacios, por ejemplo “2020-09-07 17:01”

Para este ejercicio deberá usar el modulo “net” que implemente el mecanismo de sockets y acceso a la red. Para probar su servidor conéctese como cliente usando el comando Telnet



# EJERCICIO 4

Se desea un programa cliente java que use sockets TCP para contactarse con un sitio web y despliegue la respuesta del servidor con el código HTML de la pagina web solicitado. El programa deberá recibir como argumento el nombre del sitio web, por ejemplo: [www.google.com](http://www.google.com)





# EJERCICIO 5

Desarrolle un servidor web que atienda peticiones de archivos con diferentes formatos (ASCII, JSON, HTML, PDF). Los archivos estarán distribuidos en dos únicos directorios: /pub y /doc

Cuando el servidor reciba una petición GET con la ruta '/doc', deberá recuperarlo desde el directorio /doc; el mismo procedimiento deberá seguirse con la ruta '/pub'. Por ejemplo: /pub/test.html

El servidor deberá verificar si el archivo solicitado existe antes de recuperarlo, de lo contrario deberá retornar una pagina comunicando esta condición de error (Codigo 404 – Not Found).

Use los modulos: http, url y fs de NodeJS.

Pruebas: Use un navegador para probar su servidor incluyendo distintos formatos y directorios.

Ejemplo: <http://localhost:9876/doc/test2.json>





# Recursos



## Clases de Auxiliatura

- <https://drive.google.com/drive/folders/1C5r7nF6YwbrxyGQXJw9OIOKBSOR0w3tn>

## Instalación DebianZurg

- <https://medium.com/@borisvargas/instalaci%C3%B3n-debian-zurg-abbe3d608cf3>

## INF-273 (I/2020)

- [https://drive.google.com/drive/folders/1UEUK9IMIPw1R\\_e3G7J99uwadJVFxZzzn](https://drive.google.com/drive/folders/1UEUK9IMIPw1R_e3G7J99uwadJVFxZzzn)

## Libros

- <https://drive.google.com/open?id=12ml6D4QG1kwcwmiXWQnkrxUL6l2Nukj3>

## Exámenes pasados

- [https://drive.google.com/drive/folders/1hCPdoSOn-1DCcPdVLJpfeuE05E\\_lxsc5](https://drive.google.com/drive/folders/1hCPdoSOn-1DCcPdVLJpfeuE05E_lxsc5)

## Examen de MESA

- [https://drive.google.com/drive/folders/1gC1kgglbTMFMobAAxHIBTqO\\_SKGaJkKF](https://drive.google.com/drive/folders/1gC1kgglbTMFMobAAxHIBTqO_SKGaJkKF)



# ¡GRACIAS!

¿Alguna pregunta?



borisvargaspaucara@gmail.com



+591 60514138

