



---

# MEMORIA DE PRÁCTICAS DE LABORATORIO DE INGENIERÍA DE SERVICIOS

---

Curso 2025/2026



Universidad de Oviedo  
*Universidá d'Uviéu*  
*University of Oviedo*

16 DE ENERO DE 2026

ALEJANDRO MEDINA FERNÁNDEZ | CARLOS ZAPICO BELLO  
<https://github.com/Carloszb/IngenieriaServicios>  
EPI Gijón

# CONTENIDO

SERVICIOS BASICOS: TCP/UDP .....	3
PRACTICA 1.3 .....	3
EJERCICIO 1 .....	3
EJERCICIO 2 .....	6
EJERCICIO 3 .....	8
EJERCICIO 4 .....	10
EJERCICIO 5 .....	12
EJERCICIO 6 .....	14
EJERCICIO 7 .....	17
PRACTICA 1.4 .....	18
EJERCICIO 1 .....	19
EJERCICIO 2 .....	21
EJERCICIO 3 .....	24
EJERCICIO 4 .....	28
EJERCICIO 5 .....	31
EJERCICIO 6 .....	35
SERVICIOS WEB .....	39
PRACTICA 2.1 .....	39
EJERCICIO 1 .....	39
EJERCICIO 2 .....	42
EJERCICIO 3 .....	44
PRACTICA 2.2 .....	45
EJERCICIO 4 .....	46
SERVICIOS DE NOMBRADO, ACCESO REMOTO Y TRANSFERENCIA DE ARCHIVOS .....	54
PRACTICA 3 .....	54
EJERCICIO 1 .....	54
EJERCICIO 2 .....	56
EJERCICIO 3 .....	59
EJERCICIO 4 .....	60
EJERCICIO 5 .....	63
EJERCICIO 6 .....	66
SERVICIOS DE CORREO ELECTRONICO .....	67
PRACTICA 4 .....	67
EJERCICIO 1 .....	67
EJERCICIO 2 .....	70
EJERCICIO 3 .....	70
EJERCICIO 4 .....	75
EJERCICIO 5 .....	76
EJERCICIO 6 .....	77
EJERCICIO 7 .....	78
EJERCICIO 8 .....	81
EJERCICIO 9 .....	83
EJERCICIO 10 .....	87
EJERCICIO 11 .....	88
EJERCICIO 12 .....	92
EJERCICIO 13 .....	95
EJERCICIO 14 .....	96
EJERCICIO 15 .....	97
EJERCICIO 16 .....	98
SERVICIOS MULTIMEDIA .....	99

PRACTICA 5 .....	99
EJERCICIO 1: Video bajo demanda .....	103
EJERCICIO 2: Video en vivo.....	115
EJERCICIO 3 : Streaming con bitrate adaptativo (ABR).....	119
SERVICIOS INTERACTIVOS DE MENSAJERIA .....	123
Practica 6 .....	123
EJERCICIO 1 .....	123
SERVIDOR .....	123
CLIENTE.....	126
EJERCICIO 2 .....	126
EJERCICIO 3 .....	127
EJERCICIO 4 .....	130
EJERCICIO 5 .....	133
EJERCICIO 6 .....	137
EJERCICIO 7 .....	139
EJERCICIO 8 .....	139
EJERCICIO 9 .....	140
SERVICIOS PARA MOVILES .....	141
PRACTICA 7 .....	141
EJERCICIO 4.1 .....	148
EJERCICIO 4.2.1 .....	149
EJERCICIO 4.4 y 4.5 .....	151
EJERCICIO 5 .....	152
PRACTICA 7.2 .....	154
EJERCICIO 1 .....	155
EJERCICIO 2 .....	155
EJERCICIO 3 .....	157
EJERCICIO 4 .....	158
EJERCICIO 5 .....	160
EJERCICIO 6 .....	161
EJERCICIO 7 .....	162
EJERCICIO 8 .....	163
EJERCICIO 9 .....	166
EJERCICIO 10 .....	167
EJERCICIO 11 .....	168
EJERCICIO 12 .....	169
EJERCICIO 13 .....	170
EJERCICIO 14 .....	171
EJERCICIO 15 .....	171
EJERCICIO 16 .....	172
EJERCICIO 17 .....	172
EJERCICIO 18 .....	173
EJERCICIO 19 .....	174
EJERCICIO 20 .....	174
Anexo .....	176

# **SERVICIOS BASICOS: TCP/UDP**

## **PRACTICA 1.3**

En esta práctica comenzamos a trabajar con programación de red utilizando Python y el protocolo UDP. A lo largo de la sesión implementamos distintos clientes y servidores que se comunican mediante el envío de datagramas, lo que nos permitió ver cómo funciona este tipo de comunicación sin conexión y sin garantías de entrega.

Empezamos creando ejemplos sencillos de cliente y servidor para intercambiar mensajes, y poco a poco fuimos añadiendo mejoras para simular problemas reales de red, como la pérdida de paquetes o la falta de confirmación. También trabajamos con numeración de mensajes, confirmaciones mediante respuestas del servidor y el uso de tiempos de espera y reintentos para hacer la comunicación más robusta.

Además, vimos cómo utilizar el broadcast en UDP para descubrir servicios dentro de una red sin conocer previamente la dirección del servidor, y finalmente aprendimos a desplegar nuestros programas dentro de contenedores Docker, lo que nos permitió ejecutar clientes y servidores en entornos aislados y simular escenarios más realistas.

### **EJERCICIO 1**

- 1. Escribe un servidor UDP que escuche en el puerto que se le pase por línea de comandos, o en el 9999 por defecto. El servidor estará en un bucle infinito en el que, para cada datagrama que llegue, imprimirá en pantalla el contenido del datagrama y la dirección de la cual proviene. Guárdalo como udp\_servidor1.py**

```
● ● ●

1 import socket
2 import sys
3
4 if len(sys.argv) > 2:
5     print("Uso: python udp_servidor1.py [puerto]")
6     sys.exit(1)
7 elif len(sys.argv) == 1:
8     puerto = 9999
9 else:
10    puerto = int(sys.argv[1])
11
12 # Crear el socket UDP
13 servidor = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
14 servidor.bind(('localhost', puerto))
15 print(f"Servidor UDP escuchando en el puerto {puerto}")
16
17 while True:
18     data, direccion = servidor.recvfrom(1024)
19     print(f"Recibido {data.decode()} de {direccion}")
20     servidor.sendto(b"ACK", direccion)
```

2. Escribe un cliente UDP que, en un bucle, lea una línea de texto del teclado y se la envíe en un datagrama al servidor anterior, hasta que la línea leída sea FIN. El cliente recibirá por línea de comandos la IP y puerto del servidor, o asumirá como valores por defecto “localhost” y 9999. Guárdalo como udp\_cliente1.py

```

● ○ ●

1 import socket
2 import sys
3
4 if len(sys.argv) > 3:
5     print("Uso: python udp_cliente1.py [host] [puerto]")
6     sys.exit(1)
7 elif len(sys.argv) == 1:
8     servidor = 'localhost'
9     puerto = 9999
10 elif len(sys.argv) == 2:
11     servidor = sys.argv[1]
12     puerto = 9999
13 else:
14     servidor = sys.argv[1]
15     puerto = int(sys.argv[2])
16
17 # Crear el socket UDP
18 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
19 server_address = (servidor, puerto)
20 print(f"Enviando mensajes a {servidor} en el puerto {puerto}")
21
22 while True:
23     mensaje = input("Introduce el mensaje a enviar (o 'FIN' para salir): ")
24     sock.sendto(mensaje.encode(), server_address)
25     if mensaje == "FIN":
26         break
27
28 sock.close()

```

3. Prueba a ejecutar el servidor en una terminal y el cliente en otra. Prueba también con cliente y servidor en diferentes máquinas (es decir, usa el servidor de tu compañero con tu cliente o viceversa).

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

- uo289853@is-01:~/IngenieriaServicios\$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp\_cliente1.py
 Enviando mensajes a localhost en el puerto 9999
 Introduce el mensaje a enviar (o 'FIN' para salir): hola
 Introduce el mensaje a enviar (o 'FIN' para salir): d
 Introduce el mensaje a enviar (o 'FIN' para salir): []

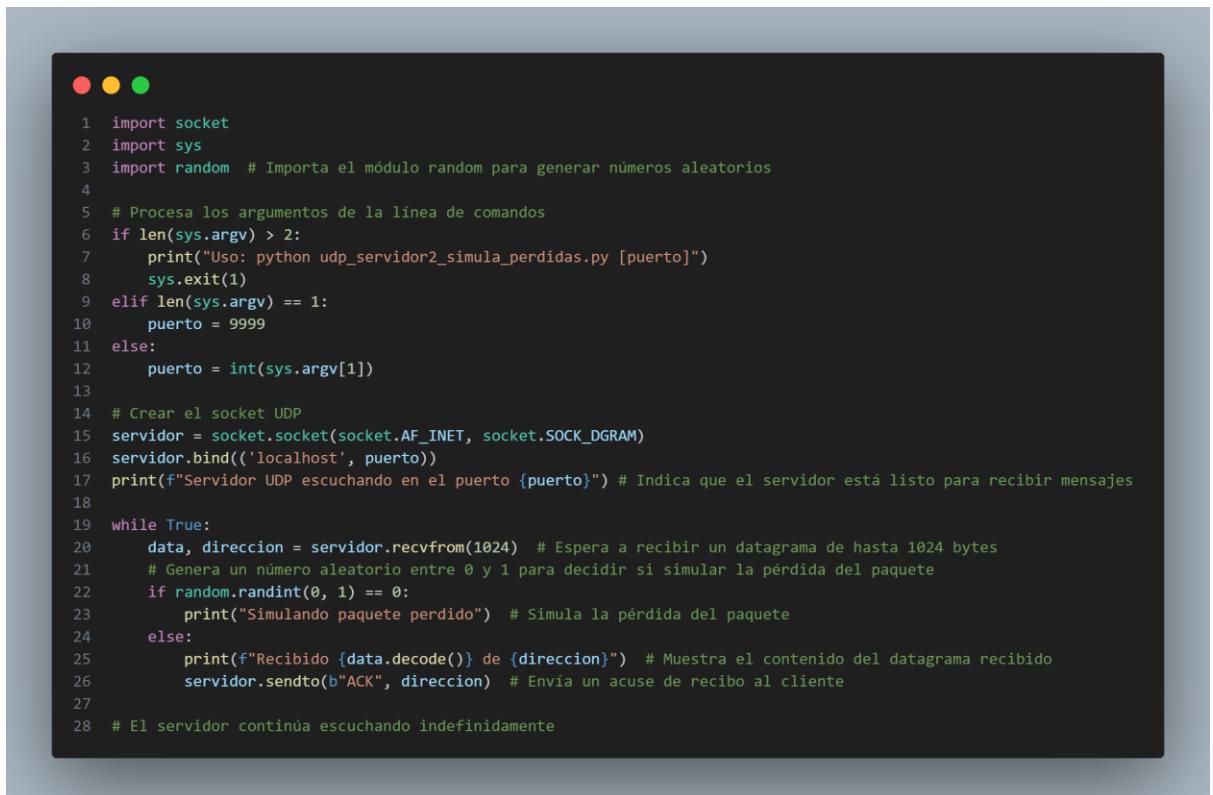
```
/bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_servidor1.py
uo289853@is-01:~/IngenieriaServicios$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_servidor1.py
Servidor UDP escuchando en el puerto 9999
Recibido hola de ('127.0.0.1', 50629)
Recibido d de ('127.0.0.1', 50629)
```

En este ejercicio implementamos por primera vez un cliente y un servidor UDP funcionales. Al principio tuvimos algunos problemas para que el servidor recibiera correctamente los mensajes, ya que olvidamos convertir los datos a bytes antes de enviarlos. Una vez corregido esto, comprobamos que

la comunicación funcionaba tanto en la misma máquina como entre equipos distintos, lo que nos ayudó a entender mejor el uso de direcciones IP y puertos.

## EJERCICIO 2

1. Modifica el servidor anterior para que, tras recibir el datagrama, decida aleatoriamente con una probabilidad del 50% si simulará no haberlo recibido. En este caso imprimirá en pantalla “Simulando paquete perdido”, en otro caso imprimirá en pantalla el contenido del datagrama tal como hacía antes. Para generar números aleatorios usar el módulo random, por ejemplo la función randint(). Guarda el programa con el nombre `udp_servidor2_simula_perdidas.py`.



```
● ● ●
1 import socket
2 import sys
3 import random # Importa el módulo random para generar números aleatorios
4
5 # Procesa los argumentos de la línea de comandos
6 if len(sys.argv) > 2:
7     print("Uso: python udp_servidor2_simula_perdidas.py [puerto]")
8     sys.exit(1)
9 elif len(sys.argv) == 1:
10     puerto = 9999
11 else:
12     puerto = int(sys.argv[1])
13
14 # Crear el socket UDP
15 servidor = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16 servidor.bind(('localhost', puerto))
17 print(f"Servidor UDP escuchando en el puerto {puerto}") # Indica que el servidor está listo para recibir mensajes
18
19 while True:
20     data, direccion = servidor.recvfrom(1024) # Espera a recibir un datagrama de hasta 1024 bytes
21     # Genera un número aleatorio entre 0 y 1 para decidir si simular la pérdida del paquete
22     if random.randint(0, 1) == 0:
23         print("Simulando paquete perdido") # Simula la pérdida del paquete
24     else:
25         print(f"Recibido {data.decode()} de {direccion}") # Muestra el contenido del datagrama recibido
26         servidor.sendto(b"ACK", direccion) # Envía un acuse de recibo al cliente
27
28 # El servidor continúa escuchando indefinidamente
```

2. Modifica el cliente para que añada al principio de cada datagrama un número (su representación como string) que vaya incrementándose por cada datagrama enviado. Así, por ejemplo, si el usuario escribe “Hola”, el datagrama enviado será “1: Hola” y el “1” se irá incrementando para sucesivos datagramas. Guarda el programa con el nombre `udp_cliente2_numera_mensajes.py`.

```

● ● ●

1 import socket
2 import sys
3
4 # Procesa los argumentos de la línea de comandos
5 if len(sys.argv) > 3:
6     print("Uso: python udp_cliente2_numera_mensajes.py [host] [puerto]")
7     sys.exit(1)
8 elif len(sys.argv) == 1:
9     servidor = 'localhost' # Host por defecto
10    puerto = 9999           # Puerto por defecto
11 elif len(sys.argv) == 2:
12    servidor = sys.argv[1] # Host proporcionado por el usuario
13    puerto = 9999           # Puerto por defecto
14 else:
15    servidor = sys.argv[1] # Host proporcionado por el usuario
16    puerto = int(sys.argv[2]) # Puerto proporcionado por el usuario
17
18 # Crear el socket UDP
19 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
20 server_address = (servidor, puerto)
21 print(f"Enviando mensajes a {servidor} en el puerto {puerto}")
22
23 contador = 1 # Inicializa el contador de mensajes
24
25 while True:
26     mensaje = input("Introduce el mensaje a enviar (o 'FIN' para salir): ")
27     datagrama = f"{contador}: {mensaje}" # Añade el número de mensaje al principio
28     sock.sendto(datagrama.encode(), server_address)
29     if mensaje == "FIN":
30         break
31     contador += 1 # Incrementa el contador
32
33 sock.close()

```

- 3. Ejecutar servidor y cliente y comprobar cómo algunos de los datagramas se pierden y por tanto la secuencia de números tiene saltos. En caso de que cliente y servidor estuvieran separados por muchos nodos intermedios podría incluso observarse que algunos de los números llegan fuera de orden.**

```

$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_cliente2_numera_mensajes.py
Envío mensajes a localhost en el puerto 9999
Introduce el mensaje a enviar (o 'FIN' para salir): ejercicio 2
Introduce el mensaje a enviar (o 'FIN' para salir): []

```

```

$ uo289853@is-01:~/IngenieriaServicios$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_servidor2_simula_perdidas.py
Servidor UDP escuchando en el puerto 9999
Recibido 1: ejercicio 2 de ('127.0.0.1', 47007)

```

Al modificar el servidor para simular la pérdida de paquetes, nos dimos cuenta de que en algunas ejecuciones parecía que no llegaba casi ningún mensaje, lo que nos hizo pensar que el programa estaba fallando. Tras revisar el código, entendimos que simplemente había coincidido una racha de valores aleatorios que provocaban muchas pérdidas seguidas.

## EJERCICIO 3

Modifica el servidor anterior para que, en caso de que decida no perder el paquete, además de mostrarlo por pantalla envíe al cliente otro datagrama conteniendo el texto “OK”. Guárdalo con el nombre `udp_servidor3_con_ok.py`.

```
● ● ●
1 import socket
2 import sys
3 import random # Importa el módulo random para generar números aleatorios
4
5 # Procesa los argumentos de la línea de comandos
6 if len(sys.argv) > 2:
7     print("Uso: python udp_servidor3_con_ok.py [puerto]")
8     sys.exit(1)
9 elif len(sys.argv) == 1:
10     puerto = 9999 # Puerto por defecto
11 else:
12     puerto = int(sys.argv[1]) # Puerto proporcionado por el usuario
13
14 # Crea un socket UDP
15 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16 sock.bind(('', puerto)) # Vincula el socket a todas las interfaces en el puerto
17 print(f"Servidor UDP escuchando en el puerto {puerto}")
18
19 while True:
20     # Espera a recibir un datagrama
21     datos, direccion = sock.recvfrom(1024) # Tamaño máximo del datagrama es 1024 bytes
22     print(f"Mensaje recibido de {direccion}: {datos.decode()}")
23
24     # Decide aleatoriamente si "pierde" el paquete o no (50% de probabilidad)
25     if random.random() < 0.5:
26         print("Paquete perdido, no se envía confirmación.")
27         continue # Simula la pérdida del paquete al no enviar respuesta
28
29     # Si no se pierde el paquete, envía una confirmación "OK" al cliente
30     respuesta = "OK"
31     sock.sendto(respuesta.encode(), direccion)
32     print(f"Confirmación enviada a {direccion}: {respuesta}")
33
34 # El servidor sigue funcionando indefinidamente
```

Modifica el cliente y guárdalo con el nombre `udp_cliente3_espera_ok.py` para que tras cada datagrama enviado espere la confirmación, pero limitando el tiempo de espera como se mostró en el código anterior.

```

1 ...
2 Ejercicio 3 (continuación)
3
4 Modifica el cliente y guárdalo con el nombre udp_cliente3_espera_ok.py para que tras cada datagrama enviado espere la confirmación, pero limitando el tiempo de espera como se mostró en el código anterior.
5
6 Ejecuta los nuevos cliente y servidor y comprueba cómo cuando el servidor "decide" simular la pérdida de un paquete, el cliente detecta correctamente la no recepción del datagrama.
7
8 Comenta el código fuente de ambos programas y envíalos a través de la plataforma.
9
10 ...
11
12 import socket
13 import sys
14
15 # Procesa los argumentos de la línea de comandos
16 if len(sys.argv) > 3:
17     print("uso: python udp_cliente3_numera_mensajes.py [host] [puerto]")
18     sys.exit(1)
19 elif len(sys.argv) == 1:
20     servidor = 'localhost' # Host por defecto
21     puerto = 9999 # Puerto por defecto
22 elif len(sys.argv) == 2:
23     servidor = sys.argv[1] # Host proporcionado por el usuario
24     puerto = 9999 # Puerto por defecto
25 else:
26     servidor = sys.argv[1] # Host proporcionado por el usuario
27     puerto = int(sys.argv[2]) # Puerto proporcionado por el usuario
28
29 # Crea el socket UDP
30 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
31 sock.settimeout(1) # Establece un tiempo de espera de 2 segundos
32 server_address = (servidor, puerto)
33 print('Enviando mensajes a {} en el puerto {}'.format(*server_address))
34 contador = 1 # Inicializa el contador de mensajes
35
36 while True:
37     mensaje = input("Introduce el mensaje a enviar (o 'FIN' para salir): ")
38     datagrama = f'{contador}:{mensaje}' # Añade el número de mensaje al principio
39     while True:
40         try:
41             sock.sendto(datagrama.encode(), server_address)
42             # Espera la confirmación del servidor
43             datos, _ = sock.recvfrom(4096)
44             print("Confirmación recibida: {}".format(datos.decode()))
45             break # Sale del bucle si se recibe la confirmación
46         except socket.timeout:
47             print("No se recibió confirmación, reenviando el mensaje...")
48     if mensaje == "FIN":
49         break
50     contador += 1 # Incrementa el contador
51
52 sock.close()

```

**Ejecuta los nuevos cliente y servidor y comprueba cómo cuando el servidor “decide” simular la pérdida de un paquete, el cliente detecta correctamente la no recepción del datagrama.**

- **uo289853@is-01:~/IngenieriaServicios\$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp\_cliente3\_espera\_ok.py**

Enviando mensajes a localhost en el puerto 9999  
 Introduce el mensaje a enviar (o 'FIN' para salir): s  
 dNo se recibió confirmación, reenviando el mensaje...  
 No se recibió confirmación, reenviando el mensaje...  
 No se recibió confirmación, reenviando el mensaje...  
 Confirmación recibida: OK  
 Introduce el mensaje a enviar (o 'FIN' para salir):  
 Confirmación recibida: OK  
 Introduce el mensaje a enviar (o 'FIN' para salir): d  
 fNo se recibió confirmación, reenviando el mensaje...  
 No se recibió confirmación, reenviando el mensaje...  
 Confirmación recibida: OK  
 Introduce el mensaje a enviar (o 'FIN' para salir): []

```
o /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_servidor3_con_ok.py
Servidor UDP escuchando en el puerto 9999
Mensaje recibido de ('127.0.0.1', 44606): 1: s
Paquete perdido, no se envía confirmación.
Mensaje recibido de ('127.0.0.1', 44606): 1: s
Paquete perdido, no se envía confirmación.
Mensaje recibido de ('127.0.0.1', 44606): 1: s
Paquete perdido, no se envía confirmación.
Mensaje recibido de ('127.0.0.1', 44606): 1: s
Confirmación enviada a ('127.0.0.1', 44606): OK
Mensaje recibido de ('127.0.0.1', 44606): 2: 
Confirmación enviada a ('127.0.0.1', 44606): OK
Mensaje recibido de ('127.0.0.1', 44606): 3: d
Paquete perdido, no se envía confirmación.
Mensaje recibido de ('127.0.0.1', 44606): 3: d
Paquete perdido, no se envía confirmación.
Mensaje recibido de ('127.0.0.1', 44606): 3: d
Confirmación enviada a ('127.0.0.1', 44606): OK
```

Cuando añadimos el envío del mensaje “OK” por parte del servidor, tuvimos un problema inicial con el cliente quedándose bloqueado esperando una respuesta. Esto ocurría cuando el servidor simulaba la pérdida del paquete y no enviaba la confirmación. La solución fue implementar un timeout en la recepción, lo que nos permitió detectar correctamente cuándo no llegaba la respuesta y evitar que el cliente se quedara bloqueado indefinidamente.

## EJERCICIO 4

Modifica el cliente anterior y guárdalo como `udp_cliente4_reintenta.py` de modo que en caso de agotar el tiempo de espera (`timeout`), el cliente repita el envío, duplicando en cada reenvío el valor del `timeout` hasta que, bien se reciba el “OK”, bien el `timeout` exceda el valor de 2 segundos. En el primer caso volverá a pedir datos al usuario para volver a enviarlos al servidor, usando el mismo esquema. En el segundo caso imprimirá un mensaje “Puede que el servidor esté caído. Inténtelo más tarde” y finalizará su ejecución. Observa que en los reintentos el número de secuencia del datagrama no se debe incrementar.

```

1 import socket
2 import sys
3
4 # Procesa los argumentos de la línea de comandos
5 if len(sys.argv) > 3:
6     print("Uso: python udp_cliente4_reintenta.py [host] [puerto]")
7     sys.exit(1)
8 elif len(sys.argv) == 1:
9     servidor = 'localhost' # Host por defecto
10    puerto = 9999 # Puerto por defecto
11 elif len(sys.argv) == 2:
12    servidor = sys.argv[1] # Host proporcionado por el usuario
13    puerto = 9999 # Puerto por defecto
14 else:
15    servidor = sys.argv[1] # Host proporcionado por el usuario
16    puerto = int(sys.argv[2]) # Puerto proporcionado por el usuario
17
18 # Crear el socket UDP
19 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
20 server_address = (servidor, puerto)
21 print(f"Enviando mensajes a {servidor} en el puerto {puerto}")
22 contador = 1 # Inicializa el contador de mensajes
23
24 while True:
25     mensaje = input("Introduce el mensaje a enviar (o 'FIN' para salir): ")
26     datagrama = f"{contador}: {mensaje}" # Añade el número de mensaje al principio
27     timeout = 0.25 # Timeout inicial en segundos
28     sock.settimeout(timeout)
29     enviado = False
30     while True:
31         try:
32             sock.sendto(datagrama.encode(), server_address)
33             datos, _ = sock.recvfrom(4096)
34             respuesta = datos.decode()
35             print(f"Confirmación recibida: {respuesta}")
36             enviado = True
37             break # Sale del bucle si se recibe la confirmación
38         except socket.timeout:
39             if timeout >= 2.0:
40                 print("Puede que el servidor esté caído. Inténtelo más tarde")
41                 sock.close()
42                 sys.exit(1)
43             timeout *= 2
44             print(f"No se recibió confirmación, reenviando el mensaje... (timeout={timeout:.2f}s)")
45             sock.settimeout(timeout)
46     if mensaje == "FIN":
47         break
48     if enviado:
49         contador += 1 # Incrementa el contador solo si se recibió OK
50
51 sock.close()
52
53
54

```

**Ejecuta el cliente anterior junto con el servidor anterior, `udp_servidor3_con_ok.py` y comprueba que funcione.**

**Ejecuta el cliente usando como IP y puerto unas en las que no haya ningún servidor escuchando. Comprueba cómo detecta correctamente el problema (tras varios reintentos) y termina su ejecución.**

```
○ uo289853@is-01:~/IngenieriaServicios$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_cliente4_reintenta.py
Enviando mensajes a localhost en el puerto 9999
Introduce el mensaje a enviar (o 'FIN' para salir): ej4
Confirmación recibida: OK
Introduce el mensaje a enviar (o 'FIN' para salir): hola
No se recibió confirmación, reenviando el mensaje... (timeout=0.50s)
Confirmación recibida: OK
Introduce el mensaje a enviar (o 'FIN' para salir):
```

En este ejercicio implementamos el reenvío de paquetes cuando no se recibía la confirmación. Uno de los errores que cometimos al principio fue incrementar el número de secuencia en cada reintento, lo que hacía que el servidor interpretara los mensajes como distintos. Tras corregirlo y mantener el mismo número de secuencia en los reenvíos, el sistema empezó a funcionar correctamente y pudimos comprobar cómo el cliente aumentaba progresivamente el timeout antes de rendirse.

## EJERCICIO 5

Implementa al menos una de las mejoras propuestas en este apartado. Llama a los ficheros `udp_servidor5_mejorado.py` y `udp_cliente5_mejorado.py`.

```

1 """
2 Mejora implementada:
3 - Uso de connect() y recv()
4 """
5
6 import socket
7 import sys
8
9 # Procesa los argumentos de la línea de comandos
10 if len(sys.argv) > 3:
11     print("Uso: python udp_cliente5_mejorado.py [host] [puerto]")
12     sys.exit(1)
13 elif len(sys.argv) == 1:
14     servidor = 'localhost' # Host por defecto
15     puerto = 9999 # Puerto por defecto
16 elif len(sys.argv) == 2:
17     servidor = sys.argv[1] # Host proporcionado por el usuario
18     puerto = 9999 # Puerto por defecto
19 else:
20     servidor = sys.argv[1] # Host proporcionado por el usuario
21     puerto = int(sys.argv[2]) # Puerto proporcionado por el usuario
22
23 # Crear el socket UDP
24 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
25 server_address = (servidor, puerto)
26
27 # *** MEJORA DE SEGURIDAD ***
28 # Usar connect() para "asociar" el socket con el servidor específico
29 # Esto hace que recv() solo acepte datagramas de esta dirección
30 sock.connect(server_address)
31
32 print(f"Enviando mensajes a {servidor} en el puerto {puerto}")
33 print("(Usando connect() para mayor seguridad - solo acepta datagramas del servidor autorizado)")
34 contador = 1 # Inicializa el contador de mensajes
35
36 while True:
37     mensaje = input("Introduce el mensaje a enviar (o 'FIN' para salir): ")
38     datagrama = f"{contador}: {mensaje}" # Añade el número de mensaje al principio
39     timeout = 0.25 # Timeout inicial en segundos
40     sock.settimeout(timeout)
41     enviado = False
42     while True:
43         try:
44             # *** MEJORA DE SEGURIDAD ***
45             # Usar send() en lugar de sendto() (ya hicimos connect())
46             sock.send(datagrama.encode())
47
48             # *** MEJORA DE SEGURIDAD ***
49             # Usar recv() en lugar de recvfrom()
50             # recv() automáticamente descarta datagramas que no vengan
51             # de la dirección especificada en connect()
52             datos = sock.recv(4096)
53
54             respuesta = datos.decode()
55             print(f"Confirmación recibida: {respuesta}")
56             enviado = True
57             break # Sale del bucle si se recibe la confirmación
58         except socket.timeout:
59             if timeout >= 2.0:
60                 print("Puede que el servidor esté caido. Inténtelo más tarde")
61                 sock.close()
62                 sys.exit(1)
63             timeout *= 2
64             print(f"No se recibió confirmación, reenviando el mensaje... (timeout={timeout:.2f}s)")
65             sock.settimeout(timeout)
66         if mensaje == "FIN":
67             break
68         if enviado:
69             contador += 1 # Incrementa el contador solo si se recibió OK
70
71 sock.close()

```

```

1 import socket
2 import sys
3 import random # Importa el módulo random para generar números aleatorios
4
5 # Procesa los argumentos de la línea de comandos
6 if len(sys.argv) > 2:
7     print("Uso: python udp_servidor5_mejorado.py [puerto]")
8     sys.exit(1)
9 elif len(sys.argv) == 1:
10     puerto = 9999 # Puerto por defecto
11 else:
12     puerto = int(sys.argv[1]) # Puerto proporcionado por el usuario
13
14 # Crea un socket UDP
15 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
16 sock.bind(('', puerto)) # Vincula el socket a todas las interfaces en el puerto
17 print(f"Servidor UDP escuchando en el puerto {puerto}")
18
19 while True:
20     # Espera a recibir un datagrama
21     datos, direccion = sock.recvfrom(1024) # Tamaño máximo del datagrama es 1024 bytes
22     print(f"Mensaje recibido de {direccion}: {datos.decode()}")
23
24     # Decide aleatoriamente si "pierde" el paquete o no (50% de probabilidad)
25     if random.random() < 0.5:
26         print("Paquete perdido, no se envía confirmación.")
27         continue # Simula la pérdida del paquete al no enviar respuesta
28
29     # Si no se pierde el paquete, envía una confirmación "OK" al cliente
30     respuesta = "OK"
31     sock.sendto(respuesta.encode(), direccion)
32     print(f"Confirmación enviada a {direccion}: {respuesta}")
33
34 # El servidor sigue funcionando indefinidamente

```

## EJERCICIO 6

**Implementa el servidor y el cliente descritos en los apartados anteriores, dándoles los nombres `udp_servidor6_broadcast.py` y `udp_cliente6_broadcast.py`, respectivamente.**

**Prueba el cliente cuando haya varios servidores funcionando (de otros compañeros) y comprueba cómo recibe varias respuestas.**

```

1 """
2 udp_cliente6_broadcast.py - Cliente de descubrimiento por broadcast
3
4 El Cliente:
5 1. Envía un broadcast "BUSCANDO HOLA" al puerto 12345
6 2. Recibe respuestas de servidores con timeout
7 3. Se conecta al primer servidor que responde y le envía "HOLA"
8 4. Muestra la respuesta final del servidor
9 """
10
11 import socket
12 import sys
13 import time
14
15 def obtener_direccion_broadcast():
16     """
17     Uso 255.255.255 (broadcast global)
18     """
19     return '255.255.255.255'
20
21 def main():
22     # Procesa los argumentos de la linea de comandos
23     if len(sys.argv) > 2:
24         print("Uso: python udp_cliente6_broadcast.py [puerto_broadcast]")
25         sys.exit(1)
26     elif len(sys.argv) == 1:
27         puerto_broadcast = 12345 # Puerto por defecto
28     else:
29         puerto_broadcast = int(sys.argv[1])
30
31     # Crear el socket UDP
32     sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
33
34     # Habilitar el modo broadcast
35     sock.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
36
37     # Vincular a cualquier puerto disponible
38     sock.bind(('', 0))
39
40     print("==> Cliente de Descubrimiento por Broadcast ===")
41     print(f"Buscando servidores en puerto {puerto_broadcast}...")
42
43     # Paso 1: Enviar broadcast "BUSCANDO HOLA"
44     direccion_broadcast = obtener_direccion_broadcast()
45     mensaje_broadcast = "BUSCANDO HOLA"
46
47     print(f"Enviando broadcast a {direccion_broadcast}:{puerto_broadcast}: '{mensaje_broadcast}'")
48     sock.sendto(mensaje_broadcast.encode('utf-8'), (direccion_broadcast, puerto_broadcast))
49
50     # Paso 2: Recibir respuestas con timeout
51     servidores_encontrados = []
52     primer_servidor = None
53     timeout = 3.0 # 3 segundos para esperar respuestas
54
55     sock.settimeout(timeout)
56     print(f"Esperando respuestas durante {timeout} segundos...")
57
58     start_time = time.time()
59     while True:
60         try:
61             datos, direccion = sock.recvfrom(1024)
62             respuesta = datos.decode('utf-8').strip()
63
64             print(f"Respuesta recibida de {direccion[0]}:{direccion[1]}: '{respuesta}'")
65
66             if respuesta == "AQUI ESTOY":
67                 servidores_encontrados.append(direccion[0])
68
69                 # Guardar la IP del primer servidor que responde
70                 if primer_servidor is None:
71                     primer_servidor = direccion[0]
72                     print(f"✓ Primer servidor encontrado: {primer_servidor}")
73
74                 # Verificar si el timeout ha expirado manualmente
75                 if time.time() - start_time > timeout:
76                     break
77
78             except socket.timeout:
79                 print("Timeout alcanzado, no se esperan más respuestas")
80                 break
81             except Exception as e:
82                 print(f"Error recibiendo respuestas: {e}")
83                 break
84
85     # Mostrar resumen de servidores encontrados
86     print("\n==> Resumen del Descubrimiento ===")
87     print(f"Servidores encontrados: [{len(servidores_encontrados)}]")
88     for i, ip in enumerate(servidores_encontrados, 1):
89         print(f" {i}: {ip}")
90
91     # Paso 3 y 4: Si hay servidores, probar el servicio con el primero
92     if primer_servidor:
93         print(f"\n==> Probando Servicio con {primer_servidor} ===")
94
95         try:
96             # Enviar "HOLA" al primer servidor
97             mensaje_hola = "HOLA"
98             print(f"Enviando '{mensaje_hola}' a {primer_servidor}:{puerto_broadcast}")
99
100            sock.settimeout(5.0) # Timeout más largo para la comunicación directa
101            sock.sendto(mensaje_hola.encode('utf-8'), (primer_servidor, puerto_broadcast))
102
103            # Esperar respuesta
104            datos, direccion = sock.recvfrom(1024)
105            respuesta_servicio = datos.decode('utf-8')
106
107            print(f"✓ Respuesta del servicio: '{respuesta_servicio}'")
108            print(f"✓ Comunicación exitosa con el servidor {direccion[0]}")
109
110            except socket.timeout:
111                print("X Timeout esperando respuesta del servicio")
112            except Exception as e:
113                print(f"X Error comunicándose con el servidor: {e}")
114        else:
115            print("\nX No se encontraron servidores disponibles")
116            print("Asegúrate de que hay al menos un servidor ejecutándose con:")
117            print(f"  python3 udp_servidor6_broadcast.py {puerto_broadcast}")
118
119            sock.close()
120            print("\nCliente finalizado")
121
122 if __name__ == "__main__":
123     main()

```

```

● ● ●

1 import socket
2 import sys
3
4 # Procesa los argumentos de la línea de comandos
5 if len(sys.argv) > 2:
6     print("Uso: python udp_servidor6_broadcast.py [puerto]")
7     sys.exit(1)
8 elif len(sys.argv) == 1:
9     puerto = 12345 # Puerto por defecto para broadcast
10 else:
11     puerto = int(sys.argv[1])
12
13 # Crear el socket UDP
14 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
15
16 # Permitir la reutilización de la dirección (importante para broadcast)
17 sock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
18
19 # Vincular el socket a todas las interfaces en el puerto especificado
20 sock.bind(('', puerto))
21
22 print(f"Servidor de broadcast escuchando en puerto {puerto}")
23 print("Esperando mensajes 'BUSCANDO HOLA' y 'HOLA'...")
24
25 while True:
26     try:
27         # Esperar a recibir un datagrama
28         datos, direccion = sock.recvfrom(1024)
29         mensaje = datos.decode('utf-8').strip()
30
31         print(f"Mensaje recibido de {direccion}: '{mensaje}'")
32
33         if mensaje == "BUSCANDO HOLA":
34             # Responder al broadcast de descubrimiento
35             respuesta = "AQUI ESTOY"
36             sock.sendto(respuesta.encode('utf-8'), direccion)
37             print(f"Respuesta de descubrimiento enviada a {direccion}: '{respuesta}'")
38
39         elif mensaje == "HOLA":
40             # Responder al saludo directo
41             respuesta = f";Hola! Soy el servidor {sock.getsockname()[0]}:{puerto}"
42             sock.sendto(respuesta.encode('utf-8'), direccion)
43             print(f"Saludo enviado a {direccion}: '{respuesta}'")
44
45         else:
46             # Mensaje no reconocido
47             respuesta = "Mensaje no reconocido. Usa 'BUSCANDO HOLA' o 'HOLA'"
48             sock.sendto(respuesta.encode('utf-8'), direccion)
49             print(f"Respuesta de error enviada a {direccion}")
50
51     except KeyboardInterrupt:
52         print("\nServidor detenido por el usuario")
53         break
54     except Exception as e:
55         print(f"Error procesando mensaje: {e}")
56
57 sock.close()
58 print("Servidor finalizado")

```

```
● uo289853@is-01:~/IngenieriaServicios$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_cliente6_broadcast.py
    === Cliente de Descubrimiento por Broadcast ===
    Buscando servidores en puerto 12345...
    Enviando broadcast a 255.255.255.255:12345: 'BUSCANDO HOLA'
    Esperando respuestas durante 3.0 segundos...
    Respuesta recibida de 10.38.18.240:12345: 'AQUI ESTOY'
    ✓ Primer servidor encontrado: 10.38.18.240
    Timeout alcanzado, no se esperan más respuestas

    === Resumen del Descubrimiento ===
    Servidores encontrados: 1
        1. 10.38.18.240

    === Probando Servicio con 10.38.18.240 ===
    Enviando 'HOLA' a 10.38.18.240:12345
    ✓ Respuesta del servicio: '¡Hola! Soy el servidor 0.0.0.0:12345'
    ✓ Comunicación exitosa con el servidor 10.38.18.240

    Cliente finalizado
```

```
● uo289853@is-01:~/IngenieriaServicios$ /bin/python3 /home/uo289853/IngenieriaServicios/Practica1.3/udp_servidor6_broadcast.py
    Servidor de broadcast escuchando en puerto 12345
    Esperando mensajes 'BUSCANDO HOLA' y 'HOLA'...
    Mensaje recibido de ('10.38.18.240', 39320): 'BUSCANDO HOLA'
    Respuesta de descubrimiento enviada a ('10.38.18.240', 39320): 'AQUI ESTOY'
    Mensaje recibido de ('10.38.18.240', 39320): 'HOLA'
    Saludo enviado a ('10.38.18.240', 39320): '¡Hola! Soy el servidor 0.0.0.0:12345'
```

Al implementar el cliente y servidor con broadcast, tuvimos dificultades para encontrar la dirección de la red, especialmente dentro de los contenedores Docker. Después de revisar la configuración de red y utilizar herramientas como ip addr y docker inspect, conseguimos sacar correctamente la dirección de broadcast.

## EJERCICIO 7

En la subred pruebas antes creada lanza tres contenedores que ejecuten `udp_servidor6_broadcast.py`. Dale un nombre diferente a cada uno, o no des nombre para que docker elija uno al azar (en este caso el nombre no importa porque el cliente no va a usarlo). Verifica que están todos en ejecución con `docker ps`.

```
uo289853@is-01:~/IngenieriaServicios/Practical.3$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
PORTS NAMES
9a6fef797ce2      python:3.7        "python /app/udp_ser..."   5 seconds ago     Up 4 seconds
ds                 thirsty_lewin
3bf2a972b969      python:3.7        "python /app/udp_ser..."   8 seconds ago     Up 7 seconds
ds                 servidor2
1d461e50569c      python:3.7        "python /app/udp_ser..."   11 seconds ago    Up 11 seconds
nds                servidor1
```

Averigua la dirección de broadcast de esta subred, entrando en uno de esos contenedores y ejecutando `ip addr` (este comando no está instalado por defecto, por lo que debes instalar antes el paquete iproute2 dentro del contenedor en marcha). Ahí podrás ver la IP de la interfaz, en su notación CIDR, por ejemplo 172.18.0.2/16 que indica que de la IP dada, los 16 primeros bits son la parte de red. Esto nos permite deducir la IP de broadcast.

```
"EndpointID": "0586d70a8c37ba8e27e7cbe44812dfb6e7c5a6699892e08e739bfa",
    "Gateway": "172.18.0.1",
    "IPAddress": "172.18.0.2",
    "IPPrefixLen": 16,
    "IPv6Gateway": "",
    "GlobalIPv6Address": "",
    "GlobalIPv6PrefixLen": 0,
    "DNSNames": [
        "servidor1",
        "servidor2",
        "servidor3"]
```

Alternativamente esta información puede obtenerse desde fuera del contenedor y por tanto sin tener que instalar nada en él, ejecutando docker inspect <nombre\_del\_contenedor> y observando las variables IPAddress e IPPrefixLen que mostrarían en este caso 172.18.0.2 y 16 respectivamente. Compruébalo.

Lanza ahora el cliente udp\_cliente6\_broadcast.py usando como IP de broadcast la que has averiguado.

Comprueba que los tres servidores responden y que el cliente elige al primero de ellos para enviarle la petición. ¿Cuál es la IP del cliente? (mírala en la respuesta del servidor).

```
uo289853@is-01:~/IngenieriaServicios/Practical.3$ docker run --rm --name cliente --network pruebas -v $(pwd):/app python:3.7 python /app/udp_cliente6_broadcast.py 5000
==> Cliente de Descubrimiento por Broadcast ==
Buscando servidores en puerto 5000...
Enviando broadcast a 255.255.255.5000: 'BUSCANDO HOLA'
Esperando respuestas durante 3.0 segundos...
Respuesta recibida de 172.18.0.3:5000: 'AQUI ESTOY'
✓ Primer servidor encontrado: 172.18.0.3
Respuesta recibida de 172.18.0.2:5000: 'AQUI ESTOY'
Timeout alcanzado, no se esperan más respuestas

==> Resumen del Descubrimiento ==
Servidores encontrados: 2
  1. 172.18.0.3
  2. 172.18.0.2

==> Probando Servicio con 172.18.0.3 ==
Enviando 'HOLA' a 172.18.0.3:5000
✓ Respuesta del servicio: '!Hola! Soy el servidor 0.0.0.0:5000'
✓ Comunicación exitosa con el servidor 172.18.0.3
```

Guarda en dos ficheros los scripts shell necesarios para lanzar a todos los servidores (udp\_docker\_lanzar\_servidores.sh) y el que lanza al cliente (udp\_docker\_lanzar\_cliente.sh).

## PRACTICA 1.4

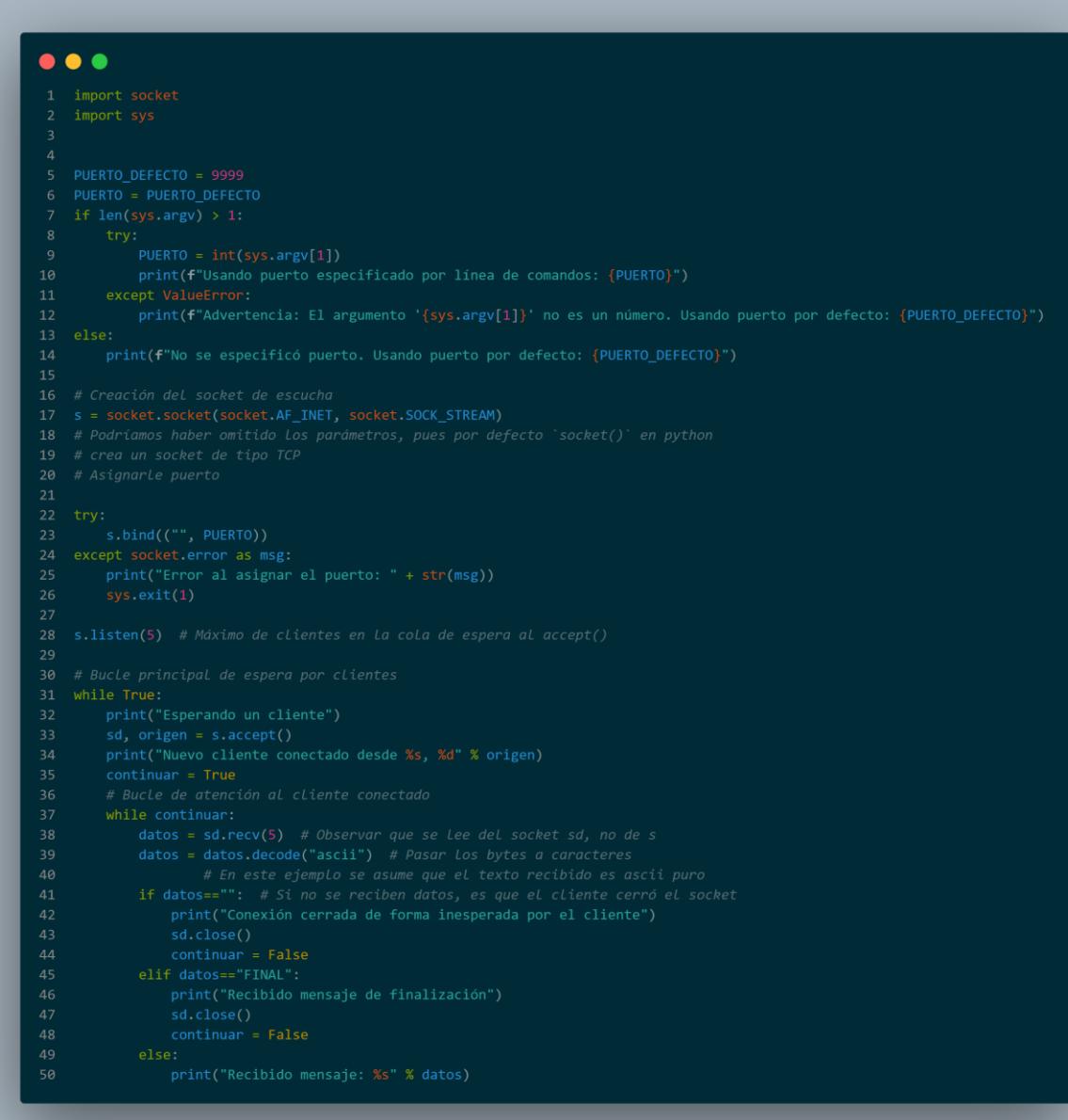
En esta práctica continuamos trabajando con programación de red en Python, pero esta vez utilizando el protocolo TCP. A lo largo de la sesión implementamos distintos clientes y servidores para ver cómo funciona la comunicación orientada a conexión y cómo TCP nos evita algunos de los problemas que aparecían al trabajar con UDP, como la pérdida de paquetes o la necesidad de implementar confirmaciones manualmente.

Empezamos creando un servidor y un cliente TCP sencillos, y poco a poco fuimos viendo que, aunque TCP garantiza la entrega y el orden de los datos, aparecen nuevos problemas relacionados con la delimitación de los mensajes. Trabajamos diferentes técnicas para saber dónde empieza y acaba cada mensaje dentro del flujo de datos, como el uso de tamaños fijos, marcas de fin de mensaje y el envío previo de la longitud del mensaje.

Además, implementamos varias versiones del mismo servicio para comparar los distintos enfoques y ver sus ventajas e inconvenientes en situaciones más realistas. Por último, desplegamos los servicios dentro de contenedores Docker, lo que nos permitió probar la comunicación entre contenedores y también el acceso desde fuera del entorno Docker mediante el mapeo de puertos.

## EJERCICIO 1

Completa el código anterior con lo necesario para que funcione. El puerto en que debe escuchar lo recibirá por línea de comandos o usará un valor por defecto de 9999 si no se especifica. Guárdalo con el nombre `tcp_servidor1_simple.py`



```
1 import socket
2 import sys
3
4
5 PUERTO_DEFECTO = 9999
6 PUERTO = PUERTO_DEFECTO
7 if len(sys.argv) > 1:
8     try:
9         PUERTO = int(sys.argv[1])
10        print(f"Usando puerto especificado por línea de comandos: {PUERTO}")
11    except ValueError:
12        print(f"Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}")
13 else:
14     print(f"No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}")
15
16 # Creación del socket de escucha
17 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18 # Podríamos haber omitido los parámetros, pues por defecto `socket()` en python
19 # crea un socket de tipo TCP
20 # Asignarle puerto
21
22 try:
23     s.bind("", PUERTO)
24 except socket.error as msg:
25     print("Error al asignar el puerto: " + str(msg))
26     sys.exit(1)
27
28 s.listen(5) # Máximo de clientes en la cola de espera al accept()
29
30 # Bucle principal de espera por clientes
31 while True:
32     print("Esperando un cliente")
33     sd, origen = s.accept()
34     print("Nuevo cliente conectado desde %s, %d" % origen)
35     continuar = True
36     # Bucle de atención al cliente conectado
37     while continuar:
38         datos = sd.recv(5) # Observar que se lee del socket sd, no de s
39         datos = datos.decode("ascii") # Pasar los bytes a caracteres
40         # En este ejemplo se asume que el texto recibido es ascii puro
41         if datos=="": # Si no se reciben datos, es que el cliente cerró el socket
42             print("Conexión cerrada de forma inesperada por el cliente")
43             sd.close()
44             continuar = False
45         elif datos=="FINAL":
46             print("Recibido mensaje de finalización")
47             sd.close()
48             continuar = False
49         else:
50             print("Recibido mensaje: %s" % datos)
```

Escribe también un cliente para probar el servidor, llamado `tcp_cliente1_simple.py`. Este cliente debe:

1. Crear un socket TCP y conectarlo con el servidor (recibirá la IP y puerto del servidor por línea de comandos o usará localhost y 9999 si no se especifican argumentos).

2. Repetir 5 veces un bucle en el que envíe el texto “ABCDE” (observa que son exactamente 5 bytes, tal como espera el servidor en cada envío)
3. Tras los 5 envíos anteriores hacer un último envío del texto “FINAL”, cerrar el socket y terminar.

```

1 import socket
2 import sys
3
4
5 PUERTO_DEFECTO = 9999
6 PUERTO = PUERTO_DEFECTO
7 if len(sys.argv) > 1:
8     try:
9         PUERTO = int(sys.argv[1])
10        print(f"Usando puerto especificado por linea de comandos: {PUERTO}")
11    except ValueError:
12        print(f"Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}")
13 else:
14    print(f"No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}")
15
16 # Creación del socket de escucha
17 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18 # Podríamos haber omitido los parámetros, pues por defecto `socket()` en python
19 # crea un socket de tipo TCP
20 # Asignarle puerto
21
22 try:
23     s.bind("", PUERTO)
24 except socket.error as msg:
25     print("Error al asignar el puerto: " + str(msg))
26     sys.exit(1)
27
28 s.listen(5) # Máximo de clientes en la cola de espera al accept()
29
30 # Bucle principal de espera por clientes
31 while True:
32     print("Esperando un cliente")
33     sd, origen = s.accept()
34     print("Nuevo cliente conectado desde %s, %d" % origen)
35     continuar = True
36     # Bucle de atención al cliente conectado
37     while continuar:
38         datos = sd.recv(5) # Observar que se lee del socket sd, no de s
39         datos = datos.decode("ascii") # Pasar los bytes a caracteres
40         # En este ejemplo se asume que el texto recibido es ascii puro
41         if datos=="": # Si no se reciben datos, es que el cliente cerró el socket
42             print("Conexión cerrada de forma inesperada por el cliente")
43             sd.close()
44             continuar = False
45         elif datos=="FINAL":
46             print("Recibido mensaje de finalización")
47             sd.close()
48             continuar = False
49         else:
50             print("Recibido mensaje: %s" % datos)

```

**Prueba el cliente contra el servidor anterior y observa si llegan bien los paquetes iniciales y el que marca el final.**

```

● uo288967@is02:~/IngenieriaServicios$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente1_simple.py
No se especificaron IP ni Puerto. Usando por defecto: localhost:9999
Intentando conectar a localhost:9999...
Conexión exitosa. Iniciando envíos de datos.
Envío 1/5: Enviado 'ABCDE'.
Envío 2/5: Enviado 'ABCDE'.
Envío 3/5: Enviado 'ABCDE'.
Envío 4/5: Enviado 'ABCDE'.
Envío 5/5: Enviado 'ABCDE'.
Envío final: Enviado 'FINAL'.
Cerrando socket cliente y terminando.
* uo288967@is02:~/IngenieriaServicios$
```

En este ejercicio implementamos un cliente y un servidor TCP que intercambiaban mensajes de tamaño fijo. Al principio nos llamó la atención que, aunque el cliente enviaba siempre 5 bytes, el servidor no siempre los recibía como esperábamos.

## EJERCICIO 2

Modifica el servidor del ejercicio 1, renombrándolo a `tcp_servidor2_recvall.py` para que incluya una función llamada `recvall()` que reciba como parámetro un socket y un entero, que indica cuántos bytes deben recibirse. La función usará un bucle llamando a `socket.recv()` para intentar recibir los bytes solicitados, tantas veces como sea necesario en un bucle hasta haberlos leído todos. Deberá hacerse de forma eficiente, es decir, intentará leer el número de bytes solicitado y si no llegan todos leer los restantes, etc. en lugar de leerlos de uno en uno. Retornará un string con los bytes recibidos (si son necesarias varias lecturas, los irá concatenando).

El servidor hará uso de esta función, en lugar de `recv()` para recibir cada paquete del cliente.



```
IngenieriaServicios [SSH: 10.38.13.5] - tcp_servidor1_simple.py

1 import socket
2 import sys
3
4
5 PUERTO_DEFECTO = 9999
6 PUERTO = PUERTO_DEFECTO
7 if len(sys.argv) > 1:
8     try:
9         PUERTO = int(sys.argv[1])
10        print(f"Usando puerto especificado por linea de comandos: {PUERTO}")
11    except ValueError:
12        print(f"Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}")
13 else:
14    print(f"No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}")
15
16 # Creación del socket de escucha
17 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18 # Podríamos haber omitido los parámetros, pues por defecto `socket()` en python
19 # crea un socket de tipo TCP
20 # Asignarle puerto
21
22 try:
23     s.bind("", PUERTO)
24 except socket.error as msg:
25     print("Error al asignar el puerto: " + str(msg))
26     sys.exit(1)
27
28 s.listen(5) # Máximo de clientes en la cola de espera al accept()
29
30 # Bucle principal de espera por clientes
31 while True:
32     print("Esperando un cliente")
33     sd, origen = s.accept()
34     print("Nuevo cliente conectado desde %s, %d" % origen)
35     continuar = True
36     # Bucle de atención al cliente conectado
37     while continuar:
38         datos = sd.recv(5) # Observar que se lee del socket sd, no de s
39         datos = datos.decode("ascii") # Pasar los bytes a caracteres
40         # En este ejemplo se asume que el texto recibido es ascii puro
41         if datos=="": # Si no se reciben datos, es que el cliente cerró el socket
42             print("Conexión cerrada de forma inesperada por el cliente")
43             sd.close()
44             continuar = False
45         elif datos=="FINAL":
46             print("Recibido mensaje de finalización")
47             sd.close()
48             continuar = False
49         else:
50             print("Recibido mensaje: %s" % datos)
```

Modifica también el cliente (llámalo `tcp_cliente2_sendall.py`) para que use la función de `python sendall()` en lugar de `send()`.

```

IngenieriaServicios [SSH: 10.38.13.5] - tcp_cliente2_sendall.py

1 import socket
2 import sys
3 import time
4
5 # --- Configuración del Servidor ---
6 IP_DEFECTO = 'localhost'
7 PUERTO_DEFECTO = 9999
8
9 IP_SERVIDOR = IP_DEFECTO
10 PUERTO_SERVIDOR = PUERTO_DEFECTO
11
12 # Verifica si se han pasado IP y Puerto por Línea de comandos
13 if len(sys.argv) > 2:
14     # Usar IP y Puerto especificados
15     IP_SERVIDOR = sys.argv[1]
16     try:
17         PUERTO_SERVIDOR = int(sys.argv[2])
18     except ValueError:
19         print(f"Advertencia: El puerto '{sys.argv[2]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}")
20         PUERTO_SERVIDOR = PUERTO_DEFECTO
21 elif len(sys.argv) == 2:
22     # Solo se pasó La IP
23     IP_SERVIDOR = sys.argv[1]
24     print(f"Usando IP: {IP_SERVIDOR}. Puerto por defecto: {PUERTO_DEFECTO}")
25 else:
26     # Usar valores por defecto
27     print(f"No se especificaron IP ni Puerto. Usando por defecto: {IP_DEFECTO}:{PUERTO_DEFECTO}")
28
29 # --- Creación y Conexión del Socket ---
30 try:
31     # Crear un socket TCP
32     c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
33
34     # Conectarse al servidor
35     print(f"Intentando conectar a {IP_SERVIDOR}:{PUERTO_SERVIDOR}...")
36     c.connect((IP_SERVIDOR, PUERTO_SERVIDOR))
37     print("Conexión exitosa. Iniciando envíos de datos.")
38
39 # --- Bucle de Envío de Datos ---
40 MENSAJE_5_BYTES = "ABCDE"
41 NUM_ENVIOS = 5
42
43 for i in range(NUM_ENVIOS):
44     # *** Usando sendall() ***: Garantiza que se envian todos Los bytes.
45     try:
46         c.sendall(MENSAJE_5_BYTES.encode('ascii'))
47         print(f"Envio {i+1}/{NUM_ENVIOS}: Enviado '{MENSAJE_5_BYTES}'")
48     except socket.error as e:
49         print(f"Error al enviar datos: {e}")
50         break
51     time.sleep(0.1)
52
53 # --- Envío de Mensaje de Finalización ---
54 MENSAJE_FINAL = "FINAL"
55 try:
56     # *** Usando sendall() ***: Garantiza que se envian todos Los bytes.
57     c.sendall(MENSAJE_FINAL.encode('ascii'))
58     print(f"Envio final: Enviado '{MENSAJE_FINAL}'")
59 except socket.error as e:
60     print(f"Error al enviar mensaje FINAL: {e}")
61
62 except ConnectionRefusedError:
63     print(f"Error: Conexión rechazada. Asegúrate de que el servidor esté corriendo en {IP_SERVIDOR}:{PUERTO_SERVIDOR}.")
64 except socket.error as e:
65     print(f"Error de socket: {e}")
66 except Exception as e:
67     print(f"Ocurrió un error inesperado: {e}")
68
69 finally:
70     # Asegurarse de cerrar el socket
71     if 'c' in locals() and c.fileno() != -1:
72         print("Cerrando socket cliente y terminando.")
73         c.close()

```

**Prueba los nuevos cliente y servidor y verifica que siguen funcionando.**

```
/bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente2_sendall.py
● uo288967@is02:~/IngenieriaServicios$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente2_sendall.py
No se especificaron IP ni Puerto. Usando por defecto: localhost:9999
Intentando conectar a localhost:9999...
Conexión exitosa. Iniciando envíos de datos.
Envío 1/5: Enviado 'ABCDE'.
Envío 2/5: Enviado 'ABCDE'.
Envío 3/5: Enviado 'ABCDE'.
Envío 4/5: Enviado 'ABCDE'.
Envío 5/5: Enviado 'ABCDE'.
Envío final: Enviado 'FINAL'.
Cerrando socket cliente y terminando.
↳ uo288967@is02:~/IngenieriaServicios$
```

Al modificar el servidor para usar una función `recvall()`, tuvimos algunos problemas iniciales porque el servidor se quedaba bloqueado esperando bytes que nunca llegaban. Esto ocurrió cuando el cliente no enviaba exactamente el número de bytes esperado.

### EJERCICIO 3

Completa el servidor anterior y guárdalo como `tcp_servidor3_oche_simplista.py`, para que espere clientes en un puerto especificado por línea de comandos (o el 9999 por defecto), y para cada cliente repita un bucle en el que implemente el servicio recién explicado. De este bucle saldrá cuando `recv()` retorne la cadena vacía, que es un indicador de que el cliente ha cerrado la conexión.

```

1 import socket
2 import sys
3
4 # --- Configuración del Puerto ---
5 PUERTO_DEFECTO = 9999
6 PUERTO = PUERTO_DEFECTO
7 BUFFER_SIZE = 80 # Máximo de bytes a recibir en una Lectura
8
9 # Obtener puerto de Línea de comandos o usar defecto
10 if len(sys.argv) > 1:
11     try:
12         PUERTO = int(sys.argv[1])
13         print(f"Usando puerto especificado por linea de comandos: {PUERTO}")
14     except ValueError:
15         print(f"Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}")
16 else:
17     print(f"No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}")
18
19 # --- Creación del socket de escucha ---
20 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21
22 # Asignarle puerto
23 try:
24     s.bind("", PUERTO)
25 except socket.error as e:
26     print(f"Error al enlazar el socket: {e}")
27     sys.exit(1)
28
29 # Ponerlo en modo pasivo
30 s.listen(5)
31 print("Servidor listo y escuchando...")
32
33 # --- Bucle principal de espera por clientes ---
34 while True:
35     print("\nEsperando un cliente...")
36     try:
37         sd, origen = s.accept()
38     except KeyboardInterrupt:
39         print("\nServidor cerrado por el usuario (Ctrl+C).")
40         s.close()
41         sys.exit(0)
42     except Exception as e:
43         print(f"Error al aceptar la conexión: {e}")
44         continue
45
46     print("Nuevo cliente conectado desde %s:%d" % origen)
47     continuar = True
48
49     # Bucle de atención al cliente conectado
50     while continuar:
51
52         # --- SERVICIO (Inicio de la parte del ejercicio) ---
53
54         # Primero recibir el mensaje del cliente
55         try:
56             # Nota: recv() puede retornar menos de 80 bytes.
57             mensaje_bytes = sd.recv(BUFFER_SIZE)
58         except socket.error as e:
59             print(f"Error de socket durante la recepción: {e}")
60             mensaje_bytes = b""
61
62         if not mensaje_bytes:
63             # Si se recibe una cadena vacía (b''), el cliente cerró el socket
64             print("Conexión cerrada por el cliente.")
65             sd.close()
66             continuar = False
67             continue
68
69         # Convertir Los bytes a caracteres (asumiendo UTF-8)
70         try:
71             mensaje = mensaje_bytes.decode("utf8")
72         except UnicodeDecodeError:
73             print("Advertencia: Error de decodificación UTF-8. Usando datos brutos.")
74             mensaje = str(mensaje_bytes)
75
76         # Segundo, quitarle el "fin de Línea" (Los 2 últimos caracteres: \r\n)
77         # Se asume que el cliente SIEMPRE envía \r\n
78         linea = mensaje[:-2]
79
80         # Tercero, darle La vuelta
81         linea_invertida = linea[::-1]
82
83         # Finalmente, enviar la respuesta con un fin de Línea añadido
84         respuesta = linea_invertida + "\r\n"
85         print(f"Recibido: '{linea}'. Enviendo: '{linea_invertida}'")
86
87         # Observa La transformación en bytes para enviarlos
88         try:
89             sd.sendall(respuesta.encode("utf8"))
90         except socket.error as e:
91             print(f"Error al enviar respuesta: {e}")
92             sd.close()
93             continuar = False
94
95     # --- SERVICIO (Fin de la parte del ejercicio) ---


```

**Para probar este servidor escribe un sencillo programa en python que se conecte al servidor, le envíe algunas líneas de prueba (recuerda finalizarlas con "\r\n"), para cada envío lea la respuesta del servidor con un recv() y la muestre por pantalla y finalmente cierre el socket de datos. Llama a este programa tcp\_cliente3\_oche.py.**

```

1 import socket
2 import sys
3
4 # --- Configuración del Puerto ---
5 PUERTO_DEFECTO = 9999
6 PUERTO = PUERTO_DEFECTO
7 BUFFER_SIZE = 80 # Máximo de bytes a recibir en una Lectura
8
9 # Obtener puerto de Línea de comandos o usar defecto
10 if len(sys.argv) > 1:
11     try:
12         PUERTO = int(sys.argv[1])
13         print(f"Usando puerto especificado por linea de comandos: {PUERTO}")
14     except ValueError:
15         print(f"Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}")
16 else:
17     print(f"No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}")
18
19 # --- Creación del socket de escucha ---
20 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
21
22 # Asignarle puerto
23 try:
24     s.bind("", PUERTO)
25 except socket.error as e:
26     print(f"Error al enlazar el socket: {e}")
27     sys.exit(1)
28
29 # Ponerlo en modo pasivo
30 s.listen(5)
31 print("Servidor listo y escuchando...")
32
33 # --- Bucle principal de espera por clientes ---
34 while True:
35     print("\nEsperando un cliente...")
36     try:
37         sd, origen = s.accept()
38     except KeyboardInterrupt:
39         print("\nServidor cerrado por el usuario (Ctrl+C).")
40         s.close()
41         sys.exit(0)
42     except Exception as e:
43         print(f"Error al aceptar la conexión: {e}")
44         continue
45
46     print("Nuevo cliente conectado desde %s:%d" % origen)
47     continuar = True
48
49     # Bucle de atención al cliente conectado
50     while continuar:
51
52         # --- SERVICIO (Inicio de la parte del ejercicio) ---
53
54         # Primero recibir el mensaje del cliente
55         try:
56             # Nota: recv() puede retornar menos de 80 bytes.
57             mensaje_bytes = sd.recv(BUFFER_SIZE)
58         except socket.error as e:
59             print(f"Error de socket durante la recepción: {e}")
60             mensaje_bytes = b""
61
62         if not mensaje_bytes:
63             # Si se recibe una cadena vacía (b''), el cliente cerró el socket
64             print("Conexión cerrada por el cliente.")
65             sd.close()
66             continuar = False
67             continue
68
69         # Convertir Los bytes a caracteres (asumiendo UTF-8)
70         try:
71             mensaje = mensaje_bytes.decode("utf8")
72         except UnicodeDecodeError:
73             print("Advertencia: Error de decodificación UTF-8. Usando datos brutos.")
74             mensaje = str(mensaje_bytes)
75
76         # Segundo, quitarle el "fin de Línea" (Los 2 últimos caracteres: \r\n)
77         # Se asume que el cliente SIEMPRE envía \r\n
78         linea = mensaje[:-2]
79
80         # Tercero, darle La vuelta
81         linea_invertida = linea[::-1]
82
83         # Finalmente, enviar la respuesta con un fin de Línea añadido
84         respuesta = linea_invertida + "\r\n"
85         print(f"Recibido: '{linea}'. Enviendo: '{linea_invertida}'")
86
87         # Observa La transformación en bytes para enviarlos
88         try:
89             sd.sendall(respuesta.encode("utf8"))
90         except socket.error as e:
91             print(f"Error al enviar respuesta: {e}")
92             sd.close()
93             continuar = False
94
95     # --- SERVICIO (Fin de la parte del ejercicio) ---


```

**Ejecuta el cliente anterior y comprueba que recibe la respuesta correcta (“al revés”) del servidor.**

```
/bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente3_oche.py
● uo288967@is02:~/IngenieriaServicios$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente3_oche.py
Usando por defecto: localhost:9999

Intentando conectar a localhost:9999...
Conexión exitosa.
CLIENTE: Enviado: 'HOLA MUNDO!'
CLIENTE: Recibido (invertido): '!ODNUM ALOH'
CLIENTE: Enviado: 'PYTHON ES GENIAL.'
CLIENTE: Recibido (invertido): '.LAINEG SE NOHTYP'
CLIENTE: Enviado: '1234567890'
CLIENTE: Recibido (invertido): '0987654321'
CLIENTE: Enviado: 'FINALIZAR'
CLIENTE: Recibido (invertido): 'RAZILANIF'

CLIENTE: Cerrando conexión.
❖ uo288967@is02:~/IngenieriaServicios$
```

En la primera versión del servicio “oche” implementamos un servidor que asumía que cada recv() devolvía una línea completa. En nuestras primeras pruebas todo parecía funcionar correctamente, pero al hacer varios envíos seguidos desde el cliente empezamos a recibir respuestas incorrectas. Esto nos ayudó a ver que, en condiciones reales, TCP puede agrupar varios envíos en una sola lectura, rompiendo las suposiciones iniciales del servidor.

## EJERCICIO 4

Implementa una función llamada `recibe_mensaje()` que retorne líneas completas que haya leído del socket, por la técnica de leer los bytes de uno en uno y concatenarlos hasta tener una línea completa.

Usa esta función tanto en el servidor como en el cliente, tomando como base `tcp_servidor3_oche_simplista` y `tcp_cliente3_oche_envia_seguido.py`, para recibir los mensajes que el otro extremo le envía. Los nuevos programas se llamarán `tcp_servidor4_oche_mejorado.py` y `tcp_cliente4_oche_mejorado.py`. Comprueba que funcionan correctamente.

Comprueba que siguen funcionando correctamente incluso si repetimos el experimento de poner un `time.sleep(1)` en el servidor tras el `accept()` para causar que varios mensajes del cliente se acumulen en el buffer del socket.

```

1 import socket
2 import sys
3
4 # --- Configuración del Puerto ---
5 PUERTO_DEFECTO = 9999
6 PUERTO = PUERTO_DEFECTO
7 BUFFER_SIZE = 88 # Máximo de bytes a recibir en una lectura
8
9 # Obtener puerto de linea de comandos o usar defecto
10 if len(sys.argv) > 1:
11     try:
12         PUERTO = int(sys.argv[1])
13         print(f"Usando puerto especificado por linea de comandos: ({PUERTO})")
14     except ValueError:
15         print(f"Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: ({PUERTO_DEFECTO})")
16 else:
17     print(f"No se especificó puerto. Usando puerto por defecto: ({PUERTO_DEFECTO})")
18
19 def recibe_mensaje(socket_cliente):
20     """
21     Recibe un mensaje completo del socket cliente leyendo byte a byte
22     hasta encontrar una linea completa (terminada en \\r\\n).
23
24     Args:
25         socket_cliente: El socket del cliente del cual leer
26
27     Returns:
28         str: La linea completa recibida (sin \\r\\n) o None si hay error/conexion cerrada
29     """
30     mensaje = ""
31
32     try:
33         while True:
34             # Leer un byte a la vez
35             byte = socket_cliente.recv(1)
36
37             # Si no se recibe nada, el cliente cerró la conexión
38             if not byte:
39                 return None
40
41             # Convertir byte a carácter
42             try:
43                 char = byte.decode('utf-8')
44             except UnicodeDecodeError:
45                 # Si hay error de decodificación, ignorar este byte
46                 continue
47
48             # Añadir el carácter al mensaje
49             mensaje += char
50
51             # Verificar si hemos recibido una linea completa (\\r\\n)
52             if mensaje.endswith("\\r\\n"):
53                 # Retornar la linea sin el \\r\\n
54                 return mensaje[:-2]
55
56     except socket.error as e:
57         print(f"Error al recibir mensaje: ({e})")
58     return None
59
60 # --- Creación del socket de escucha ---
61 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
62
63 # Asignarle puerto
64 try:
65     s.bind("", PUERTO)
66 except socket.error as e:
67     print(f"Error al enlazar el socket: ({e})")
68     sys.exit(1)
69
70 # Ponerlo en modo pasivo
71 s.listen(5)
72 print("Servidor listo y escuchando...")
73
74 # --- Bucle principal de espera por clientes ---
75 while True:
76     print("\nEsperando un cliente...")
77     try:
78         sd, origen = s.accept()
79     except KeyboardInterrupt:
80         print("\nServidor cerrado por el usuario (Ctrl+C).")
81         s.close()
82         sys.exit(0)
83     except Exception as e:
84         print(f"Error al aceptar la conexión: ({e})")
85         continue
86
87     print(f"Nuevo cliente conectado desde {sd}:{origen}")
88     continuar = True
89
90     # Bucle de atención al cliente conectado
91     while continuar:
92
93         # --- SERVICIO (Inicio de la parte del ejercicio) ---
94
95         # Primero recibir el mensaje del cliente usando la función recibe_mensaje()
96         linea = recibe_mensaje(sd)
97
98         if linea is None:
99             # Si recibe_mensaje() retorna None, el cliente cerró la conexión o hubo error
100            print("Conexión cerrada por el cliente.")
101            sd.close()
102            continuar = False
103            continue
104
105        # Tercero, darle la vuelta
106        linea_invertida = linea[::-1]
107
108        # Finalmente, enviar la respuesta con un fin de linea añadido
109        respuesta = linea_invertida + "\\r\\n"
110        print(f"Recibido: '{linea}'. Enviendo: '{(linea_invertida)}'")
111
112        # Observa la transformación en bytes para enviarlos
113        try:
114            sd.sendall(respuesta.encode("utf8"))
115        except socket.error as e:
116            print(f"Error al enviar respuesta: ({e})")
117            sd.close()
118            continuar = False
119
120    # --- SERVICIO (Fin de la parte del ejercicio) ---
121
```

```

1 import socket
2 import sys
3
4 # --- Configuración ---
5 IP_DEFECTO = 'localhost'
6 PUERTO_DEFECTO = 9999
7
8 IP_SERVIDOR = IP_DEFECTO
9 PUERTO_SERVIDOR = PUERTO_DEFECTO
10
11 # Obtener IP y Puerto de línea de comandos o usar valores por defecto
12 if len(sys.argv) > 2:
13     IP_SERVIDOR = sys.argv[1]
14     try:
15         PUERTO_SERVIDOR = int(sys.argv[2])
16     except ValueError:
17         print(f"Advertencia: Puerto '{sys.argv[2]}' no es válido. Usando {PUERTO_DEFECTO}.")
18         PUERTO_SERVIDOR = PUERTO_DEFECTO
19 elif len(sys.argv) == 2:
20     IP_SERVIDOR = sys.argv[1]
21     print(f"Usando IP: {IP_SERVIDOR}. Puerto por defecto: {PUERTO_DEFECTO}")
22 else:
23     print(f"Usando por defecto: {IP_DEFECTO}:{PUERTO_DEFECTO}")
24
25 def recibe_mensaje(socket_servidor):
26     """
27     Recibe un mensaje completo del socket servidor leyendo byte a byte
28     hasta encontrar una linea completa (terminada en \\r\\n).
29
30     Args:
31         socket_servidor: El socket del servidor del cual leer
32
33     Returns:
34         str: La linea completa recibida (sin \\r\\n) o None si hay error/conexión cerrada
35     """
36     mensaje = ""
37
38     try:
39         while True:
40             # Leer un byte a la vez
41             byte = socket_servidor.recv(1)
42
43             # Si no se recibe nada, el servidor cerró la conexión
44             if not byte:
45                 return None
46
47             # Convertir byte a carácter
48             try:
49                 char = byte.decode('utf-8')
50             except UnicodeDecodeError:
51                 # Si hay error de decodificación, ignorar este byte
52                 continue
53
54             # Añadir el carácter al mensaje
55             mensaje += char
56
57             # Verificar si hemos recibido una línea completa (\\r\\n)
58             if mensaje.endswith('\\r\\n'):
59                 # Retornar la linea sin el \\r\\n
60                 return mensaje[:-2]
61
62     except socket.error as e:
63         print(f"Error al recibir mensaje: {e}")
64         return None
65
66 # --- Datos de prueba ---
67 MENSAJES = [
68     "HOLA MUNDO\\r\\n",
69     "PYTHON ES GENIAL.\\r\\n",
70     "1234567890\\r\\n",
71     "FINALIZAR\\r\\n"
72 ]
73
74 # --- Lógica del Cliente ---
75 try:
76     # 1. Crear y Conectar Socket
77     c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
78     print(f"\nIntentando conectar a {IP_SERVIDOR}:{PUERTO_SERVIDOR}...")
79     c.connect((IP_SERVIDOR, PUERTO_SERVIDOR))
80     print("Conexión exitosa.")
81
82     # 2. Bucle de Envío y Recepción
83     for mensaje in MENSAJES:
84
85         # Enviar mensaje. Recordar que ya contiene "\\r\\n".
86         datos_a_enviar = mensaje.encode('utf8')
87         c.sendall(datos_a_enviar)
88         print(f"CLIENTE: Enviado: '{mensaje.strip()}'")
89
90         # Recibir la respuesta del servidor usando recibir_mensaje()
91         respuesta = recibe_mensaje(c)
92
93         if respuesta is None:
94             print("CLIENTE: El servidor cerró la conexión inesperadamente.")
95             break
96
97         print(f"CLIENTE: Recibido (invertido): '{respuesta}'")
98
99     # 3. Cerrar Socket
100    print("\nCLIENTE: Cerrando conexión.")
101    c.close()
102
103 except ConnectionRefusedError:
104     print(f"\nError: Conexión rechazada. Asegúrate de que '{IP_SERVIDOR}:{PUERTO_SERVIDOR}' esté funcionando.")
105 except socket.error as e:
106     print(f"\nError de socket: {e}")
107 except Exception as e:
108     print(f"\nOcurrió un error inesperado: {e}")

```

## EJERCICIO 5

Modifica cliente y servidor del ejercicio 4 para que usen la técnica readline() para recibir los mensajes que el otro extremo les envía. Llama a estas nuevas versiones `tcp_servidor5_oche_readline.py` y `tcp_cliente5_oche_readline.py` respectivamente. Comprueba que funcionan incluso con la modificación de `time.sleep(1)` en el servidor.

```

Practica1.4 [SSH: 10.38.15.30] - tcp_servidor5_oche_readline.py

1 import socket
2 import sys
3 import time
4
5 # --- Configuración del Puerto ---
6 PUERTO_DEFECTO = 9999
7 PUERTO = PUERTO_DEFECTO
8 BUFFER_SIZE = 80 # Máximo de bytes a recibir en una lectura
9
10 # Obtener puerto de linea de comandos o usar defecto
11 if len(sys.argv) > 1:
12     try:
13         PUERTO = int(sys.argv[1])
14         print(f'Usando puerto especificado por linea de comandos: {PUERTO}')
15     except ValueError:
16         print(f'Advertencia: El argumento [{sys.argv[1]}] no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}')
17 else:
18     print(f'No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}')
19
20 def recibe_mensaje(socket_cliente):
21     """
22     Recibe un mensaje completo del socket cliente usando readline()
23     """
24     try:
25         f = socket_cliente.makefile(encoding="utf8", newline="\r\n")
26         mensaje = f.readline()
27         if mensaje == '':
28             return None
29         return mensaje.rstrip("\r\n")
30     except Exception as e:
31         print(f'Error al recibir mensaje: {e}')
32         return None
33
34 # --- Creación del socket de escucha ---
35 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
36
37 # Asignarle puerto
38 try:
39     s.bind("", PUERTO)
40 except socket.error as e:
41     print(f'Error al enlazar el socket: {e}')
42     sys.exit(1)
43
44 # Ponerlo en modo pasivo
45 s.listen(5)
46 print("Servidor listo y escuchando...")
47
48 # --- Bucle principal de espera por clientes ---
49 while True:
50     print("\nEsperando un cliente...")
51     try:
52         sd, origen = s.accept()
53     except KeyboardInterrupt:
54         print("\nServidor cerrado por el usuario (Ctrl+C).")
55         s.close()
56         sys.exit(0)
57     except Exception as e:
58         print(f'Error al aceptar la conexión: {e}')
59         continue
60
61     print("Nuevo cliente conectado desde %s:%d" % origen)
62     continuar = True
63
64     # Bucle de atención al cliente conectado
65     while continuar:
66         # --- SERVICIO (Inicio de la parte del ejercicio) ---
67         # Primero recibir el mensaje del cliente usando la función recibe_mensaje()
68         linea = recibe_mensaje(sd)
69         if linea is None:
70             # Si recibe_mensaje() retorna None, el cliente cerró la conexión o hubo error
71             print("Conexión cerrada por el cliente.")
72             sd.close()
73             continuar = False
74             continue
75         # Simular retraso para pruebas
76         time.sleep(1)
77         # Tercero, darle la vuelta
78         linea_invertida = linea[::-1]
79         # Finalmente, enviar la respuesta con un fin de Línea añadido
80         respuesta = linea_invertida + "\r\n"
81         print(f'Recibido: "{linea}'. Enviando: '{linea_invertida}'')
82         # Observa la transformación en bytes para enviarlos
83         try:
84             sd.sendall(respuesta.encode("utf8"))
85         except socket.error as e:
86             print(f'Error al enviar respuesta: {e}')
87             sd.close()
88             continuar = False
89
90     # --- SERVICIO (Fin de la parte del ejercicio) ---


```

```
o uo288967@is02:~/IngenieriaServicios/Practica1.4$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_servidor5_oche_readline.py
No se especificó puerto. Usando puerto por defecto: 9999
Servidor listo y escuchando...

Esperando un cliente...
Nuevo cliente conectado desde 127.0.0.1:45386
Recibido: 'HOLA MUNDO!'. Enviando: '100NUM ALOH'
Recibido: 'PYTHON ES GENIAL.'. Enviando: '.LAINEG SE NOHTYP'
Recibido: '1234567890'. Enviando: '0987654321'
Recibido: 'FINALIZAR'. Enviando: 'RAZILANIF'
Conexión cerrada por el cliente.

Esperando un cliente...
□
```

```

Practica1.4 [SSH: 10.38.15.30] - tcp_cliente5_oche_readline.py

1 import socket
2 import sys
3
4 # --- Configuración ---
5 IP_DEFECTO = 'localhost'
6 PUERTO_DEFECTO = 9999
7
8 IP_SERVIDOR = IP_DEFECTO
9 PUERTO_SERVIDOR = PUERTO_DEFECTO
10
11 # Obtener IP y Puerto de Línea de comandos o usar valores por defecto
12 if len(sys.argv) > 2:
13     IP_SERVIDOR = sys.argv[1]
14     try:
15         PUERTO_SERVIDOR = int(sys.argv[2])
16     except ValueError:
17         print(f"Advertencia: Puerto '{sys.argv[2]}' no es válido. Usando {PUERTO_DEFECTO}.")
18         PUERTO_SERVIDOR = PUERTO_DEFECTO
19     elif len(sys.argv) == 2:
20         IP_SERVIDOR = sys.argv[1]
21         print(f"Usando IP: {IP_SERVIDOR}, Puerto por defecto: {PUERTO_DEFECTO}")
22     else:
23         print(f"Usando por defecto: {IP_DEFECTO}:{PUERTO_DEFECTO}")
24
25 def recibe_mensaje(socket_servidor):
26     """
27     Recibe un mensaje completo del socket servidor usando readline()
28     """
29     try:
30         f = socket_servidor.makefile(encoding="utf8", newline="\r\n")
31         mensaje = f.readline()
32         if mensaje == '':
33             return None
34         return mensaje.rstrip("\r\n")
35     except Exception as e:
36         print(f"Error al recibir mensaje: {e}")
37         return None
38
39 # --- Datos de prueba ---
40 MENSAJES = [
41     "HOLA MUNDO!\r\n",
42     "PYTHON ES GENIAL.\r\n",
43     "1234567890\r\n",
44     "FINALIZAR\r\n"
45 ]
46
47 # --- Lógica del Cliente ---
48 try:
49     # 1. Crear y Conectar Socket
50     c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
51     print(f"\nIntentando conectar a {IP_SERVIDOR}:{PUERTO_SERVIDOR}...")
52     c.connect((IP_SERVIDOR, PUERTO_SERVIDOR))
53     print("Conexión exitosa.")
54
55     # 2. Bucle de Envío y Recepción
56     for mensaje in MENSAJES:
57         # Enviar mensaje. Recordar que ya contiene "\r\n".
58         datos_a_enviar = mensaje.encode('utf8')
59         c.sendall(datos_a_enviar)
60         print(f"CLIENTE: Enviado: '{mensaje.strip()}'")
61
62         # Recibir la respuesta del servidor usando recibe_mensaje()
63         respuesta = recibe_mensaje(c)
64         if respuesta is None:
65             print("CLIENTE: El servidor cerró la conexión inesperadamente.")
66             break
67         print(f"CLIENTE: Recibido (invertido): '{respuesta}'")
68
69     # 3. Cerrar Socket
70     print("\nCLIENTE: Cerrando conexión.")
71     c.close()
72
73 except ConnectionRefusedError:
74     print(f"\nError: Conexión rechazada. Asegúrate de que '{IP_SERVIDOR}:{PUERTO_SERVIDOR}' esté funcionando.")
75 except socket.error as e:
76     print(f"\nError de socket: {e}")
77 except Exception as e:
78     print(f"\nOcurrió un error inesperado: {e}")
79

```

```
/bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente5_oche_readline.py
uo288967@is02:~/IngenieriaServicios/Practica1.4$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente5_oche_readline.py
Usando por defecto: localhost:9999

Intentando conectar a localhost:9999...
Conexión exitosa.
CLIENTE: Enviado: 'HOLA MUNDO!'
CLIENTE: Recibido (invertido): '!ODNUM ALOH'
CLIENTE: Enviado: 'PYTHON ES GENIAL.'
CLIENTE: Recibido (invertido): '.LAINEG SE NOHTYP'
CLIENTE: Enviado: '1234567890'
CLIENTE: Recibido (invertido): '0987654321'
CLIENTE: Enviado: 'FINALIZAR'
CLIENTE: Recibido (invertido): 'RAZILANIF'

CLIENTE: Cerrando conexión.
uo288967@is02:~/IngenieriaServicios/Practica1.4$
```

En este ejercicio sustituimos nuestra función manual por el uso de readline(). Al principio tuvimos problemas porque mezclábamos lecturas directas del socket con lecturas desde el fichero creado con makefile(), lo que provocaba comportamientos raros. Una vez corregido y usando siempre el fichero para leer, comprobamos que esta solución era mucho más limpia y eficiente que leer byte a byte.

## EJERCICIO 6

Modifica el cliente y servidor del servicio “oche” (y renómbralos como `tcp_servidor6_oche_len.py` y `tcp_cliente6_oche_len.py`) para que envíen sus mensajes y respuestas con la técnica descrita.

```

Practical4 [SSH: 10.38.15.30] - tcp_servidor6_oche_len.py

1 import socket
2 import sys
3 import time
4
5 # --- Configuración del Puerto ---
6 PUERTO_DEFECTO = 9999
7 PUERTO = PUERTO_DEFECTO
8 BUFFER_SIZE = 80
9
10 # Obtener puerto de linea de comandos o usar defecto
11 if len(sys.argv) > 1:
12     try:
13         PUERTO = int(sys.argv[1])
14         print(f'Usando puerto especificado por linea de comandos: {PUERTO}')
15     except ValueError:
16         print(f'Advertencia: El argumento '{sys.argv[1]}' no es un número. Usando puerto por defecto: {PUERTO_DEFECTO}')
17 else:
18     print(f'No se especificó puerto. Usando puerto por defecto: {PUERTO_DEFECTO}')
19
20 def recibe_mensaje(sock):
21     """
22     Recibe primero la longitud (ASCII, delimitada por '\n'), luego el mensaje de esa longitud
23     """
24     f = sock.makefile('rb')
25     # Leer La Longitud
26     longitud_str = b''
27     while True:
28         c = f.read(1)
29         if not c:
30             return None
31         if c == b'\n':
32             break
33     longitud_str += c
34     try:
35         longitud = int(longitud_str.decode('utf8'))
36     except Exception:
37         return None
38     # Leer el mensaje completo
39     mensaje_bytes = f.read(longitud)
40     if not mensaje_bytes or len(mensaje_bytes) < longitud:
41         return None
42     return mensaje_bytes.decode('utf8')
43
44 def enviar_mensaje(sock, mensaje):
45     """
46     Envía la longitud en ASCII (con delimitador '\n') seguida del mensaje codificado en utf8
47     """
48     datos = mensaje.encode('utf8')
49     longitud = f'{len(datos)}\n'
50     sock.sendall(longitud.encode('utf8') + datos)
51
52 # --- Creación del socket de escucha ---
53 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
54
55 try:
56     s.bind("", PUERTO)
57 except socket.error as e:
58     print(f'Error al enlazar el socket: {e}')
59     sys.exit(1)
60
61 s.listen(5)
62 print("Servidor listo y escuchando...")
63
64 while True:
65     print("\nEsperando un cliente...")
66     try:
67         sd, origen = s.accept()
68     except KeyboardInterrupt:
69         print("\nServidor cerrado por el usuario (Ctrl+C).")
70         s.close()
71         sys.exit(0)
72     except Exception as e:
73         print(f'Error al aceptar la conexión: {e}')
74         continue
75
76     print("Nuevo cliente conectado desde %s:%d" % origen)
77     continuar = True
78
79     while continuar:
80         linea = recibe_mensaje(sd)
81         if linea is None:
82             print("Conexión cerrada por el cliente.")
83             sd.close()
84             continuar = False
85             continue
86         time.sleep(1)
87         linea_invertida = linea[::-1]
88         print(f'Recibido: '{linea}'. Enviando: '{linea_invertida}'')
89         enviar_mensaje(sd, linea_invertida)
90

```

```
/bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_servidor6_oche_len.py
▷ uo288967@is02:~/IngenieriaServicios/Practica1.4$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_servidor6_oche_len.py
No se especificó puerto. Usando puerto por defecto: 9999
Servidor listo y escuchando...

Esperando un cliente...
Nuevo cliente conectado desde 127.0.0.1:51250
Recibido: 'HOLA MUNDO!'. Enviando: '!ODNUM ALOH'
Recibido: 'PYTHON ES GENIAL!'. Enviando: '.LAINEG SE NOHTYP'
Recibido: '1234567890'. Enviando: '0987654321'
Recibido: 'FINALIZAR'. Enviando: 'RAZILANIE'
Conexión cerrada por el cliente.

Esperando un cliente...
```

```

Práctica1.4 [SSH: 10.38.15.30] - tcp_cliente6_oche_len.py

1 import socket
2 import sys
3
4 # --- Configuración ---
5 IP_DEFECTO = 'localhost'
6 PUERTO_DEFECTO = 9999
7
8 IP_SERVIDOR = IP_DEFECTO
9 PUERTO_SERVIDOR = PUERTO_DEFECTO
10
11 # Obtener IP y Puerto de Línea de comandos o usar valores por defecto
12 if len(sys.argv) > 2:
13     IP_SERVIDOR = sys.argv[1]
14     try:
15         PUERTO_SERVIDOR = int(sys.argv[2])
16     except ValueError:
17         print(f'Advertencia: Puerto {sys.argv[2]} no es válido. Usando {PUERTO_DEFECTO}.')
18         PUERTO_SERVIDOR = PUERTO_DEFECTO
19 elif len(sys.argv) == 2:
20     IP_SERVIDOR = sys.argv[1]
21     print(f'Usando IP: {IP_SERVIDOR}. Puerto por defecto: {PUERTO_DEFECTO}')
22 else:
23     print(f'Usando por defecto: {IP_DEFECTO}:{PUERTO_DEFECTO}')
24
25 def enviar_mensaje(sock, mensaje):
26     """
27     Envía la longitud en ASCII (con delimitador '\n') seguida del mensaje codificado en utf8
28     """
29     datos = mensaje.encode('utf8')
30     longitud = f'{len(datos)}\n'
31     sock.sendall(longitud.encode('utf8') + datos)
32
33 def recibe_mensaje(sock):
34     """
35     Recibe primero la longitud (ASCII, delimitada por '\n'), luego el mensaje de esa longitud
36     """
37     f = sock.makefile('rb')
38     # Leer la Longitud
39     longitud_str = b''
40     while True:
41         c = f.read(1)
42         if not c:
43             return None
44         if c == b'\n':
45             break
46         longitud_str += c
47     try:
48         longitud = int(longitud_str.decode('utf8'))
49     except Exception:
50         return None
51     # Leer el mensaje completo
52     mensaje_bytes = f.read(longitud)
53     if not mensaje_bytes or len(mensaje_bytes) < longitud:
54         return None
55     return mensaje_bytes.decode('utf8')
56
57 # --- Datos de prueba ---
58 MENSAJES = [
59     "HOLA MUNDO!",
60     "PYTHON ES GENIAL.",
61     "1234567890",
62     "FINALIZAR"
63 ]
64
65 # --- Lógica del Cliente ---
66 try:
67     c = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
68     print(f'\nIntentando conectar a {IP_SERVIDOR}:{PUERTO_SERVIDOR}...')
69     c.connect((IP_SERVIDOR, PUERTO_SERVIDOR))
70     print("Conexión exitosa.")
71
72     for mensaje in MENSAJES:
73         enviar_mensaje(c, mensaje)
74         print(f'CLIENTE: Enviado: "{mensaje}"')
75         respuesta = recibe_mensaje(c)
76         if respuesta is None:
77             print("CLIENTE: El servidor cerró la conexión inesperadamente.")
78             break
79         print(f'CLIENTE: Recibido (invertido): "{respuesta}"')
80
81     print("\nCLIENTE: Cerrando conexión.")
82     c.close()
83
84 except ConnectionRefusedError:
85     print(f'\nError: Conexión rechazada. Asegúrate de que "{IP_SERVIDOR}:{PUERTO_SERVIDOR}" esté funcionando.')
86 except socket.error as e:
87     print(f'\nError de socket: {e}')
88 except Exception as e:
89     print(f'\nOcurrió un error inesperado: {e}')
90

```

```
/bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente6_oche_len.py
● uo288967@is02:~/IngenieriaServicios/Practica1.4$ /bin/python3 /home/uo288967/IngenieriaServicios/Practica1.4/tcp_cliente6_oche_len.py
Usando por defecto: localhost:9999

Intentando conectar a localhost:9999...
Conexión exitosa.
CLIENTE: Enviado: 'HOLA MUNDO!'
CLIENTE: Recibido (invertido): 'I0DNUM ALOH'
CLIENTE: Enviado: 'PYTHON ES GENIAL.'
CLIENTE: Recibido (invertido): '.LAINEG SE NOHTYP'
CLIENTE: Enviado: '1234567890'
CLIENTE: Recibido (invertido): '0987654321'
CLIENTE: Enviado: 'FINALIZAR'
CLIENTE: Recibido (invertido): 'RAZILANIF'

CLIENTE: Cerrando conexión.
✧ uo288967@is02:~/IngenieriaServicios/Practica1.4$
```

## **SERVICIOS WEB**

### **PRACTICA 2.1**

En esta práctica empezamos a trabajar con la instalación y configuración de una infraestructura web básica. A lo largo de la sesión montamos un entorno completo para alojar páginas web, viendo cómo se relacionan los distintos componentes que intervienen en una arquitectura web real, como el sistema operativo, el servidor web y los servicios asociados.

Comenzamos preparando el entorno de trabajo y realizando las configuraciones iniciales necesarias, para después instalar y poner en funcionamiento un servidor web. También trabajamos con la estructura de directorios, los permisos y la configuración básica de los servicios, lo que nos permitió entender mejor cómo se organiza y gestiona un servidor web desde el punto de vista del administrador.

Además, realizamos distintas comprobaciones para verificar que los servicios estaban funcionando correctamente y que el contenido web era accesible desde el navegador

### **EJERCICIO 1**

**Escribe, dentro de la carpeta sitios\_nginx un segundo fichero de configuración que llamarás sitio2.conf. Usa default.conf como punto de partida, pero modifícalo para que el sitio 2 tenga como directorio raíz /usr/share/nginx/html2/ y escuche en el puerto 81. Añade un fichero index.html distinto a ese sitio.**

```
gnome-terminal uo288967@is02: ~/IngenieriaServicios/Practica2.1/sitios_nginx
GNU nano 7.2 sitio2.conf
server {
    listen      81;
    server_name localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root    /usr/share/nginx/html2/;
        index  index.html index.htm;
    }

    #error_page  404          /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root    /usr/share/nginx/html2;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass    http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
    #
    #location ~ \.php$ {
    #    root            html;
    #    fastcgi_pass    127.0.0.1:9000;
    #    fastcgi_index   index.php;
    #    fastcgi_param   SCRIPT_FILENAME  /scripts$fastcgi_script_name;
    #    include         fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
    #location ~ /\.ht {
    #    deny  all;
    #}
}
```

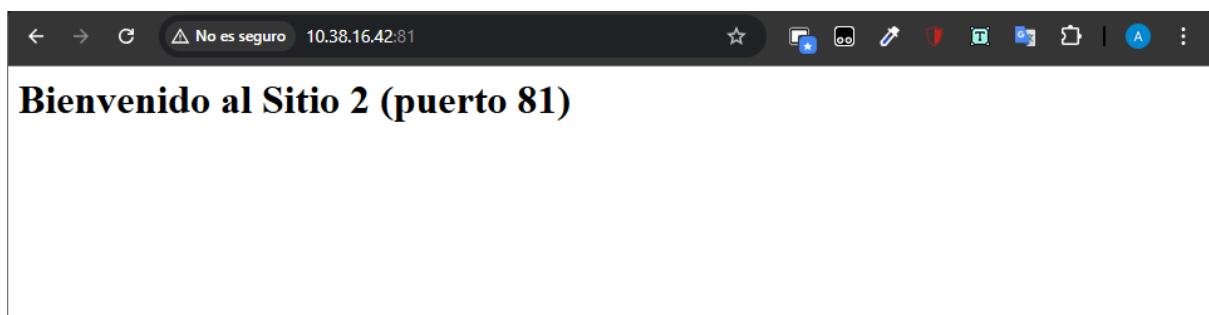
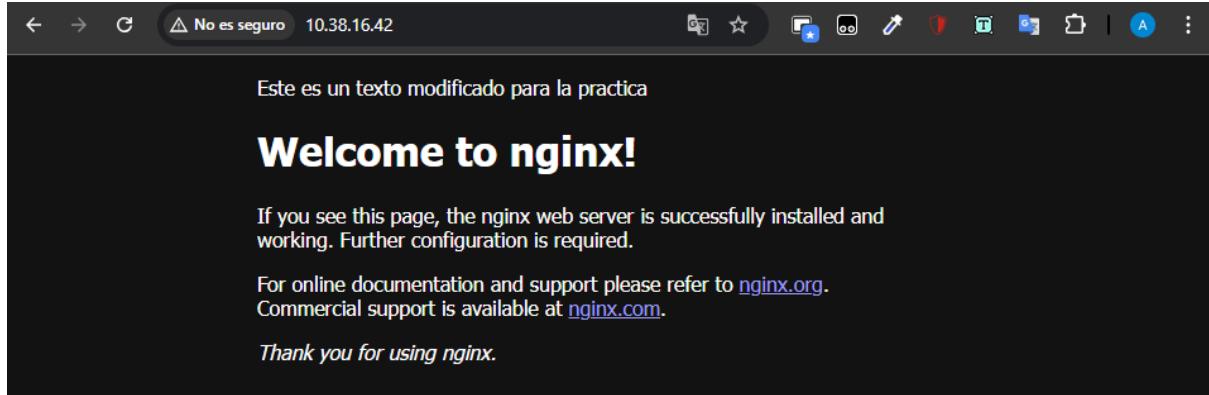
```
gnome-terminal uo288967@is02: ~/IngenieriaServicios/Practica2.1/html2
GNU nano 7.2
<h1>Bienvenido al Sitio 2 (puerto 81)</h1>
```

Escribe un shell script para lanzar nginx en un contenedor. Será como el que usaste la última vez, pero con un volumen adicional para acceder al html del segundo sitio (no hace falta un volumen adicional para el segundo fichero .conf, pues ambos están la misma carpeta). También

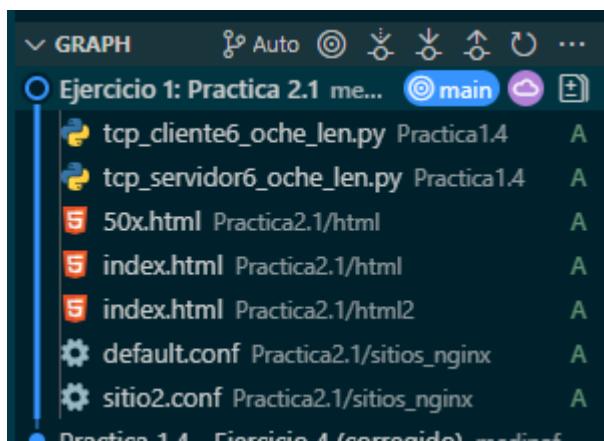
tendrás que mapear el puerto 81 del contenedor en el puerto 81 del anfitrión, para poder probarlo desde fuera.

```
uo288967@is02:~/IngenieriaServicios
uo288967@is02:~/IngenieriaServicios$ docker run --rm -d --network pruebas --name nginx -p 80:80 -p 81:81 -v $(pwd)/html:/usr/share/nginx/html -v $(pwd)/html2:/usr/share/nginx/html2 -v $(pwd)/sitios_nginx:/etc/nginx/conf.d/nginx
```

Conecta el navegador web a la IP de linux y comprueba que se sigue viendo correctamente la página que ya veíamos desde el principio. Despues conéctalo a la misma IP, puerto 81, y comprueba que ahora se ve la que pusiste en html2.



Guarda bajo control de versiones los contenidos de esta carpeta y sus subcarpetas (es decir, los ficheros html, los de configuración y los scripts de lanzamiento).

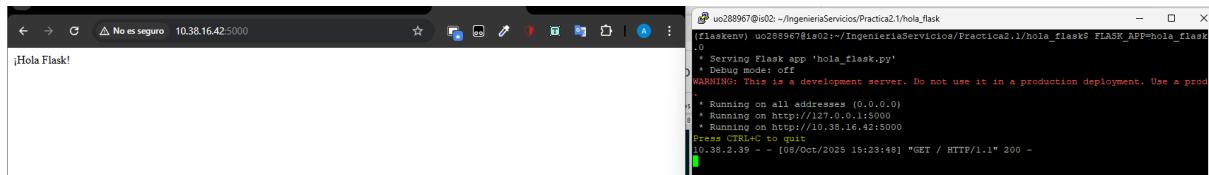


Crea una carpeta (hermana de html) llamada hola\_flask y crea dentro de ella el fichero hola\_flask.py con el contenido anterior.

```
uo288967@is02: ~/IngenieriaServicios/Practica2.1/hola_flask
GNU nano 7.2                                     hola_flask.py

from flask import Flask
app = Flask(__name__)

@app.route('/')
def hola_flask():
    return '¡Hola Flask!'
```



```
uo288967@is02: ~/IngenieriaServicios/Practica2.1/hola_flask
flask run
 * Serving Flask app 'hola_flask'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a prod
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.38.16.42:5000
Press CTRL+C to quit
10.38.2.39 - - [08/Oct/2025 18:23:48] "GET / HTTP/1.1" 200 -
```

## EJERCICIO 2

Copia lo siguiente a un fichero llamado Dockerfile dentro de la carpeta hola\_flask. Lee con atención los comentarios:

```
(flaskenv) uo289853@is-01:~/practica2.1/hola flask$ docker run --rm -d -p 5050:5000 gunicorn
632a831936998bae92f57235f2de744b405a555bd35d4c96835ee1b8a35547b6
(flaskenv) uo289853@is-01:~/practica2.1/hola_flask$ cat Dockerfile
# La instrucción FROM indica en qué imagen nos basamos
FROM python:3.7

# Si vamos a publicar esta imagen porque pueda ser útil a otros
# convendría dar una dirección de contacto con MAINTAINER
# MAINTAINER Jose Luis Diaz <jldiaz@uniovi.es>

# La orden RUN ejecuta los comandos siguientes dentro de un contenedor
# (que se crea a partir de la imagen FROM), y se usan para modificar
# el sistema de archivos del contenedor, crear carpetas, instalar
# paquetes, etc.

# Cada línea RUN es una capa nueva para el sistema de archivos docker
# por lo que a veces se agrupan comandos en un solo RUN para no crear
# muchas capas intermedias
RUN pip install flask && pip install gunicorn

# Observa que lo anterior usa el comando pip que esté presente en la
# imagen python:3.7. La instalación es global, no se usan aquí entornos
# virtuales. Pero es global dentro del contenedor, no afecta al
# anfitrión

# Lo siguiente "instala" nuestra aplicación dentro del contenedor
RUN mkdir /app
COPY hola_flask.py /app

# WORKDIR permite especificar en qué carpeta se hallará el contenedor
# cuando arranque
WORKDIR /app

# EXPOSE permite especificar qué puertos del contenedor serán
# accesibles desde el anfitrión o desde otros contenedores en la
# misma red virtual
EXPOSE 5000

# Finalmente CMD sirve para especificar qué proceso será ejecutado
# cuando arranque el contenedor con esta imagen
CMD ["/usr/local/bin/gunicorn", "-b", "0.0.0.0:5000", "-w", "4", "hola_flask:app"]
```

Después, desde esa carpeta, ejecuta el siguiente comando:

```
$ docker build -t gunicornflask:1.0 .
```

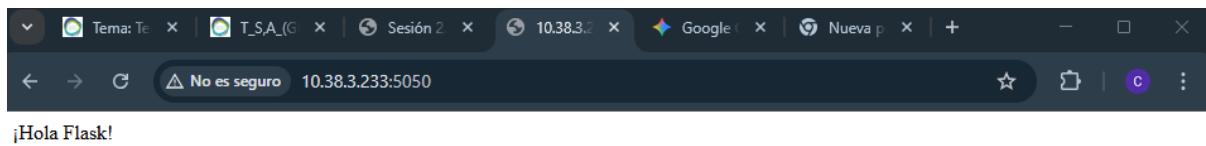
La opción **-t** es para dar un nombre a la imagen (y un número de versión). El punto al final del comando representa a la carpeta en que docker ha de buscar el Dockerfile (carpeta actual)

Si todo va bien, la imagen habrá sido creada. Compruébalo con docker images.

```
(flaskenv) uo289853@is-01:~/practica2.1/hola_flask$ docker images
REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
gunicornflask   1.0      8f87ed0f8a88    7 minutes ago  1.01GB
nginx           latest    07ccdb783875    18 hours ago  160MB
python          3.7      16d93ae3411b    2 years ago   994MB
(flaskenv) uo289853@is-01:~/practica2.1/hola_flask$ █
```

Prueba a lanzar el contenedor con:

```
$ docker run --rm -d -p 5050:5000 gunicornflask:1.0
```



Durante la instalación del servidor web todo parecía funcionar correctamente, pero al intentar acceder desde el navegador no se mostraba la página por defecto. Una vez arrancado manualmente y comprobado su estado, el servidor empezó a funcionar correctamente.

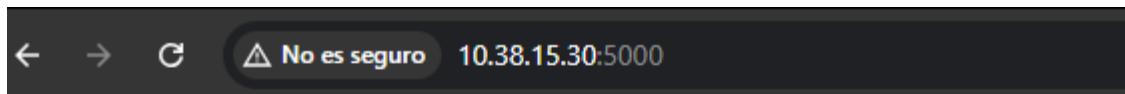
## EJERCICIO 3

1. Escribe los dos ficheros anteriores.
2. Probemos a lanzar la aplicación desde línea de comandos en la forma siguiente:

```
$ FLASK_APP=lanzar.py flask run -h 0.0.0.0
```

3. Conecta un navegador con la IP de tu máquina Ubuntu, puerto 5000 y verifica que recibes el mensaje de Flask.

```
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ nano config.py
y
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ cd instance/
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/instance$ ls
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/instance$ nano
config.py
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/instance$ cd .
.
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ ls
app config.py instance lanzar.py __pycache__
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ FLASK_APP=lan
zar.py flask run -h 0.0.0.0
 * Serving Flask app 'lanzar.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
 Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.38.15.30:5000
Press CTRL+C to quit
```



Hola Flask con MariaDB!

```

flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ FLASK_APP=lanzar.py flask run -h 0.0.0.0
* Serving Flask app 'lanzar.py'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on all addresses (0.0.0.0)
* Running on http://127.0.0.1:5000
* Running on http://10.38.15.30:5000
Press CTRL+C to quit
[0.38.5.14] "GET / HTTP/1.1" 200 -
[0.38.5.14] "GET /favicon.ico HTTP/1.1" 404 -

```

## PRACTICA 2.2

En esta práctica trabajamos con el desarrollo de una aplicación web completa utilizando Flask y el protocolo HTTP. A lo largo de la sesión fuimos construyendo paso a paso una aplicación que permite almacenar y consultar información sobre la localización de distintos amigos, lo que nos ayudó a entender cómo se comunican un cliente web y un servidor.

Durante la práctica configuramos una base de datos para guardar la información, desarrollamos las distintas partes de la aplicación siguiendo el patrón modelo-vista-controlador y creamos vistas tanto en formato HTML como en JSON. También trabajamos con peticiones HTTP de distintos tipos y analizamos su funcionamiento utilizando herramientas del navegador.

Finalmente, desplegamos toda la aplicación utilizando contenedores Docker y configuramos nginx como proxy inverso, lo que nos permitió acceder a la aplicación desde el navegador como si se tratara de un entorno real de producción.

1. **Modifica amigos/app/\_\_init\_\_.py para que soporte migraciones, como se muestra en el código siguiente:**

```

2. # parte de app/__init__.py
3. from flask_migrate import Migrate
4. #
5. def create_app():
6.     # ... todo igual que antes, pero al final...
7.     migrate = Migrate(app, db)
8.     from app import models

```

return app

9. **Ejecuta (desde la carpeta amigos con el entorno virtual activado):**

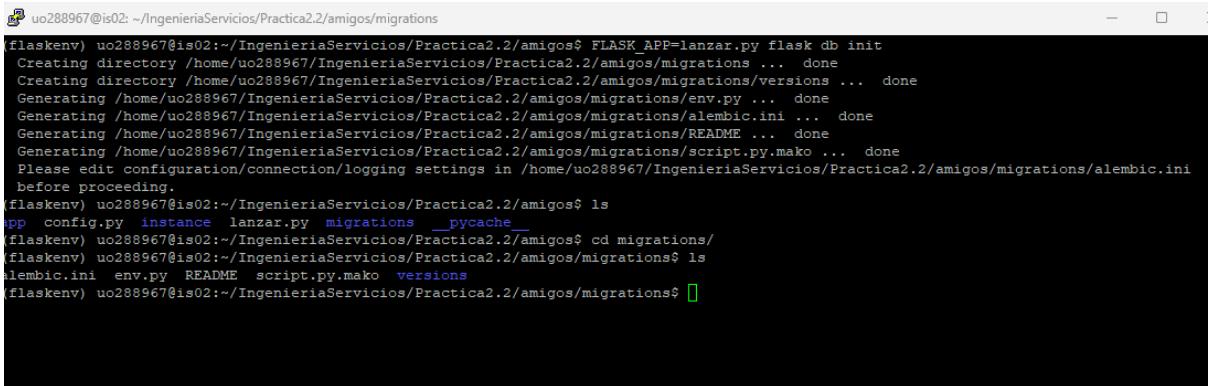
**\$ FLASK\_APP=lanzar.py flask db init**

10. **Comprueba que ha aparecido una carpeta nueva llamada migrations, y dentro de ella versions. No necesitamos entrar en ella, es donde flask-migrations guarda sus scripts para hacer los upgrade o downgrade cuando se lo pidamos. Puedes ver que de momento está vacía.**

11. **Ejecuta el siguiente comando:**

**\$ FLASK\_APP=lanzar.py flask db migrate**

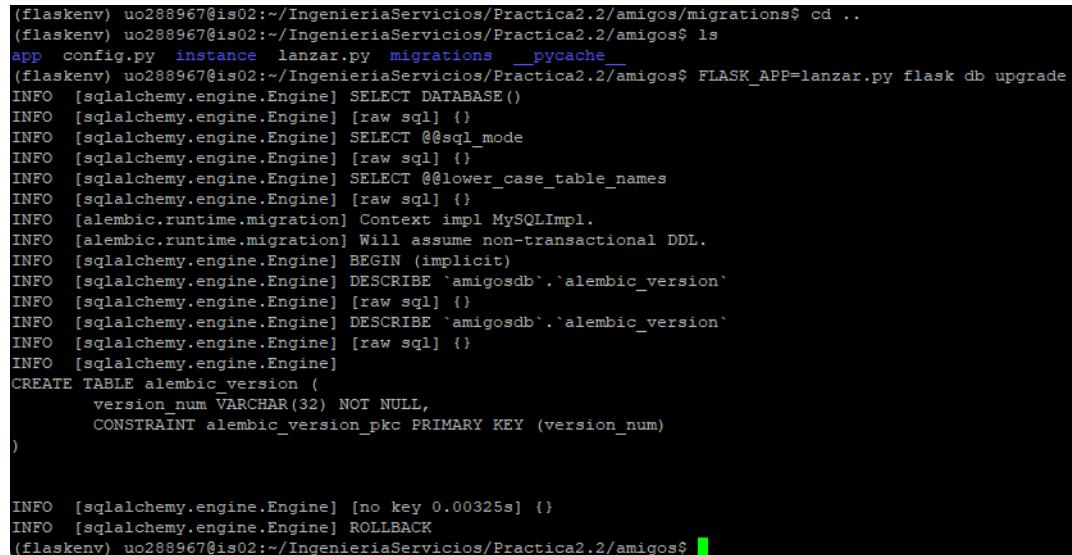
**12. Comprueba que han aparecido ficheros en migrations/versions. Serán los ejecutados ante un *upgrade* o *downgrade*.**



```
uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/migrations
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ FLASK_APP=lanzar.py flask db init
Creating directory /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations ... done
Creating directory /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations/versions ... done
Generating /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations/env.py ... done
Generating /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations/alembic.ini ... done
Generating /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations/README ... done
Generating /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations/script.py.mako ... done
Please edit configuration/connection/logging settings in /home/uo288967/IngenieriaServicios/Practica2.2/amigos/migrations/alembic.ini
before proceeding.
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ ls
app config.py instance lanzar.py migrations __pycache__
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ cd migrations/
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/migrations$ ls
alembic.ini env.py README script.py.mako versions
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/migrations$ 
```

**13. Ejecuta el siguiente comando:**

\$ **FLASK\_APP=lanzar.py flask db upgrade**



```
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos/migrations$ cd ..
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ ls
app config.py instance lanzar.py migrations __pycache__
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ FLASK_APP=lanzar.py flask db upgrade
INFO [sqlalchemy.engine.Engine] SELECT DATABASE()
INFO [sqlalchemy.engine.Engine] [raw sql] {}
INFO [sqlalchemy.engine.Engine] SELECT @@sql_mode
INFO [sqlalchemy.engine.Engine] [raw sql] {}
INFO [sqlalchemy.engine.Engine] SELECT @@lower_case_table_names
INFO [sqlalchemy.engine.Engine] [raw sql] {}
INFO [alembic.runtime.migration] Context impl MySQLImpl.
INFO [alembic.runtime.migration] Will assume non-transactional DDL.
INFO [sqlalchemy.engine.Engine] BEGIN (implicit)
INFO [sqlalchemy.engine.Engine] DESCRIBE `amigosdb`.`alembic_version`
INFO [sqlalchemy.engine.Engine] [raw sql] {}
INFO [sqlalchemy.engine.Engine] DESCRIBE `amigosdb`.`alembic_version`
INFO [sqlalchemy.engine.Engine] [raw sql] {}
INFO [sqlalchemy.engine.Engine]
CREATE TABLE alembic_version (
    version_num VARCHAR(32) NOT NULL,
    CONSTRAINT alembic_version_pkc PRIMARY KEY (version_num)
)

INFO [sqlalchemy.engine.Engine] [no key 0.00325s] {}
INFO [sqlalchemy.engine.Engine] ROLLBACK
(flaskenv) uo288967@is02:~/IngenieriaServicios/Practica2.2/amigos$ 
```

**14. Repite el Experimento 1 y verás que ahora la base de datos ya tiene un par de tablas. Una es la tabla amigos creada por nuestra aplicación. La otra es para uso interno de flask-migrations, donde guarda en qué número de versión se halla la tabla. Haz un describe amigos; para observar el esquema de la base de datos que ha sido creada.**

## EJERCICIO 4

1. Guarda el código anterior en app/templates/tabla\_amigos.html

2. Modifica app/\_\_init\_\_.py:

- Añade from flask import render\_template en la cabecera, para tener acceso a la funcionalidad de jinja2
- Cambia la función prueba por:
  - @app.route('/amigos')

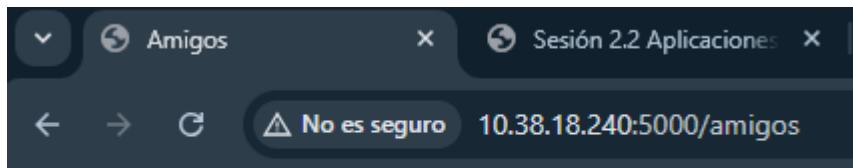
- **def hola\_mundo():**
- **# Obtener lista de amigos de la base de datos**
- **from app.models import Amigo**
- **amigos = Amigo.query.all()**
- **# Retornar el HTML con la tabla de amigos**
- **return render\_template('tabla\_amigos.html',**

**amigos=amigos)**

### 3. Ejecuta el comando para lanzar el servidor de desarrollo:

```
$ FLASK_APP=lanzar.py flask run -h 0.0.0.0
```

### 4. Conecta un navegador web a URL `http://<ip-de-tu-ubuntu>:5000/amigos` y comprueba que obtienes la tabla con los dos amigos que antes introdujimos en la base de datos.

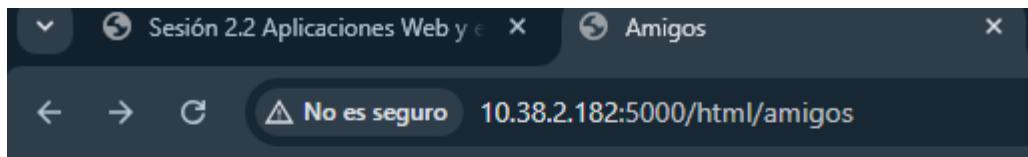


## Amigos

### Nombre Latitud Longitud

Manolito	0	0
Luis	0	0

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ FLASK_APP=lanzar.py flask run -h 0.0.0.0
 * Serving Flask app 'lanzar.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.38.18.240:5000
Press CTRL+C to quit
2025-10-15 15:24:17,864 INFO sqlalchemy.engine.Engine SELECT DATABASE()
2025-10-15 15:24:17,866 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-15 15:24:17,875 INFO sqlalchemy.engine.Engine SELECT @@sql_mode
2025-10-15 15:24:17,875 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-15 15:24:17,878 INFO sqlalchemy.engine.Engine SELECT @@lower_case_table_names
2025-10-15 15:24:17,878 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-15 15:24:17,892 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-15 15:24:17,903 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
2025-10-15 15:24:17,905 INFO sqlalchemy.engine.Engine [generated in 0.00213s] {}
2025-10-15 15:24:17,923 INFO sqlalchemy.engine.Engine ROLLBACK
10.38.9.135 - - [15/Oct/2025 15:24:17] "GET /amigos HTTP/1.1" 200 -
```

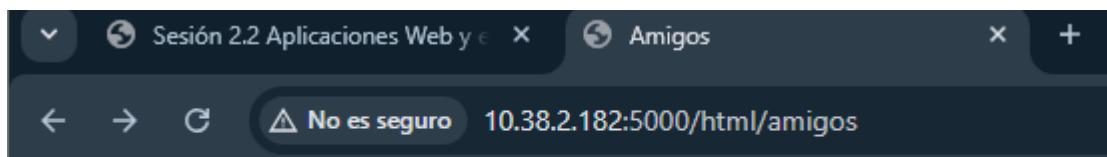


## Amigos

<b>Id</b>	<b>Nombre</b>	<b>Latitud</b>	<b>Longitud</b>	<b>Modificar</b>	<b>Borrar</b>
1	Manolito	0	0	<a href="#">Modificar</a>	<a href="#">Borrar</a>
2	Luis	0	0	<a href="#">Modificar</a>	<a href="#">Borrar</a>

[Añadir Amigo](#)

```
C(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ FLASK_APP=lanzar.py
 * Serving Flask app 'lanzar.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:5000
 * Running on http://10.38.2.182:5000
Press CTRL+C to quit
2025-10-29 15:57:34,420 INFO sqlalchemy.engine.Engine SELECT DATABASE()
2025-10-29 15:57:34,420 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-29 15:57:34,424 INFO sqlalchemy.engine.Engine SELECT @@sql_mode
2025-10-29 15:57:34,424 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-29 15:57:34,427 INFO sqlalchemy.engine.Engine SELECT @@lower_case_table_names
2025-10-29 15:57:34,428 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-29 15:57:34,431 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 15:57:34,436 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.longi AS amigos_longi, amigos.lat AS amigos_lat
FROM amigos
2025-10-29 15:57:34,436 INFO sqlalchemy.engine.Engine [generated in 0.00045s] {}
2025-10-29 15:57:34,447 INFO sqlalchemy.engine.Engine ROLLBACK
10.38.18.28 - - [29/Oct/2025 15:57:34] "GET /html/amigos HTTP/1.1" 200 -
10.38.18.28 - - [29/Oct/2025 15:57:37] "GET /html/aniadir_amigo HTTP/1.1" 404 -
```



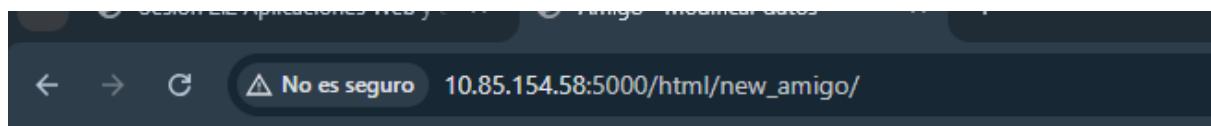
## Amigos

<b>Id</b>	<b>Nombre</b>	<b>Latitud</b>	<b>Longitud</b>	<b>Modificar</b>	<b>Borrar</b>
1	Manolito	0	0	<a href="#">Modificar</a>	<a href="#">Borrar</a>

[Añadir Amigo](#)

```
2025-10-29 16:02:22,225 INFO sqlalchemy.engine.Engine SELECT @@lower_case_table_names
2025-10-29 16:02:22,225 INFO sqlalchemy.engine.Engine [raw sql] {}
2025-10-29 16:02:22,227 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 16:02:22,230 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
2025-10-29 16:02:22,230 INFO sqlalchemy.engine.Engine [generated in 0.00016s] {}
2025-10-29 16:02:22,236 INFO sqlalchemy.engine.Engine ROLLBACK
10.38.18.28 - - [29/Oct/2025 16:02:22] "GET /html/amigos HTTP/1.1" 200 -
2025-10-29 16:02:25,032 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 16:02:25,033 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
WHERE amigos.id = %(pk_1)s
2025-10-29 16:02:25,034 INFO sqlalchemy.engine.Engine [generated in 0.00031s] {'pk_1': 2}
2025-10-29 16:02:25,039 INFO sqlalchemy.engine.Engine DELETE FROM amigos WHERE amigos.id = %(id)s
2025-10-29 16:02:25,039 INFO sqlalchemy.engine.Engine [generated in 0.00014s] {'id': 2}
2025-10-29 16:02:25,040 INFO sqlalchemy.engine.Engine COMMIT
10.38.18.28 - - [29/Oct/2025 16:02:25] "GET /html/delete_amigo/2 HTTP/1.1" 302 -
2025-10-29 16:02:25,053 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 16:02:25,053 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
2025-10-29 16:02:25,053 INFO sqlalchemy.engine.Engine [cached since 2.824s ago] {}
2025-10-29 16:02:25,056 INFO sqlalchemy.engine.Engine ROLLBACK
10.38.18.28 - - [29/Oct/2025 16:02:25] "GET /html/amigos HTTP/1.1" 200 -
10.38.18.28 - - [29/Oct/2025 16:02:29] "GET /html/aniadir_amigo HTTP/1.1" 404 -

```



## Amigo - Modificar datos

Nombre

Logitud

Latitud

```
uo289853@is-01: ~/IngenieriaServicios/practica2.2/amigos
2025-10-29 16:25:52,187 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
WHERE amigos.id = %(pk_1)s
2025-10-29 16:25:52,187 INFO sqlalchemy.engine.Engine [cached since 30.81s ago]
{'pk_1': 1}
2025-10-29 16:25:52,195 INFO sqlalchemy.engine.Engine UPDATE amigos SET longi=%(longi)s, lati=%(lati)s WHERE amigos.id = %(amigos_id)s
2025-10-29 16:25:52,195 INFO sqlalchemy.engine.Engine [generated in 0.00076s] {'longi': '3', 'lati': '3', 'amigos_id': 1}
2025-10-29 16:25:52,198 INFO sqlalchemy.engine.Engine COMMIT
10.85.154.139 - - [29/Oct/2025 16:25:52] "POST /html/save_amigo HTTP/1.1" 302 -
2025-10-29 16:25:52,216 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 16:25:52,216 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
2025-10-29 16:25:52,216 INFO sqlalchemy.engine.Engine [cached since 36.65s ago]
{}
2025-10-29 16:25:52,222 INFO sqlalchemy.engine.Engine ROLLBACK
10.85.154.139 - - [29/Oct/2025 16:25:52] "GET /html/amigos HTTP/1.1" 200 -
10.85.154.139 - - [29/Oct/2025 16:27:35] "GET /html/new_amigo/ HTTP/1.1" 200 -
```



## Amigo - Modificar datos

Nombre

Longitud

Latitud

```
uo289853@is-01: ~/IngenieriaServicios/practica2.2/amigos
2025-10-29 16:29:21,009 INFO sqlalchemy.engine.Engine [generated in 0.00051s] {}
2025-10-29 16:29:21,015 INFO sqlalchemy.engine.Engine ROLLBACK
10.85.154.139 - - [29/Oct/2025 16:29:21] "GET /html/amigos HTTP/1.1" 200 -
2025-10-29 16:29:46,617 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 16:29:46,618 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
WHERE amigos.id = %(pk_1)s
2025-10-29 16:29:46,618 INFO sqlalchemy.engine.Engine [cached since 47.15s ago]
{'pk_1': 1}
2025-10-29 16:29:46,626 INFO sqlalchemy.engine.Engine ROLLBACK
10.85.154.139 - - [29/Oct/2025 16:29:46] "GET /html/edit_amigo/1 HTTP/1.1" 200 -
2025-10-29 16:29:52,133 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2025-10-29 16:29:52,135 INFO sqlalchemy.engine.Engine SELECT amigos.id AS amigos_id, amigos.name AS amigos_name, amigos.longi AS amigos_longi, amigos.lati AS amigos_lati
FROM amigos
WHERE amigos.id = %(pk_1)s
2025-10-29 16:29:52,139 INFO sqlalchemy.engine.Engine [cached since 52.67s ago]
{'pk_1': 3}
2025-10-29 16:29:52,143 INFO sqlalchemy.engine.Engine ROLLBACK
10.85.154.139 - - [29/Oct/2025 16:29:52] "GET /html/edit_amigo/3 HTTP/1.1" 200 -
^[[23~
```

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http GET localhost:5000/api/amigo/1
HTTP/1.1 200 OK
Connection: close
Content-Length: 50
Content-Type: application/json
Date: Wed, 29 Oct 2025 16:36:06 GMT
Server: Werkzeug/3.1.3 Python/3.12.3

{
    "id": 1,
    "lati": "3",
    "longi": "3",
    "name": "Manolito"
}

(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http GET localhost:5000/api/amigo/byName/Manolito
HTTP/1.1 200 OK
Connection: close
Content-Length: 50
Content-Type: application/json
Date: Wed, 29 Oct 2025 16:39:37 GMT
Server: Werkzeug/3.1.3 Python/3.12.3

{
    "id": 1,
    "lati": "3",
    "longi": "3",
    "name": "Manolito"
}

(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http GET localhost:5000/api/amigo/byName/palom
HTTP/1.1 404 NOT FOUND
Connection: close
Content-Length: 181
Content-Type: text/html; charset=utf-8
Date: Wed, 29 Oct 2025 16:39:56 GMT
Server: Werkzeug/3.1.3 Python/3.12.3

<!doctype html>
<html lang=en>
<title>404 Not Found</title>
<h1>Not Found</h1>
<p>No se encuentra ningún amigo con ese nombre</p>

(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ _
```

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http GET localhost:5000/api/amigos
HTTP/1.1 200 OK
Connection: close
Content-Length: 100
Content-Type: application/json
Date: Wed, 29 Oct 2025 16:47:26 GMT
Server: Werkzeug/3.1.3 Python/3.12.3

[
    {
        "id": 1,
        "lati": "3",
        "longi": "3",
        "name": "Manolito"
    },
    {
        "id": 3,
        "lati": "532",
        "longi": "23",
        "name": "fsd"
    }
]

(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$
```

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http --json PUT localhost:5000/api/amigo/1 lati=12
HTTP/1.1 200 OK
Connection: close
Content-Length: 51
Content-Type: application/json
Date: Wed, 29 Oct 2025 16:48:41 GMT
Server: Werkzeug/3.1.3 Python/3.12.3

{
    "id": 1,
    "lati": "12",
    "longi": "3",
    "name": "Manolito"
}

(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ _
```

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http DELETE localhost:5000/api/amigo/3
HTTP/1.1 204 NO CONTENT
Connection: close
Content-Type: text/html; charset=utf-8
Date: Wed, 29 Oct 2025 16:50:51 GMT
Server: Werkzeug/3.1.3 Python/3.12.3
```

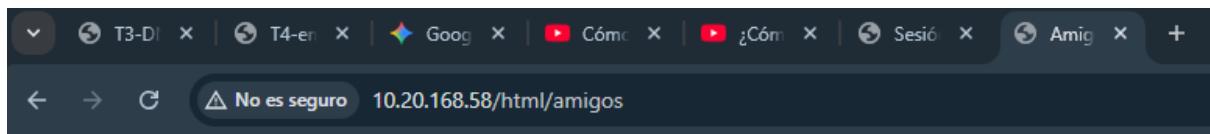
```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ _
```

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ http --json POST localhost:5000/api/amigos name=Javi
HTTP/1.1 200 OK
Connection: close
Content-Length: 48
Content-Type: application/json
Date: Wed, 29 Oct 2025 16:52:55 GMT
Server: Werkzeug/3.1.3 Python/3.12.3

{
    "id": 4,
    "lati": "0",
    "longi": "0",
    "name": "Javier"
}

(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ _
```

```
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ docker run --name basedatos \
-e MYSQL_ROOT_PASSWORD=clavero root \
-e MYSQL_USER=amigosuser \
-e MYSQL_DATABASE=amigosdb \
-e MYSQL_PASSWORD=amigospass \
-v $(pwd)/basedatos:/var/lib/mysql \
--rm --network pruebas -d mariadb
a01251f4edelf7c134d3fe4e731fd78affe92ae477c2594acb297ba9af8b0db8
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ docker run --name amigos \
-e DEPLOYMENT_MODE=production \
-e DATABASE_URI=mysql+pymysql://amigosuser:amigospass@basedatos/amigosdb \
--rm -d --network pruebas amigos:1.0
fcd84e2e84d626309efbe390bd544a788391e7132186d159105df1c2f0650419
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ docker run --rm -d --network pruebas \
-v $(pwd)/sitios_nginx:/etc/nginx/conf.d \
-p 80:80 nginx
d238b43e099dae9b3faed5e5ee3ae4d0fa4c2bc8fld1128749a8d7b43fab644a
(flaskenv) uo289853@is-01:~/IngenieriaServicios/practica2.2/amigos$ _
```



<b>Id</b>	<b>Nombre</b>	<b>Latitud</b>	<b>Longitud</b>	<b>Modificar</b>	<b>Borrar</b>
4	Javier	0	0	<a href="#">Modificar</a>	<a href="#">Borrar</a>

[Añadir Amigo](#)

En este práctica tuvimos algunos problemas al lanzar el contenedor de MariaDB porque ya lo habíamos lanzado el contenedor antes usando el mismo nombre. Al principio no nos dimos cuenta y el contenedor no arrancaba correctamente. Tras listar los contenedores y eliminar los anteriores, conseguimos desplegar la base de datos sin problemas

además, otros de los problemas que encontramos fue que el navegador no permitía probar fácilmente las peticiones PUT y DELETE aunque finalmente lo resolvimos.

## **SERVICIOS DE NOMBRADO, ACCESO REMOTO Y TRANSFERENCIA DE ARCHIVOS**

### **PRACTICA 3**

En esta práctica trabajamos con distintos servicios de acceso remoto y transferencia de archivos, que nos permiten conectarnos a otras máquinas y gestionar información sin necesidad de estar físicamente delante de ellas. A lo largo de la sesión utilizamos varias herramientas habituales en sistemas Linux para acceder remotamente a un servidor y transferir archivos entre equipos.

Comenzamos estableciendo conexiones remotas mediante terminal, lo que nos permitió ejecutar comandos en otra máquina como si estuviéramos trabajando localmente. A continuación, trabajamos con diferentes métodos de transferencia de archivos, tanto para copiar ficheros individuales como para mover directorios completos, entendiendo las diferencias entre cada herramienta.

Finalmente, realizamos varias pruebas para comprobar el funcionamiento correcto de las conexiones y las transferencias, prestando especial atención a aspectos como los permisos, las rutas de destino y la autenticación, que resultan fundamentales en un entorno real.

#### **EJERCICIO 1**

Consulta la [documentación](#) del módulo telnetlib, y en particular copia el ejemplo que aparece al final, con el nombre telnet1\_ejemplo.py y prueba a ejecutarlo desde Unix y desde Windows, si tienes python instalado en Windows. Cuando lo ejecutes desde Windows deberás cambiar “localhost” por la IP del servidor Unix.

Modifícalo para que en lugar de la carpeta del usuario muestre el contenido de la carpeta /home

```
● ○ ● Practica3.1 [SSH: 10.38.10.100] - telnet1_ejemplo.py

1 import getpass
2 import telnetlib
3
4 HOST = "10.38.10.100"
5 user = input("Enter your remote account: ")
6 password = getpass.getpass()
7
8 tn = telnetlib.Telnet(HOST)
9
10 tn.read_until(b"login: ")
11 tn.write(user.encode('utf-8') + b"\n")
12 if password:
13     tn.read_until(b>Password: ")
14     tn.write(password.encode('utf-8') + b"\n")
15
16 tn.write(b"cd /home\n")
17 tn.write(b"exit\n")
18
19 print(tn.read_all().decode('utf-8'))
```

La función `read_all()` utilizada al final, llamará repetidamente a `recv()` sobre el socket de conexión, hasta que detecte el cierre de dicho socket (fin de transmisión). Nos devolverá todo el texto recibido (eliminando los comandos binarios del protocolo), en forma de cadena de bytes. Verás que recibimos el mensaje de bienvenida del sistema, el eco del comando que enviamos (`ls`) y la respuesta de ese comando, así como el eco del `exit` final.

Aunque el ejemplo tomado de la documentación de Python decodifica lo recibido con el encoding "ascii", esto podría fallar si en la respuesta viniera algún carácter no ascii (por ejemplo, si algún fichero de los listados en la carpeta tiene acentos o eñes en su nombre). Necesitamos saber qué codificación usa el shell remoto para este caso, y así poder decodificarlo apropiadamente. En nuestro caso el encoding sería UTF-8, que es el usado por defecto por Ubuntu Server 18.04. Por desgracia el protocolo telnet no puede informarnos de tal cosa ya que para él esto es transparente y asume que el shell remoto y la terminal local usan ambos la misma codificación.

En lugar de `read_all()` podemos usar también `read_until()`, la cual también recibe bytes del socket y los va procesando dejando el texto limpio, pero se detiene cuando recibe cierta secuencia de bytes que le pasamos como parámetro. Este es el truco usado para detenerse cuando el sistema remoto está pidiendo el `login:` y la `Password:` y así poder enviarle estos datos en el momento oportuno.

Piensa cómo usarías `read_until()` para leer (y descartar) el mensaje de bienvenida del sistema, y así quedarse sólo con el comando `ls` y su salida. Pista: el *prompt* del shell remoto termina siempre en dolar (\$). Impleméntalo en el fichero `telnet1_ejemplo_mejorado.py`.

```

Practica3.1 [SSH: 10.38.10.100] - telnet1_ejemplo_mejorado.py

1 import getpass
2 import telnetlib
3
4 HOST = "10.38.10.100"
5 user = input("Enter your remote account: ")
6 password = getpass.getpass()
7
8 tn = telnetlib.Telnet(HOST)
9
10 tn.read_until(b"login: ")
11 tn.write(user.encode('utf-8') + b"\n")
12 if password:
13     tn.read_until(b>Password: ")
14     tn.write(password.encode('utf-8') + b"\n")
15
16 # Esperar hasta que aparezca el prompt del shell ($), descartando el mensaje de bienvenida
17 tn.read_until(b"$")
18
19 # Ejecutar el comando y leer su salida
20 tn.write(b"ls -la /home\n")
21 output = tn.read_until(b"$").decode('utf-8', errors='replace')
22
23 tn.write(b"exit\n")
24
25 print(output)

```

```

Enter your remote account: uo288967
Password:
total 12
drwxr-xr-x  3 root      root      4096 sep 17 14:59 .
drwxr-xr-x 23 root      root      4096 sep 17 14:55 ..
drwxr-x--- 11 uo288967 uo288967 4096 oct 22 15:11 uo288967
uo288967@is02:~$ 
uo288967@is02:~/IngenieriaServicios/Practica3.1$ 

```

## EJERCICIO 2

Copia en la máquina Linux, a tu carpeta *home*, el programa servidor que escribiste en la sesión sobre protocolos UDP, llamado *udp\_servidor3\_con\_ok.py*. Este era un servidor que esperaba por datagramas en el puerto 9999 y devolvía otros datagramas de confirmación conteniendo un “OK”.

En este ejercicio vamos a usar telnet para arrancar remotamente ese servidor, en el caso de que no estuviera ya en ejecución. Para ello debes escribir un programa llamado *telnet2\_lanza\_servidor.py* que:

1. Se conecte usando telnetlib con el servidor Linux, usando la misma técnica que en el ejercicio anterior, pero en esta ocasión sin preguntar al usuario nombre y clave, sino incrustando éstas directamente en el código fuente (es una mala práctica de cara a la seguridad, pero lo haremos así en igualmente para este ejercicio)
2. Una vez conectado, lee la respuesta del telnet usando *read\_until(b"\$")* esto leerá lo que el shell remoto nos envía, hasta que se reciba un signo dólar. Ya que el signo dólar es

parte del *prompt*, al recibirla sabremos que el *shell* remoto está listo para recibir comandos.

3. Envía el comando `ps -ef\n` que lista todos los procesos del usuario (¡no olvides enviar el `\n` al final!)
4. Usa de nuevo `read_until()` para esperar al próximo *prompt* y captura la respuesta en una variable. Esa respuesta contendrá la salida del comando `ps -ef` que le habíamos pedido.
5. Comprueba si la respuesta recibida contiene la cadena `udp_servidor3_con_ok.py`, lo que denotaría que el servicio está en ejecución. En ese caso imprime “El servidor ya está en ejecución” y terminará.
6. Si esa cadena no está presente (el servidor no está en ejecución), lo lanzaremos, enviando por telnet la orden `nohup python3 udp_servidor3_con_ok.py &\n` (¡no olvides el retorno de carro!).
  - El `&` final es para que el *shell* devuelva el *prompt* y así poder lanzarle un `exit` después.
  - El `nohup` inicial es para que el servidor siga corriendo cuando cerramos la conexión telnet (de lo contrario, al estar lanzado desde esta “terminal”, moriría al cerrar telnet). Se creará un fichero llamado `nohup.out` que contendrá la salida del comando, en lugar de ir a la terminal.
7. Espera un segundo para dar tiempo a que el servidor arranque (`time.sleep(1)`)
8. Envía el comando `exit\n` para terminar el *shell* remoto y por tanto la sesión telnet.
9. Recoge la respuesta del servidor con `read_all()` y muéstralala en pantalla para depuración.

Prueba a lanzar este programa, si es posible desde otra máquina. Entra después en la máquina Ubuntu y verifica que el servidor ha sido arrancado y está en ejecución. Si algo ha ido mal, puedes mirar el contenido del fichero `~/nohup.out`, que contiene la salida del comando que has lanzado con `nohup` y puede darte pistas sobre el problema.

Lanza de nuevo el programa que has escrito y comprueba si esta vez te dice que el servidor ya estaba en ejecución. Desde una terminal en Ubuntu, mata al servidor (`kill` del proceso cuyo PID averiguarás con `ps -ef`), y lanza de nuevo el script desde otra máquina para ver que arranca de nuevo el servidor.

```

IngenieriaServicios [SSH: 10.38.10.100] - telnet2_lanza_servidor.py

1 import telnetlib
2 import time
3
4 HOST = "10.38.10.100"
5 user = "uo288967"
6 password = "Amefer03."
7
8 tn = telnetlib.Telnet(HOST)
9
10 tn.read_until(b"login: ")
11 tn.write(user.encode('utf-8') + b"\n")
12 tn.read_until(b>Password: ")
13 tn.write(password.encode('utf-8') + b"\n")
14
15 # Espera hasta que aparezca el prompt del shell remoto ($)
16 tn.read_until(b"$")
17
18 # Envía el comando ps -ef
19 tn.write(b"ps -ef\n")
20
21 # Usa read_until() para esperar al próximo prompt y capturar la salida completa
22 respuesta = tn.read_until(b"$").decode('utf-8', errors='replace')
23
24 # Comprueba si la salida contiene el nombre del servidor UDP
25 if "udp_servidor3_con_ok.py" in respuesta:
26     print("El servidor ya está en ejecución.")
27 else:
28     print("El servidor no está en ejecución. Lanzando servidor...")
29     # Envía el comando para lanzarlo en background con nohup
30     tn.write(b"nohup python3 udp_servidor3_con_ok.py &\n")
31     # Espera a que aparezca el prompt otra vez
32     tn.read_until(b"$")
33     print("Servidor lanzado correctamente.")
34
35     # 7 Espera un segundo para dar tiempo a que el servidor arranque
36     time.sleep(1)
37
38 # 8 Envía el comando exit para cerrar el shell remoto
39 tn.write(b"exit\n")
40
41 # 9 Recoge la respuesta final del servidor con read_all() y muéstralala
42 salida_final = tn.read_all().decode('utf-8', errors='replace')
43 print("== Respuesta completa del servidor ==")
44 print(salida_final)
45

```

```
16 tn.read_until(b"$")  
uo288967@is02: ~/IngenieriaServicios/Practica3.1  
uo288967@is02:/home$ ls  
uo288967  
uo288967@is02:/home$ cd uo288967/  
uo288967@is02:~/uo288967$ ls  
basedatos flaskenv IngenieriaServicios nohup.out pruebas.py snap  
uo288967@is02:~/uo288967$ cd IngenieriaServicios/  
uo288967@is02:~/IngenieriaServicios$ cd Practica3.1/  
uo288967@is02:~/IngenieriaServicios/Practica3.1$ ls  
telnetl_ejemplo_mejorado.py telnet2_lanza_servidor.py  
telnetl_ejemplo.py udp_servidor3_con_ok.py  
uo288967@is02:~/IngenieriaServicios/Practica3.1$ python3 telnet2_lanza_servidor.py  
/home/uo288967/IngenieriaServicios/Practica3.1/telnet2_lanza_servidor.py:1: DeprecationWarning: 'telnetlib' is deprecated and slated for removal in Python 3.13  
    import telnetlib  
El servidor no está en ejecución. Lanzando servidor...  
Servidor lanzado correctamente.  
==== Respuesta completa del servidor ====  
nohup: ignoring input and appending output to 'nohup.out'  
exit  
logout  
[1]+ Exit 2 nohup python3 udp_servidor3_con_ok.py  
uo288967@is02:~/IngenieriaServicios/Practica3.1$
```

Esto significa que el script ha detectado que el servidor no corría, lo ha lanzado y ha cerrado la sesión Telnet correctamente.

Al desarrollar el script para lanzar remotamente el servidor UDP, uno de los principales problemas fue detectar correctamente si el proceso ya estaba en ejecución. En una de las primeras pruebas, el script siempre lanzaba el servidor aunque ya estuviera activo, porque estábamos leyendo mal la salida del comando ps -ef. Tras ajustar el uso de `read_until()` y revisar bien el contenido recibido antes del prompt, conseguimos detectar correctamente la presencia del proceso. También tuvimos algún fallo al lanzar el servidor con nohup, pero finalmente se solucionó.

## EJERCICIO 3

Usando la consola de la máquina virtual Ubuntu, entra en sesión. Esta es la forma más segura posible de entrar en el sistema, ya que lo estaremos haciendo directamente en su consola, sin conexiones de red.

Dentro de ella, averigua el *fingerprint sha256* de la clave pública para el algoritmo ed25519. Anota (al menos sus primeras cifras) en un papel.

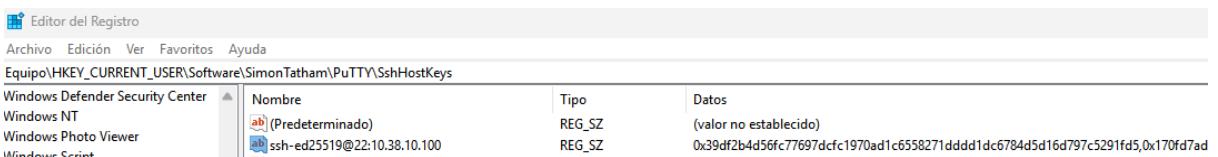
Usando putty, conecta a la máquina. Si no es la primera vez que lo haces no te preguntará nada y no veremos la pantalla de advertencia con el *fingerprint*.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS ssh - IngenieriaServicios + × [ ] ×
uo288967@is02:~/IngenieriaServicios/Practica3.1$ ssh-keygen -lf /etc/ssh/ssh_host_ed25519_key.pub
256 SHA256:ry9jNJoFsMkoj15j9J5D92eDmI9fLudkeXJ11dWFtI4 root@is02 (ED25519)
uo288967@is02:~/IngenieriaServicios/Practica3.1$
```

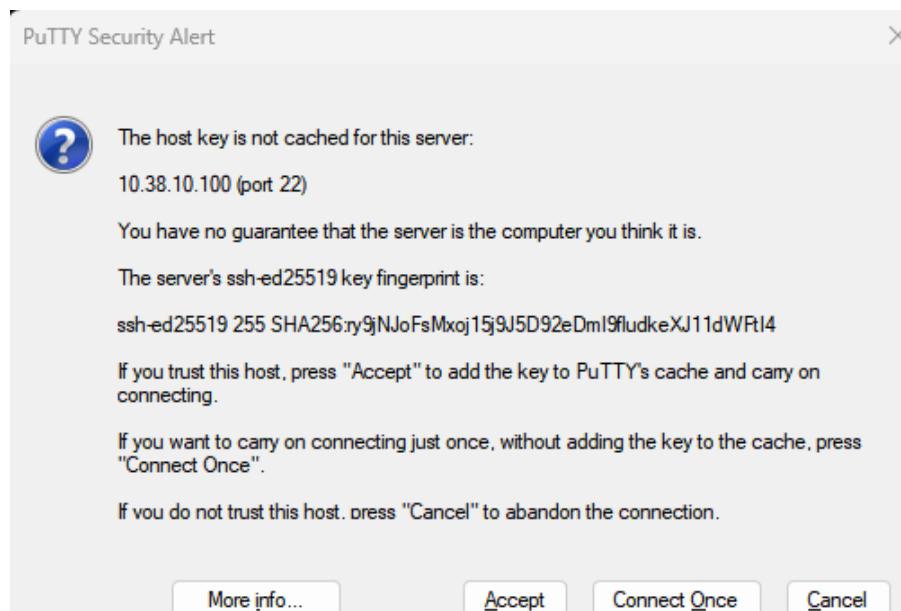
Si borramos el almacén donde putty guarda las claves por él conocidas (lo que sería el archivo known hosts de unix), e intentamos reconectar, entonces nos aparecerá de nuevo esa

pantalla como la primera vez. Por desgracia putty no guarda esta información en un archivo, sino en claves de registro de Windows, que son algo más difíciles de borrar:

- Desde el menú **Inicio** lanza el comando regedit (puedes hacerlo por su nombre)
- Dentro de regedit, busca la clave HKEY\_CURRENT\_USER → Software → SimonTatham → PuTTY → SshHostKeys
- Verás todas las claves públicas que tiene almacenadas de todos los sitios a los que te hayas conectado usando PuTTY. Busca la que corresponde a tu servidor Ubuntu (en la columna “nombre” podrás encontrarlo por la IP o nombre de dominio de la máquina).
- **Borra esa entrada usando la tecla Supr.**



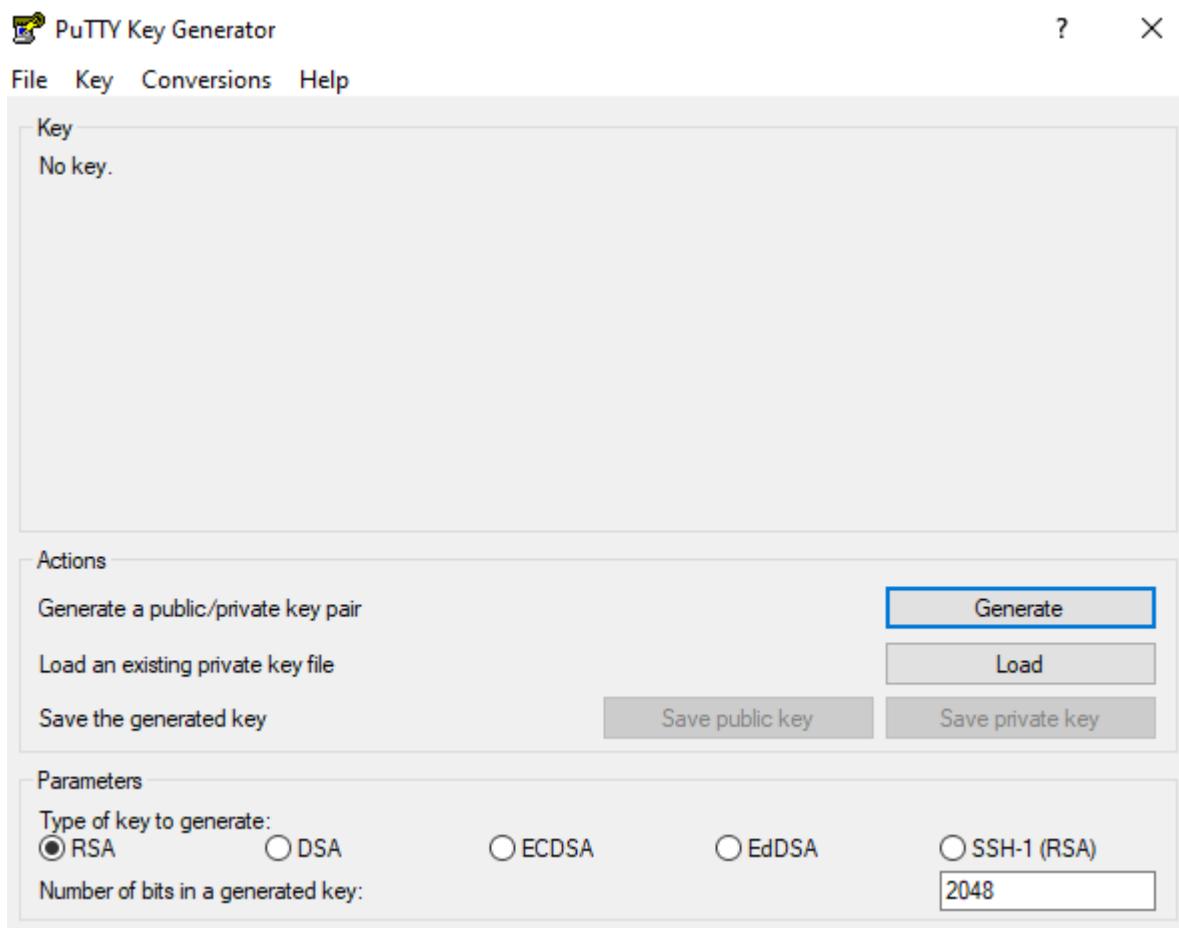
Conecta de nuevo al servidor y observa cómo ahora te sale de nuevo la advertencia y el **fingerprint**. Comprueba que coincide con el que obtuviste en la consola. Si no concordaran, comprueba que realmente putty y el servidor están intentando usar ED25519 (el mensaje antes del **fingerprint** te indica qué algoritmo usan). Deberían ser iguales, de lo contrario ¡la universidad no es un lugar seguro!



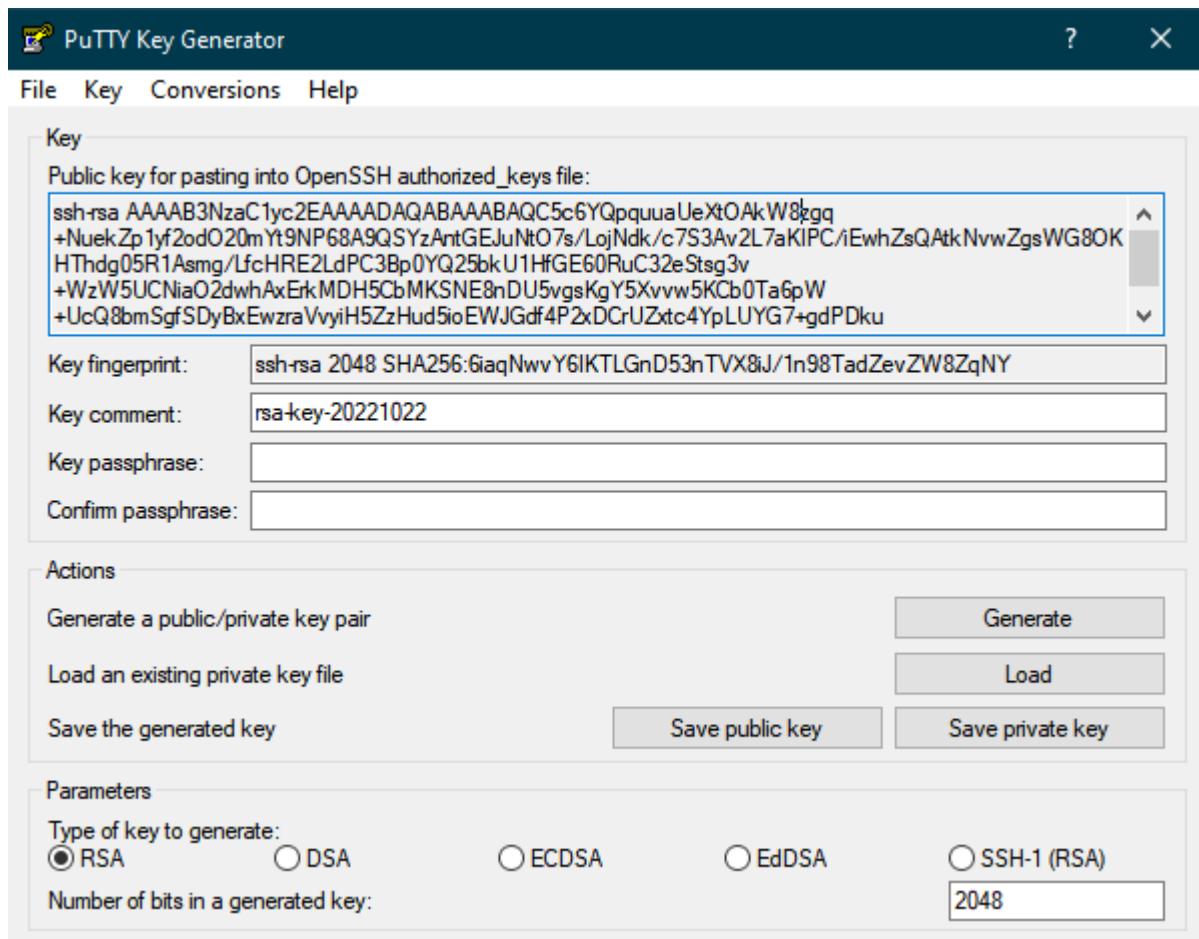
En este ejercicio tuvimos problemas al principio para establecer la conexión SSH porque intentábamos acceder usando el nombre del host en lugar de la dirección IP correcta. El comando no daba un error claro y parecía que el servidor no respondía. Tras comprobar la configuración de red y usar directamente la IP, conseguimos conectarnos sin problemas y verificar que el acceso remoto funcionaba correctamente.

## EJERCICIO 4

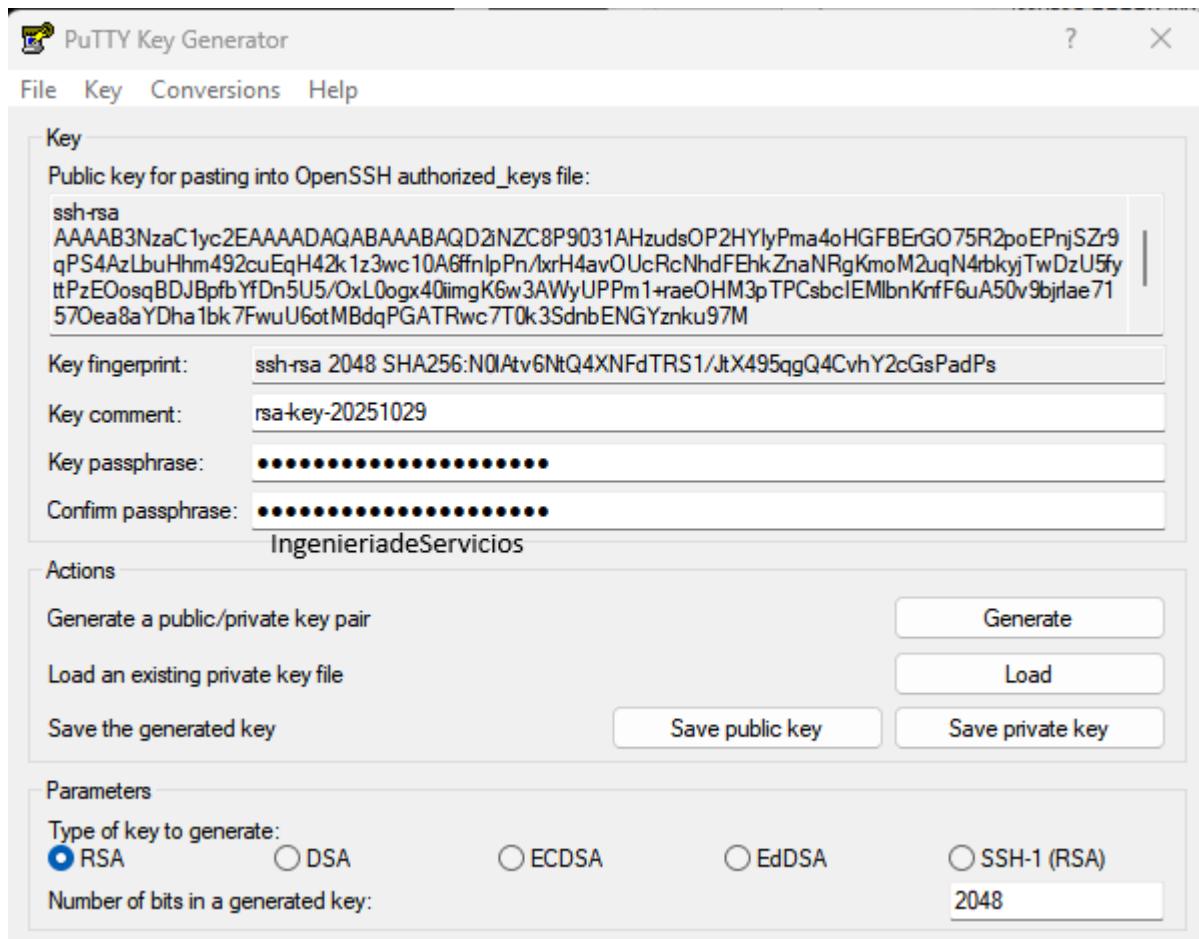
Ejecuta el PuTTYgen y verás una ventana como la siguiente:



Selecciona el tipo de clave RSA (seleccionado por defecto), y pulsa el botón Generate. Mueve el ratón aleatoriamente para generar aleatoriedad, hasta que la barra verde se complete. Al cabo de un rato obtendrás una pantalla como la siguiente:



**Introduce una frase de paso (y repítela debajo). Es la contraseña que servirá para cifrar la parte privada de la clave. Asegúrate de que la contraseña sea suficientemente larga y de que no se te olvide. No uses la misma contraseña que hayas puesto al usuario alumno dentro de la máquina Ubuntu, para diferenciar mejor cuándo se usa una u otra.**



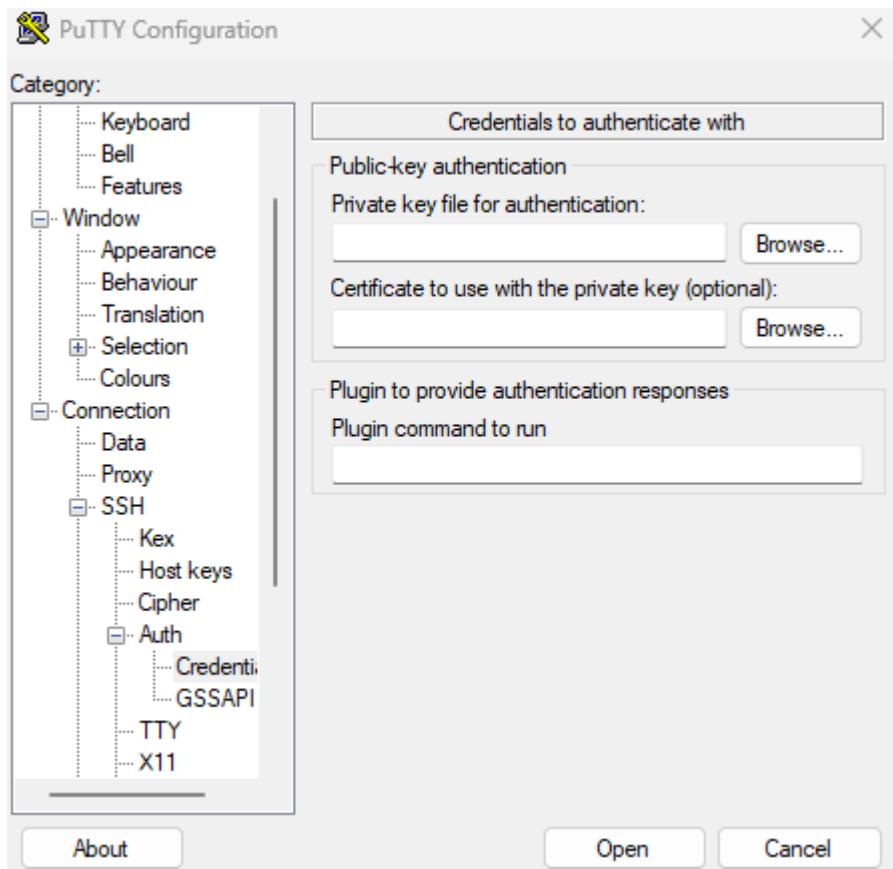
Pulsa el botón **Save private key** y elige un lugar donde guardar el archivo, al que puedes darle cualquier nombre. El almacén puede ser un lápiz USB. Recuerda que si pierdes el lápiz y alguien se hace con ese archivo, aún no podrá hacer uso de él mientras no conozca la frase de paso que has puesto en el punto anterior.

Pulsa el botón **Save public key** y guarda la parte pública en la misma carpeta en que guardaste la parte privada.

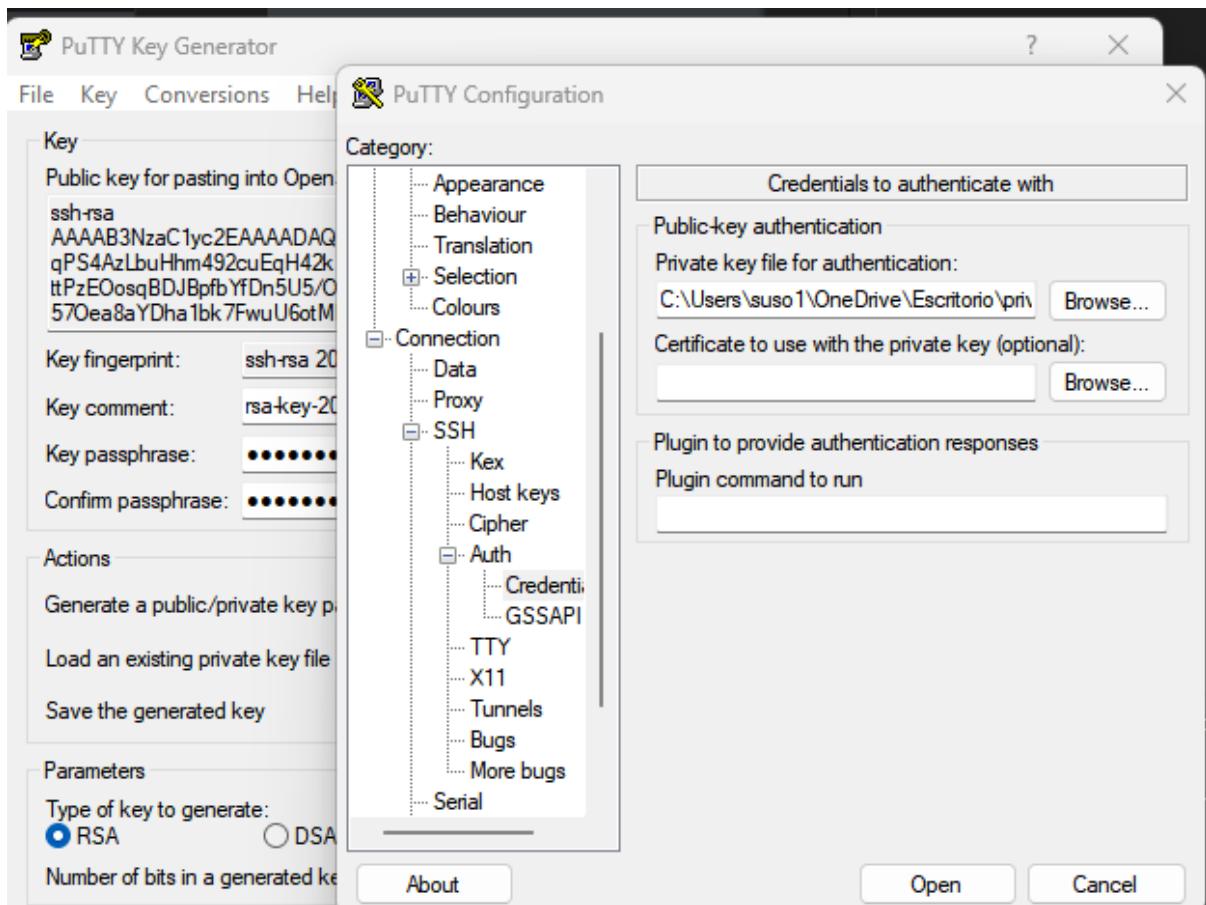
## EJERCICIO 5

Ejecuta el programa PuTTY. Si tenías una sesión guardada (**Saved Sessions**) para acceder al servidor Ubuntu, carga esa sesión, para modificarla. Si no la tenías, introduce la IP del servidor, asegúrate de que esté elegido el protocolo SSH (puerto 22), e introduce un nombre para la sesión (por ejemplo IngServ) y pulsa el botón **Save**.

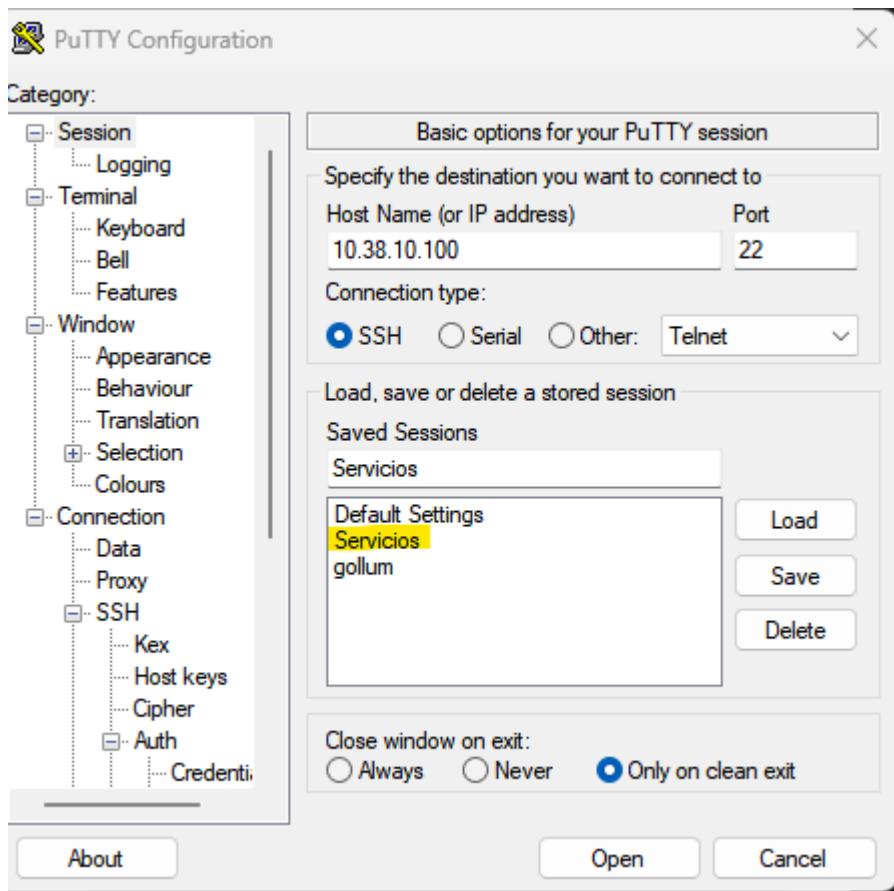
En la lista de opciones del panel izquierdo, elige **Connection → SSH → Auth → Credentials**. Verás un panel como el siguiente:



Pulsa el botón **Browse...** y elige el archivo con la clave privada que guardaste antes, al ejecutar PuTTYgen.



Vuelve a la opción primera, **Session** y pulsa de nuevo **Save** para guardar el cambio que acabas de hacer en esta sesión (esto es, añadirle un mecanismo de autenticación por clave pública).



Al configurar PuTTY para usar autenticación por clave pública, olvidamos guardar la sesión después de seleccionar la clave privada. Esto provocó que, al cerrar y volver a abrir PuTTY, siguiera intentando autenticarse solo con usuario y contraseña. Tras repetir la configuración y guardar correctamente la sesión, PuTTY empezó a usar la clave privada sin problemas.

## EJERCICIO 6

**Entra en sesión en el servidor SSH. Esta vez todavía tendrás que autenticarte por el método clásico de usuario y contraseña Unix.**

- Una vez dentro, ve a la carpeta `~/.ssh`
- Edita el fichero llamado `authorized_keys`. Este fichero contiene copias de todas las claves públicas que el usuario quiera usar para autenticarse ante este servidor. Si no existía el fichero, créalo.
- Ve de nuevo a la ventana en que estaba ejecutándose PuTTYgen. ¿La has cerrado? No importa, ejecútalo de nuevo y pulsa el botón `Load` para cargar el fichero con la clave privada que guardaste en su momento al generarlas. En el momento de cargar ese fichero te pedirá tu frase de paso.
- Verás que en la parte superior de la ventana, debajo del rótulo que dice "`_Public key for pasting into OpenSSH authorized_keys file`" hay una larga secuencia de letras y números. Selecciona todo ello con el ratón (o más rápido, pulsando Alt-P).
- En la terminal en que estabas editando `authorized_keys`, pega la secuencia de letras y números que acabas de copiar de PuTTYGen.

- Guarda el archivo y sal del servidor.

```

GNU nano 7.2          authorized_keys
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQD2iNZC8P9031AHzudsOP2HYIyPma4oHGFBErGO75R>

[ Read 1 line ]
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^/ Go To Line

```

## SERVICIOS DE CORREO ELECTRONICO

### PRACTICA 4

En esta práctica nos metimos con DNS y correo electrónico para entender “qué hay debajo” cuando escribimos un dominio o enviamos un mail. Por un lado, resolvimos nombres a IPs usando herramientas como dig/host/nslookup, comparando qué información aporta cada una y viendo que a veces un mismo dominio puede resolver distinto según el servidor DNS o la caché. También hicimos resolución iterativa paso a paso (desde un DNS raíz hasta el autoritativo) para ver el camino real que sigue una consulta.

Por otro lado, pasamos a correo electrónico, primero enviando mensajes “a bajo nivel” hablando SMTP por sockets, y después usando librerías de Python (smtplib, email).

### **EJERCICIO 1**

Averigua cuál es la IP de [www.uniovi.es](http://www.uniovi.es) y de [en.wikipedia.org](http://en.wikipedia.org), usando las tres herramientas mencionadas en los párrafos anteriores. Compara la salida producida por cada uno de ellos ¿cuál es más concisa? ¿cuál la que da información más próxima a la implementación del protocolo?

Compara ahora el resultado de [en.wikipedia.org](http://en.wikipedia.org) (versión inglesa) con [es.wikipedia.org](http://es.wikipedia.org) (versión española) ¿qué conclusión obtienes? ¿Qué saldrá si, desde el navegador web, accedes a wikipedia directamente por la IP?

```
uo288967@is02:~$ dig @156.35.41.4 www.uniovi.es

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @156.35.41.4 www.uniovi.es
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11507
;; flags: qr aa rd; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1
;; WARNING: recursion requested but not available

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 9ab281bbf9dc64b7105blb40691104dd533c04588d3245d6 (good)
;; QUESTION SECTION:
;www.uniovi.es.           IN      A

;; ANSWER SECTION:
www.uniovi.es.        1800    IN      CNAME   crisbl01.sic233.uniovi.es.
crisbl01.sic233.uniovi.es. 300    IN      A       156.35.233.101

;; Query time: 29 msec
;; SERVER: 156.35.41.4#53(156.35.41.4) (UDP)
;; WHEN: Sun Nov 09 21:17:17 UTC 2025
;; MSG SIZE rcvd: 125
```

uniovi

```
uo288967@is02:~$ dig en.wikipedia.org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> en.wikipedia.org
; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23683
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;en.wikipedia.org.           IN      A

;; ANSWER SECTION:
en.wikipedia.org.        23121   IN      CNAME   dyna.wikimedia.org.
dyna.wikimedia.org.       45      IN      A       185.15.58.224

;; Query time: 13 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Sun Nov 09 21:18:10 UTC 2025
;; MSG SIZE rcvd: 90

uo288967@is02:~$ 
```

EN wikipedia

```
uo288967@is02:~$ dig es.wikipedia.org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> es.wikipedia.org
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11733
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

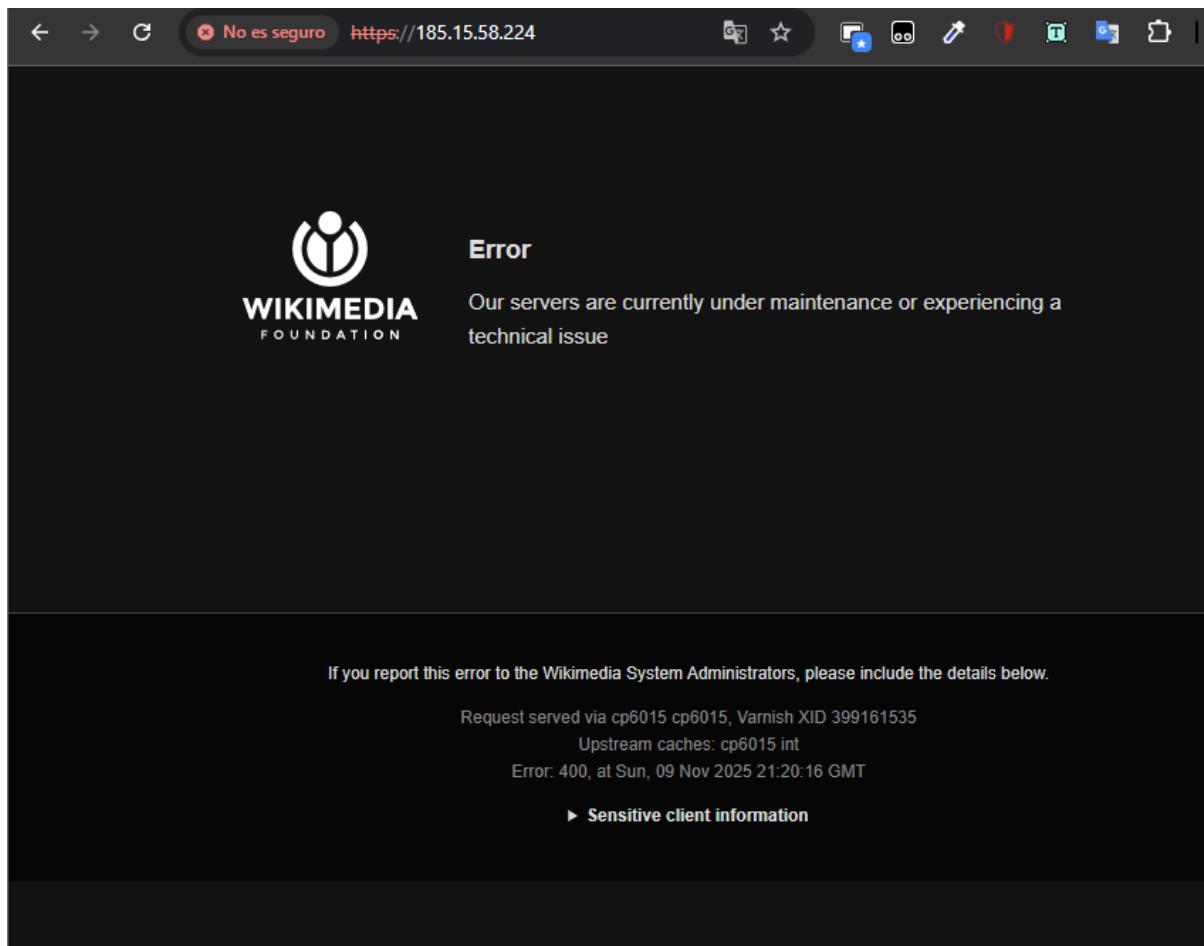
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;es.wikipedia.org.           IN      A

;; ANSWER SECTION:
es.wikipedia.org.    16808   IN      CNAME   dyna.wikimedia.org.
dyna.wikimedia.org. 12       IN      A       185.15.58.224

;; Query time: 16 msec
;; SERVER: 127.0.0.53#53(127.0.0.53) (UDP)
;; WHEN: Sun Nov 09 21:18:43 UTC 2025
;; MSG SIZE  rcvd: 90
```

ES Wikipedia

Si accedes desde el navegador:



No cargaría, necesita el nombre de dominio en la cabecera HTTP.

## EJERCICIO 2

Repite cualquiera de las consultas anteriores usando otro servidor de nombres. En /etc/resolv.conf podrás encontrar la IP de otro. Además Google mantiene un servidor de nombres público en la IP 8.8.8.8 (fácil de recordar).

```
uo288967@is02:~$ dig @8.8.8.8 www.uniovi.es

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @8.8.8.8 www.uniovi.es
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 24965
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;www.uniovi.es.           IN      A

;; ANSWER SECTION:
www.uniovi.es.        1641    IN      CNAME   crisbl01.sic233.uniovi.es.
crisbl01.sic233.uniovi.es. 141    IN      A       156.35.233.101

;; Query time: 22 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Sun Nov 09 21:26:07 UTC 2025
;; MSG SIZE rcvd: 88

uo288967@is02:~$
```

## EJERCICIO 3

Durante todo este ejercicio usa siempre la opción +norecurse para lanzar las consultas en modo iterativo.

Realiza la siguiente consulta al DNS público de Google:

```
$ dig @8.8.8.8 +norecurse en.wikipedia.org
```

Observa la respuesta. ¿Puedes leer la IP de wikipedia? Es posible que aparezca, a pesar de que el servidor 8.8.8.8 no sea autoridad para ese dominio. Esto se debe a que este servidor es propiedad de Google y cachea las IPs de nodos frecuentemente consultados. Si pruebas a hacer la misma consulta al servidor 156.35.41.4 verás que éste en cambio no incluye en su respuesta la IP de wikipedia. Vamos a realizar nosotros mismos la resolución iterativa, comenzando por un DNS-raíz.

- Averigua la IP de un DNS-raíz. Para ello consulta a un DNS (el mismo 8.8.8.8) información sobre el dominio raíz (valor: .). ¿Qué IP has averiguado? Anótala (nota, dig no te devuelve la IP sino el nombre, pero puedes usarlo de nuevo para resolver ese nombre en una IP).
- Usa de nuevo dig pero poniendo tras la @ la IP averiguada en el paso anterior (de modo que haces la consulta a ese servidor top-level), y consulta acerca del dominio org, para obtener la IP de otro servidor.

- Usa de nuevo dig pero poniendo tras la @ la IP anterior, y así la consulta irá a un servidor first-level, y pregúntale sobre wikipedia.org. Anota la IP de alguno de sus servidores de nombres (tiene tres).
- Finalmente, usa dig para interrogar a uno de esos servidores acerca de en.wikipedia.org
- Finalmente obtendrás la IP buscada. ¿Cuál es? Compárala con la que sale usando el comando host.

```
uo288967@is02:~$ dig @8.8.8.8 +norecurse en.wikipedia.org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @8.8.8.8 +norecurse en.wikipedia.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: SERVFAIL, id: 64485
;; flags: qr ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;en.wikipedia.org.           IN      A

;; Query time: 20 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Sun Nov 09 21:30:28 UTC 2025
;; MSG SIZE  rcvd: 45

uo288967@is02:~$ 
```

```
uo288967@is02:~$ dig @156.35.41.4 +norecurse en.wikipedia.org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @156.35.41.4 +norecurse en.wikipedia.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: REFUSED, id: 63741
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
; COOKIE: 7b466f37183ef34f6a5e20016911080b63d35f447181d0b0 (good)
;; QUESTION SECTION:
;en.wikipedia.org.           IN      A

;; Query time: 37 msec
;; SERVER: 156.35.41.4#53(156.35.41.4) (UDP)
;; WHEN: Sun Nov 09 21:30:51 UTC 2025
;; MSG SIZE  rcvd: 73

uo288967@is02:~$ 
```

Averiguar un DNS raíz

```
uo288967@is02:~$ dig @8.8.8.8 +norecurse .

; <>> DiG 9.18.30-Ubuntu0.24.04.2-Ubuntu <>> @8.8.8.8 +norecurse .
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 324
;; flags: qr ra ad; QUERY: 1, ANSWER: 0, AUTHORITY: 1, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;.

; IN A

;; AUTHORITY SECTION:
. 86393 IN SOA a.root-servers.net. ns1d.verisign-grs.com. 2025110901 1800 900 60480
0 86400

;; Query time: 21 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Sun Nov 09 21:31:12 UTC 2025
;; MSG SIZE rcvd: 103

uo288967@is02:~$
```

### Obtener IP raíz

```
uo288967@is02:~$ dig @8.8.8.8 a.root-servers.net

; <>> DiG 9.18.30-Ubuntu0.24.04.2-Ubuntu <>> @8.8.8.8 a.root-servers.net
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 44286
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 512
;; QUESTION SECTION:
;a.root-servers.net. IN A

;; ANSWER SECTION:
a.root-servers.net. 259196 IN A 198.41.0.4

;; Query time: 23 msec
;; SERVER: 8.8.8.8#53(8.8.8.8) (UDP)
;; WHEN: Sun Nov 09 21:32:57 UTC 2025
;; MSG SIZE rcvd: 63

uo288967@is02:~$
```

### Preguntar al DNS raíz por el dominio

```

uo288967@is02:~$ dig @198.41.0.4 +norecurse org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @198.41.0.4 +norecurse org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 12162
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 6, ADDITIONAL: 13

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;org.                      IN      A

;; AUTHORITY SECTION:
org.           172800  IN      NS      a2.org.afiliias-nst.info.
org.           172800  IN      NS      b2.org.afiliias-nst.org.
org.           172800  IN      NS      d0.org.afiliias-nst.org.
org.           172800  IN      NS      a0.org.afiliias-nst.info.
org.           172800  IN      NS      b0.org.afiliias-nst.org.
org.           172800  IN      NS      c0.org.afiliias-nst.info.

;; ADDITIONAL SECTION:
a2.org.afiliias-nst.info. 172800  IN      A      199.249.112.1
a2.org.afiliias-nst.info. 172800  IN      AAAA    2001:500:40::1
b2.org.afiliias-nst.org. 172800  IN      A      199.249.120.1
b2.org.afiliias-nst.org. 172800  IN      AAAA    2001:500:48::1
d0.org.afiliias-nst.org. 172800  IN      A      199.19.57.1
d0.org.afiliias-nst.org. 172800  IN      AAAA    2001:500:f::1
a0.org.afiliias-nst.info. 172800  IN      A      199.19.56.1
a0.org.afiliias-nst.info. 172800  IN      AAAA    2001:500:e::1
b0.org.afiliias-nst.org. 172800  IN      A      199.19.54.1
b0.org.afiliias-nst.org. 172800  IN      AAAA    2001:500:c::1
c0.org.afiliias-nst.info. 172800  IN      A      199.19.53.1
c0.org.afiliias-nst.info. 172800  IN      AAAA    2001:500:b::1

;; Query time: 36 msec
;; SERVER: 198.41.0.4#53(198.41.0.4) (UDP)
;; WHEN: Sun Nov  9 21:34:04 UTC 2025
;; MSG SIZE  rcvd: 434

uo288967@is02:~$ 

```

Preguntar al DNS del TLD .org por wikipedia.org

```

uo288967@is02:~$ dig @199.19.56.1 +norecurse wikipedia.org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @199.19.56.1 +norecurse wikipedia.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 55617
;; flags: qr; QUERY: 1, ANSWER: 0, AUTHORITY: 3, ADDITIONAL: 4

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
;; QUESTION SECTION:
;wikipedia.org.           IN      A

;; AUTHORITY SECTION:
wikipedia.org.      3600    IN      NS      ns2.wikimedia.org.
wikipedia.org.      3600    IN      NS      ns0.wikimedia.org.
wikipedia.org.      3600    IN      NS      ns1.wikimedia.org.

;; ADDITIONAL SECTION:
ns0.wikimedia.org.   3600    IN      A       208.80.154.238
ns1.wikimedia.org.   3600    IN      A       208.80.153.231
ns2.wikimedia.org.   3600    IN      A       198.35.27.27

;; Query time: 20 msec
;; SERVER: 199.19.56.1#53(199.19.56.1) (UDP)
;; WHEN: Sun Nov 09 21:34:52 UTC 2025
;; MSG SIZE rcvd: 154

uo288967@is02:~$ 

```

Preguntar al servidor autoritativo por en.wikipedia.org

```

uo288967@is02:~$ dig @208.80.154.238 +norecurse en.wikipedia.org

; <>> DiG 9.18.30-0ubuntu0.24.04.2-Ubuntu <>> @208.80.154.238 +norecurse en.wikipedia.org
; (1 server found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 40030
;; flags: qr aa; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 0ef6c3076c404661049117e890f07891 (good)
;; QUESTION SECTION:
;en.wikipedia.org.           IN      A

;; ANSWER SECTION:
en.wikipedia.org.     86400   IN      CNAME   dyna.wikimedia.org.

;; Query time: 109 msec
;; SERVER: 208.80.154.238#53(208.80.154.238) (UDP)
;; WHEN: Sun Nov 09 21:35:43 UTC 2025
;; MSG SIZE rcvd: 94

uo288967@is02:~$ 

```

Resultado final:

en.wikipedia.org → 185.15.58.224

Lo comprobamos con host

```
uo288967@is02:~$ host en.wikipedia.org
en.wikipedia.org is an alias for dyna.wikimedia.org.
dyna.wikimedia.org has address 185.15.58.224
dyna.wikimedia.org has IPv6 address 2a02:ec80:600:edla::1
uo288967@is02:~$
```

Coincide con nuestra resolución iterativa.

## EJERCICIO 4 dns\_ej1.py

Instala el módulo dnspython como se indica más arriba y escribe el siguiente programa (dns\_ej1.py) para probarlo:

```
import dns.resolver
```

```
respuesta = dns.resolver.query('en.wikipedia.org')
print(respuesta[0].address)
```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 dns_ej1.py
185.15.58.224
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

Como ves, la biblioteca te da un acceso de alto nivel a la información buscada, pero si lo necesitas también tienes acceso de bajo nivel. En la variable respuesta.response tienes directamente la respuesta del servidor, que es convertida a una cadena de texto legible si tratas de imprimirla con print. Añade por ejemplo la siguiente línea al script anterior:

```
print(respuesta.response)
```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 dns_ej1.py
185.15.58.224
id 26128
opcode QUERY
rcode NOERROR
flags QR RD RA
edns 0
payload 65494
;QUESTION
en.wikipedia.org. IN A
;ANSWER
en.wikipedia.org. 54 IN CNAME dyna.wikimedia.org.
dyna.wikimedia.org. 54 IN A 185.15.58.224
;AUTHORITY
;ADDITIONAL
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

También la puedes obtener en la representación binaria usada por el protocolo (sólo como curiosidad), así:

```
print(respuesta.response.to_wire())
```

## EJERCICIO 5 dns\_ips\_apple.py

Escribe un pequeño script en python que muestre por pantalla todas las IPs asignadas al nodo apple.com, usando el módulo dns.resolver en forma análoga al ejemplo anterior. Guárdalo con el nombre dns\_ips\_apple.py

Pista: el resultado devuelto por dns.resolver.query es una especie de lista, sobre la que se puede iterar con un bucle for variable in ...:

```
import dns.resolver

dominio = 'apple.com'
respuesta = dns.resolver.resolve(dominio, 'A')

print(f"Direcciones IP asociadas a {dominio}:\n")
for ip in respuesta:
    print(ip.address)

print("\nRespuesta completa del servidor DNS:\n")
print(respuesta.response)
```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 dns_ips_apple.py
Direcciones IP asociadas a apple.com:

17.253.144.10

Respuesta completa del servidor DNS:

id 64748
opcode QUERY
rcode NOERROR
flags QR RD RA
edns 0
payload 65494
;QUESTION
apple.com. IN A
;ANSWER
apple.com. 253 IN A 17.253.144.10
;AUTHORITY
;ADDITIONAL
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

## EJERCICIO 6

Escribe un pequeño programa en python (llámalo dns\_getaddrinfo\_wikipedia.py) que se conecte a la página web de Wikipedia, solicite el contenido raíz (/) y muestre en pantalla la respuesta del servidor. Para ello:

- Usa `getaddrinfo()` para obtener información de conexión con el sitio, como en el ejercicio anterior.
- Usa los valores devueltos por `getaddrinfo()` para crear el socket y conectar al servidor, por IPv4.
- Una vez establecida la conexión envía una cabecera HTTP que solicite el recurso mencionado.
- Recibe la respuesta del servidor y muéstralala en pantalla.

Después intenta la conexión por IPv6. No lo conseguirás porque la red de la Universidad aún no soporta ese protocolo.

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 dns_getaddrinfo_wikipedia.py
Resolviendo www.wikipedia.org...

Intentando conectar con ('185.15.58.224', 80)...
--- Respuesta del servidor ---

HTTP/1.1 301 Moved Permanently
content-length: 0
Location: https://www.wikipedia.org/
server: HAProxy
x-cache: cp6009 int
x-cache-status: int-tls
connection: close

(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

Intento de conexión por IPv6:

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ ping6 2620:0:862:edla::1
ping6: connect: Network is unreachable
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

La red no lo soporta

En el programa con getaddrinfo() + socket para pedir GET / a Wikipedia, lo más “pesado” fue que la primera vez nos salía una respuesta rara/incompleta porque no estábamos cerrando bien la cabecera HTTP (nos faltaba el salto doble \r\n\r\n\r\n).

## EJERCICIO 7 email\_envia\_crudo.py

Escribe un programa en python llamado `email_envia_crudo.py` que envíe un correo electrónico. Para ello sigue los siguientes pasos:

1. Define una función denominada `RecvReply` con dos parámetros: un socket y un código (cadena de bytes). Esta función debe recibir hasta un máximo de 1024 bytes del socket e imprimir por pantalla lo recibido. A continuación, extraerá los tres primeros caracteres (bytes) de lo que ha recibido y verificará si coinciden con el código recibido como parámetro. En caso de que no coincidan imprimirá por pantalla un error y terminará el programa (`sys.exit()`). Si quieres, puedes imprimir también la respuesta recibida por el socket, para depuración.
2. Dentro del programa principal define las siguientes variables y asignales un valor adecuado: `server` (dirección del servidor SMTP), `port` (puerto del servidor), `fromaddr` (dirección de correo desde la que se envía), `toaddr` (dirección de correo a la que se envía), `subject` (asunto del mensaje), `data` (texto que va a contener el mensaje). Configura el servidor como `relay.uniovi.es` y el puerto como 25. La dirección de envío y la de recepción será la que tengas en uniovi.
3. Crea una conexión de tipo TCP al servidor usando las variables creadas.
4. A través de ese socket ya conectado, envía los comandos SMTP necesarios para enviar el correo electrónico. Tras cada comando llama a la función `RecvReply` para recibir la respuesta y verificar que es correcta. Puede servirte de ayuda el [ejemplo de diálogo SMTP](#) que aparece en Wikipedia.

Para configurar los datos del mensaje se puede realizar lo siguiente:

```
message = """To: %s  
From: %s  
Subject: %s\r\n\r\n%  
\r\n.\r\n"""\n    % (toaddr, fromaddr, subject, data)
```

Fíjate como entre las cabeceras y el cuerpo del mensaje hay una secuencia \r\n\r\n y al final del mensaje hay una secuencia \r\n.\r\n.

Ejecuta el script (sólo te funcionará si lo haces desde un ordenador de prácticas, pues relay.uniovi.es rechaza conexiones exteriores). Lee tu correo para verificar que ha llegado.

```

IngenieriaServicios [SSH: 192.168.1.147] - email_envia_crudo.py

1 import socket
2 import sys
3
4 # --- Función para recibir y verificar respuestas del servidor SMTP ---
5 def RecvReply(sock, expected_code):
6     """
7     Recibe respuesta del servidor y comprueba el código esperado.
8     """
9     reply = sock.recv(1024)
10    print("Servidor:", reply.decode().strip()) # Mostrar lo que llega
11
12    code = reply[:3] # primeros 3 bytes -> código de estado
13    if code != expected_code:
14        print(f"✗ Error: se esperaba {expected_code.decode()}, pero se recibió {code.decode()}")
15        sys.exit(1)
16    return reply
17
18
19 # --- Datos de configuración ---
20 server = "relay.uniovi.es"
21 port = 25
22
23 fromaddr = "tu_usuario@uniovi.es" # <- pon tu correo Uniovi
24 toaddr = "tu_usuario@uniovi.es" # <- destinatario (puede ser el mismo)
25 subject = "Prueba SMTP crudo desde Python"
26 data = "Este mensaje ha sido enviado manualmente usando sockets TCP."
27
28 # --- Construir el mensaje con formato SMTP ---
29 message = """To: %s
30 From: %s
31 Subject: %s\r\n\r\n
32 %s
33 \r\n.\r\n"""\n (toaddr, fromaddr, subject, data)
34
35 # --- Crear socket TCP ---
36 print(f"Conectando a {server}:{port} ...")
37 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
38 sock.connect((server, port))
39
40 # --- Comienzo del diálogo SMTP ---
41 RecvReply(sock, b"220") # Banner inicial del servidor
42 sock.send(b"HELO uniovi.es\r\n") # Identificación
43 RecvReply(sock, b"250")
44
45 sock.send(f"MAIL FROM:<{fromaddr}>\r\n".encode()) # Remitente
46 RecvReply(sock, b"250")
47
48 sock.send(f"RCPT TO:<{toaddr}>\r\n".encode()) # Destinatario
49 RecvReply(sock, b"250")
50
51 sock.send(b"DATA\r\n") # Comienza el cuerpo del mensaje
52 RecvReply(sock, b"354")
53
54 sock.send(message.encode()) # Enviar cuerpo completo (termina en \r\n.\r\n)
55 RecvReply(sock, b"250")
56
57 sock.send(b"QUIT\r\n") # Terminar sesión
58 RecvReply(sock, b"221")
59
60 sock.close()
61 print("✓ Mensaje enviado correctamente.")
62

```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 email_envia_crudo.py
Conectando a relay.uniovi.es:25 ...
Servidor: 220 sablon.net.uniovi.es ESMTP Postfix
Servidor: 250 sablon.net.uniovi.es
Servidor: 250 2.1.0 Ok
Servidor: 554 5.7.1 <tu_usuario@uniovi.es>: Relay access denied
✗ Error: se esperaba 250, pero se recibió 554
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

En el envío SMTP “a mano” con `email_envia_crudo.py`, el problema fue que desde casa el servidor `relay.uniovi.es` nos rechazaba la conexión. Al principio pensábamos que era un fallo del código (porque el diálogo SMTP ni llegaba a empezar), pero revisando el guion entendimos que solo funciona desde la red de prácticas.

## EJERCICIO 8

Modifica el ejercicio `email_envia_crudo.py` para que construya el mensaje como acabamos de mostrar (pero haciendo uso de las variables `toaddr`, `fromaddr`, `subject` y `data`). Recuerda usar `.as_bytes()` y añadir la línea final que contiene sólo un punto a la secuencia de bytes retornada.

```

IngenieriaServicios [SSH: 192.168.1.147] - email_envia_crudo_bytes.py

1 import socket
2 import sys
3 from email.message import EmailMessage
4
5 # --- Función para recibir y verificar respuesta del servidor SMTP ---
6 def RecvReply(sock, expected_code):
7     reply = sock.recv(1024)
8     print("Servidor:", reply.decode().strip())
9
10    code = reply[:3]
11    if code != expected_code:
12        print(f"🔴 Error: se esperaba {expected_code.decode()}, pero se recibió {code.decode()}")
13        sys.exit(1)
14    return reply
15
16
17 # --- Datos de configuración ---
18 server = "relay.uniovi.es"
19 port = 25
20
21 fromaddr = "tu_usuario@uniovi.es"    # Cambia por tu dirección Uniovi
22 toaddr = "tu_usuario@uniovi.es"
23 subject = "Prueba SMTP con EmailMessage (Ejercicio 8)"
24 data = "Este mensaje se ha construido usando EmailMessage y enviado con sockets TCP crudos."
25
26 # --- Crear el mensaje con la librería email ---
27 msg = EmailMessage()
28 msg["From"] = fromaddr
29 msg["To"] = toaddr
30 msg["Subject"] = subject
31 msg.set_content(data)
32
33 # Convertir a bytes (añadiremos la línea final del punto más adelante)
34 msg_bytes = msg.as_bytes()
35
36 # --- Crear socket TCP ---
37 print(f"Conectando a {server}:{port} ...")
38 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
39 sock.connect((server, port))
40
41 # --- Diálogo SMTP ---
42 RecvReply(sock, b"220")                      # Banner inicial
43 sock.send(b"HELO uniovi.es\r\n")
44 RecvReply(sock, b"250")
45
46 sock.send(f"MAIL FROM:<{fromaddr}>\r\n".encode())
47 RecvReply(sock, b"250")
48
49 sock.send(f"RCPT TO:<{toaddr}>\r\n".encode())
50 RecvReply(sock, b"250")
51
52 sock.send(b"DATA\r\n")
53 RecvReply(sock, b"354")
54
55 # Enviar cuerpo del mensaje y secuencia final
56 sock.send(msg_bytes)
57 sock.send(b"\r\n.\r\n")
58 RecvReply(sock, b"250")
59
60 sock.send(b"QUIT\r\n")
61 RecvReply(sock, b"221")
62
63 sock.close()
64 print("Mensaje enviado correctamente (si el servidor lo permite).")
65
66

```

```
(dnsenv) uc288967@is02:~/IngenieriaServicios/Practica4$ python3 email_envia_crudo_bytes.py
Conectando a relay.uniovi.es:25 ...
Servidor: 220 sablon.net.uniovi.es ESMTP Postfix
Servidor: 250 sablon.net.uniovi.es
Servidor: 250 2.1.0 Ok
Servidor: 554 5.7.1 <tu.usuario@uniovi.es>: Relay access denied
X Error: se esperaba 250, pero se recibió 554
(dnsenv) uc288967@is02:~/IngenieriaServicios/Practica4$
```

Lo rechaza debido a que no permite reenviar correos desde equipos no autenticados o externos a la red universitaria.

## EJERCICIO 9 (opcional)

Descarga el fichero logoatc.gif con el siguiente comando:

```
$ wget http://www.atc.uniovi.es/estilo_atc/logoatc.gif
```

Modifica email\_crear\_mensaje\_prueba() para que incluya el adjunto, y observa la salida al ejecutar el programa. Verás que ahora el cuerpo de mensaje es multipart/mixed y que la biblioteca se ha ocupado de elegir una cadena aleatoria como *boundary*. Observa también como el adjunto ha sido codificado en base64.

Prueba a modificar email\_envia\_crudo.py para que envíen un mensaje con ese adjunto y después verifica que lo hayas recibido correctamente.



IngenieriaServicios [SSH: 192.168.1.147] - email\_crear\_mensaje\_prueba.py

```
1 import email.message
2 import email.policy
3 import email.utils
4
5 # Crear el mensaje principal
6 mensaje = email.message.EmailMessage()
7 mensaje['To'] = 'Desti Natario <destinatario@example.com>'
8 mensaje['From'] = 'Remi Tente <remitente@example.com>'
9 mensaje['Subject'] = 'Mensaje de prueba con adjunto'
10 mensaje['Date'] = email.utils.formatdate(localtime=True)
11 mensaje['Message-ID'] = email.utils.make_msgid()
12
13 # Cuerpo de texto
14 mensaje.set_content("Esto es una prueba con el logo ATC adjunto.\n\nSaludos.")
15
16 # Añadir adjunto (imagen logoatc.gif)
17 with open("logoatc.gif", "rb") as adjunto:
18     contenido = adjunto.read()
19     mensaje.add_attachment(contenido,
20                           maintype='image',
21                           subtype='gif',
22                           filename='logoatc.gif')
23
24 # Mostrar el mensaje en formato MIME (texto completo)
25 binario = mensaje.as_bytes(policy=email.policy.default)
26 print(binario.decode("utf-8"))
27
28
```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 email_crear_mensaje_prueba.py
To: Desti Natario <destinatario@example.com>
From: Remi Tente <remitente@example.com>
Subject: Mensaje de prueba con adjunto
Date: Sun, 09 Nov 2025 22:11:09 +0000
Message-ID: <176272626981.3952.4084878005271312256@is02>
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary="=====0720082976295181468=="

=====0720082976295181468==
Content-Type: text/plain; charset="utf-8"
Content-Transfer-Encoding: 7bit

Esto es una prueba con el logo ATC adjunto.

Saludos.

=====0720082976295181468==
Content-Type: image/gif
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="logoatc.gif"
MIME-Version: 1.0

R0lGODlhRgBGAPcAAAA2bPT2+QFKkuTq7wFEhwFAfwJNmOzu8AAtYQ5KhiFNegARTQFSooQAiWA04
YaOyxkVYggA4cJahuQA0Zw1WngBFng9apgg4a1MixQ9ZgE9eAANSejr7gAuXvHz9g5BcgFChQA0
aAApXwEnTQAsWA43XwE1SAFIjgAyZQFCgwFUqGiFowFHjGKBoQE4agApU+nu8xhAaW18kcjT3gA7
dQFFiI6kult+ngAwYAE/fAAyZgAfPRdGdA9dq+7x9QAmXHyXsWuRtwAkW+vv9AEjRJ+2yQ0zgBT
pYCZszFYgsXR3AdVoudrixNeniTru3y+GZzluvw9QAADAAtWebr8AFGiwAKGAwlPwkuVBhCbQAA
OQAZMQFOmgAxYgopSGqIpqAnncgA1aQJasgAVLoWasw5R1AABLXKNqd7166y9zgA2aA5UmwB0mAlE
egAwax1HdQBE1AFBggE6cxY+Zj5kiwE/fsrU3w0+bgExYQs1XgAUTwAwZAFBgQErVFJ8q5XoBxk
qyAuYwo6aw5TmQ4yVQEeuWwAJRA5HgQ5YowATNAUzYRQ+aAE9fgE8dwAeagAFRAAOlgAbVAJUpg09
bgEsVgJQngJ0mwAbNgBNmQFL1QA5cktvk9ff5wAoTwFNmQNGiwFllwAyYwFK1AJUpwQ1ZgBNmgQ/
dwASiWFPn2qA1wBMmI6fsG+MqQAlawAXUr/M2QE+fAFL1AFM1gArYPP1+EltkgA3bQE9eQ08aWh3
me/0+Q9itgA6fQQ+eQFcunNnf5SVTfzBHdg4/btLX4Q5Oj9Te6Qo9beHj6gUv2AQ2aICIpSvS3Rg/
ZszV32Z8kgJhwGV9nQ0+cAFarwAibAI1ZxE6ZAYxZUZrkQ+A+j/X7/AM8Fzquw6O1yLC+zgcrYIKV
p1F011R2mFh7nNTd5UhskR9LeQAIJgIyZdjh6hBPkQBNmAFNmPn6+/n6/Pn+/v39/RpIdgsoRg0z
WAw5agAeUA09bdn17CxckQA0eUxwk+vx+DdTfgFAgAFL1//wAzZiH5BAAAAAAALAAAAABGAEYA
AAj/AOFNGkiwoAMMu3ooXMiQwqA1ECNCtHCIImLGDOmQyRRIqIP8hyIHdkyy7+TKFP+m0JAjKSX
MF+OkuRJVL+bOG+yuDWCKs+fPvv0+VQhJ05QqAB0msK0qdNJK1WydBkT5syA0nEyONAT6E+hRLP2
Q6rUqVkJUKOinFrVKK2bYnd29UoJbNGsZJeabYpW7dqWbWW+Fdtv61yvdsXm3cs3rV+2ga/CzSqX
btChd40uZjylr9+VgCMPjsvVc13MipPqZe5cei2kg1XtpwYr2rOnR2rhQx7NOXStFhbLsu59e7X
VWPH5Wm6tubbxXVHJVHlmZjr2LOr2M69u684RF6I/x8vv1MnASoqqV+PCpWKVjtGyJ8/n5B01Tho
sCDAv79/AHj4FYIieRRo4IF59BJHHQw22KAcGtwi4YQTHnJfSiHgQAJnhRQyxikhiiFVe8EcOJ
KKaoYBQdtOhiiy+YIs0RNNZYw8XfqZSCBPssMU1QAYJ5BheZJHBkUgiOQcpOvjVgSkM1ILJ1FRO
SUGOOqLEo49CCkmkkUkmuWSTaj0ZZZVVXpm1lj1v+2OWQRYYpJpNOQi klm1ZiuWabb8IJppwZjFnn
```

```

SUGOOoqLEo49CCkmkkUkmuWSTaj0ZZZVVXpmIj1v+20WQRYYpJpNOQiklmlZiuWabb8IJppwZjFnn
mXhiouaaAvb05ptfAnqkoGxaWaiembJZ5+NOgppVGbeiehiEZ1KaNxakpnPIR+WimbiVZ5SaaA
bv+qUqeTghochj20scWuvO46Rjx2zCHssMMyk0IyCabLA6mMHFs9BC+8eqUe2BgAg//JDIGDvs
kAgk4IJrzg6KdGhuIXkskglPzTg7rvv0uMGHNjAwYY6quSbr606ToCAKwu48sMeFwhyQOM4/CCC
wRc03LAgeEcsyA+zUNMAFhnrLEXF2CQAC/4EMDHcMnsKoaIiwgRBjkldNMmgkP4MEQRerxRjC5P
5PyEPkPAAMMOQEfhwyvILAAEBwckrfTSQwiNzjlgOFiHyf2qoUck0AzjzgZcc7lE12HLbY/8yww
wNho++EIP2yzTfVnYRjSxDJcH/MAGWdAwY0/ARj/E0k+xwt+BRRpBPlALoJ/QucDQtjhzy8y0JPK
5JTLsEIQQfihsDSntu421Ggvs487WEkCwxwIbbGDGNxx/0IAeqKnubtj+/KPFBrG7gsAPM/jzjRRm
BC988IaA4cg0jKTQOT9vRzXBt5k4489wTyygAgoLdAC33cIodICtWxdzhIqISBE70yYQd98ROBB
wPKdNx/VBkX4Q4URj0wwQurac+89/5vYmgLIlxLz9W4V61vfCNr3Pvh5zi8N+MDWbqAFF0S1fwHo
3vcC6i8Blu98/kCgAhfoPgc+UCOb+II/fHCO/31vex104UkwEEPFhCE1QgA03ivAuhQA+F48YC
//Z3QRhqEIAJCJCBKDBjCBK5vhyuU3xJBiIQN+AWDR8xeDZV4EiYi0ArqDKMJdoCHE4DgjGhEIwYu
JII9oMEfLdDCEu2IxzlqARz+22ASP3hAQ2zILH1AQSPkQMhCFrINF/oBNajgDyQEgqGQjCQkA+GE
PCXg1z8BxNtoIcQtOKToATAR3DQhVKa0pS4SCQ1zoa2sHkgizPcIh/9YYMfoOBWKbmIWhtJy1Zy
7ZUy/AcN93jD3tXylrj8hy6jwkt/bAMCm4imNKMTARv4gxawFKYsi01LWyztTmao8mzjMsIBymrOc
ZghfDPWIyVke85vLVEkzWWFFtWAxmMNsjzedYP+G1Pnzn61bwB7UoJJ4pmSe9Swi34SBz20u8QeO
m4EykEDRiloUCSuwBgiKEiEhuOAdjTSEXZzwhq25TiUbEmfWgBEJ1cyiAb305dbCkdCTGHSKr2AH
OVzhFlfQoR1KEMEsVLKEJtDifj9QCQqWIpzcOKpUI0qVF+hgAZwVC0TmIUoECACIKY1DCLgAQP
EAaVqOEHHxCECAiakv0tAQFwjatc5dqAHlgwJtDfCcraigAKvqmQCKKgrW9s6AcaV1X1jFYFiF8vY
xSLArzalljcnC0/JUvayiMorZjeTMIy9rNr81x0R6sW0ZL2tJFFrWpLtwEAuPalsLVFGJA52RD/
wPa2uM3tb82ziTZE4LfDa4mAdb2KIgBjoIrnKXq9zhF1c1d2iDjqZL3erKwRbPvRUKwkCD6nr3
u969bnZREo0200C86E2vBmwRCsputxHpja9843sLAHTg1PitxyEoxN9b5CACXaBsKAAAi/4a2MCw
kEMrbMHgBjf4A4fIgYQnDIj8yECOEABCkLBQYQ6Xsr2f6QIA6jbhCpf4xDmocIpzUIAWu9jFfDhE
AnjIDxDIIRTKQ1YYdnxYv4g4DjQOMo35kAAMpODISE7yfkQhybHIckEKA/xvuPltiCACBQspa3
zOUkggyABxPCPmPlTgzKb+cwsIMOU/YKDCFRhw8xwjrOcCVADDKSjCnjOs573zGczrlktbT4BnwdN
6EJXgQXEKMMJF3oRjv60QIgwIsm3QESAEAAj860pj96DIMQgCgDrWoR03qTLDgt+DVhClukQ1S
u/rVsAZ1GdaQiVrb+ta4znUs2BGtXj9rHbHItbCHTexMyGIQFDCAspfN7GY7+9nQjra0p63sNfzB
EtjOtra3zelue/vb4A43tv+wnnKb+9zTre6181udAPCAvCo7znTe962/ve+M43vAEREEAA7
=====0720082976295181468=====
```

(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4\$

```

# --- Crear el mensaje con adjunto usando EmailMessage ---
msg = EmailMessage()
msg["From"] = fromaddr
msg["To"] = toaddr
msg["Subject"] = subject
msg.set_content(data)

# Adjuntar el archivo logoatc.gif (descargado previamente con wget)
try:
    with open("logoatc.gif", "rb") as adjunto:
        contenido = adjunto.read()
        msg.add_attachment(contenido,
                           maintype="image",
                           subtype="gif",
                           filename="logoatc.gif")
except FileNotFoundError:
    print("⚠️ No se encontró 'logoatc.gif'. Asegúrate de haberlo descargado antes:")
    print("wget http://www.atc.uniovi.es/estilo_atc/logoatc.gif")
    sys.exit(1)
```

He cambiado el código de creación del mensaje para que envíe un mensaje con ese adjunto y después verifique que lo hayas recibido

```

(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 email_envia_crudo.py
Conectando a relay.uniovi.es:25 ...
Servidor: 220 sablon.net.uniovi.es ESMTP Postfix
Servidor: 250 sablon.net.uniovi.es
Servidor: 250 2.1.0 Ok
Servidor: 554 5.7.1 <tu_usuario@uniovi.es>: Relay access denied
✖ Error: se esperaba 250, pero se recibió 554
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

Al estar fuera, sucede lo mismo

## EJERCICIO 10 dns\_mx\_gmail.py

Para obtener el campo MX usando el módulo dns de python, basta añadir el parámetro "MX" en la llamada a query(), por ejemplo:

```
respuesta = dns.resolver.query("dominio.com", "MX")
```

Tomando como punto de partida el código del ejercicio dns\_ips\_apple.py cópialo al fichero dns\_mx\_gmail.py y modifícalo para que muestre por pantalla las IPs de las máquinas designadas para recibir correos dirigidos a \*@gmail.com. Debes usar el campo exchange en lugar de address para obtener el nombre de la máquina, y usar de nuevo una consulta DNS sobre ese nombre para obtener la IP.

Intenta además que la lista de IPs salga ordenada en el orden en que el MTA tendría que probarlas (cada elemento de la respuesta, además del campo exchange tiene el campo preference que es la prioridad). Ejemplo de salida:

```
5 gmail-smtp-in.l.google.com. 108.177.119.27
10 alt1.gmail-smtp-in.l.google.com. 74.125.130.27
20 alt2.gmail-smtp-in.l.google.com. 108.177.125.26
30 alt3.gmail-smtp-in.l.google.com. 74.125.195.26
40 alt4.gmail-smtp-in.l.google.com. 108.177.104.26
```



IngenieriaServicios [SSH: 192.168.1.147] - dns\_mx\_gmail.py

```
1 import dns.resolver
2
3 # Dominio al que consultaremos los registros MX
4 dominio = "gmail.com"
5
6 print(f"Servidores MX para el dominio {dominio}:\n")
7
8 # Consulta de tipo MX
9 respuesta_mx = dns.resolver.resolve(dominio, "MX")
10
11 # Crear una lista con los resultados (preferencia, exchange)
12 mx_registros = []
13 for rdata in respuesta_mx:
14     prioridad = rdata.preference
15     servidor = str(rdata.exchange)
16     mx_registros.append((prioridad, servidor))
17
18 # Ordenar por prioridad
19 mx_registros.sort(key=lambda x: x[0])
20
21 # Para cada servidor MX, obtener su IP (registro A)
22 for prioridad, servidor in mx_registros:
23     try:
24         respuesta_a = dns.resolver.resolve(servidor, "A")
25         for ip in respuesta_a:
26             print(f"{prioridad}\t{servidor}\t{ip.address}")
27     except Exception as e:
28         print(f"{prioridad}\t{servidor}\tX No se pudo resolver IP ({e})")
29
```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 dns_mx_gmail.py
Servidores MX para el dominio gmail.com:

5      gmail-smtp-in.l.google.com.    64.233.167.26
10     alt1.gmail-smtp-in.l.google.com.    192.178.213.26
20     alt2.gmail-smtp-in.l.google.com.    142.250.147.27
30     alt3.gmail-smtp-in.l.google.com.    74.125.131.26
40     alt4.gmail-smtp-in.l.google.com.    142.250.4.27
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ █
```

## EJERCICIO 11 email\_envia\_crudo\_ssl.py

Copia el fichero `email_envia_crudo.py` a otro llamado `email_envia_crudo_ssl.py` sobre el que haremos modificaciones.

En este caso el servidor será `smtp.gmail.com` y el puerto `587`. Tras la conexión con el servidor usa el comando `EHLO` en lugar del comando `HELO`. Observa la respuesta. Además del código `250`, contiene información sobre los comandos extendidos que soporta el servidor. Si contiene `STARTTLS` quiere decir que soporta cifrado. Para poder usar el cifrado, tras el

comando EHLO hay que enviar el comando starttls. Si la respuesta es correcta (código 220) se debe empezar la negociación de la conexión segura. Se puede hacer de la siguiente forma:

```
context = ssl.create_default_context()  
sc = context.wrap_socket(s, server_hostname=server)
```

Para utilizar esta función es necesario import ssl al principio del programa. Fíjate que .wrap\_socket() devuelve un nuevo socket que sirve para enviar la información a través del canal seguro. El resto de comandos y respuestas se tratarán con este socket.

Tras la negociación debes enviar el comando auth login, y el servidor debe responder con el código 334. A continuación hay que enviar el nombre de usuario y la contraseña codificados en base 64 (debes importar el módulo base64). Tras el nombre de usuario el servidor responderá con el código 334 para indicar que espera la contraseña, y después, tras recibir y verificar ésta, responderá con el código 235.

Usa el siguiente código para leer el nombre de usuario y la contraseña desde teclado y para enviarla a través del socket seguro:

```
username = input("Usuario: ")
```

```
password = getpass.getpass("Contraseña: ")
```

```
sc.send(base64.b64encode(username.encode("ascii"))+b'\r\n')
```

*# Leer respuesta (esperamos 334)*

```
sc.send(base64.b64encode(password.encode("utf8"))+b'\r\n')
```

*# Leer respuesta (esperamos 235)*

Para poder usar la función getpass que no muestra los caracteres escritos por pantalla hay que incluir su módulo al principio del programa.

Envía el resto de los comandos y recibe la respuesta por el socket sc (el que está envuelto por ssl).

Si lo has hecho bien, el programa anterior te permite enviar correo (usando gmail como MTA) a cualquier dirección del mundo. Prueba a enviarte un correo a tí mismo/a (tanto a tu correo de gmail como al de Uniovi). Verifica que han llegado correctamente, aunque posiblemente habrás recibido el email en el apartado de Correo no deseado.

```

IngenieriaServicios [SSH: 192.168.1.147] - email_envia_crudo_bytes.py

1 import socket
2 import sys
3 from email.message import EmailMessage
4
5 # --- Función para recibir y verificar respuesta del servidor SMTP ---
6 def RecvReply(sock, expected_code):
7     reply = sock.recv(1024)
8     print("Servidor:", reply.decode().strip())
9
10    code = reply[:3]
11    if code != expected_code:
12        print(f"🔴 Error: se esperaba {expected_code.decode()}, pero se recibió {code.decode()}")
13        sys.exit(1)
14    return reply
15
16
17 # --- Datos de configuración ---
18 server = "relay.uniovi.es"
19 port = 25
20
21 fromaddr = "tu_usuario@uniovi.es"    # Cambia por tu dirección Uniovi
22 toaddr = "tu_usuario@uniovi.es"
23 subject = "Prueba SMTP con EmailMessage (Ejercicio 8)"
24 data = "Este mensaje se ha construido usando EmailMessage y enviado con sockets TCP crudos."
25
26 # --- Crear el mensaje con la librería email ---
27 msg = EmailMessage()
28 msg["From"] = fromaddr
29 msg["To"] = toaddr
30 msg["Subject"] = subject
31 msg.set_content(data)
32
33 # Convertir a bytes (añadiremos la línea final del punto más adelante)
34 msg_bytes = msg.as_bytes()
35
36 # --- Crear socket TCP ---
37 print(f"Conectando a {server}:{port} ...")
38 sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
39 sock.connect((server, port))
40
41 # --- Diálogo SMTP ---
42 RecvReply(sock, b"220")                      # Banner inicial
43 sock.send(b"HELO uniovi.es\r\n")
44 RecvReply(sock, b"250")
45
46 sock.send(f"MAIL FROM:<{fromaddr}>\r\n".encode())
47 RecvReply(sock, b"250")
48
49 sock.send(f"RCPT TO:<{toaddr}>\r\n".encode())
50 RecvReply(sock, b"250")
51
52 sock.send(b"DATA\r\n")
53 RecvReply(sock, b"354")
54
55 # Enviar cuerpo del mensaje y secuencia final
56 sock.send(msg_bytes)
57 sock.send(b"\r\n.\r\n")
58 RecvReply(sock, b"250")
59
60 sock.send(b"QUIT\r\n")
61 RecvReply(sock, b"221")
62
63 sock.close()
64 print("Mensaje enviado correctamente (si el servidor lo permite).")
65
66

```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 email_envia_crudo_ssl.py
Dirección remitente (tu cuenta Gmail): alejandromedinafernandez@gmail.com
Dirección destino: alejandromedinafernandez@gmail.com
Conectando a smtp.gmail.com:587 ...
Servidor: 220 smtp.gmail.com ESMTP 5blf17bl1804bl-4777d9447eesm4481475e9.16 - gsmtp
Servidor: 250-smtp.gmail.com at your service, [85.152.106.33]
250-SIZE 35882577
250-8BITMIME
250-STARTTLS
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8
Servidor: 220 2.0.0 Ready to start TLS
Conexión segura establecida (TLS).
Servidor: 250-smtp.gmail.com at your service, [85.152.106.33]
250-SIZE 35882577
250-8BITMIME
250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH
250-ENHANCEDSTATUSCODES
250-PIPELINING
250-CHUNKING
250 SMTPUTF8
Servidor: 334 VXNlcm5hbWU6
Usuario Gmail (sin @gmail.com si ya lo incluye): alejandromedinafernandez
Contraseña o clave de aplicación:
Servidor: 334 UGFzc3dvcmQ6
Servidor: 235 2.7.0 Accepted
Autenticación correcta.
Servidor: 250 2.1.0 OK 5blf17bl1804bl-4777d9447eesm4481475e9.16 - gsmtp
Servidor: 250 2.1.5 OK 5blf17bl1804bl-4777d9447eesm4481475e9.16 - gsmtp
Servidor: 354 Go ahead 5blf17bl1804bl-4777d9447eesm4481475e9.16 - gsmtp
Servidor: 250 2.0.0 OK 1762727679 5blf17bl1804bl-4777d9447eesm4481475e9.16 - gsmtp
Servidor: 221 2.0.0 closing connection 5blf17bl1804bl-4777d9447eesm4481475e9.16 - gsmtp
Mensaje enviado correctamente.
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$
```

## Prueba SMTP con SSL/TLS desde Python (Ejercicio 11)



Recibidos x

A

alejandromedinafern... 23:34 (hace 0 minutos)  
para mí ▾



Este mensaje se ha enviado de forma segura usando STARTTLS.

Saludos desde Ingeniería de Servicios.

Responder

Reenviar



## EJERCICIO 12 email\_smtp.py

Crea un nuevo programa llamado `email_smtp.py` que use `smtplib` para enviar un correo a través de tu cuenta de gmail. Consulta [la documentación de este módulo](#). Se comienza por instanciar un objeto de la clase `SMTP`:

```
s = smtplib.SMTP(server, port)
```

A continuación activa la depuración del protocolo para poder ver los comandos y las respuestas:

```
s.set_debuglevel(1)
```

El siguiente paso es usar la función `starttls()` para negociar el canal seguro y usar la función `login()` para enviar el usuario y la contraseña. Haz que pida ambos por teclado igual que en los ejemplos anteriores.

Por último se envía el mensaje (que habrá sido construido con `email.message` como vimos en `email_crear_mensaje.py`) usando `sendmail()` y después se cierra la conexión. El último parámetro de `sendmail()` es el mensaje en sí, y se puede usar la forma binario que obtuvimos de `mensaje.as_bytes()`, en este caso sin necesidad de añadir el punto final.

```
● ● ● Ingeniería de Servicios [SSH: 192.168.1.147] - email_smtp.py

1 import smtplib
2 import ssl
3 import getpass
4 from email.message import EmailMessage
5
6 # --- Configuración del servidor SMTP de Gmail ---
7 server = "smtp.gmail.com"
8 port = 587 # STARTTLS
9
10 # --- Leer usuario y contraseña desde teclado ---
11 fromaddr = input("Tu cuenta Gmail: ")
12 password = getpass.getpass("Contraseña o clave de aplicación: ")
13
14 # --- Destinatario y contenido ---
15 toaddr = input("Correo de destino: ")
16 subject = "Prueba de envío con smtplib (Ejercicio 12)"
17 body = "Mensaje enviado correctamente usando smtplib, STARTTLS y autenticación segura."
18
19 # --- Crear el mensaje con email.message ---
20 msg = EmailMessage()
21 msg["From"] = fromaddr
22 msg["To"] = toaddr
23 msg["Subject"] = subject
24 msg.set_content(body)
25
26 # --- Crear conexión SMTP ---
27 print(f"Conectando a {server}:{port} ...")
28 context = ssl.create_default_context()
29 s = smtplib.SMTP(server, port)
30 s.set_debuglevel(1) # Muestra comandos SMTP y respuestas
31
32 # --- Iniciar comunicación y cifrado ---
33 s.ehlo()
34 s.starttls(context=context)
35 s.ehlo()
36
37 # --- Autenticación ---
38 try:
39     s.login(fromaddr, password)
40     print("Autenticación correcta.")
41 except smtplib.SMTPAuthenticationError as e:
42     print("Error de autenticación:", e)
43     s.quit()
44     exit()
45
46 # --- Enviar el mensaje ---
47 s.sendmail(fromaddr, toaddr, msg.as_string())
48 print(" Mensaje enviado correctamente.")
49
50 # --- Cerrar conexión ---
51 s.quit()
52 print("Conexión cerrada.")
```

```
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ python3 email_smtp.py
Tu cuenta Gmail: alejandromedinafernandez@gmail.com
Contraseña o clave de aplicación:
Correo de destino: alejandromedinafernandez@gmail.com
Conectando a smtp.gmail.com:587 ...
send: 'ehlo [192.168.1.147]\r\n'
reply: b'250-smtp.gmail.com at your service, [85.152.106.33]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-STARTTLS\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250-SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [85.152.106.33]\nSIZE 35882577\n8BITMIME\nSTARTTLS\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'
send: 'STARTTLS\r\n'
reply: b'220 2.0.0 Ready to start TLS\r\n'
reply: retcode (220); Msg: b'2.0.0 Ready to start TLS'
send: 'ehlo [192.168.1.147]\r\n'
reply: b'250-smtp.gmail.com at your service, [85.152.106.33]\r\n'
reply: b'250-SIZE 35882577\r\n'
reply: b'250-8BITMIME\r\n'
reply: b'250-AUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\r\n'
reply: b'250-ENHANCEDSTATUSCODES\r\n'
reply: b'250-PIPELINING\r\n'
reply: b'250-CHUNKING\r\n'
reply: b'250-SMTPUTF8\r\n'
reply: retcode (250); Msg: b'smtp.gmail.com at your service, [85.152.106.33]\nSIZE 35882577\n8BITMIME\nAUTH LOGIN PLAIN XOAUTH2 PLAIN-CLIENTTOKEN OAUTHBEARER XOAUTH\nENHANCEDSTATUSCODES\nPIPELINING\nCHUNKING\nSMTPUTF8'
send: 'AUTH PLAIN AGFsZWphbmRyb2l1ZGluYWZ1cm5hbmrlepaZ21haWwuY29tAGR0ZHRzcmJsaGF2dmt1Ynk=\r\n'
reply: b'235 2.7.0 Accepted\r\n'
reply: retcode (235); Msg: b'2.7.0 Accepted'
Autenticación correcta.
send: 'mail FROM:<alejandromedinafernandez@gmail.com> size=351\r\n'
reply: b'250 2.1.0 OK ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.0 OK ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp'
send: 'rcpt TO:<alejandromedinafernandez@gmail.com>\r\n'
reply: b'250 2.1.5 OK ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.1.5 OK ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp'
send: 'data\r\n'
reply: b'354 Go ahead ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp\r\n'
reply: retcode (354); Msg: b'Go ahead ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp'
data: (354, b'Go ahead ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp')
send: b'From: alejandromedinafernandez@gmail.com\r\nTo: alejandromedinafernandez@gmail.com\r\nSubject: Prueba de ==?utf-8?q?env=C3=ADo?= con smtplib (Ejercicio 12)\r\nContent-Type: text/plain; charset="utf-8"\r\nContent-Transfer-Encoding: quoted-printable\r\nMIME-Version: 1.0\r\n\r\nMensaje enviado correctamente usando smtplib, STARTTLS y autenticaci=C3=B3n s=vr\negura.\r\n\r\n'
reply: b'250 2.0.0 OK 1762727972 ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp\r\n'
reply: retcode (250); Msg: b'2.0.0 OK 1762727972 ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp'
data: (250, b'2.0.0 OK 1762727972 ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp')
Mensaje enviado correctamente.
send: 'quit\r\n'
reply: b'221 2.0.0 closing connection ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp\r\n'
reply: retcode (221); Msg: b'2.0.0 closing connection ffacd0b85a97d-42abe63e13lsm18805673f8f.20 - gsmtp'
Conexión cerrada.
(dnsenv) uo288967@is02:~/IngenieriaServicios/Practica4$ 
```

## Prueba de envío con smtplib (Ejercicio 12) Recibidos x



**alejandromedinafernandez**... dom, 9 nov, 23:39 (hace 31 minutos)

para mí ▾



Mensaje enviado correctamente usando smtplib, STARTTLS y autenticación segura.

En el Ejercicio 12 se utilizó el módulo `smtplib` para enviar un correo a través de Gmail con conexión segura STARTTLS.

Tras autenticarse correctamente con una clave de aplicación, el servidor respondió con 235 2.7.0 Accepted, confirmando la autenticación, y posteriormente 250 2.0.0 OK, indicando que el mensaje fue aceptado para su entrega.

El flujo se cerró con 221 2.0.0 closing connection.

Por tanto, el programa cumple todos los objetivos del ejercicio y permite enviar correo de forma segura usando Python.

## EJERCICIO 13 email\_pop3\_recibe\_crudo.py

Vamos a realizar un programa en python que, usando el protocolo POP3, reciba correo electrónico de una cuenta de gmail. En primer lugar entra a través de la aplicación web de gmail y vete a la configuración (ícono de engranaje en la parte superior derecha y opción “Ver todos los ajustes”). Sobre la opción “Reenvío y correo POP/IMAP”, habilita POP para la recepción de mensajes. Ahora utiliza algunos de los programas anteriores para enviar varios correos a esa cuenta de correo. Con la configuración establecida de POP3 solo se accederá a los nuevos correos enviados.

Ahora escribe un programa en python llamado email\_pop3\_recibe\_crudo.py que reciba el correo electrónico. Para ello sigue los siguientes pasos:

1. Define una función denominada RecvReply con un parámetro: un socket. Esta función debe recibir hasta un máximo de 1024 bytes e imprimir por pantalla lo recibido, para depuración. A continuación, extraerá los tres primeros bytes recibidos y verificará si coinciden con el código de éxito: b"+OK". En caso de que no coincidan imprimirá por pantalla un error y terminará el programa (sys.exit()).
2. Dentro del programa principal define las siguientes variables: server (dirección del servidor POP3) y port (puerto del servidor). Configura el servidor como pop.gmail.com y el puerto como 995.
3. Añade el código para leer el nombre de usuario y la contraseña por teclado igual que en ejemplos anteriores.
4. Crea la conexión del socket y establece el canal seguro:
  5. s = socket.socket()
  6. s.connect((server, port))
  7. context = ssl.create\_default\_context()

```
sc = context.wrap_socket(s, server_hostname=server)
```

Tras la conexión con éxito el servidor enviará un primer mensaje que debes leer con RecvReply()

8. Envía el comando USER y PASS para entrar en sesión (leyendo la respuesta del servidor tras cada uno).
9. Envía el comando STAT y verifica cuantos correos hay disponibles.
10. Lee el contenido del primer correo. Implementa un bucle para recibir el correo completo. Recuerda que debe terminar en \r\n.\r\n.

## 11. Extrae de las cabeceras el campo “Subject” y el campo “From” e imprimelo por pantalla.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

/usr/bin/python3 /home/uo288967/IngenieriaServicios/Practica4/email_pop3_recibe_crudo.py
● uo288967@is02:~/IngenieriaServicios$ /usr/bin/python3 /home/uo288967/IngenieriaServicios/Practica4/email_pop3_recibe_crudo.py
Conectando a pop.gmail.com:995 ...
Servidor: +OK Gpop ready for requests from 85.152.106.33 5c2dc8763939e-486175db0bam261210537122
Usuario Gmail (con @gmail.com): ALEJANDROMEDINAfernandez@gmail.com
Clave de aplicación (la de 16 letras):
Servidor: +OK send PASS
Servidor: +OK Welcome.
Servidor: +OK 439 19587978
Hay 439 mensajes disponibles.

--- Cabeceras del mensaje ---
From: Alejandro Medina <alejandromedinafernandez@gmail.com>
Subject: Filosofía chuli
Servidor: +OK Farewell.
Conexión cerrada correctamente.
✉ uo288967@is02:~/IngenieriaServicios$
```

```
--- Mensaje 4 de 5 (ID real: 435) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0 with SMTP id ga23csp3271411pb;
    Sun, 11 Apr 2021 08:27:16 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJzh81RqVmkrrySuuu2swuRxNmiG0Eq5c1kGkdNKwBkUP4kpn5kbpfJUpeuQQLJyYYiKBh0I
X-Received: by 2002:a25:bec8:: with SMTP id k8mr12330456yb.m.186.1618154836046;
    Sun, 11 Apr 2021 08:27:16 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1618154836; cv=none;
    d=google.com; s=arc-20160816;
    b=BmB9nhgZ9895YuoAc+E7s+DvYj0IM0gu8++pCbEm4PqKwJALcl+5fL2tyztzF3b5rr
    mx8PrE+38m9RgcNJE9PSYwPdiPrwRXE6jaqGHRs8TyOAJaXIx95oA+iSwXpgyDTQ1GW
... (contenido truncado)

--- Mensaje 5 de 5 (ID real: 434) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0 with SMTP id ga23csp819274pb;
    Tue, 6 Apr 2021 11:50:36 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJwc0Qe3WF1YmhZ76TsEv2YKcUz2MzsXbsz3exYUCYhe/t0r06uHA6xCEzm06aRQSxFsy0F9
X-Received: by 2002:a63:c807:: with SMTP id z7mr27951580pgg.363.1617735036555;
    Tue, 06 Apr 2021 11:50:36 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1617735036; cv=none;
    d=google.com; s=arc-20160816;
    b=S1+cBKeOCE+6nX9dFftxLsY21R6R3pj6ypNqW7157PTHnfei2rnccipTENWN5e0Nlc
    lYWLwzp/mMmGTRb0//IdlN0FCxjF+i6KZdsDK15+rQmocf6GgfQHuJGloeDNcdP6zRKQ
... (contenido truncado)
```

## EJERCICIO 14 email\_pop3\_recibe\_crudo2.py

Copia el ejercicio anterior al fichero email\_pop3\_recibe\_crudo2.py. Modifica el programa para que se muestre el “Subject” y el “From” de todos los mensajes disponibles.

```

● uo288967@is02:~/IngenieriaServicios$ /usr/bin/python3 /home/uo288967/IngenieriaServicios/Practica4/email_pop3_recibe_crudo2.py
Conectando a pop.gmail.com:995 ...
Servidor: +OK Gpop ready for requests from 85.152.106.33 e09e53db590de-87f01e43e1fm198708381d85
Usuario Gmail (con @gmail.com): alejandromedinafernandez@gmail.com
Clave de aplicación (la de 16 letras):
Servidor: +OK send PASS
Servidor: +OK Welcome.
Servidor: +OK 439 19463403
Hay 439 mensajes disponibles.

--- Mensaje 5 de 5 (ID real: 439) ---
From: =?UTF-8?Q?=C3=81_=28v=C3=ADA_Twitter=29?= <notify@twitter.com>
Subject: =?UTF-8?Q?=C3=81_(@angelaprados=5F)_te_envi=C3=B3_?=

--- Mensaje 4 de 5 (ID real: 438) ---
From: =?UTF-8?Q?=C3=81_=28v=C3=ADA_Twitter=29?= <notify@twitter.com>
Subject: =?UTF-8?Q?=C3=81_(@angelaprados=5F)_te_envi=C3=B3_?=

--- Mensaje 3 de 5 (ID real: 437) ---
From: =?UTF-8?Q?=C3=81_=28v=C3=ADA_Twitter=29?= <notify@twitter.com>
Subject: =?UTF-8?Q?=C3=81_(@angelaprados=5F)_te_envi=C3=B3_?=

--- Mensaje 2 de 5 (ID real: 436) ---
From: =?UTF-8?Q?=C3=81_=28v=C3=ADA_Twitter=29?= <notify@twitter.com>
Subject: =?UTF-8?Q?=C3=81_(@angelaprados=5F)_te_envi=C3=B3_?=

--- Mensaje 1 de 5 (ID real: 435) ---
From: =?UTF-8?Q?=C3=81_=28v=C3=ADA_Twitter=29?= <notify@twitter.com>
Subject: =?UTF-8?Q?=C3=81_(@angelaprados=5F)_te_envi=C3=B3_?=

Servidor: +OK Farewell.
Conexión cerrada correctamente.
❖ uo288967@is02:~/IngenieriaServicios$ []

```

En este ejercicio se ha decidido mostrar los 5 primeros, ya que era un correo que tenía uso y se sobresaturaba.

## EJERCICIO 15

### email\_pop3\_poplib.py

Crea un nuevo programa para recibir correo, en este caso usando la librería poplib. Llama al programa email\_pop3\_poplib.py. Instancia el objeto de la siguiente forma y activa la depuración del protocolo:

```

pop3_mail = poplib.POP3_SSL(server)

pop3_mail.set_debuglevel(2);

```

A continuación envía el usuario y la contraseña con las funciones user() y pass\_(). Recibe el contenido del primer mensaje y muéstralolo por pantalla, tal como poplib te lo devuelve. La función para obtener un mensaje es retr(). Esta función retorna una tupla con 3 elementos: el número de mensaje, el contenido del mensaje, y el tamaño ocupado por el mismo.

El contenido del mensaje está almacenado en una lista, siendo cada elemento de la lista una línea del mensaje (incluye cabeceras), de la que se ha eliminado el \r\n al final de cada línea. Puedes encontrar más información sobre las funciones disponibles en la [documentación de poplib](#).

```

● uo288967@is02:~/IngenieriaServicios$ /usr/bin/python3 /home/uo288967/IngenieriaServicios/Practica4/email_pop3_poplib.py
Conectando a pop.gmail.com:995 ...
Usuario Gmail (con @gmail.com): alejandromedinafernandez@gmail.com
Clave de aplicación (la de 16 letras):

Número total de mensajes: 438, Tamaño total: 19470060 bytes

--- Mensaje 1 de 5 (ID real: 438) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0:0 with SMTP id ga23csp4193529pb;
      Mon, 12 Apr 2021 11:00:53 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJxeCh9dYaiFY+kpdNozRtf4uk3QRxbWH3/gAUzvvMZDyvXev9cqzaEdZ5iQuvbL9Ur0VkEk
X-Received: by 2002:a17:90b:397:0:0:0:0 with SMTP id gg12mr397662pb.184.1618250452817;
      Mon, 12 Apr 2021 11:00:52 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1618250452; cv=none;
      d=google.com; s=arc-20160816;
      b=q2pkau88PZkmYxlnyPqyC0F7K1fgXGypdPp1mBpKuHkKZbuD8XBBepekrCuZmbi5
      ZeoasIUodofI2W6x8CZA8m4c8CpNL1VAScujtLeQwK6M2KHcrr9T1GZAxY1xSNMGuCTo
... (contenido truncado)

--- Mensaje 2 de 5 (ID real: 437) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0:0 with SMTP id ga23csp3860664pb;
      Mon, 12 Apr 2021 03:43:53 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJygMiqiIAv0f3o1mvDAXNmae/2ZSIUj8NRXQYLmyEzDzmxOzEh/T7dkFbMYabMwsk3EJXJC
X-Received: by 2002:a25:7c47:: with SMTP id x68mr36801730ybc.358.1618224233631;
      Mon, 12 Apr 2021 03:43:53 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1618224233; cv=none;
      d=google.com; s=arc-20160816;
      b=Itwcpk8ly19myeljuUDUCwYowAYtNxP5uGwuvvgQBDhXJWvr9fMDC82gvj6qNeIMPI
      sd2aVee0fTaafn2HQYGl4noIcwuaAfSyDTC83MpMPFpgpDq+oPLfxkj/wmtzAAhtxmFZ
... (contenido truncado)

--- Mensaje 3 de 5 (ID real: 436) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0:0 with SMTP id ga23csp3271478pb;
      Sun, 11 Apr 2021 08:27:24 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJxiHkxrXGTnhbY+VeI9V3tS1dDiNZg+c7qFwZf6DxJ1eqEVFcXMF7hsICY+6tDnNqIPzUq
X-Received: by 2002:a25:6a88:: with SMTP id f130mr31614296ybc.234.1618154844388;
      Sun, 11 Apr 2021 08:27:24 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1618154844; cv=none;
      d=google.com; s=arc-20160816;
      b=tqe9BjznChZMxnxxThc0Vku0zCDdmTQ636XE1BgxRRWJAJnw9WvvAs6+eSzRVDZ
      lU/opQu5ZUBNzQFGNh9GxxbLrj2n+A3j9RagYHNy3DAU/Lzp01HKtbC9//p6PcWzcwXZ
... (contenido truncado)

--- Mensaje 4 de 5 (ID real: 435) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0:0 with SMTP id ga23csp3271411pb;
      Sun, 11 Apr 2021 08:27:16 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJzh81RqYwkrriySuuu2swuRxNmig0Eq5c1kGkdNKw3KUP4kpn5kbpfJUpeuQQLJyYYikBh0I

```

## EJERCICIO 16 (opcional)

Crea un programa `email_pop3_poplib2.py` que haga uso de la función anterior para mostrar un breve resumen de cada mensaje recibido mediante `poplib`. Ten en cuenta que, ya que `poplib` te devuelve el mensaje en forma de lista de líneas, debes concatenarlas todas con `b"\r\n".join()` para convertirlo en el array de bytes que `email.message_from_bytes()` espera.

No nos salía el ejercicio, hemos estado buscando soluciones y hemos encontrado esto:

**IMPORTANTE:** Desde 2024, Google ha restringido el acceso POP3 con autenticación básica incluso para cuentas personales con claves de aplicación.

Solo permite POP3 a clientes oficiales (Outlook, Thunderbird, Apple Mail) o a apps que usen OAuth2, no a scripts directos de Python con `poplib` o `sockets`.

Por eso no nos salía correctamente, aunque este bien implementado.

```

--- Mensaje 4 de 5 (ID real: 435) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0 with SMTP id ga23csp3271411pb;
Sun, 11 Apr 2021 08:27:16 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJzh81RqYkrrySLuu2swuRxNmiG0Eq5c1kGkdNkW3KUP4kpn5kbpfJUpeuQQLJyYYikBh0I
X-Received: by 2002:a25:bec8:: with SMTP id k8mr12330456ybm.186.1618154836046;
Sun, 11 Apr 2021 08:27:16 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1618154836; cv=none;
d=google.com; s=arc-20160816;
b=Bm89hgZ9895YuAc+E7s+DvYj0IM0gu8+pCbEm4PqK0ALc1+5fL2tyztF3b5rr
mx8PpE+38m9RGcNTE9PSYwPdiPrnRXE6jaqGRs8TyOAJaXlx95oA+i5MsXpgy0TQlGW
... (contenido truncado)

--- Mensaje 5 de 5 (ID real: 434) ---
--- Contenido parcial del mensaje ---
Delivered-To: alejandromedinafernandez@gmail.com
Received: by 2002:a17:90b:397:0:0:0 with SMTP id ga23csp819274pb;
Tue, 6 Apr 2021 11:50:36 -0700 (PDT)
X-Google-Smtp-Source: ABdhPJwc8Qe3MF1YmhZ76TsEv2YKcUz2MzsXbsz3exYUCYhe/t0r06uHA6xCEm06aRQ5xFsyOF9
X-Received: by 2002:a63:c807:: with SMTP id z7mr27951580pgg.363.1617735036555;
Tue, 06 Apr 2021 11:50:36 -0700 (PDT)
ARC-Seal: i=1; a=rsa-sha256; t=1617735036; cv=none;
d=google.com; s=arc-20160816;
b=S1+cBKeoCE+6nX9dfFtxNsY21R6R3pj6ypNgW71S7PTHnfe12rnccipTEMNNSe0Nlc
1YwLwzp/mMgTRb//IdlN0FCxjF+i6KZdsDKl5+rQnacf6Ggf0HuJGloeDNcdP6zRKQ
... (contenido truncado)

```

## **SERVICIOS MULTIMEDIA**

### **PRACTICA 5**

En esta práctica exploramos el funcionamiento de los servicios multimedia, enfocándonos en la distribución de vídeo mediante tecnologías de streaming. Aprendimos a configurar y gestionar el servidor **Wowza Streaming Engine** para servir contenidos tanto bajo demanda (VOD) como en directo (Live). Durante la sesión, trabajamos con diversos protocolos como HLS, MPEG-DASH, RTMP y RTSP, analizando sus diferencias y compatibilidades. Además, implementamos el streaming con bitrate adaptativo (ABR) mediante el uso de ficheros SMIL para ajustar la calidad del vídeo según las condiciones de la red. Finalmente, integramos estos servicios en páginas web utilizando la biblioteca **Video.js**.

**En la carpeta dedicada a la sesión de hoy, crea un subdirectorio llamado html y después otro html/video5/.**

**Descarga y descomprime dentro de html/video5/ el fichero <http://www.atc.uniovi.es/grado/4is/VideosHTML5.zip>. Contiene el mismo video con distintos formatos. Se trata del video [Big Buck Bunny](#), utilizado como el “Hola mundo” en temas de video. No pongas estos archivos bajo control de versiones (de hecho, añade esta carpeta a .gitignore) pues ocupan mucho y no es necesario tenerlos en el repositorio ya que siempre pueden bajarse de nuevo.**

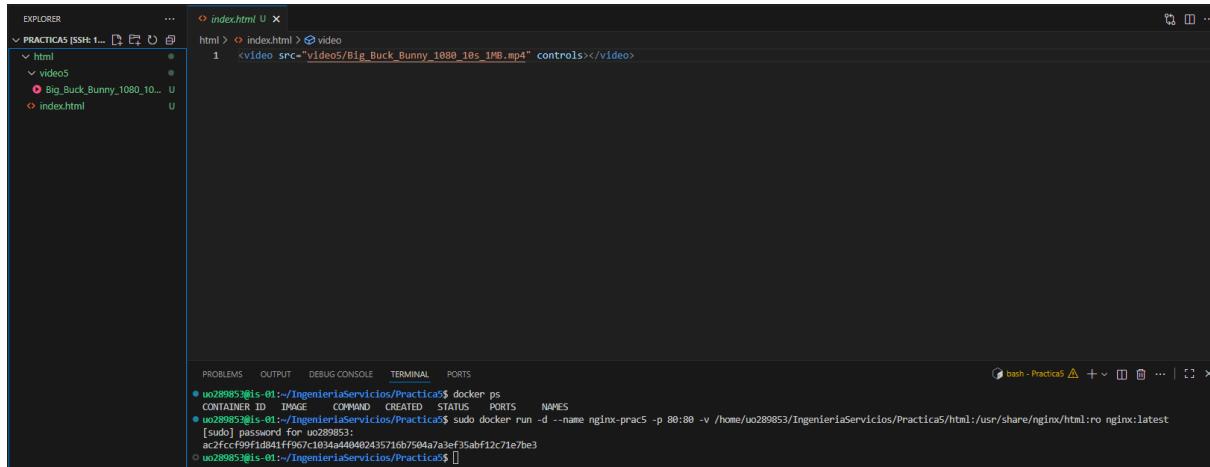
**Crea una página web en esa carpeta (es decir, un fichero .html) e incrusta uno de los videos. Este fichero sí irá bajo control de versiones.**

**Lanza la imagen docker nginx como ya hemos hecho en otras sesiones, montando la carpeta html en la ruta apropiada del contenedor. No necesitamos en esta ocasión montar otro volumen para la configuración de nginx, puesto que la configuración por defecto es adecuada.**

**Abre la página con el navegador Firefox (o Chrome) y verifica que puedes visualizar el video.**

Mientras lo estás reproduciendo abre las herramientas de desarrollador web y observa los mensajes HTTP. Avanza el video y verás nuevos mensajes (a menos que el vídeo ya haya sido descargado por completo y esté en la caché del navegador, en cuyo caso no necesita solicitarlo al servidor).

Fíjate en la cabecera HTTP que se utiliza para acceder a una parte no descargada del fichero.

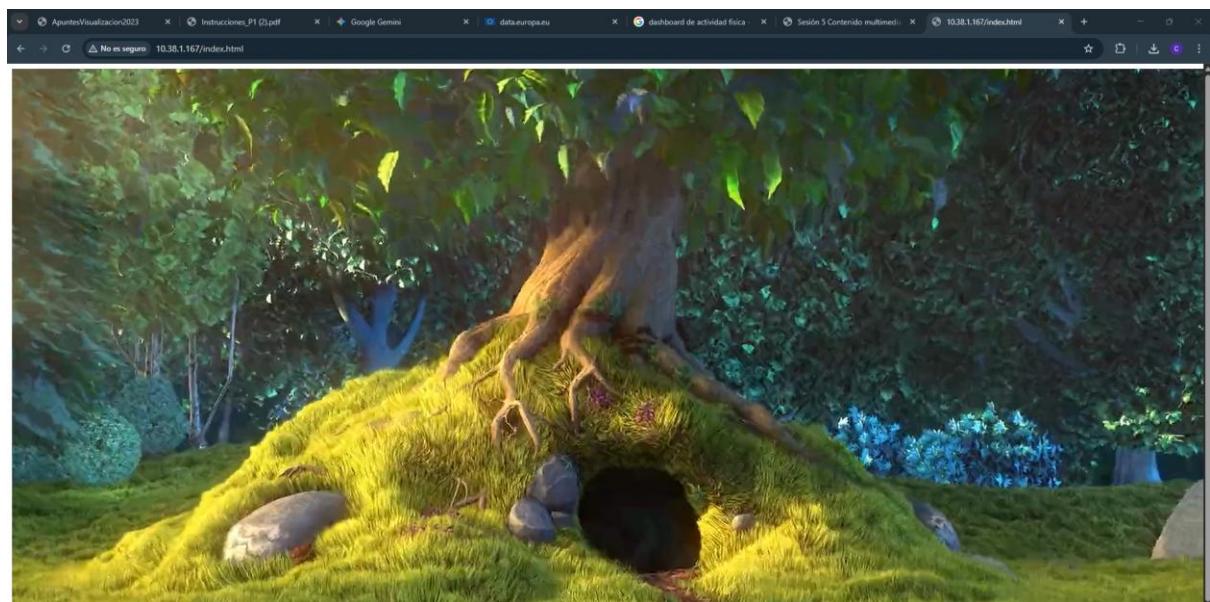


The screenshot shows a terminal window with the following logs:

```
● uo289853@ls-01:~/IngenieriaServicios/Practicas$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
● uo289853@ls-01:~/IngenieriaServicios/Practicas$ sudo docker run -d --name nginx-prac5 -p 80:80 -v /home/uo289853/IngenieriaServicios/Practicas/html:/usr/share/nginx/html:ro nginx:latest
[sudo] password for uo289853:
ac2fccf99f1d841ff967c1034a40402435716b7504a7a3ef35abf12c71e7be3
● uo289853@ls-01:~/IngenieriaServicios/Practicas$
```

Below the terminal is a code editor showing the content of index.html:

```
<html>
  <head>
    <title>index.html</title>
  </head>
  <body>
    <video src="video5/Big_Buck_Bunny_1080_10s_1MB.mp4" controls></video>
  </body>
</html>
```

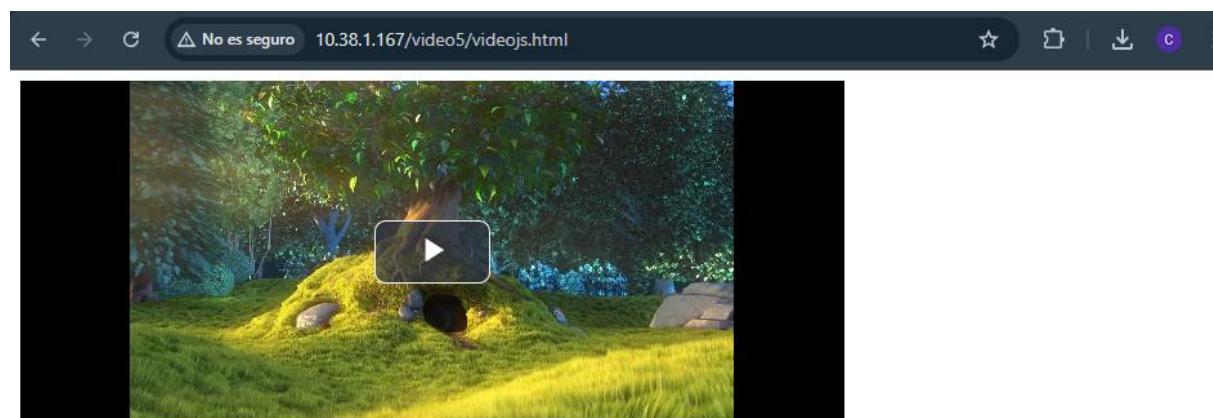


Crea un nuevo fichero en html/video5 llamado videojs.html

Adapta el [código HTML de ejemplo](#) que aparece en la página de video.js para que reproduzca el vídeo “Big Buck Bunny” que has bajado antes. Ese código contiene enlaces a través del tag `<link>` a direcciones públicas de las que el navegador obtendrá el javascript necesario, por tanto *no necesitas instalar nada* en tu ordenador, y las únicas modificaciones son para señalar al vídeo que quieras reproducir.

Verifica su funcionamiento.

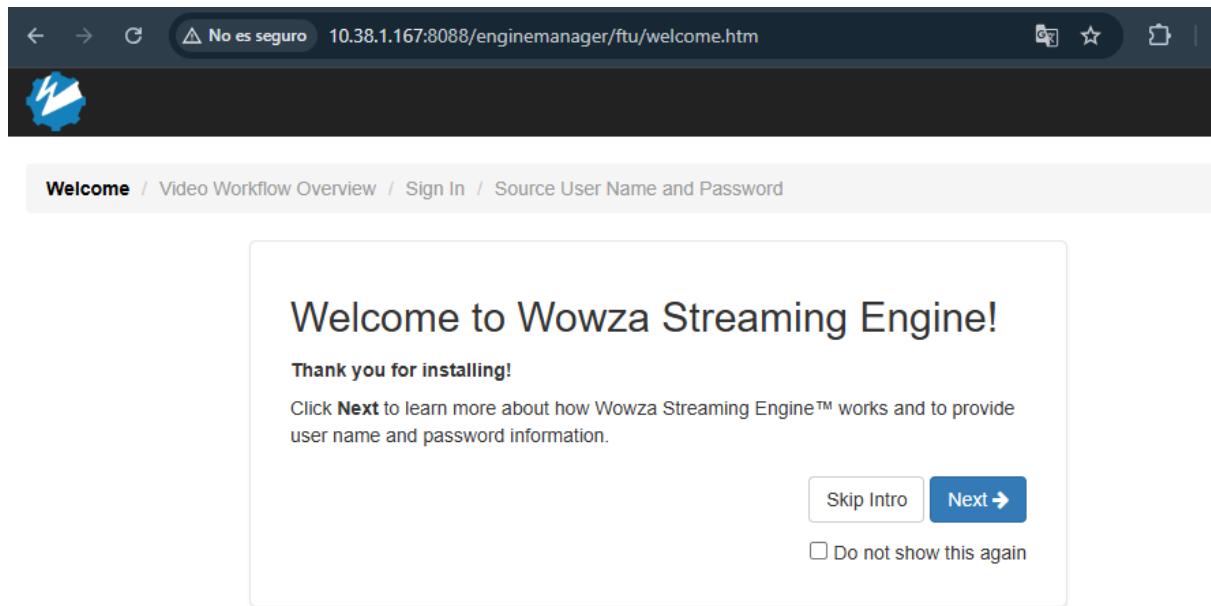
```
1 <head>
2   <link href="https://vjs.zencdn.net/8.23.4/video-js.css" rel="stylesheet" />
3
4   <!-- If you'd like to support IE8 (for Video.js versions prior to v7) -->
5   <!-- <script src="https://vjs.zencdn.net/ie8/1.1.2/videojs-ie8.min.js"></script> -->
6 </head>
7
8 <body>
9   <video
10    id="my-video"
11    class="video-js"
12    controls
13    preload="auto"
14    width="640"
15    height="264"
16    data-setup="{}"
17   >
18   <source src="Big_Buck_Bunny_1080_10s_1MB.mp4" type="video/mp4" />
19   <p class="vjs-no-js">
20     To view this video please enable JavaScript, and consider upgrading to a
21     web browser that
22     <a href="https://videojs.com/html5-video-support/" target="_blank"
23       >supports HTML5 video</a>
24   >
25   </p>
26 </video>
27
28 <script src="https://vjs.zencdn.net/8.23.4/video.min.js"></script>
29 </body>
```



Lanza el contenedor con el comando anterior. Comprueba con docker ps que está en ejecución. Puedes ver su log con docker logs wowza

Conecta un navegador a <http://ip.de.tu.ubuntu:8088>

Debe responderle Wowza. Tras ver (o saltarte) la intro llegarás a un punto en que te pedirá nombre y clave. Introduce wowza para ambos y verás la interfaz administrativa de este servidor.



1. Junto a la carpeta html que teníamos de antes, crea otra llamada wowza
2. Entra en la carpeta wowza y ejecuta los comandos siguientes:
3. `$ docker cp wowza:/usr/local/WowzaStreamingEngine/applications .`
4. `$ docker cp wowza:/usr/local/WowzaStreamingEngine/conf .`
5. `$ mkdir content`
6. `$ touch applications/vod/.vacio applications/live/.vacio content/.vacio`

`$ git add .`

Eso crea tres carpetas llamadas applications, conf y content. Las dos primeras se copian del contenedor, la tercera la creamos vacía. El comando touch es para crear un algunos ficheros en las carpetas vacías ya que de lo contrario git no las pondría bajo control de versiones con el comando git add . final.

```

uo288967@is02:~/IngenieriaServicios/Practica5$ mkdir wowza
uo288967@is02:~/IngenieriaServicios/Practica5$ cd wowza/
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ docker cp wowza:/usr/local/
WowzaStreamingEngine/applications .
Successfully copied 2.56kB to /home/uo288967/IngenieriaServicios/Practica5-wowza
/.
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ docker cp wowza:/usr/local/WowzaStreamingEngine/conf .
Successfully copied 97.8kB to /home/uo288967/IngenieriaServicios/Practica5-wowza..
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ mkdir content
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ touch applications/vod/.vacio applications/live/.vacio content/ .
vacío
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ git add .
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ ls
applications conf content
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ docker stop wowza
wowza
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ docker run -it --rm -d --name wowza \
-p 1935:1935 -p 8086:8086 -p 8087:8087 -p 8088:8088 \
-e WSE_MGR_USER=wowza \
-e WSE_MGR_PASS=clavesecreta \
-v $(pwd)/applications:/usr/local/WowzaStreamingEngine/applications \
-v $(pwd)/conf:/usr/local/WowzaStreamingEngine/conf \
-v $(pwd)/content:/usr/local/WowzaStreamingEngine/content \
--entrypoint /sbin/entrypoint.sh \
-wowzamedia/wowza-streaming-engine-linux:4.8.17
ldl9d9b202f782dd7794b54f95983eb117b1210880554689dc3a4b6855396da0
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ 

```

The screenshot shows the Wowza Streaming Engine management interface. At the top, there's a navigation bar with links for Home, Server, Applications, and a user icon. Below that is a main title "Welcome to Wowza Streaming Engine!" followed by a subtitle "for Docker 4.8.17+1 build20211216162410 License". Under "Status", there's a "Connections" section with a chart showing 1 connection from 03:30 pm to 04:00 pm. In the "Usage" section, there are four horizontal bars: "Wowza CPU" (orange, ~12%), "Wowza Heap" (blue, ~5%), "Total Memory" (yellow, ~95%), and "Total Disk" (brown, ~45%).

## EJERCICIO 1: VÍDEO BAJO DEMANDA

Entra en la carpeta applications y crea un subdirectorio que se llame videodemanda.

Ahora entra en la carpeta llamada conf. Crea otra carpeta denominada videodemanda (se debe llamar igual). En esta carpeta se debe almacenar el fichero de configuración que se debe llamar Application.xml.

Para evitar crear un fichero de configuración desde cero vamos a copiar un ejemplo que ya viene hecho. Se encuentra en la carpeta de Wowza, dentro de conf/vod/Application.xml. Copia este fichero al directorio de configuración de la nueva aplicación.

Edita el fichero y observa su contenido. Fíjate en concreto que la etiqueta StreamType que sirve para indicar el tipo de stream. Wowza soporta distintas opciones, pero sólo vamos a trabajar con dos: default para servir fichero en disco, y live para servir video en directo.

No hace falta que modifiques nada en el fichero.

El contenido multimedia servido por Wowza se almacena en la carpeta content. Copia el fichero big\_buck\_bunny.mp4 de antes a esta carpeta, llamándolo sample.mp4 (si te da errores de permisos verifica que el propietario de la carpeta content es tu usuario, y no root, si es necesario usa chown tu\_usuario:tu\_usuario content para cambiarlo, desde la carpeta wowza).

Detén el contenedor wowza y lánzalo de nuevo, con el mismo comando que la última vez (es necesario para que durante el arranque el servidor Wowza descubra la nueva aplicación que has creado, no sería necesario si hubiéramos creado la aplicación a través de la interfaz Web de Wowza, como haremos más adelante).

```
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ ls
applications conf content
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ cd conf/
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf$ ls
Admin.guid          jmxremote.password      PushPublishMap.txt      Tune.xml
admin.password      LicenseDomainCache.properties PushPublishProfilesCustom.xml VHosts.xml
Application.xml     live                  Server.guid           VHost.xml
clientaccesspolicy.xml log4j2-config.xml    Server.license        videodemanda
crossdomain.xml     MediaCache.xml       Server.xml            vod
jmxremote.access    publish.password      StartupStreams.xml
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf$ cd vod/
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf/vod$ ls
Application.xml
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf/vod$ cp conf/vod/Application.xml conf/videodemanda/Application.xml
cp: cannot stat 'conf/vod/Application.xml': No such file or directory
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf/vod$ ^C
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf/vod$ cp Application.xml ../videodemanda/
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf/vod$ cp ../../content/Big_Buck_Bunny_1080_10s_1MB.mp4 ../../content/sample.mp4
cp: cannot stat '../../content/Big_Buck_Bunny_1080_10s_1MB.mp4': No such file or directory
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf/vod$ cd ..
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/conf$ cd ..
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ docker stop wowza
wowza
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$ docker run -it --rm -d --name wowza \
-p 1935:1935 -p 8086:8086 -p 8087:8087 -p 8088:8088 \
-p 5004-5008:5004-5008/udp \
-e WSE_MGR_USER=wowza \
-e WSE_MGR_PASS=clavesecreta \
-v $(pwd)/applications:/usr/local/WowzaStreamingEngine/applications \
-v $(pwd)/conf:/usr/local/WowzaStreamingEngine/conf \
-v $(pwd)/content:/usr/local/WowzaStreamingEngine/content \
--entrypoint /sbin/entrypoint.sh \
wowzamedia-wowza-streaming-engine-linux:4.8.17
332e2600a3alle829520e7165979771lad0a52a16c71cecc0a4cf4936f809c2a4
uo288967@is02:~/IngenieriaServicios/Practica5-wowza$
```

Conecta con <http://ip.de.tu.ubuntu:8088> y ve a “Applications”, donde deberás ver una llamada “videodemanda” que es la que has creado. Arriba a la derecha hay un botón “▶ Test Playback...” que debes pulsar, para abrir el diálogo que te muestra las diferentes URLs a probar.

Por defecto, la IP que te pone arriba a la izquierda esa ventana es la IP del servidor Wowza, el cual “no sabe” que está dentro de un contenedor, por lo que te muestra en realidad la IP del contenedor (172.17.0.2 por defecto). Debes cambiar esa IP por la que tenga tu máquina Ubuntu, pues al lanzar el contenedor hemos publicado (con -p) el puerto 1935 utilizado para el straming, de modo que esté accesible a través de la máquina anfitrión.

**Cambia esa IP y deja el resto de campos como estaban (en *Application* deja “videodemanda” y en *Media file name* deja “mp4:sample.mp4”, que es el tipo de contenedor y el nombre de fichero con el vídeo a servir, el cual ha de coincidir con el que hemos dejado en la carpeta content. Toma nota de las URLs que Wowza te muestra para los protocolos HLS, MPEG-DASH, RTMP y RTSP.**

The screenshot shows the Wowza Streaming Engine management interface. At the top, there's a navigation bar with icons for Home, Server, Applications (which is highlighted in orange), and user authentication (wowza, Help, Sign Out). Below the navigation bar, a search bar contains the text "videodemanda". Underneath it, a dropdown menu labeled "Select a page" has "Main" selected. The main content area is titled "videodemanda" and describes it as a "Video on Demand Single Server or Origin". It includes four action buttons: "Test Playback...", "Copy...", "Restart...", and "Delete...". Below these buttons is a navigation bar with tabs: "Setup" (selected), "Properties", and "Modules". The "Edit" tab is highlighted with a blue background. The "Application Description" section states: "Default application for video on demand (VOD) streaming created when Wowza Streaming Engine is installed. Use this application with its default configuration or modify the configuration as needed. You can also copy it to create another VOD application." The "Playback Types" section lists five options, all of which are checked: "MPEG-DASH", "Apple HLS", "Adobe RTMP", and "RTSP/RTP".

## Test Playback

X

Server 192.168.1.162:1935

Application videodemanda

Media File Name mp4:sample.mp4

### Streaming URLs

To play your streams, enter the URLs below into your player or mobile browser.

#### HLS

**Secure URL**

<http://192.168.1.162:1935/videodemanda/mp4:sample.mp4/playlist.m3u8>



#### MPEG-DASH

**Secure URL**

<http://192.168.1.162:1935/videodemanda/mp4:sample.mp4/manifest.mpd>



#### RTMP

**Secure URL**

<rtmp://192.168.1.162:1935/videodemanda/mp4:sample.mp4>



### HLS

**Secure URL**

<http://192.168.1.162:1935/videodemanda/mp4:sample.mp4/playlist.m3u8>



### MPEG-DASH

**Secure URL**

<http://192.168.1.162:1935/videodemanda/mp4:sample.mp4/manifest.mpd>



### RTMP

**Secure URL**

<rtmp://192.168.1.162:1935/videodemanda/mp4:sample.mp4>



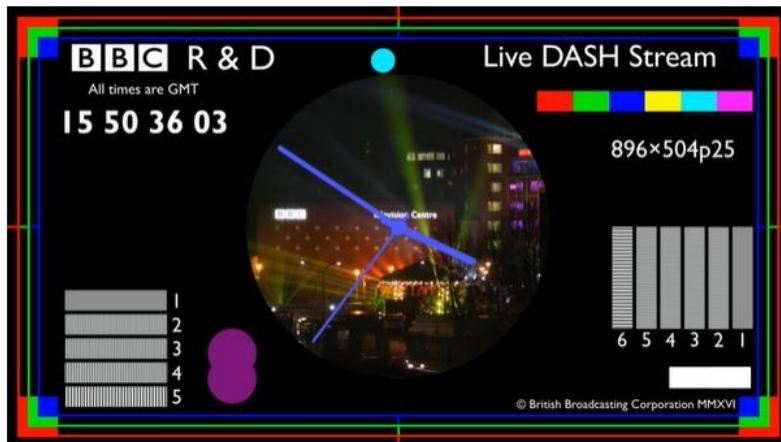
### RTSP

**Secure URL**

<rtsp://192.168.1.162:1935/videodemanda/sample.mp4>



**Test Players**



### Flowplayer Event Log

```
timeupdate - 914.28s  
timeupdate - 914.03s  
timeupdate - 913.77s  
state - 913.27s  
timeupdate - 913.02s  
timeupdate - 912.27s  
timeupdate - 910.26s  
timeupdate - 910.01s  
timeupdate - 909.76s  
timeupdate - 909.26s  
timeupdate - 909.01s
```

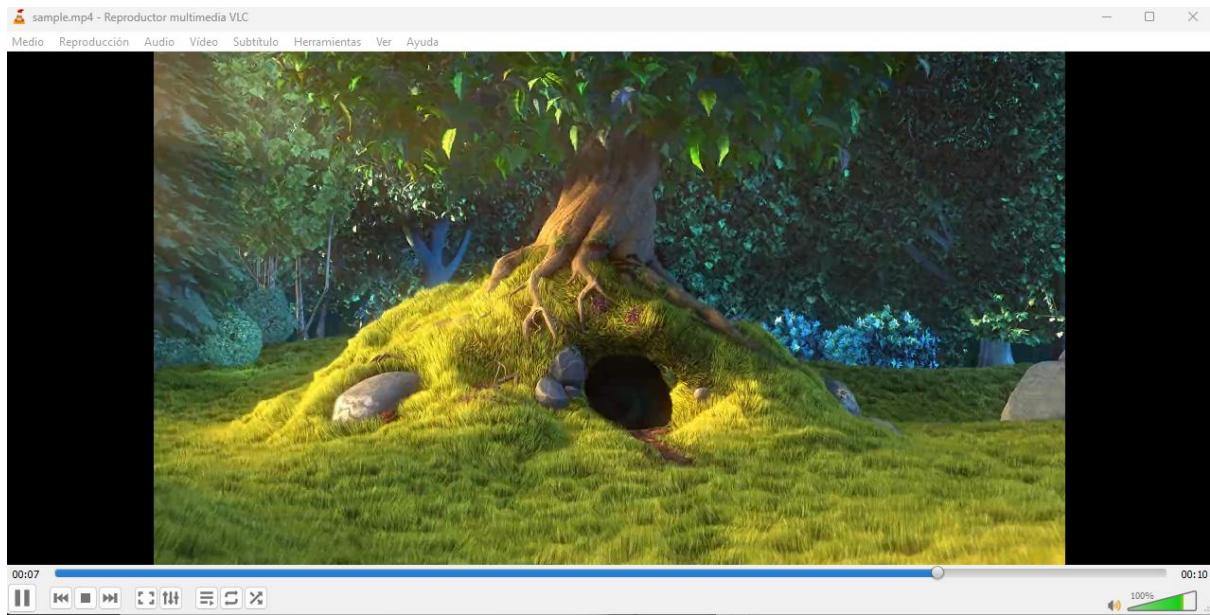
Este cliente soporta todos los protocolos antes vistos. Para verificarlo abre el programa VLC y ve al menú Medio > Abrir Ubicación de Red, o pulsa Ctrl+N. En el diálogo que aparece introduce una de las URLs de las que hemos tomado nota en el apartado anterior, por ejemplo la que corresponde a HLS, y después pulsa el botón “Reproducir”. Verifica que el vídeo se está sirviendo correctamente.

Repite lo anterior para los restantes tres protocolos y verifica que VLC puede reproducirlos todos sin problemas.

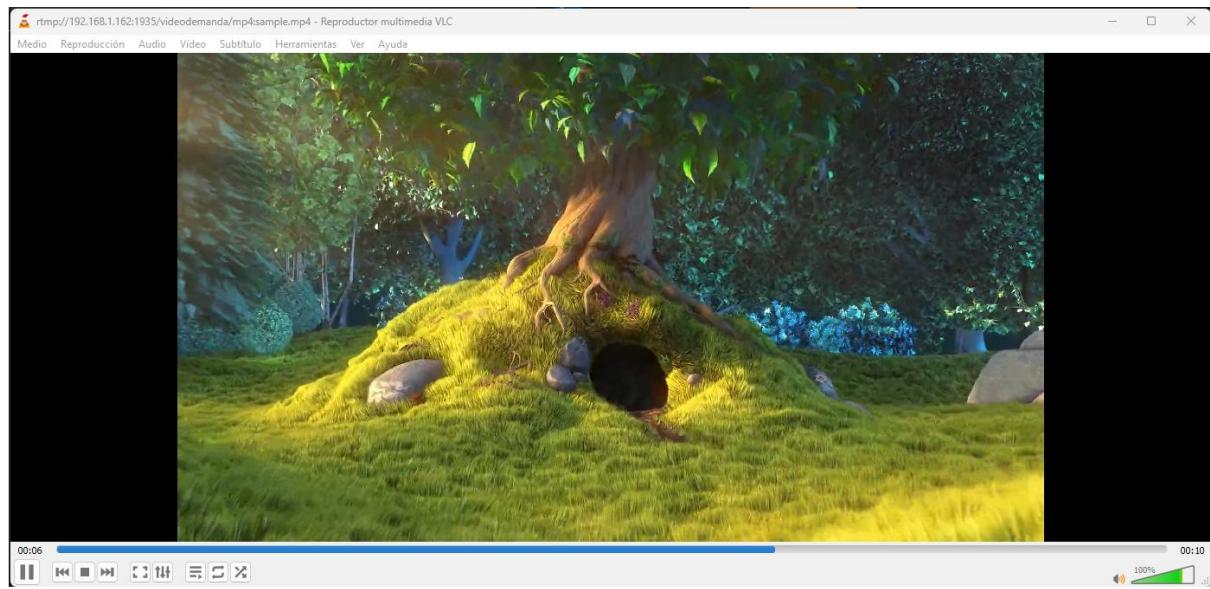
Con HLS:



## Con MPEG-DASH



## Con RTMP



## Con RTSP

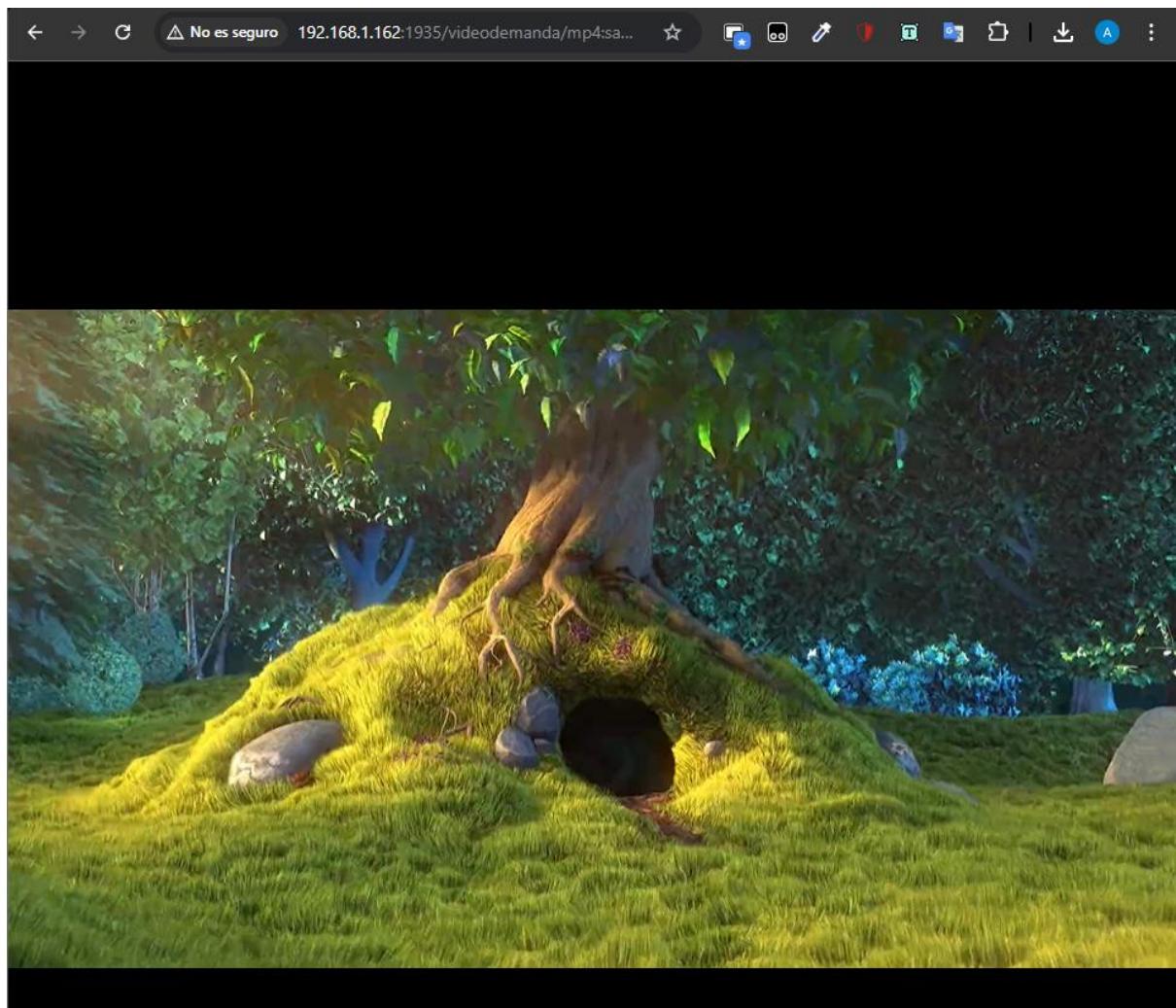


Si observas las URLs antes utilizadas, verás que las dos últimas (RTMP y RTSP) comienzan por `rtmp://` y `rtsp://` respectivamente, que no son protocolos soportados por el navegador. Esto indica que si intentamos pegar esas URLs directamente en la barra de dirección del vamos a obtener un error. Compruébalo.

En cambio las dos primeras comienzan por `http://` por lo que el navegador debería aceptarlas. Pero si te fijas en el resto de la URL no parece que el fichero que está solicitando sea el vídeo a reproducir. En el caso del protocolo HLS el fichero solicitado tiene extensión `.m3u8` y en el caso de MPEG-DASH tiene extensión `.mpd`. No son formatos contenedores de vídeo, sino ficheros de texto utilizados como parte de estos protocolos. ¿Qué ocurrirá si usamos directamente esas URLs en el navegador? Intentalo. Verás que el navegador, al no conocer qué hacer con ese tipo de ficheros, se limita a descargarlos como si fuera un archivo de datos.

Prueba a abrir con un editor los ficheros que se ha descargado y verás que HLS proporciona un archivo de texto plano, mientras que MPEG-DASH proporciona un XML.

HLS



MPEG-DASH

```

<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns="urn:mpeg:dash:schema:mpd:2011"
      xmlns:xlink="http://www.w3.org/1999/xlink"
      xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011 http://standards.iso.org/ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd"
      profiles="urn:mpeg:dash:profile:isoff-live:2011"
      type="static"
      publishTime="2025-12-26T16:05:00.134Z"
      mediaPresentationDuration="PT10.0S"
      minBufferTime="PT1.5S">
  <ProgramInformation>
    <Title>sample.mp4</Title>
  </ProgramInformation>
  <Location>http://192.168.1.162:1935/videodemanda/mp4:sample.mp4/manifest_w162580627.mpd</Location>
  <Period id="0" start="PT0.0S">
    <AdaptationSet id="0" group="1" mimeType="video/mp4" width="1920" height="1080" par="16:9" frameRate="60"
      segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true" subsegmentStartsWithSAP="1">
      <SegmentTemplate presentationTimeOffset="0" timescale="90000" media="segment_ctvideo_rid$RepresentationID$_cs
      $Time$_w162580627_mpd.m4s" initialization="segment_ctvideo_rid$RepresentationID$_cinit_w162580627_mpd.m4s">
        <SegmentTimeline>
          <S t="0" d="900000"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="p0va0br832398" codecs="avc1.64002a" sar="1:1" bandwidth="832398" />
    </AdaptationSet>
  </Period>
</MPD>

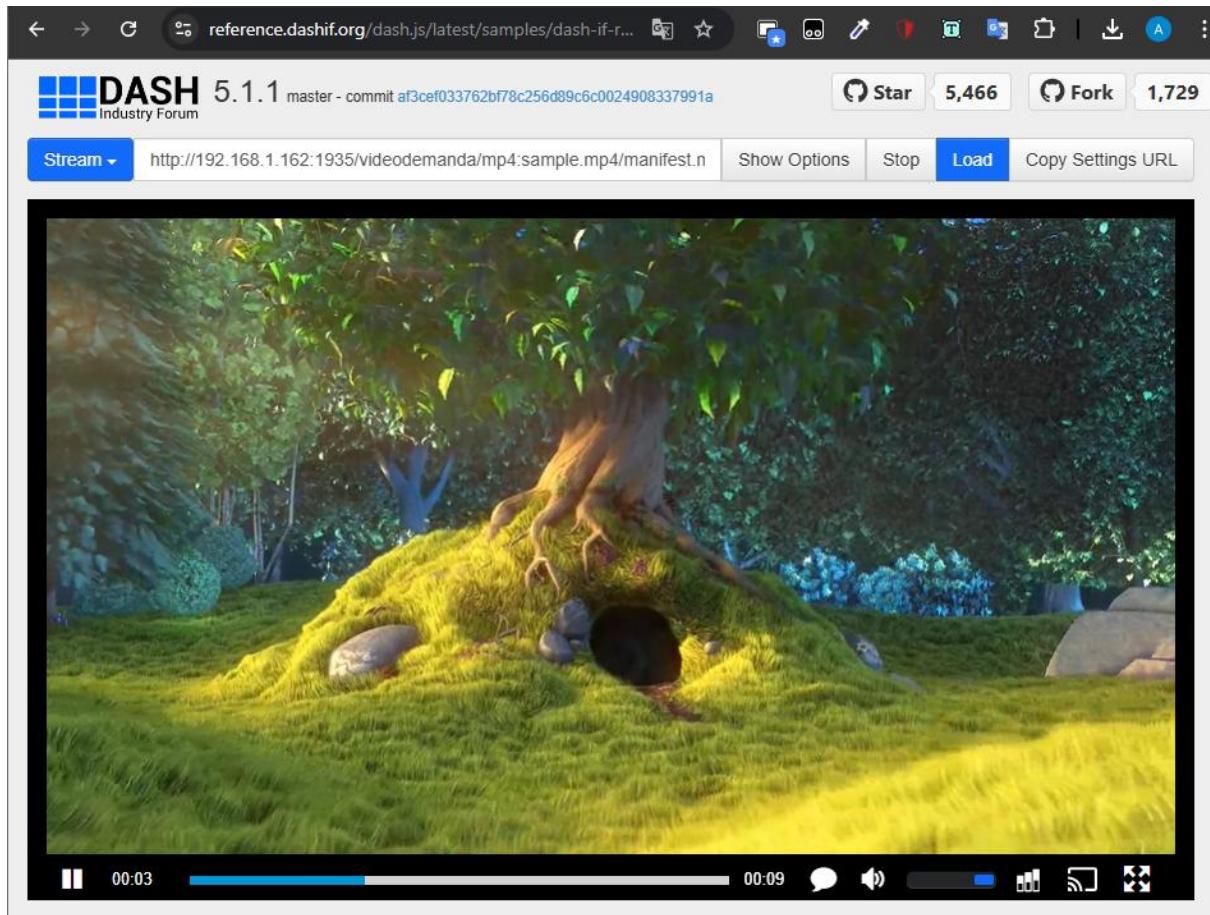
```

- Ve a la página <https://reference.dashif.org/dash.js/> en la que el consorcio MPEG-DASH proporciona un cliente de referencia (en el que pueden basarse otros reproductores javascript que quieran ser compatibles con este protocolo)
- Pincha en el enlace [Reference GUI](#)
- Asegúrate de que el navegador está accediendo a esta página con el protocolo HTTP y no con HTTPS pues de lo contrario no podría reproducir nuestro vídeo (ya que Wowza lo sirve bajo HTTP). Si ves que el navegador accede por HTTPS, edita la URL en la barra de navegación para que use HTTP (<http://reference.dashif.org/dash.js/latest/samples/dash-if-reference-player/index.html>)
- En la caja de texto que aparece al lado de “Stream” pega la URL que has sacado de Wowza para el protocolo MPEG-DASH y luego pulsa el botón “Load”

La reproducción del vídeo debería comenzar. Observa que el vídeo está siendo solicitado desde tu navegador a tu servidor Wowza. El vídeo no pasa por los servidores de dashif.org que tan solo se ha limitado a proporcionar el código JS que el navegador está ejecutando. Ese ese código el que, en base al XML que ha recibido en el fichero manifest.mpd va haciendo peticiones de vídeo al servidor Wowza.

**Nota:** Hemos detectado que algunos navegadores se niegan a reproducir el vídeo, debido a que la URL para reproducirlo (la que has pegado para MPEG-DASH que apunta a nuestro servidor Wowza) no es segura por comenzar por http. El síntoma es que se muestra un error “Dash.js:25” diciendo que el recurso no está disponible, y el vídeo no se muestra. El problema no es que el recurso no esté disponible, sino que el navegador ha bloqueado el acceso al mismo. Si abres la consola JavaScript para desarrolladores en el navegador podrás ver errores de CORS del

estilo “*Mixed Content: [...] the content must be served over HTTPS*”, que hacen que el navegador se niegue a acceder a ese recurso. Si te sucede esto, simplemente adjunta en la memoria una captura de pantalla que muestre el problema y pasa al ejercicio siguiente.



En la caperta html que creaste al inicio de la práctica crea otra llamada videojsHLS y dentro de ella el fichero index.html a partir del siguiente ejemplo que debes adaptar:

```
<head>

<title>Apple HLS con Video.js</title>

<link href="https://vjs.zencdn.net/7.1.0/video-js.css" rel="stylesheet">

</head>

<body>

<h1>HLS con Video.js</h1>

<video id="my-video" class="video-js" controls

preload="auto" width="640" height="264">

<source src="URL-a-averiguar" type="application/x-mpegURL">

<p class="vjs-no-js">
```

Para ver este video debes activar JavaScript y tener un navegador moderno que <a href="https://videojs.com/html5-video-support/">soporte HTML5 video</a>

</p>

</video>

```
<script src="https://vjs.zencdn.net/7.1.0/video.js"></script>

<script>
    // Esto instancia el reproductor del video usando el elemento cuyo
    // id le pasemos como parámetro
    var player = videojs('my-video')

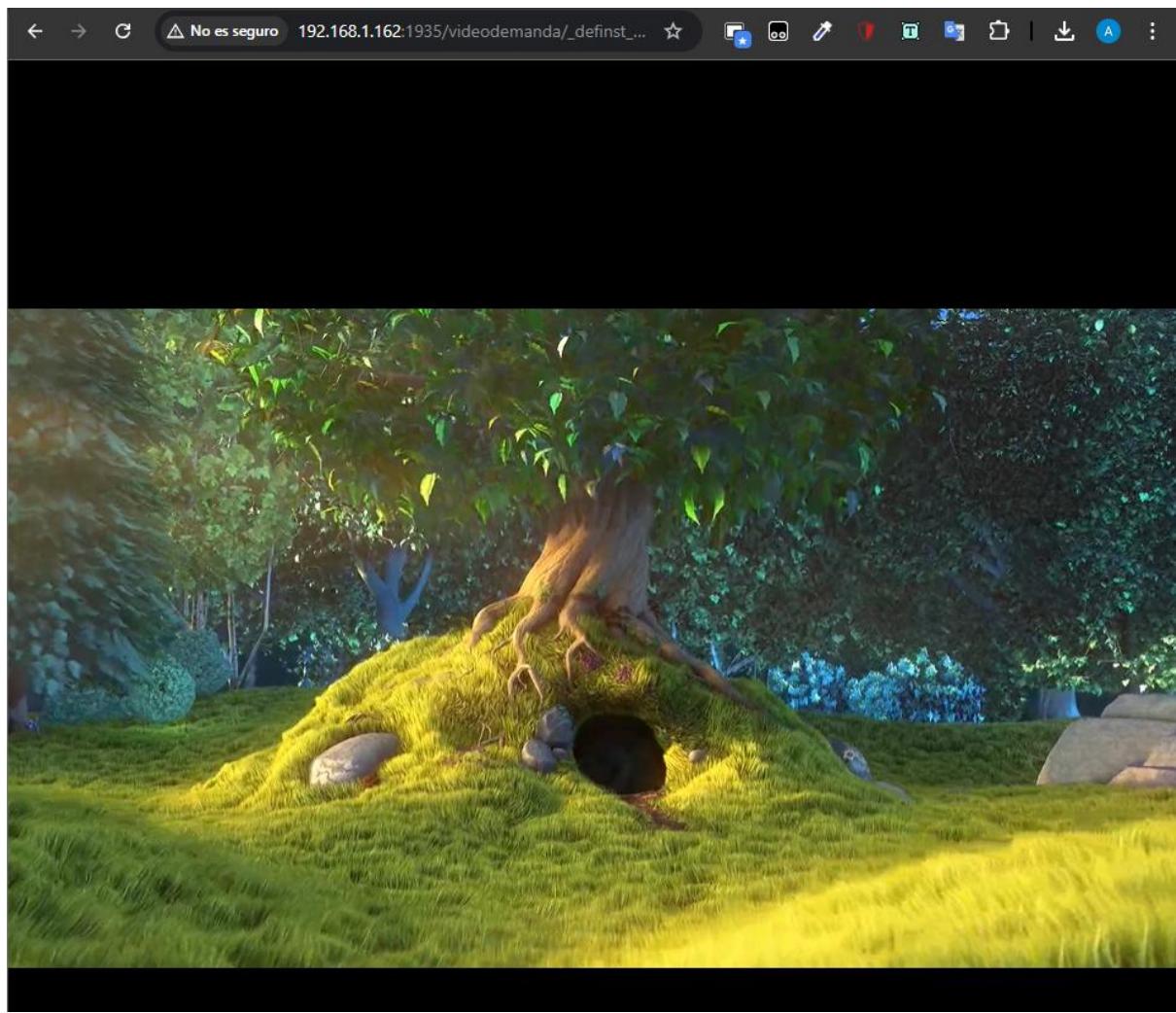
</script>

</body>
```

Averigua cuál es la URL que debes poner en el recurso donde Wowza servirá el *stream* para la aplicación videodemanda usando el protocolo HLS. Aunque el protocolo fue creado para la plataforma iOS y un navegador no-Apple en principio no podría reproducir ese protocolo, gracias a video.js sí se podrá, pues en el fondo utiliza el protocolo HTTP y las capacidades de video de HTML5.

Recuerda poner como IP del servidor la de tu máquina Ubuntu en lugar de la del contenedor docker.

La url será [http://192.168.1.162:1935/videodemanda/\\_definst\\_/mp4:sample.mp4/playlist.m3u8](http://192.168.1.162:1935/videodemanda/_definst_/mp4:sample.mp4/playlist.m3u8)



## EJERCICIO 2: VIDEO EN VIVO

Descarga en Windows algún vídeo de longitud suficiente (media hora). Puedes usar por ejemplo este: <http://www.atc.uniovi.es/grado/4is/tv.mp4>.

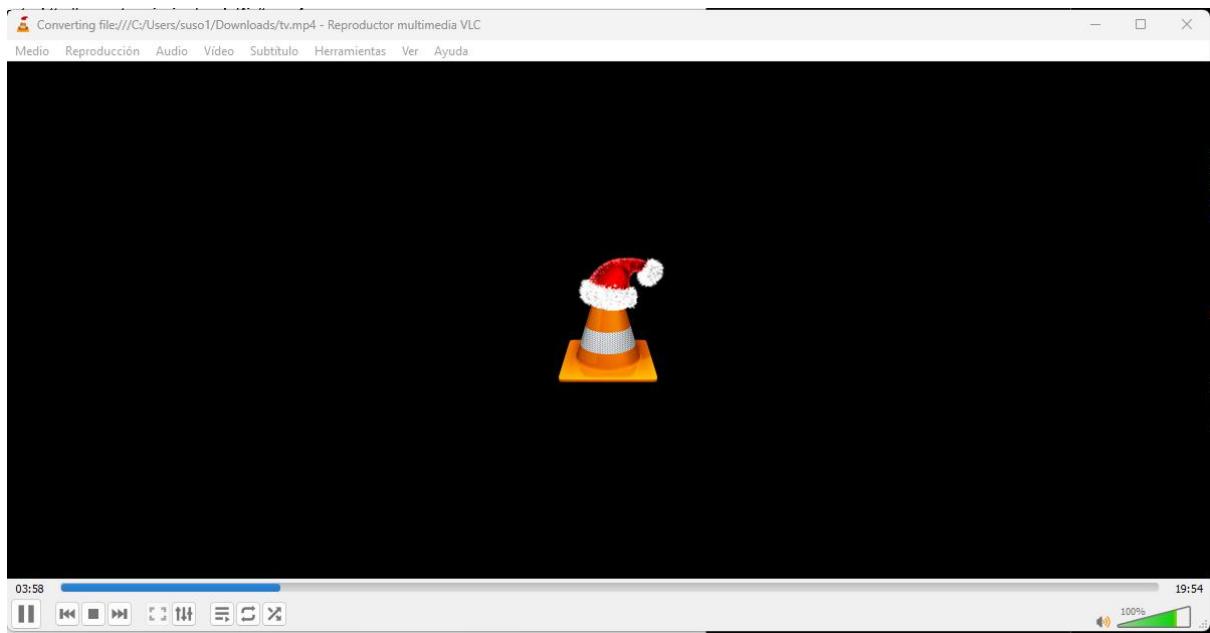
Abre VLC y elige el menú “Medio->Emitir” (si trabajas con OSX, deberás usar “Abrir archivo -> Abrir archivo avanzado” y adaptar lo que sigue).

En la pestaña “Archivo” elige el video que has descargado. Pulsa el botón “Emitir”.

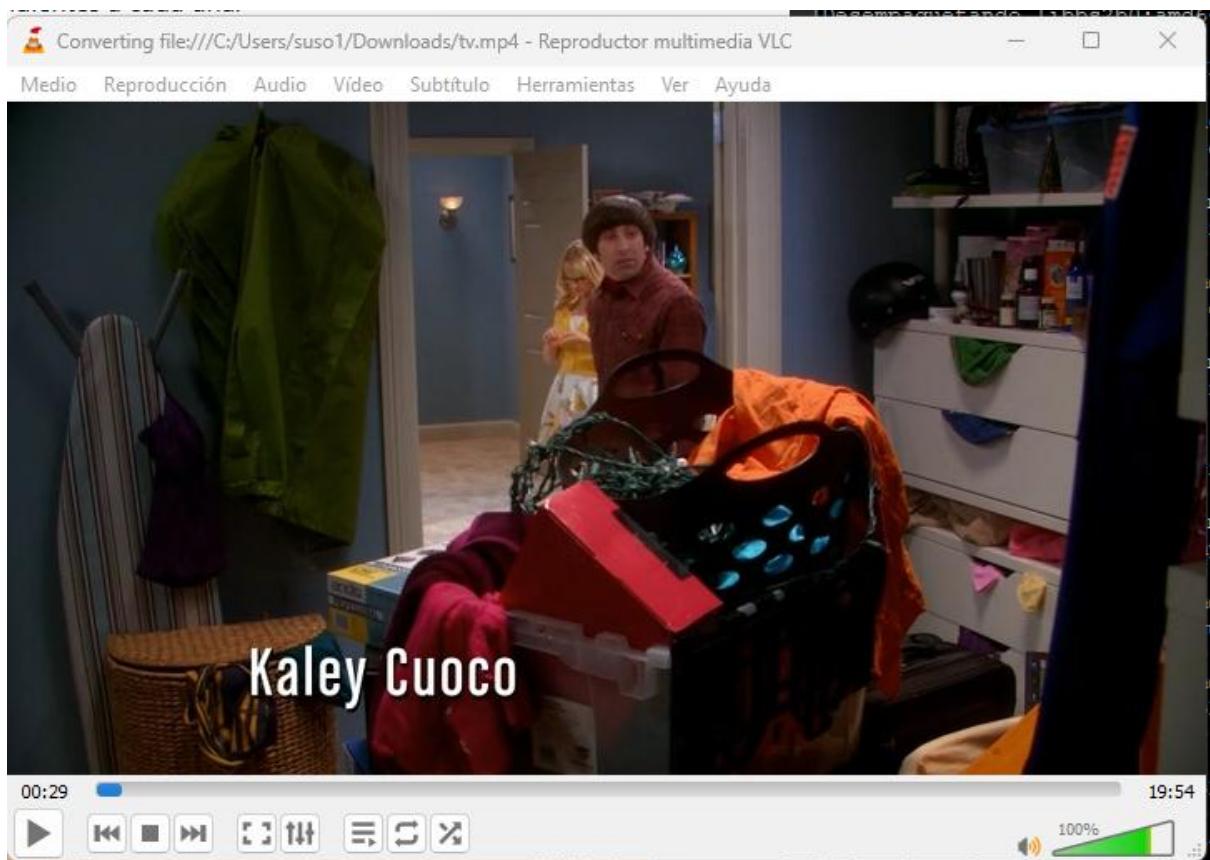
Pulsa “siguiente” y en la pantalla de configuración de destino selecciona la opción “RTP / MPEG Transport Stream” y pulsa el botón “añadir”.

Selecciona como dirección la IP del servidor linux y como puerto 5004. Pulsa “siguiente”.

Selecciona transcodificar a “Video H.264 + MP3 (MP4)”. Pulsa “siguiente” y luego “stream” o “emitir”.



Como se puede ver y como dice la práctica, el video se reproduce pero no se muestra.



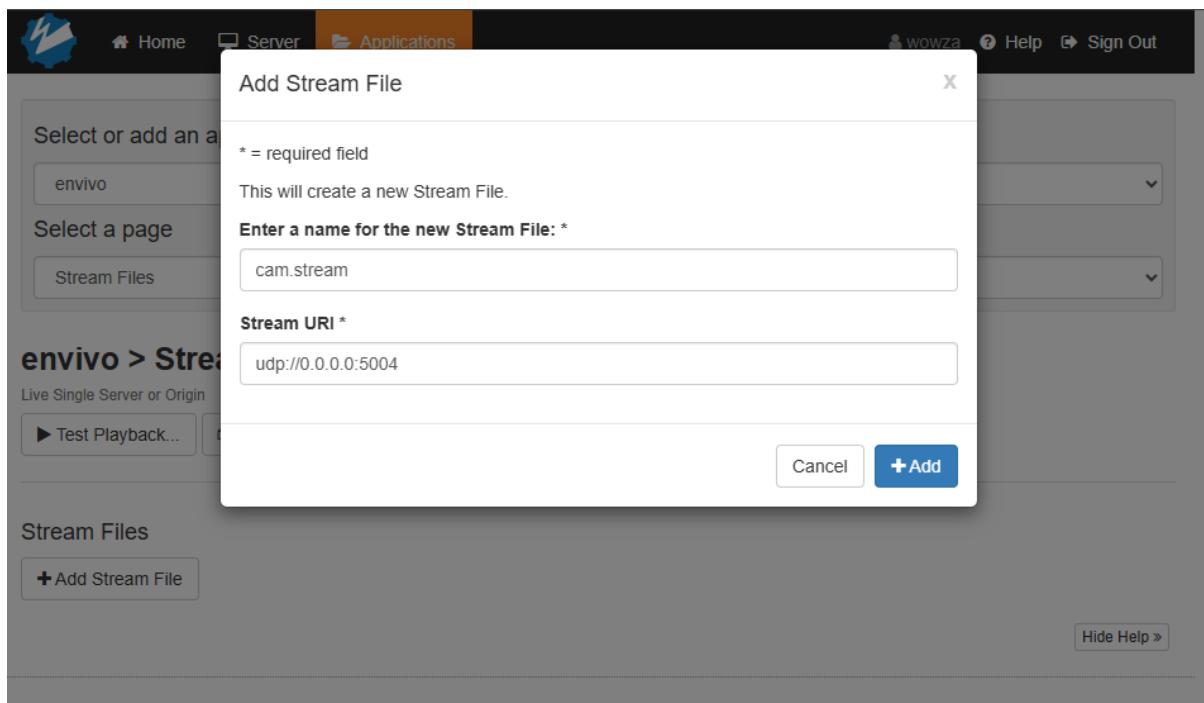
\*Cargó después de 10 minutos.

**A través de la interfaz web de Wowza crea una nueva aplicación de video en vivo, llamada por ejemplo “envivo”.**

Dentro de la opción “Stream Files” añade un nuevo fichero llamado cam.stream, cuya URI sea udp://0.0.0.0:5004, (la IP 0.0.0.0 hará que escuche en la interfaz de red del contenedor docker donde se está ejecutando Wowza).

Pincha de nuevo en “Stream files” y en la lista de streams, pulsa el icono que representa “Connect to this stream”. Dentro de la pantalla “Incoming Streams” deberías ver que tienes un flujo de video conectado.

Verifica que eres capaz de reproducir el video con el reproductor web que ya usamos para el ejercicio de video bajo demanda. Para obtener la URL apropiada usa “Test Playback...” Fíjate que el nombre del stream debe ser cam.stream y la IP debe ser la del servidor Ubuntu. Al igual que en el ejercicio anterior, se requiere Flash para ciertos protocolos.



Screenshot of the Wowza Media Server interface showing the 'Applications' section.

Header navigation: Home, Server, Applications, wowza, Help, Sign Out.

Form fields:

- Select or add an application: envivo
- Select a page: Stream Files

Message bar: Connection successful cam.stream

Section title: envivo > Stream Files

Sub-section title: Live Single Server or Origin

Action buttons: Test Playback..., Copy..., Restart..., Delete...

Section title: Stream Files

Action buttons: Copy Stream File, Add Stream File

Name	Actions
cam.stream	→ + 🖊️ 🗑️

Link: Hide Help »



## EJERCICIO 3 : STREAMING CON BITRATE ADAPTATIVO (ABR)

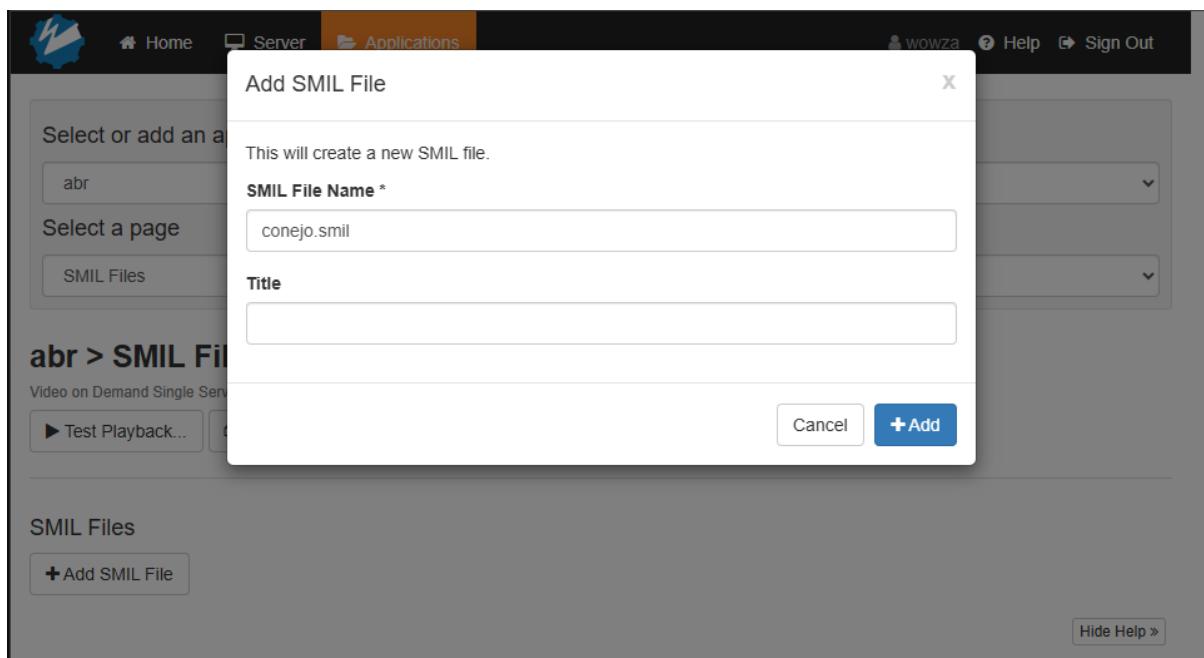
Descomprime el fichero <http://www.atc.uniovi.es/grado/4is/VideosABR.zip> dentro de la carpeta wowza/content.

Crea una nueva aplicación de video bajo demanda mediante la interfaz web (llámala abr)

Las calidades disponibles se describen en un fichero denominado "smil". Desde la interfaz web de la aplicación añade un nuevo fichero SMIL a la aplicación denominado conejo.smil.

Dentro del fichero smil debes añadir tantos flujos ("+ Add SMIL stream") como videos tengas disponibles. Utiliza el programa ffprobe para obtener la información que necesitas, con la opción -show\_entries stream=bit\_rate te mostrará el bitrate del vídeo y el del audio (en ese orden). Repítelo para cada uno de los ficheros.

Verifica que puedes reproducir el contenido ABR con el cliente de referencia DASH que hemos usado anteriormente. Ahora el nombre del medio es smil:conejo.smil. Verás que el reproductor comienza correctamente, pero puede que genera errores de "recurso no encontrado" al poco tiempo, así como si intentas cambiar la calidad del video con el selector de calidades. Probablemente se trata de alguna incompatibilidad entre este reproductor concreto y los contenidos servidos por Wowza, por lo que intentaremos seguidamente utilizar otro reproductor.



Para usar ffprobe, tuvimos que instalarlo con el comando:

```
sudo apt install ffmpeg
```

```
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/content$ ffprobe -v error -select_streams v:0 -show_entries stream=bit_rate -of default=noprint_wrappers=l conejo_300.mp4
bit_rate=170819
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/content$ ffprobe -v error -select_streams v:0 -show_entries stream=bit_rate -of default=noprint_wrappers=l conejo_450.mp4
bit_rate=344453
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/content$ ffprobe -v error -select_streams v:0 -show_entries stream=bit_rate -of default=noprint_wrappers=l conejo_750.mp4
bit_rate=644274
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/content$ ffprobe -v error -select_streams v:0 -show_entries stream=bit_rate -of default=noprint_wrappers=l conejo_1100.mp4
bit_rate=993719
uo288967@is02:~/IngenieriaServicios/Practica5-wowza/content$
```

## abr > conejo.smil > [new]

Video on Demand Single Server or Origin

[← Return to SMIL File](#)

Basic

\* = Required field

[+ Add](#)

[Cancel](#)

**Tag Type**

video

**Source (src) \***

mp4:conejo\_300.mp4

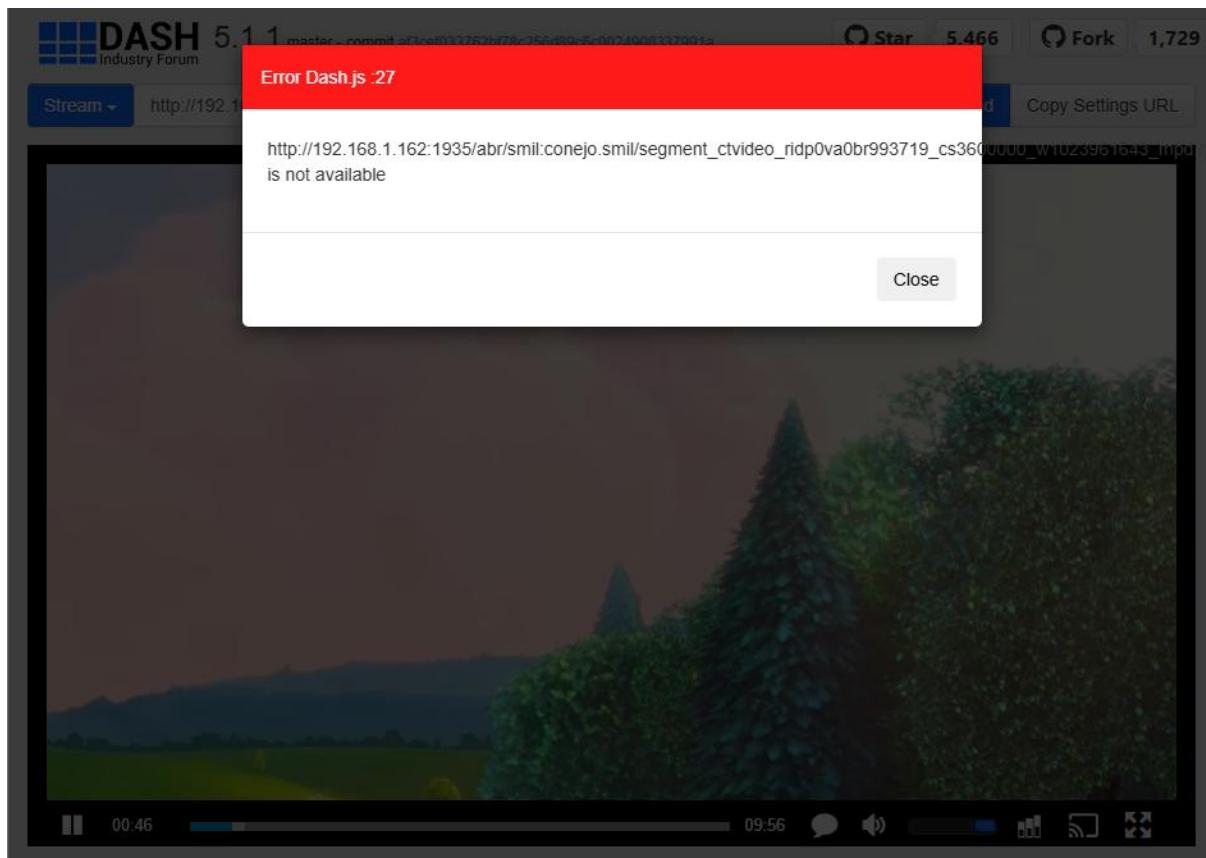
**System Language**

**Video Bitrate \***

170819

bps

\*Repetimos esto con los otros 3 videos



Como hay incompatibilidad se probará con otro reproductor

**En la misma carpeta, copia index.html como indexABR.html.**

**Modifícalo para que el título mostrado sea “HLS con ABR con Video.js”**

**Modifica el atributo src del vídeo para que sea el del recurso SMIL (puedes usar el reproductor que te da Wowza en “Test Playback...” y seleccionar la URL para el protocolo HLS).**

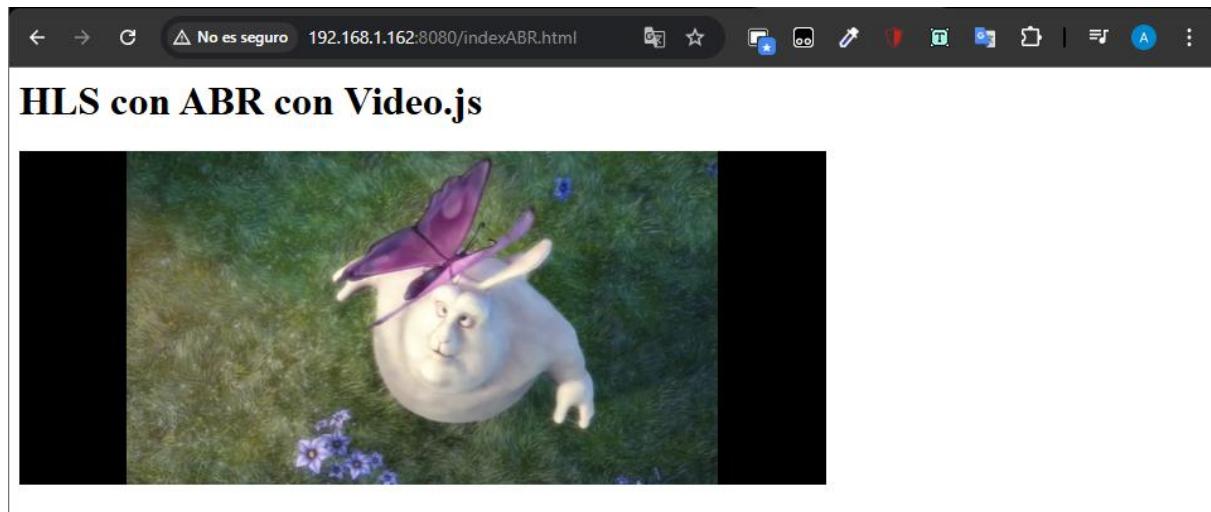
**Usa un navegador web para acceder a <http://ip.de.ubuntu/videojsHLS/indexABR.html> y comprueba que se ve el vídeo.**

```
uo288967@is02: ~/IngenieriaServicios/Practica5/html/videojsHLS
GNU nano 7.2                               indexABR.html *

<head>
  <title>Apple HLS con Video.js</title>
  <link href="https://vjs.zencdn.net/7.1.0/video-js.css" rel="stylesheet">
</head>

<body>
  <h1>HLS con ABR con Video.js</h1>
  <video id="my-video" class="video-js" controls
    preload="auto" width="640" height="264">
    <source src="http://192.168.1.162:1935/abr/smil:conejo.smil/playlist.m3u8" type="application/x-mpegURL">
    <p class="vjs-no-js">
      Para ver este video debes activar JavaScript y tener un navegador
      moderno que <a href="https://videojs.com/html5-video-support/">
      soporte HTML5 video</a>
    </p>
  </video>

  <script src="https://vjs.zencdn.net/7.1.0/video.js"></script>
  <script>
    // Esto instancia el reproductor del video usando el elemento cuyo
    // id le pasemos como parámetro
    var player = videojs('my-video')
  </script>
</body>
```



**Haz las modificaciones anteriores y comprueba que el reproductor ofrece ahora un nuevo control que permite elegir la calidad del vídeo. Prueba a cambiar esa calidad mientras observas las peticiones de red para comprobar cómo solicita recursos diferentes según la calidad elegida.**

```
GNU nano 7.2 /home/uo288967/IngenieriaServicios/Practica5/html/videojsHLS/indexABR.html
<head>
  <title>Apple HLS con Video.js</title>
  <link href="https://vjs.zencdn.net/7.1.0/video-js.css" rel="stylesheet">
</head>

<body>
  <h1>HLS con ABR con Video.js</h1>
  <video id="my-video" class="video-js vjs-default-skin" controls
    preload="auto" width="640" height="264">
    <source src="http://192.168.1.162:1935/abr/smil:conejo.smil/playlist.m3u8" type="application/x-mpegURL">
    <p class="vjs-no-js">
      Para ver este video debes activar JavaScript y tener un navegador
      moderno que <a href="https://videojs.com/html5-video-support/">
      soporte HTML5 video</a>
    </p>
  </video>

  <script src="https://vjs.zencdn.net/7.1.0/video.js"></script>

  <script src="https://cdn.jsdelivr.net/npm/videojs-contrib-quality-levels@2.0.7/dist/videojs-contrib-quality-levels">
  <script src="https://cdn.jsdelivr.net/npm/videojs-hls-quality-selector@0.0.8/dist/videojs-hls-quality-selector.min">

  <script>
    // Esto instancia el reproductor del video usando el elemento cuyo
    // id le pasemos como parámetro
    var player = videojs('my-video');

    // Inicializa el selector de calidad para HLS
    player.hlsQualitySelector();
  </script>
</body>
```

Se han añadido los 3 scripts y el qualitySelector

Despues de varios intentos el reproductor sigue sin mostrar el engranaje para cambiar la calidad, pero creemos que el código esta correcto, no encontramos solución a esto.

## **SERVICIOS INTERACTIVOS DE MENSAJERIA**

### **PRACTICA 6**

Esta práctica se centró en la implementación de servicios de mensajería instantánea utilizando el protocolo abierto **XMP**P. Comenzamos desplegando y configurando un servidor **Prosody** mediante contenedores Docker, creando cuentas de usuario y probando el intercambio de mensajes y estados a través del cliente **Pidgin**. Una parte fundamental consistió en el desarrollo de un bot interactivo en Python utilizando la biblioteca **slixmpp**. A través del bot, aprendimos a gestionar eventos de sesión, procesar mensajes entrantes (stanzas XML) y detectar cambios de estado de los contactos. Implementamos diversas funcionalidades, desde un simple bot de eco hasta un sistema capaz de evaluar expresiones matemáticas en tiempo real y responder de forma automatizada a los usuarios.

### **EJERCICIO 1**

#### **Despliegue del servidor**

**Ponte de acuerdo con tu compañero de grupo en quién de los dos desplegará el servidor. No es necesario que lo hagais ambos. Quien vaya a instalarlo es quien debe realizar este ejercicio tal como se describe a continuación.**

#### **SERVIDOR**

```

uo288967@is02:~/ingenieriaServicios$ mkdir Practica6
uo288967@is02:~/IngenieriaServicios$ cd Practica6
uo288967@is02:~/IngenieriaServicios/Practica6$ docker pull unclerv/prosody-docker-extended:0.10
0.10: Pulling from unclerv/prosody-docker-extended
35b42117c431: Pull complete
ad9c569a8d98: Pull complete
293b44f45162: Pull complete
0c175077525d: Pull complete
af974fc95e36: Pull complete
170a840f43db: Pull complete
3cb5aee783c9: Pull complete
1750fff36165: Pull complete
b0e96c85189c: Pull complete
ac456b41e72c: Pull complete
54f700743882: Pull complete
Digest: sha256:4b835061eb5d5d13alb3da7cda0bf9573a7e221a0fe5e74b3087271f864a4a2
Status: Downloaded newer image for unclerv/prosody-docker-extended:0.10
docker.io/unclerv/prosody-docker-extended:0.10
uo288967@is02:~/IngenieriaServicios/Practica6$ docker run --rm -d --name prosody unclerv/prosody-docker-extended:0.10
76e48cadb7217a2c153c05289e67d7a3c64d0ac7f84adcfd1b88892fe75ab79
uo288967@is02:~/IngenieriaServicios/Practica6$ mkdir prosody
uo288967@is02:~/IngenieriaServicios/Practica6$ cd prosody/
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ mkdir etc data
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ docker cp prosody:/etc/prosody etc
Successfully copied 24.1kB to /home/uo288967/IngenieriaServicios/Practica6/prosody/etc
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ LS
LS: command not found
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ ls
data  etc
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ █

```

Quien haya desplegado el servidor, deberá editar el fichero prosody.cfg.lua y buscar la línea que comienza por VirtualHost, cambiando su valor "localhost" para que sea "ingservXX" donde XX es vuestro número de grupo. Si no estás seguro del número usa en lugar de XX una cadena arbitraria de tres dígitos. CUIDADO, no uses ningún carácter en mayúsculas como nombre del host, pues eso causaría problemas más adelante.

Cada vez que este documento mencione el nombre ingservXX en lo sucesivo, debeis cambiarlo por el que hayais usado en este paso.

```

----- Virtual hosts -----
-- You need to add a VirtualHost entry for each domain you want to support
-- Settings under each VirtualHost entry apply *only* to that domain

VirtualHost "ingserv02"

--VirtualHost "example.com"
--      certificate = "/path/to/example.crt"

```

Detén el contenedor (si aún estaba en marcha) con docker stop prosody

Escribe lo siguiente en un script llamado lanza-prosody.sh (que debe estar en la misma carpeta en que se hallan los directorios etc y data antes creados)

```

$ docker run --rm -d --name prosody --network pruebas \
-p 5222:5222 \
-v $(pwd)/etc/prosody:/etc/prosody \
-v $(pwd)/data:/var/lib/prosody \
unclev/prosody-docker-extended:0.10

```

Lanza el script anterior y comprueba con docker ps que todo haya ido bien.

```

uo288967@is02:~/IngenieriaServicios/Practica6/prosody
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ ./lanza-prosody.sh
a0884a5db53c2b009f326daeae1ddaf3c6f12eclc36ee47790e3c0d5826b43al
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a0884a5db53c unclerv/prosody-docker-extended:0.10 "/usr/bin/entrypoint..." 20 seconds ago Up 18 seconds 80/tcp
, 443/tcp, 5269/tcp, 5280-5281/tcp, 5347/tcp, 0.0.0.0:5222->5222/tcp, [::]:5222->5222/tcp prosody
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ 

```

**Quien haya instalado Prosody, debe conectar con el contenedor docker en que se está ejecutando, mediante el comando:**

```
$ docker exec -it prosody bash
```

Una vez “dentro” del contenedor, tenemos acceso al comando prosodyctl, como puede comprobarse con prosodyctl help por ejemplo. Usémoslo para crear un certificado en la forma siguiente:

```
$ prosodyctl cert generate ingservXX
```

Puedes pulsar intro para aceptar la opción por defecto ante las diferentes preguntas que te irá haciendo, pero puede ser conveniente que cuando te pregunte el email pongas tu dirección de correo de uniovi, pues es la dirección a que enviará posibles problemas de funcionamiento (que de otro modo se perderían).

Una vez finalizado comprueba con ls que tienes los dos ficheros .crt y .key que acabas de generar en la carpeta /var/lib/prosody.

Puedes salir del contenedor con el comando exit.

```

uo288967@is02:~/IngenieriaServicios/Practica6/prosody
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ ./lanza-prosody.sh
a0884a5db53c2b009f326daeae1ddaf3c6f12eclc36ee47790e3c0d5826b43al
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ docker ps
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
a0884a5db53c unclerv/prosody-docker-extended:0.10 "/usr/bin/entrypoint..." 20 seconds ago Up 18 seconds 80/tcp
, 443/tcp, 5269/tcp, 5280-5281/tcp, 5347/tcp, 0.0.0.0:5222->5222/tcp, [::]:5222->5222/tcp prosody
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ docker exec -it prosody bash
prosody@a0884a5db53c:~$ prosodyctl cert generate ingserv02
Choose key size (2048):
Generating RSA private key, 2048 bit long modulus
.....+++
.....+++
e is 65537 (0x10001)
Key written to /var/lib/prosody/ingserv02.key
Please provide details to include in the certificate config file.
Leave the field empty to use the default value or '.' to exclude the field.
countryName (GB):
localityName (The Internet):
organizationName (Your Organisation):
organizationalUnitName (XMPP Department):
commonName (ingserv02):
emailAddress (xmpp@ingserv02): uo288967@uniovi.es

Config written to /var/lib/prosody/ingserv02.cnf
Certificate written to /var/lib/prosody/ingserv02.crt

prosody@a0884a5db53c:~$ ls
bin boot dev etc home lib lib64 media mnt opt proc root run sbin srv sys tmp usr var
prosody@a0884a5db53c:~$ 

```

```

prosody@a0884a5db53c:~$ ls
ingserv02.cnf  ingserv02.crt  ingserv02.key
prosody@a0884a5db53c:~$ 

```

Mueve los certificados de data/ a etc/prosody/certs.

**Detén el contenedor con docker stop prosody y lánzalo de nuevo con el script que hemos escrito antes.**

**Comprueba con docker logs prosody que no haya problemas en el arranque.**

```
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ la  
data etc lanza-prosody.sh  
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ mv data/*.crt etc/prosody/certs/  
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ mv data/*.key etc/prosody/certs/
```

**En la máquina en que se está ejecutando el servidor, entra al contenedor docker como hemos hecho antes:**

```
$ docker exec -it prosody bash
```

**De este modo ya tenemos acceso al comando prosodyctl con el que crearemos los usuarios:**

```
$ prosodyctl adduser manolo@ingservXX
```

```
$ prosodyctl adduser ramon@ingservXX
```

```
$ exit
```

**En cada uno de estos dos casos será necesario introducir una contraseña para el correspondiente usuario. Esta contraseña se necesitará más tarde en el cliente.**

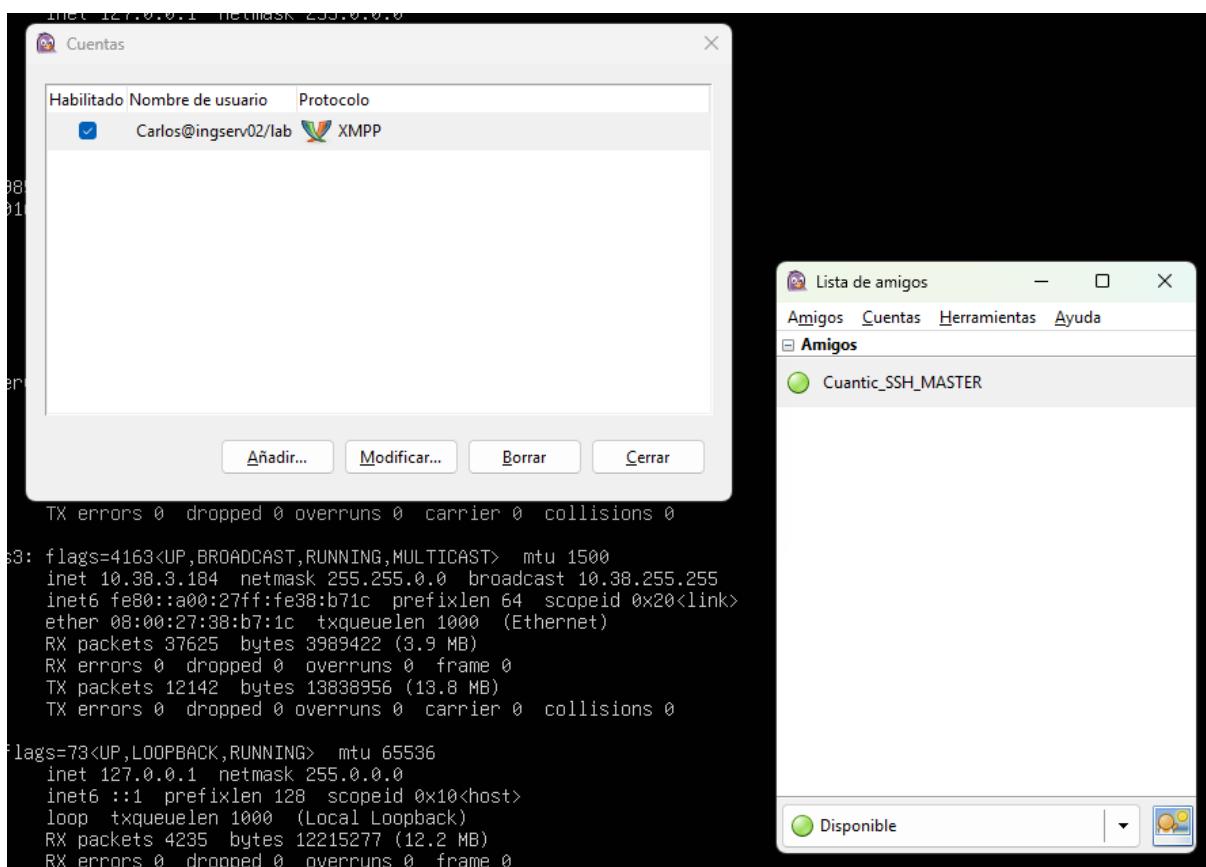
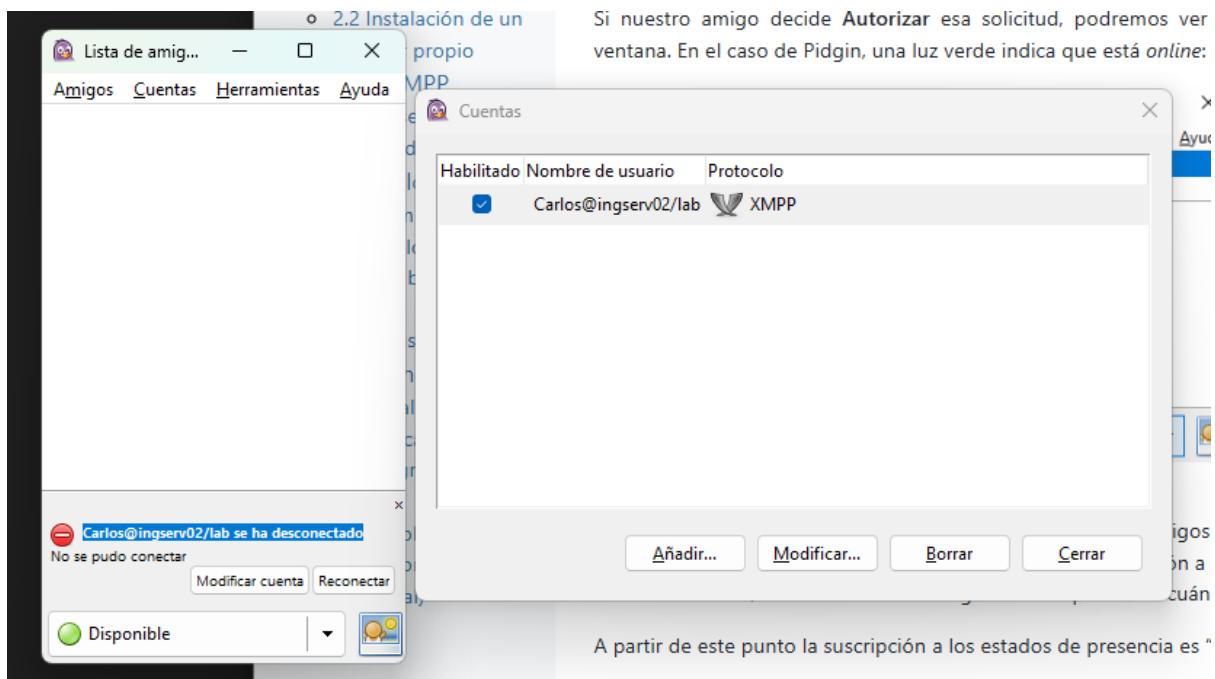
```
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ ./lanza-prosody.sh  
f4fa653fadec8d72df5cef9742268d6b3c0703bd0257079cadd3e96530fae576  
uo288967@is02:~/IngenieriaServicios/Practica6/prosody$ docker exec -it prosody bash  
prosody@f4fa653fadec:$ prosodyctl adduser Carlos@ingserv02  
Enter new password:  
Retype new password:  
prosody@f4fa653fadec:$ prosodyctl adduser Medina@ingserv02  
Enter new password:  
Retype new password:  
prosody@f4fa653fadec:$ █
```

## CLIENTE

### EJERCICIO 2

#### Instalación del cliente

Descarga [la versión portable de Pidgin](#) e instálala en cualquier carpeta de tu máquina Windows en la que tengas permisos de escritura (es posible instalarla también en un lápiz USB, pero es más rápido instalarlo en el disco duro y después copiar al lápiz la carpeta en cuestión).

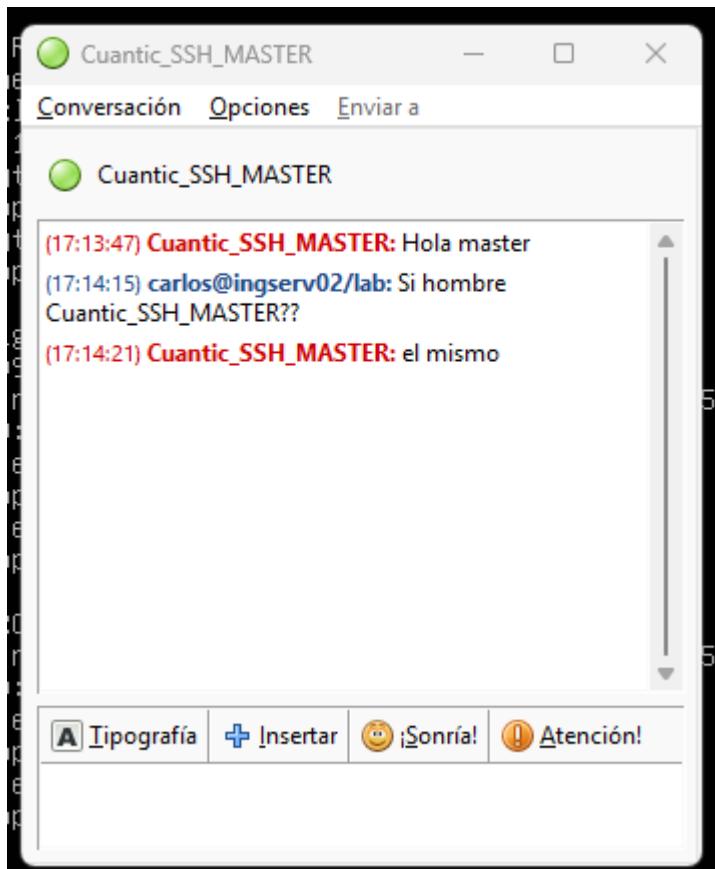


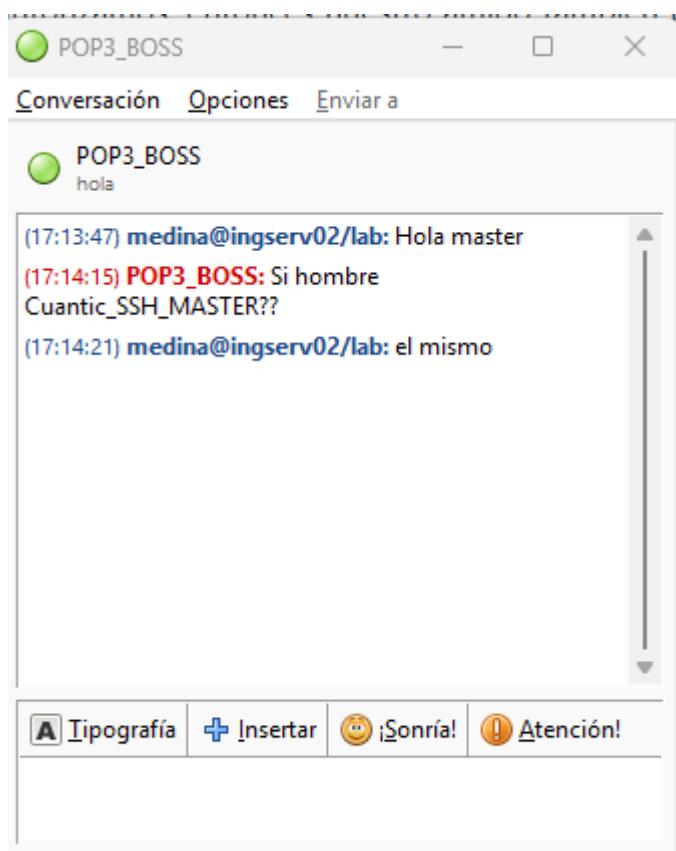
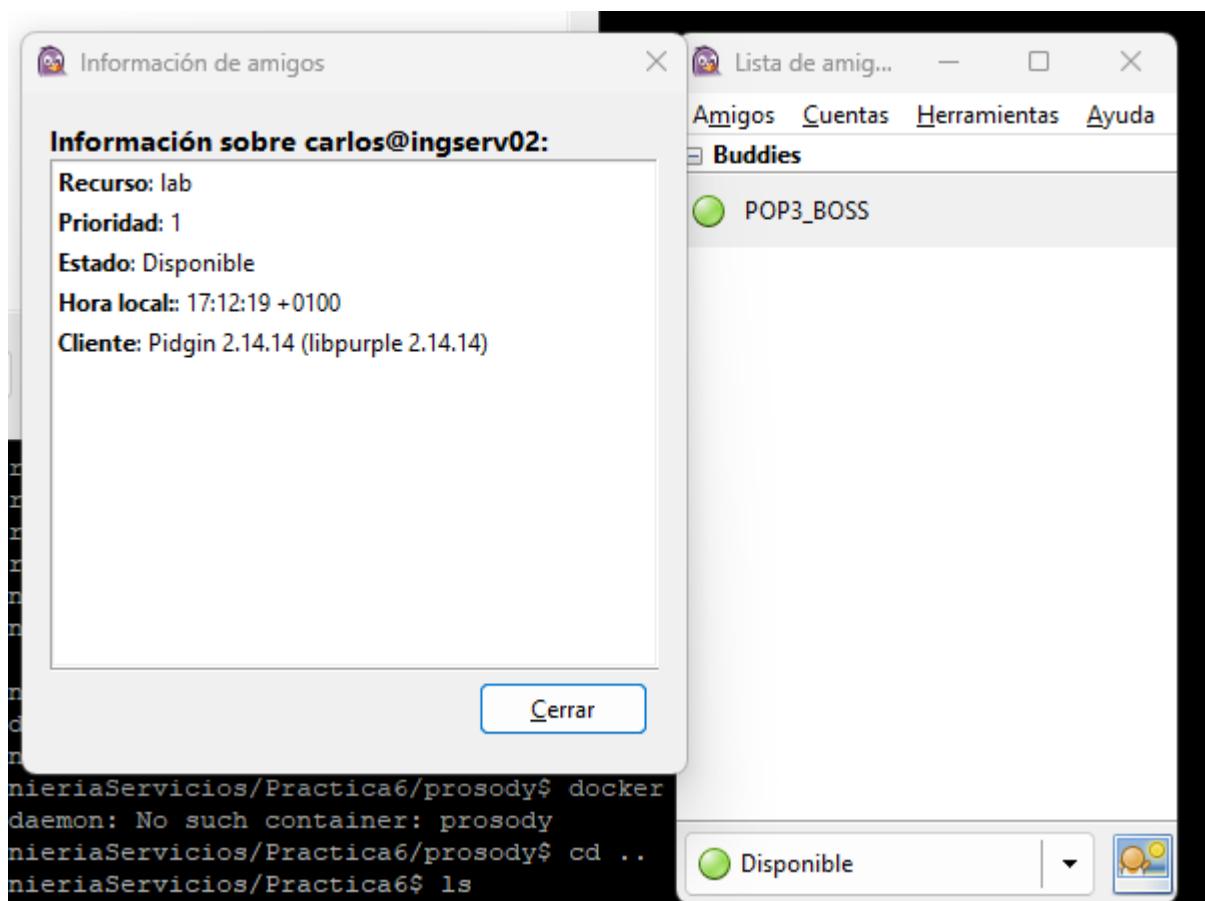
## EJERCICIO 3

### Probando a chatear

**Prueba a cambiar tu estado entre las diferentes opciones (el desplegable de la parte inferior de Pidgin) y comprueba cómo tu compañero de grupo recibe correctamente las notificaciones de tu cambio de estado.**

Prueba a iniciar un *chat* con tu compañero y comprueba que los mensajes llegan correctamente. Observa también cómo mientras uno de los dos está escribiendo, o hace una pausa, al otro le llegan notificaciones de su cambio de estado del chat.





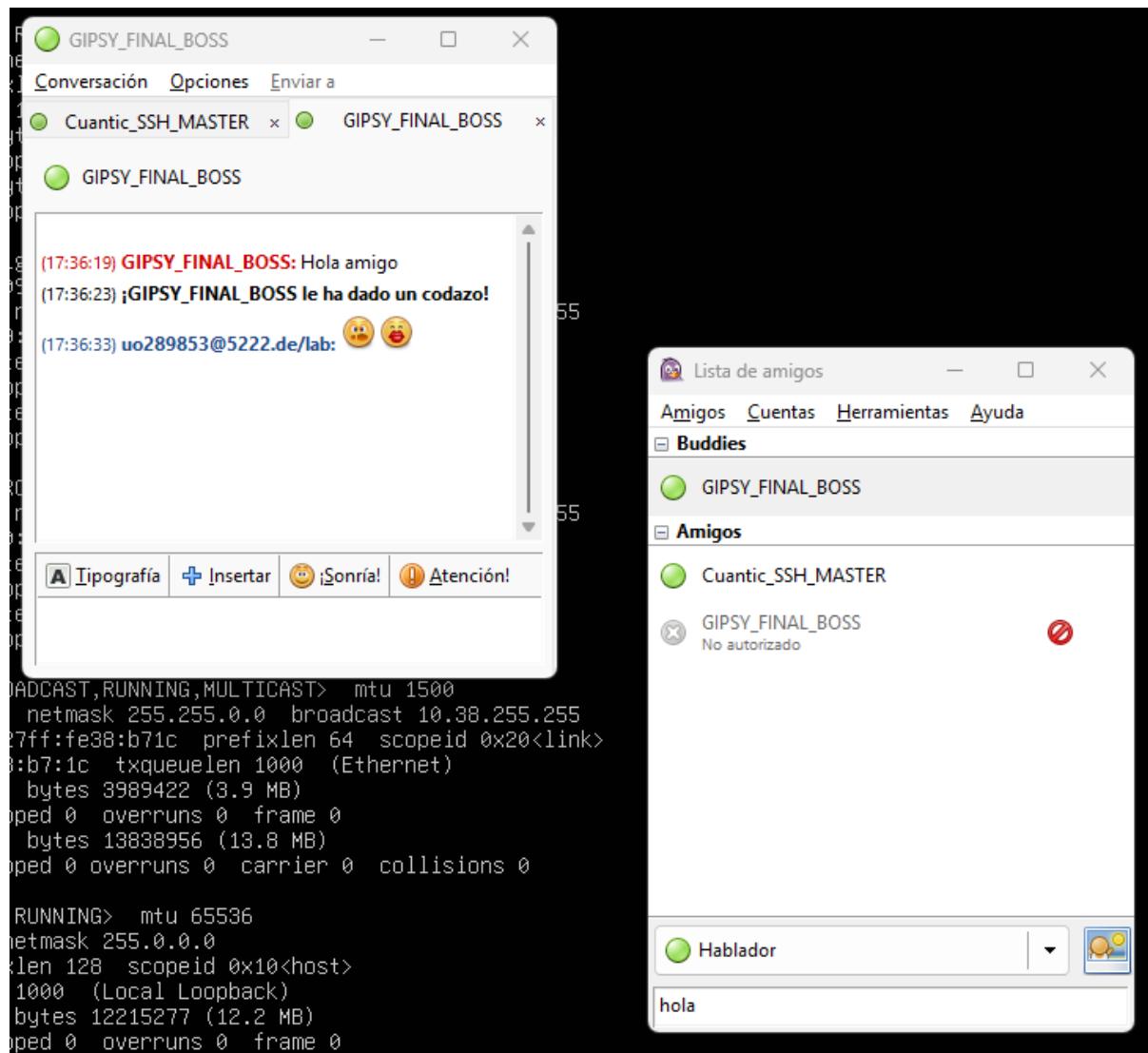
## EJERCICIO 4

Tuvimos problemas al reenviar los mensajes a todos los clientes, ya que el emisor recibía una copia de su propio mensaje. Lo solucionamos comparando la dirección del socket emisor antes de realizar el reenvío, asegurándonos de que la información solo llegara al resto del grupo.

### Servidor XMPP público

Elige un servidor donde crear una cuenta. [En esta página](#) tienes una lista de ellos. Procura elegir un servidor distinto al de tu compañero de grupo, para comprobar cómo los mensajes se comunican correctamente entre dos servidores diferentes. Para usuarios de habla hispana se tienen [jabberes.org](#) y [suchat.org](#), pero no es requisito que uses ninguno de estos dos. Al estar todos federados, no importa en cuál abras la cuenta, pues podrás chatear también con usuarios de otros servidores. En muchos de estos servidores, para darse de alta se usa el propio cliente (pidgin) en lugar de un registro vía web.

Una vez te has dado de alta en uno de ellos, añade la cuenta a pidgin, a través del menú **Cuentas → Gestionar cuentas** (o Ctrl-A). Sigue los mismos pasos que cuando añadiste tu primera cuenta en el servidor ingservXX, pero añade ahora la cuenta que hayas creado en un servidor público.





DARKNET.IM

Home

Register

Account

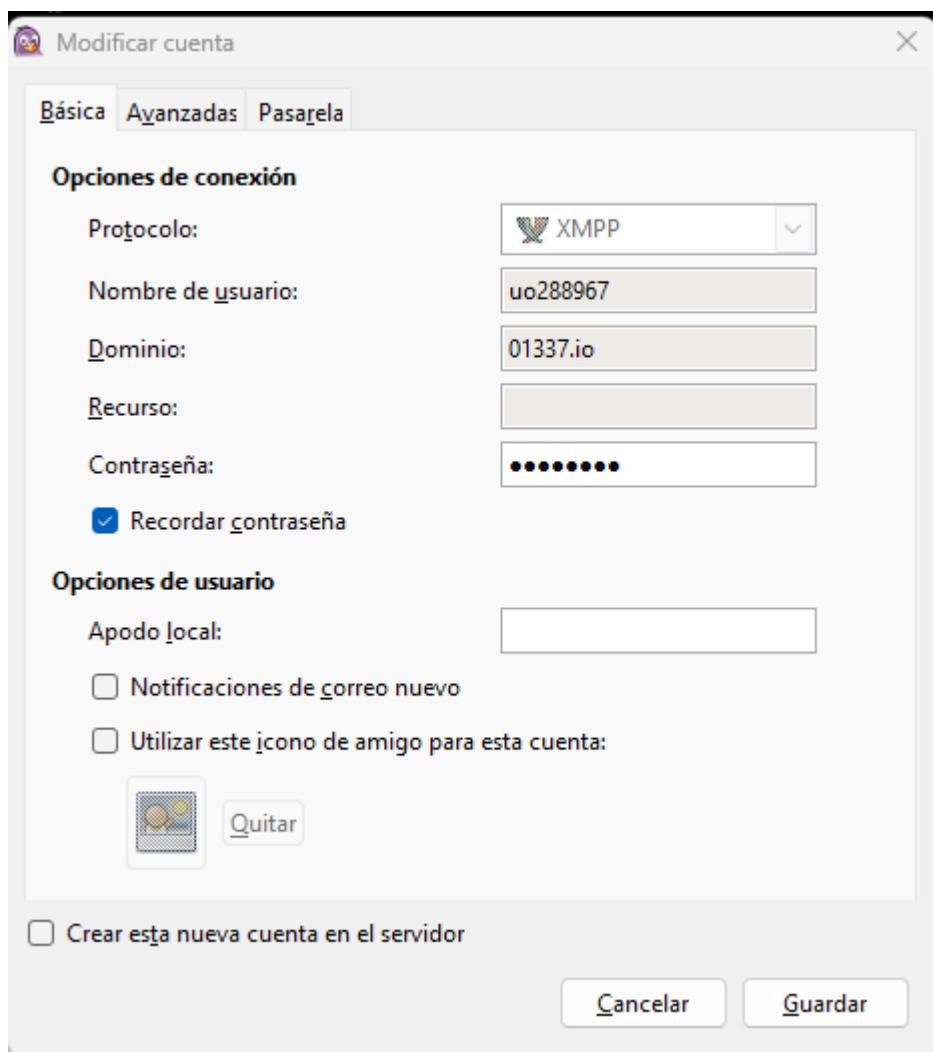
Privacy

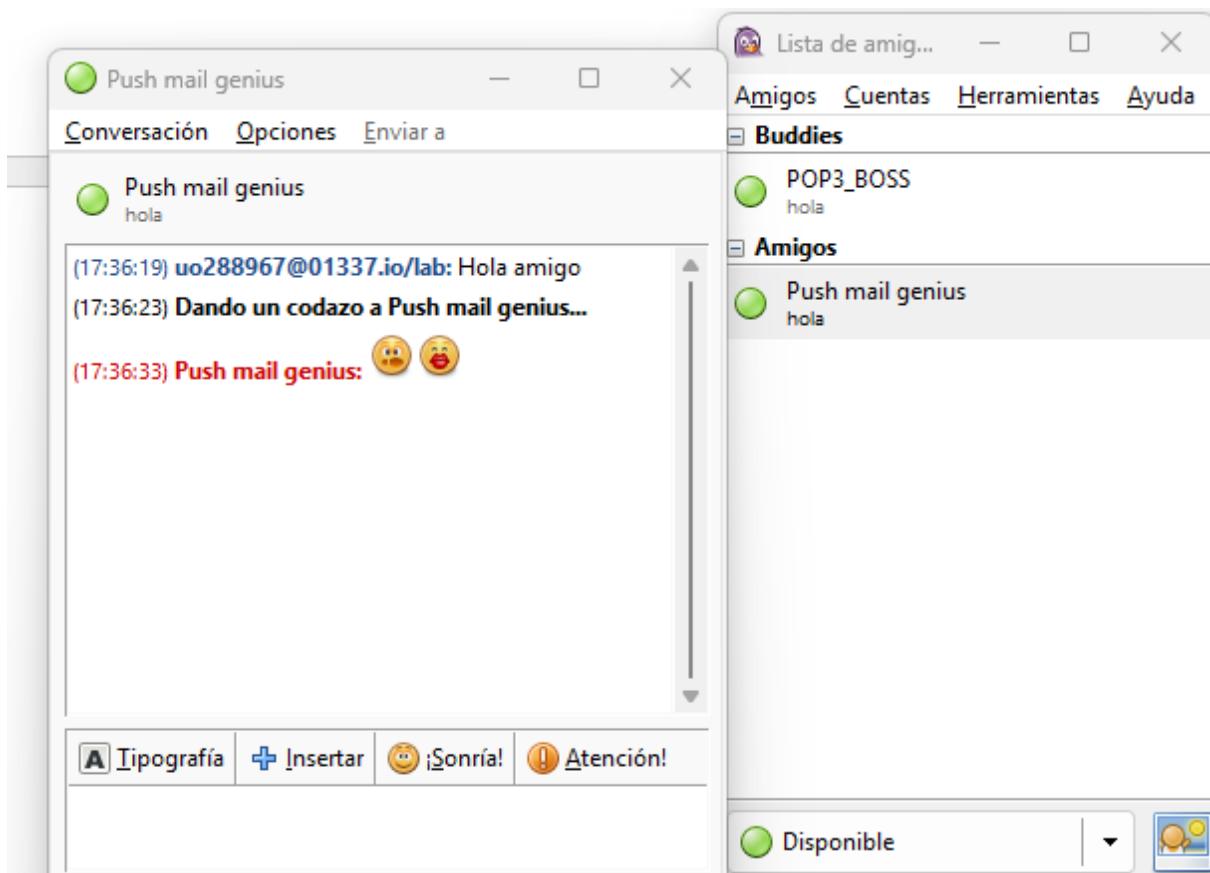
# Create an account

**Create account**

Account created successfully!

[01337.io](http://01337.io)



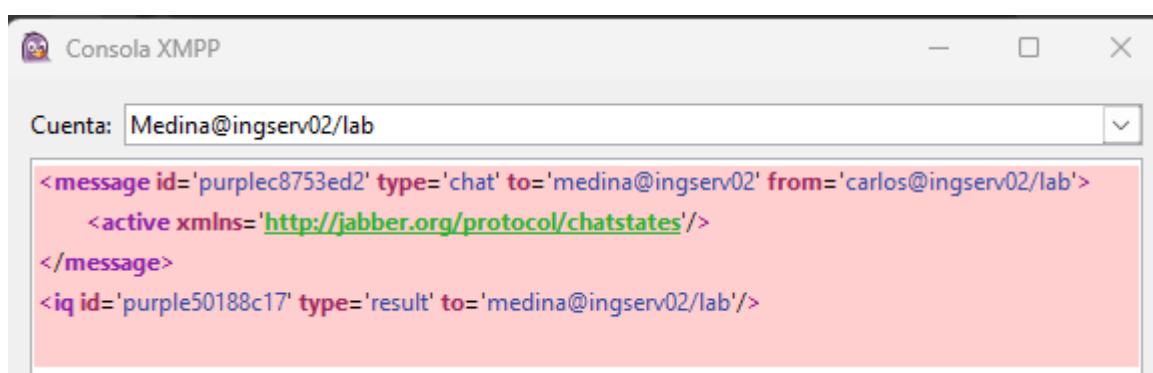
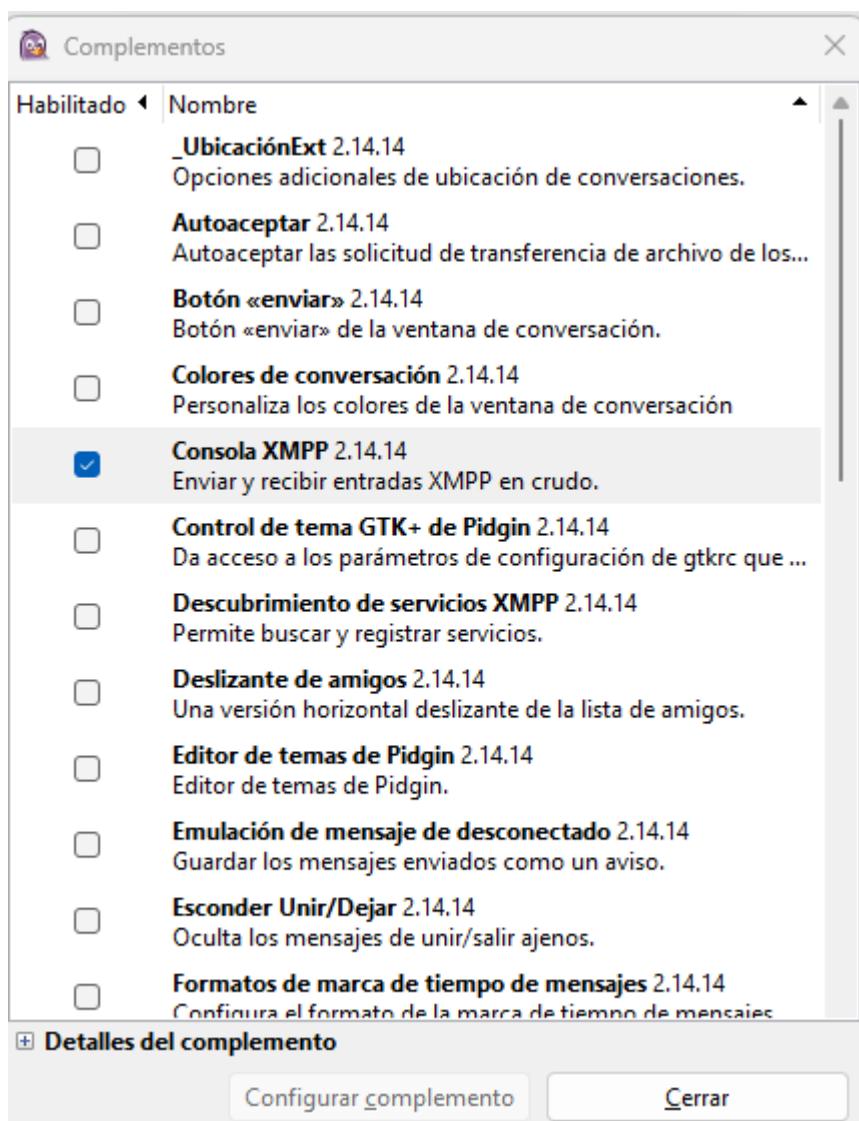


## EJERCICIO 5

### Stanzas en XML

Activa la consola XMPP que te permitirá observar las stanzas transmitidas entre cliente y servidor. Para ello ve al menú **Herramientas** → **Complementos** (o pulsa **Ctrl-U**) y marca el complemento **Consola XMPP**, como muestra la siguiente figura

Una vez hecho esto, el menú para acceder a esta consola estará en **Herramientas** → **Consola XMPP** como muestra la figura siguiente



### Ejercicio 5.1

Crea un nuevo entorno virtual para trabajar con xmpp, mediante el comando:

```
$ python3 -m venv ~/xmppenv
```

Actívalo mediante el comando:

```
$ source ~/xmppenv/bin/activate
```

```
(xmppenv) $
```

Observarás que el *prompt* del sistema ha cambiado y pone ahora (xmppenv) como recordatorio de que estás trabajando en ese entorno virtual. Las bibliotecas de terceros que instales a partir de ahora irán a parar a esa carpeta y no entrarán en conflicto con las del sistema. El desarrollo de aplicaciones python que usen ese entorno puedes hacerlo en cualquier otra carpeta. Con tal de que el entorno virtual esté “activado” cada vez que invoques a python se ejecutará el allí instalado y la aplicación funcionará con las bibliotecas que hubieras instalado en ese entorno.

Para salir del entorno virtual y volver a la instalación global de python basta poner deactivate en cualquier momento. Recuerda que para volver a hacer uso de las bibliotecas instaladas en ese entorno debes volver a activarlo con source ~/xmppenv/bin/activate.

Con el entorno activado, instala slixmpp, versión 1.8.6 usando pip:

```
(xmppenv)$ pip install wheel slixmpp==1.8.6
```

Para comprobar que ha quedado correctamente instalado puedes probar a importar el módulo desde el intérprete python:

```
(xmppenv) $ python3
```

```
>>> import slixmpp
```

```
>>>
```

no deberían producirse errores

```

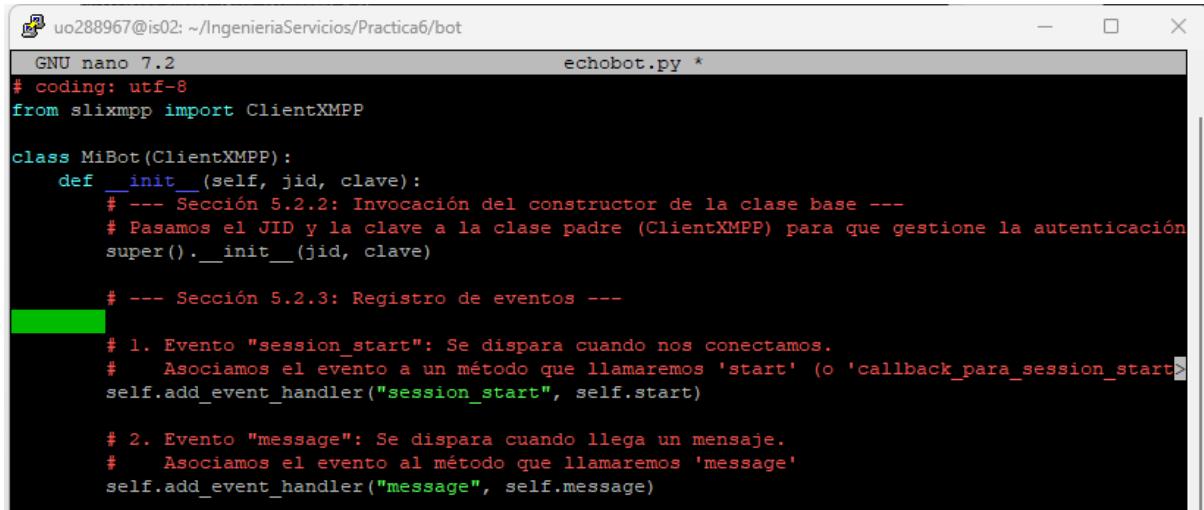
uo288967@is02:~/IngenieriaServicios/Practica6$ python3 -m venv ~xmppenv
uo288967@is02:~/IngenieriaServicios/Practica6$ source ~xmppenv/bin/activate
(xmppenv) uo288967@is02:~/IngenieriaServicios/Practica6$ pip install wheel slixmpp==1.8.6
Collecting wheel
  Using cached wheel-0.45.1-py3-none-any.whl.metadata (2.3 kB)
Collecting slixmpp==1.8.6
  Downloading slixmpp-1.8.6.tar.gz (581 kB)
    581.6/581.6 kB 2.6 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Collecting aiodns>=1.0 (from slixmpp==1.8.6)
  Downloading aiodns-3.5.0-py3-none-any.whl.metadata (5.8 kB)
Collecting pyasn1 (from slixmpp==1.8.6)
  Downloading pyasn1-0.6.1-py3-none-any.whl.metadata (8.4 kB)
Collecting pyasn1_modules (from slixmpp==1.8.6)
  Downloading pyasn1_modules-0.4.2-py3-none-any.whl.metadata (3.5 kB)
Collecting pycares>=4.9.0 (from aiodns>=1.0->slixmpp==1.8.6)
  Downloading pycares-4.11.0-cp312-cp312-manylinux_2_28_x86_64.whl.metadata (4.5 kB)
Collecting cffi>=1.5.0 (from pycares>=4.9.0->aiodns>=1.0->slixmpp==1.8.6)
  Downloading cffi-2.0.0-cp312-cp312-manylinux2014_x86_64_manylinux_2_17_x86_64.whl.metadata (2.6 kB)
Collecting pycparser (from cffi>=1.5.0->pycares>=4.9.0->aiodns>=1.0->slixmpp==1.8.6)
  Downloading pycparser-2.23-py3-none-any.whl.metadata (993 bytes)
Using cached wheel-0.45.1-py3-none-any.whl (72 kB)
Downloading aiodns-3.5.0-py3-none-any.whl (8.1 kB)
Downloading pyasn1-0.6.1-py3-none-any.whl (83 kB)
  83.1/83.1 kB 1.2 MB/s eta 0:00:00
Downloading pyasn1_modules-0.4.2-py3-none-any.whl (181 kB)
  181.3/181.3 kB 2.9 MB/s eta 0:00:00
Downloading pycares-4.11.0-cp312-cp312-manylinux_2_28_x86_64.whl (641 kB)
  641.1/641.1 kB 8.3 MB/s eta 0:00:00
Downloading cffi-2.0.0-cp312-cp312-manylinux2014_x86_64_manylinux_2_17_x86_64.whl (219 kB)
  219.6/219.6 kB 3.8 MB/s eta 0:00:00
Downloading pycparser-2.23-py3-none-any.whl (118 kB)
  118.1/118.1 kB 1.9 MB/s eta 0:00:00
Building wheels for collected packages: slixmpp
  Building wheel for slixmpp (pyproject.toml) ... done
  Created wheel for slixmpp: filename=slixmpp-1.8.6-py3-none-any.whl size=518471 sha256=5606b64ef256d594220dafb94c8805ae488f522859d0ff8065d4d8039665cfb
  Stored in directory: /home/uo288967/.cache/pip/wheels/3d/28/82/e42f763e07b2ee302786097d07c8958alef4ddcb138f82b51f
Successfully built slixmpp
Installing collected packages: wheel, pycparser, pyasn1, pyasn1_modules, cffi, pycares, aiodns, slixmpp
Successfully installed aiodns-3.5.0 cffi-2.0.0 pyasn1-0.6.1 pyasn1_modules-0.4.2 pycares-4.11.0 pycparser-2.23 slixmpp-1.8.6 wheel-0.45.1
(xmppenv) uo288967@is02:~/IngenieriaServicios/Practica6$ python3
Python 3.12.3 (main, Aug 14 2025, 17:47:21) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import slixmpp
Using slower stringprep, consider compiling the faster cython/libidn one.
>>> 
```

**Crea una carpeta llamada bot (hermana de las carpetas etc y data creadas anteriormente) y entra en ella para trabajar. Todos los scripts relacionados con la programación del bot irán en esta carpeta.**

```

(xmppenv) uo288967@is02:~/IngenieriaServicios/Practica6$ ls
bot  data  etc  lanza-prosody.sh
(xmppenv) uo288967@is02:~/IngenieriaServicios/Practica6$ cd bot/
(xmppenv) uo288967@is02:~/IngenieriaServicios/Practica6/bot$ nano echobot.py
(xmppenv) uo288967@is02:~/IngenieriaServicios/Practica6/bot$ 
```

**Completa el código del constructor.**



```
GNU nano 7.2                               echobot.py *
# coding: utf-8
from slixmpp import ClientXMPP

class MiBot(ClientXMPP):
    def __init__(self, jid, clave):
        # --- Sección 5.2.2: Invocación del constructor de la clase base ---
        # Pasamos el JID y la clave a la clase padre (ClientXMPP) para que gestione la autenticación
        super().__init__(jid, clave)

        # --- Sección 5.2.3: Registro de eventos ---
        # 1. Evento "session_start": Se dispara cuando nos conectamos.
        #   Asociamos el evento a un método que llamaremos 'start' (o 'callback_para_session_start')
        self.add_event_handler("session_start", self.start)

        # 2. Evento "message": Se dispara cuando llega un mensaje.
        #   Asociamos el evento al método que llamaremos 'message'
        self.add_event_handler("message", self.message)
```

Completa el código de callback\_para\_session\_start

```
async def callback_para_session_start(self, evento):
    # --- Callback de inicio (Sección 5.2.4) ---
    # 1. Anunciar presencia (Online)
    self.send_presence()

    # 2. Solicitar roster (lista de contactos)
    # Usamos await para esperar a que el servidor procese la petición
    await self.get_roster()

async def callback_para_message(self, evento):
    # --- Callback de mensajes ---
    # (Aqui pondremos el código de la siguiente sección)
    pass
```

## EJERCICIO 6

programación del bot inicial

Completa el código del bot y ejecútalo. Como IP del servidor puedes poner "localhost". Asegúrate de que prosody se está ejecutando (en un contenedor) y que su puerto 5222 ha sido exportado al host. Tras arrancar, el bot permanecerá en un bucle infinito esperando y procesando eventos. La única forma de terminar ese bucle es matar el proceso, lo que puede hacerse con Ctrl-C.

Si el bot se conecta con éxito al servidor, desde otros clientes como Pidgin podrás ver su estado "online". Si le envías mensajes desde Pidgin, deberás ver que el robot reacciona mostrando información sobre los mismos en la consola (procesados por el callback para el evento message)

```

IngenieríaServicios [SSH: 192.168.1.157] - echobot.py

1 # coding: utf-8
2 import sys
3 import logging
4 import ssl
5 from getpass import getpass
6 from slixmpp import ClientXMPP
7
8 class MiBot(ClientXMPP):
9     def __init__(self, jid, clave):
10         # --- Constructor (Sección 5.2.2) ---
11         super().__init__(jid, clave)
12
13         # --- Registro de eventos (Sección 5.2.3) ---
14         self.add_event_handler("session_start", self.callback_para_session_start)
15         self.add_event_handler("message", self.callback_para_message)
16
17     async def callback_para_session_start(self, evento):
18         # --- Callback de inicio (Sección 5.2.4) ---
19         print("Bot conectado. Enviando presencia...")
20         self.send_presence()
21         await self.get_roster()
22
23     async def callback_para_message(self, evento):
24         # --- Callback de mensajes (Lógica descripta en 5.2.1) ---
25         # Solo procesamos mensajes tipo "chat" o "normal" (ignoramos errores, etc.)
26         if evento['type'] in ('chat', 'normal'):
27             body = evento['body']
28             sender = evento['from']
29
30             # Verificar que el mensaje tenga contenido
31             if body:
32                 print(f"Mensaje recibido de {sender}: {body}")
33
34             # Componer la respuesta
35             respuesta = "¿S?" % body
36
37             # Responder usando el método reply() de la propia stanza recibida
38             evento.reply(respuesta).send()
39             print(f"Respuesta enviada: {respuesta}")
40
41     # --- Programa Principal (Sección 5.2.5) ---
42 if __name__ == '__main__':
43     # Configuración de logs para depuración
44     logging.basicConfig(level=logging.INFO,
45                         format='%(levelname)-8s %(message)s')
46
47     print("---- Configuración del Bot ----")
48
49     # 1. Obtener credenciales
50     # Pedimos el JID y la contraseña al usuario.
51     # Nota: En un entorno real podrías usar os.environ.get('BOT_PASSWORD')
52     jid = input("Introduce el JID del bot (ej: bot@ingser02): ")
53     clave = getpass("Introduce la contraseña del bot: ")
54     ip_servidor = input("Introduce la IP del servidor (o 'localhost'): ")
55
56     # 2. Instanciar el cliente
57     bot = MiBot(jid, clave)
58
59     # 3. Configurar SSL (Contexto personalizado)
60     # Creamos un contexto por defecto
61     ssl_context = ssl.create_default_context()
62
63     # OPCIÓN 1 (Para entorno de pruebas):
64     # Desactivamos la verificación del certificado y del nombre de host
65     # para que acepte el certificado autofirmado de Prosody sin errores.
66     ssl_context.check_hostname = False
67     ssl_context.verify_mode = ssl.CERT_NONE
68
69     # Asignamos el contexto al bot
70     bot.ssl_context = ssl_context
71
72     # Registrar plugins útiles (opcional pero recomendado para compatibilidad)
73     bot.register_plugin('xep_0030') # Service Discovery
74
75     # 4. Conectar al servidor
76     print(f"Conectando a {ip_servidor} en el puerto 5222...")
77     # Si usamos 'localhost', forzamos la tupla (host, puerto)
78     if ip_servidor:
79         bot.connect((ip_servidor, 5222))
80     else:
81         bot.connect()
82
83     # 5. Iniciar el bucle de eventos
84     print("Bot iniciado. Presiona Ctrl+C para salir.")
85     try:
86         # process(forever=True) mantiene el script corriendo escuchando eventos
87         bot.process(forever=True)
88     except KeyboardInterrupt:
89         print("\nDeteniendo el bot...")
90         bot.disconnect()
91

```

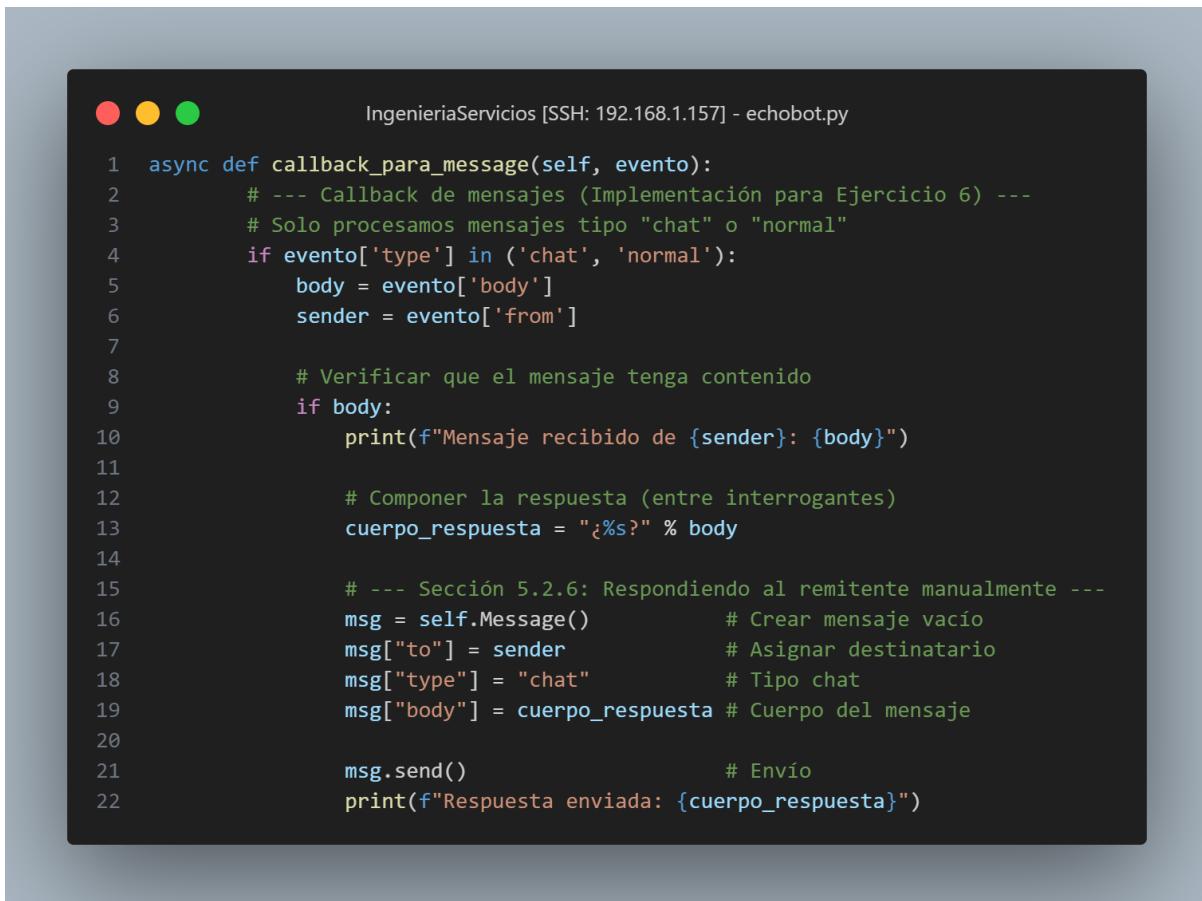
## EJERCICIO 7

añadir eco al bot

Completa el código del método `callback_para_message` de modo que responda como se ha especificado. Recuerda que esta respuesta sólo debe darla a los mensajes de tipo "chat", ignorando los que sean de otro tipo.

Comprueba que funciona, chateando con el bot desde un cliente como Pidgin.

\*Se modifica la siguiente función ( para no volver a enviar todo el código)



```
1  async def callback_para_message(self, evento):
2      # --- Callback de mensajes (Implementación para Ejercicio 6) ---
3      # Solo procesamos mensajes tipo "chat" o "normal"
4      if evento['type'] in ('chat', 'normal'):
5          body = evento['body']
6          sender = evento['from']
7
8          # Verificar que el mensaje tenga contenido
9          if body:
10              print(f"Mensaje recibido de {sender}: {body}")
11
12              # Componer la respuesta (entre interrogantes)
13              cuerpo_respuesta = "?%s?" % body
14
15              # --- Sección 5.2.6: Respondiendo al remitente manualmente ---
16              msg = self.Message()           # Crear mensaje vacío
17              msg["to"] = sender            # Asignar destinatario
18              msg["type"] = "chat"          # Tipo chat
19              msg["body"] = cuerpo_respuesta # Cuerpo del mensaje
20
21              msg.send()                  # Envío
22              print(f"Respuesta enviada: {cuerpo_respuesta}")
```

## EJERCICIO 8

añadir detección de estado al bot

Aumenta el código del bot para que imprima por pantalla mensajes del estilo de:

**manolo@infservn.org** está activo

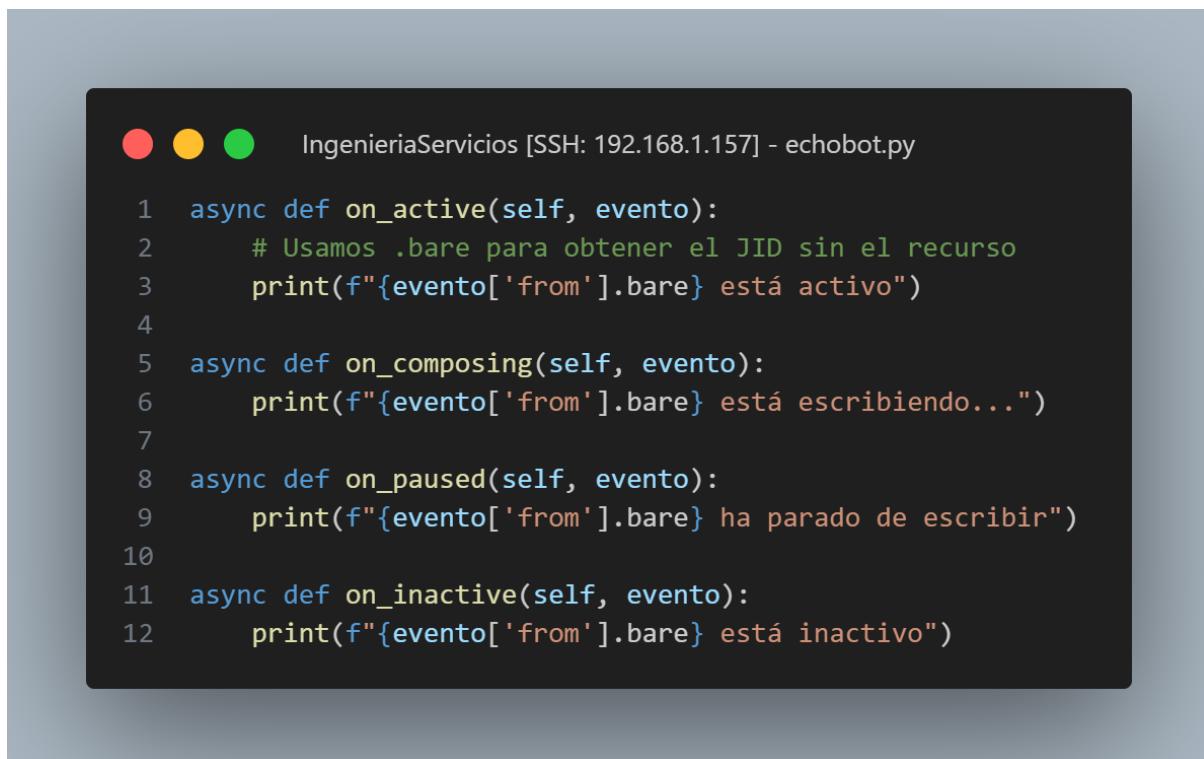
**manolo@infservn.org** está escribiendo...

**manolo@infservn.org** ha parado de escribir

**manolo@infservn.org** está escribiendo...

cuando estos eventos tengan lugar. Para ello sólo necesitas saber que cada una de las funciones `callback` que asocies con esos eventos recibirán como parámetros `self` y un evento del cual puedes extraer el campo `from` como se ha visto en el caso del mensaje.

Por defecto, `evento["from"]` te da el *full JID* del remitente, incluyendo su *recurso* (el identificador del dispositivo desde el cual escribe). Esto es conveniente para enviarle la respuesta al mismo dispositivo desde el que nos escribió, si es que está en varios. Pero si quieras sacar por pantalla lo que es solamente su *bare JID* (sin el nombre de recurso), puedes usar `evento["from"].bare`



```
● ● ● IngenieriaServicios [SSH: 192.168.1.157] - echobot.py

1  async def on_active(self, evento):
2      # Usamos .bare para obtener el JID sin el recurso
3      print(f"{evento['from'].bare} está activo")
4
5  async def on_composing(self, evento):
6      print(f"{evento['from'].bare} está escribiendo...")
7
8  async def on_paused(self, evento):
9      print(f"{evento['from'].bare} ha parado de escribir")
10
11 async def on_inactive(self, evento):
12     print(f"{evento['from'].bare} está inactivo")
```

## EJERCICIO 9

el bot sabe calcular

Puedes hacer un bot ligeramente más útil. Por ejemplo, uno que evalúe expresiones matemáticas y te responda con el resultado. Puedes hacer que esta funcionalidad se dispare sólo ante mensajes que comiencen por el signo `=`, de modo que si en el chat le pones `=2**10` te responda con `1024`.

Python tiene la función `eval(cadena)` que evalúa la cadena en cuestión como una expresión python y te devuelve el resultado. En general se considera peligroso el uso de esta función, ya que no sabes lo que el usuario enviará en el mensaje y te expones por tanto a inyección de código (un usuario podría pedirte una operación que tarde mucho en computarse, o que requiera tanta memoria que rompa el bot, o que comprometa la seguridad<sup>2</sup>). No debe usarse `eval()` en un servidor en producción, si bien implementar una solución segura no es asunto trivial. De todas formas, como experimento de prácticas puede ser adecuado. Una vez `eval()` te ha retornado el valor de la expresión, no tienes más que convertir ese resultado en un `string` y enviarlo como `body` en un mensaje de respuesta al remitente.

Para ello hemos modificado la función callback\_para\_message de la siguiente manera:

```
● ● ● IngenieriaServicios [SSH: 192.168.1.157] - echobot.py

1 if cuerpo.startswith("="):
2     # Extraer la expresión (quitando el '=')
3     expresion = cuerpo[1:]
4     try:
5         resultado = str(eval(expresion))
6         cuerpo_respuesta = f"Resultado: {resultado}"
7     except Exception as e:
8         cuerpo_respuesta = f"Error al calcular: {e}"
9     else:
10        # Comportamiento anterior (Eco)
11        cuerpo_respuesta = "%s" % cuerpo
```

## SERVICIOS PARA MOVILES

### PRACTICA 7

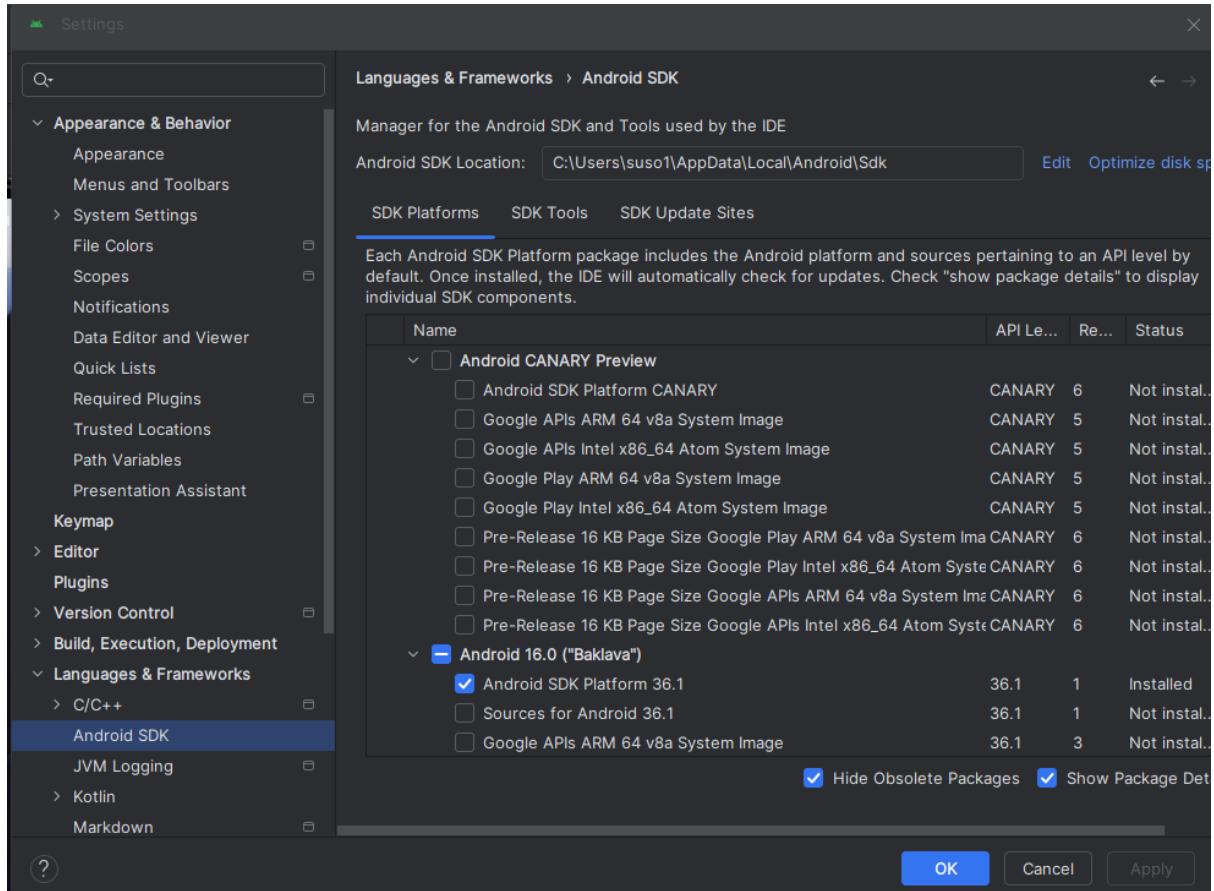
En esta práctica vimos el desarrollo de aplicaciones para dispositivos móviles utilizando **Android Studio** y el lenguaje **Kotlin**. El objetivo principal fue crear un conversor de divisas funcional que consume datos en tiempo real de una API externa. Aprendimos a diseñar interfaces de usuario y a gestionar las comunicaciones de red de forma eficiente mediante la biblioteca **Retrofit**, utilizando corrotinas para evitar el bloqueo del hilo principal de la interfaz. Además, aplicamos el patrón de diseño **MVVM** (Model-View-ViewModel) junto con **LiveData**, lo que nos permitió separar la lógica de negocio de la interfaz y asegurar que la aplicación mantenga su estado y los datos actualizados incluso ante cambios de configuración, como la rotación de la pantalla.

**Abre Android Studio. Es posible que descargue componentes adicionales la primera vez. Si te pide instalar un hipervisor (HAXM) y no tienes permisos, cancela la opción: en los laboratorios ya está instalado.**

Crea un nuevo proyecto seleccionando “*New Project*” y elige la plantilla **Empty Views Activity** para “*Phone and Tablet*”. Usa **Kotlin** como lenguaje y selecciona la opción de configuración **Gradle** en **Kotlin DSL** (`build.gradle.kts`).

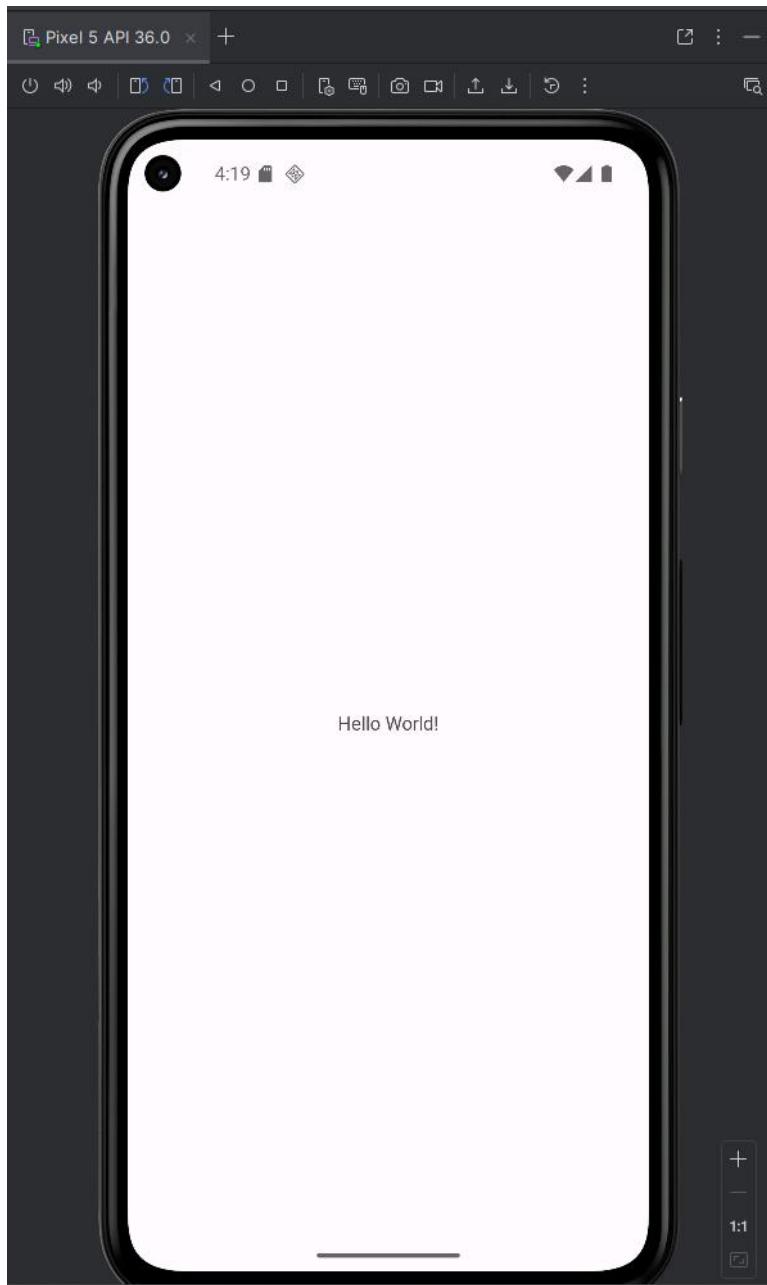
Cuando termine la creación, observa la barra de progreso de **Gradle** en la parte inferior y espera a que finalicen todas las tareas.

Accede a **Tools > SDK Manager** para comprobar las versiones instaladas del **SDK**. Marca **Show Package Details** y verifica que tienes una versión reciente (por ejemplo, **Android SDK Platform 35**).



Abre ahora el **Device Manager**. En esta aplicación se configuran los emuladores. Si tienes alguno creado aparecerá una ventana similar a la que se muestra a continuación, situado en la parte derecha de la interfaz.





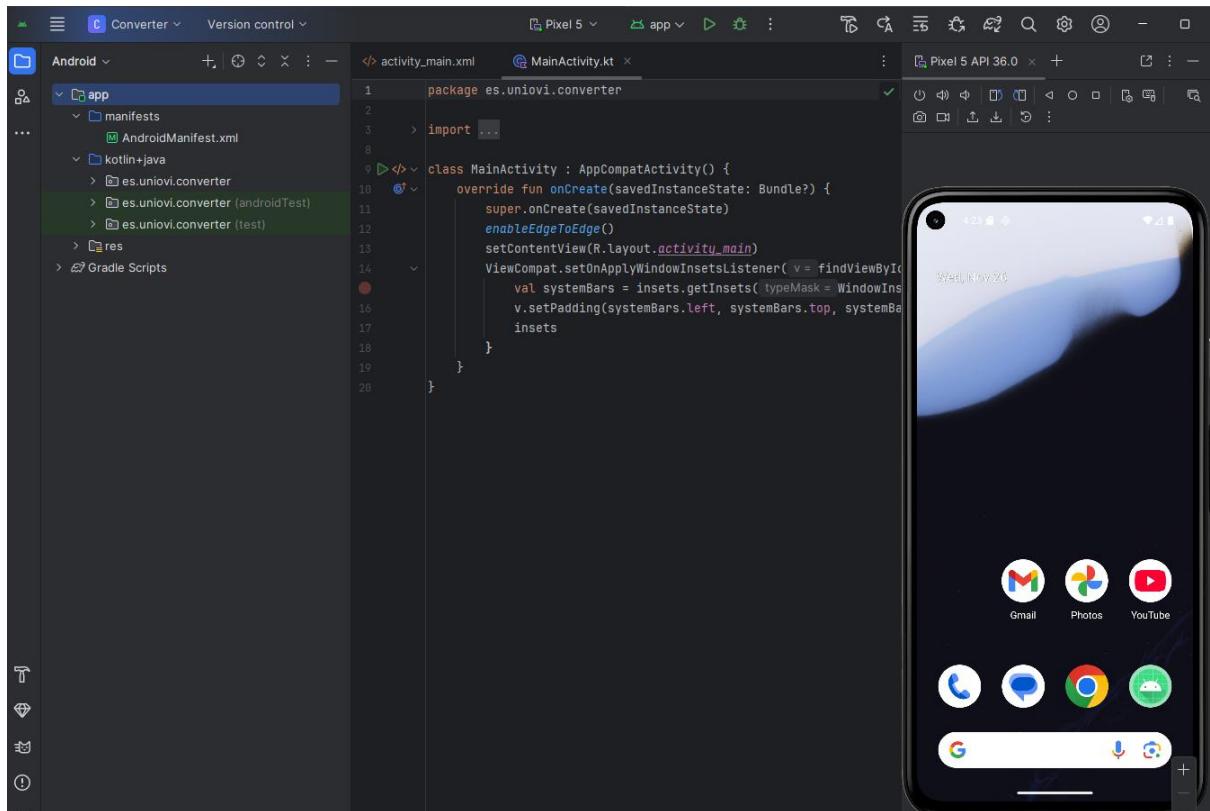
Dentro del menú *File* de Android Studio selecciona *New > New Project*.

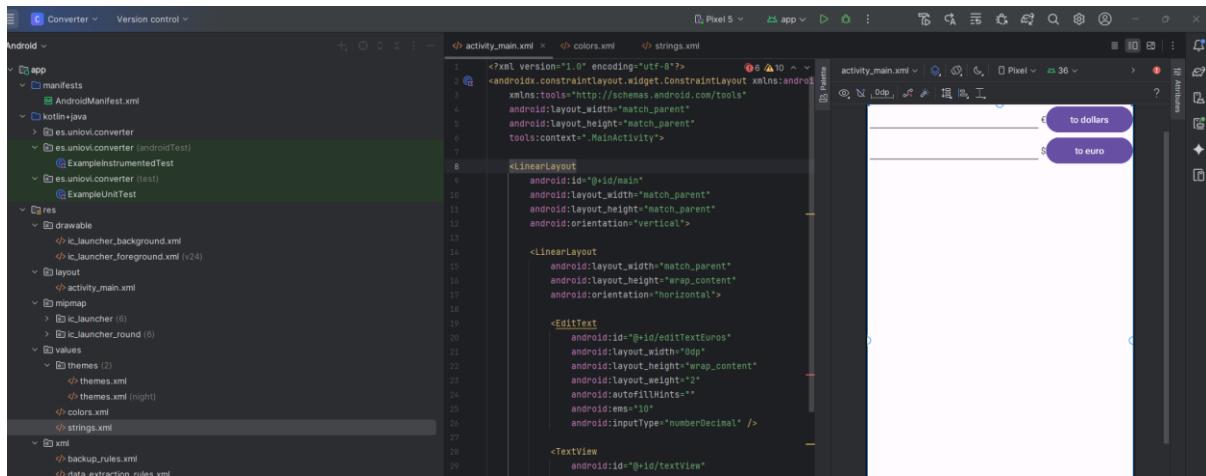
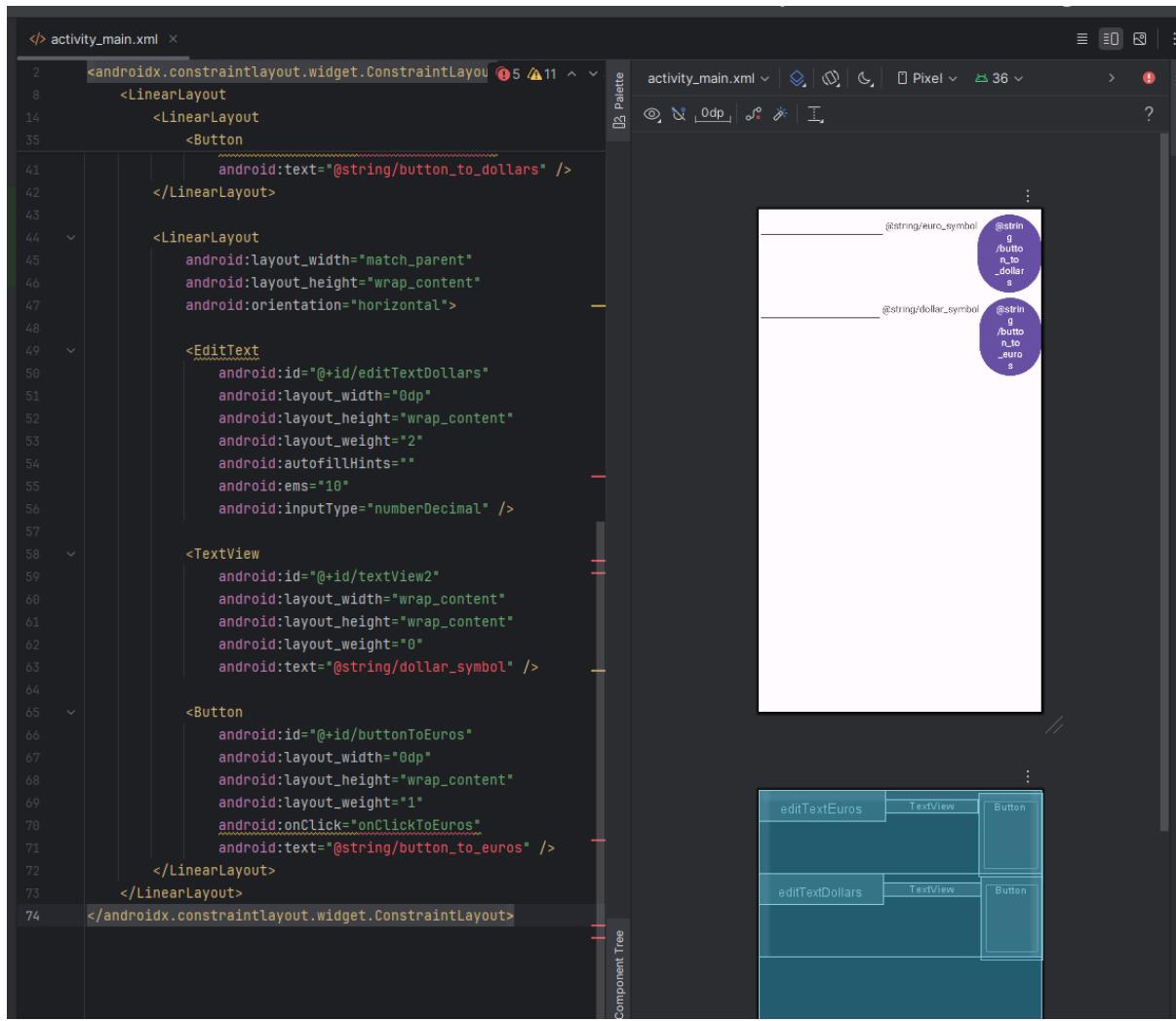
Lo primero será elegir el tipo de aplicación **Empty Views Activity**, tras lo cual aparecerá un formulario donde debes seleccionar diversas opciones.

Como primer ejemplo vamos a realizar un conversor de moneda entre euros y dólares, por tanto el nombre de la aplicación será **Converter**, y el nombre de paquete será **es.uniovi.converter**. Asegúrate de que el lenguaje seleccionado es **Kotlin**.

Para nuestro ejemplo vamos a elegir la versión **API 21**, correspondiente a **Android 5.0 Lollipop**. Si elegimos versiones más antiguas, la aplicación podrá correr en más dispositivos que aún no tengan **Android 5.0**, pero habrá funciones que no podremos utilizar. Por el contrario, si marcamos una versión más moderna tendremos acceso a las últimas novedades de la API, pero habrá menos dispositivos que puedan ejecutarlo (sólo los que tengan esa versión o posterior).

**La API 21 es un buen compromiso, al soportar las funcionalidades que necesitaremos para la práctica y a la vez poder funcionar en prácticamente todos los dispositivos, según reporta Android Studio. Tras seguir estos pasos, tendrías una configuración de proyecto como:**





**Declara las variables que se han indicado antes, en la clase MainActivity, justo después de la línea que declara la clase.**

A continuación, dentro del método onCreate, que se ejecuta cuando se crea la actividad, añade la inicialización de las variables editTextEuros y editTextDollars, usando findViewById, como se muestra a continuación:

```
editTextEuros = findViewById(R.id.editTextEuros)  
editTextDollars = findViewById(R.id.editTextDollars)
```

La función findViewById permite obtener una referencia a partir del identificador del recurso.

Añade el método convert que reciba como parámetro un EditText fuente de la conversión, un EditText destino de la conversión y el ratio de conversión. Este método realiza la conversión y carga el resultado en el campo destino:

```
private fun convert(source: EditText, destination: EditText, factor: Double) {  
  
    val text = source.text.toString()  
  
    val value = text.toDoubleOrNull()  
  
    if (value == null) {  
  
        destination.setText("")  
  
        return  
  
    }  
  
    destination.setText((value * factor).toString())  
  
}
```

Observa el código fuente del método. No continúes hasta que no comprendas su funcionamiento.

Llama a este método de forma adecuada como respuesta a la pulsación de los botones.

Ejecuta la aplicación y verifica que la conversión funciona correctamente.

The screenshot shows the Android Studio interface. On the left is the code editor for `MainActivity.kt`, which contains Kotlin code for a currency converter. On the right is the emulator window showing a smartphone screen with two text input fields and a button. The top field has '55.0' and the bottom field has '63.8'. A button labeled 'to dollars' is above the top field, and a button labeled 'to euro' is above the bottom field.

```
11 class MainActivity : AppCompatActivity() {
12     private var euroToDollar: Double = 1.16
13     private lateinit var editTextEuros: EditText
14     private lateinit var editTextDollars: EditText
15
16     override fun onCreate(savedInstanceState: Bundle?) {
17         super.onCreate(savedInstanceState)
18         enableEdgeToEdge()
19         setContentView(R.layout.activity_main)
20
21         editTextEuros = findViewById(id = R.id.editTextEuros)
22         editTextDollars = findViewById(id = R.id.editTextDollars)
23
24         ViewCompat.setOnApplyWindowInsetsListener(v = findViewById(id = R.id.main)) { v, insets ->
25             val systemBars = insets.getInsets(typeMask = WindowInsetsCompat.Type.systemBars())
26             v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)
27             insets
28         }
29
30     }
31
32     private fun convert(source: EditText, destination: EditText, factor: Double) {
33         val text = source.text.toString()
34         val value = text.toDoubleOrNull()
35         if (value == null) {
36             destination.setText("")
37             return
38         }
39         destination.setText((value * factor).toString())
40
41     }
42     fun onClickToDollars(view: View) {
43         convert(source = editTextEuros, destination = editTextDollars, factor = euroToDollar)
44     }
45
46     fun onClickToEuro(view: View) {
47         convert(source = editTextDollars, destination = editTextEuros, factor = 1 / euroToDollar)
48     }
49
50     private fun updateUI() {
51         val euros = editTextEuros.text.toString().toDoubleOrNull()
52         val dollars = editTextDollars.text.toString().toDoubleOrNull()
53         if (euros != null & dollars != null) {
54             editTextDollars.setText((euros * euroToDollar).toString())
55             editTextEuros.setText((dollars / euroToDollar).toString())
56         }
57     }
58
59     override fun onWindowFocusChanged(hasFocus: Boolean) {
60         super.onWindowFocusChanged(hasFocus)
61         if (hasFocus) {
62             editTextEuros.requestFocus()
63         }
64     }
65
66     companion object {
67         const val TAG = "MainActivity"
68     }
69 }
```

> Task :app:mergeDebugJavaResource  
> Task :app:packageDebug  
> Task :app:createDebugApkListingFileRedirect

## EJERCICIO 4.1

```
private fun fetchExchangeRate() {
```

```
    val url = URL("https://api.frankfurter.app/latest?from=EUR&to=USD")
    val connection = url.openConnection() as HttpURLConnection
    connection.requestMethod = "GET"
    val inputStream = connection.inputStream
    val reader = BufferedReader(InputStreamReader(inputStream))
    val response = StringBuilder()
    var line: String?
    while (reader.readLine().also { line = it } != null) {
        response.append(line)
    }
    reader.close()
    val jsonResponse = response.toString()
```

```

    val jsonObject = JSONObject(jsonResponse)
    val rates = jsonObject.getJSONObject("rates")
    euroToDollar = rates.getDouble("USD")
}

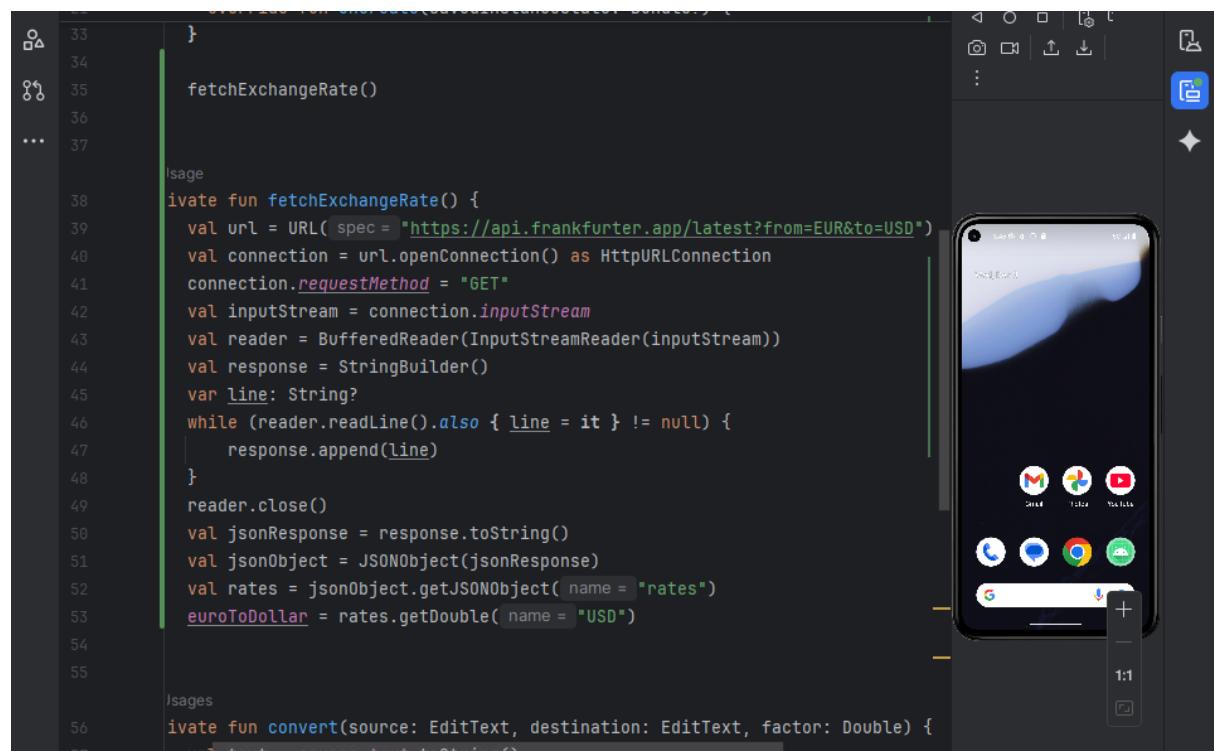
```

Como ves el código es un poco farragoso, al tener que utilizar varias clases Java para recibir la respuesta como cadena y luego parsearla. No obstante, el código es sencillo de entender.

Añade el método antes mostrado a tu clase `MainActivity`, y llámalo desde `onCreate()` para que se ejecute al iniciar la aplicación.

Ejecuta la aplicación. Verás que no funciona. La aplicación se cierra en el teléfono nada más arrancar. En el logcat de Android Studio podrás ver que se ha generado la excepción `NetworkOnMainThreadException`. Básicamente indica que no se puede hacer una operación de red en el hilo principal de la aplicación. Para hacer una operación de red es necesario usar un hilo aparte.

Esto es así porque Android no permite hacer operaciones de red en el hilo principal de la aplicación, que es donde se ejecuta el método `onCreate`. Si se permitiera, la interfaz de usuario se bloquearía mientras se espera la respuesta del servidor, lo que sería una mala experiencia para el usuario.



## EJERCICIO 4.2.1

1. Añade Retrofit y el convertidor Gson en `build.gradle.kts`:

```

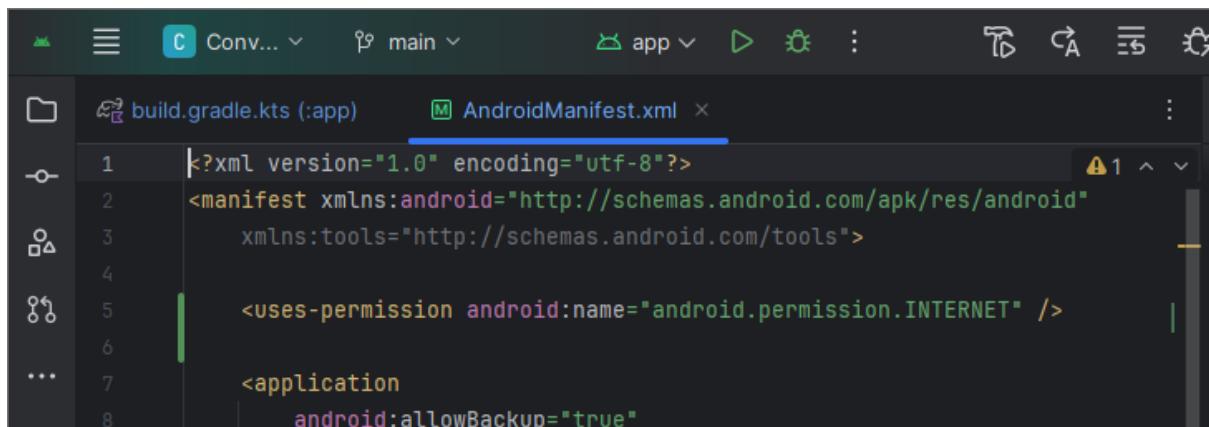
dependencies {
    // Otras dependencias previas...
    implementation("com.squareup.retrofit2:retrofit:2.9.0")
}

```

```
implementation("com.squareup.retrofit2:converter-gson:2.9.0")  
37     }  
38 }  
39  
40 dependencies {  
41     implementation(libs.androidx.core.ktx)  
42     implementation(libs.androidx.appcompat)  
43     implementation(libs.material) 💡  
44     implementation(libs.androidx.activity)  
45     implementation(libs.androidx.constraintlayout)  
46     testImplementation(libs.junit)  
47     androidTestImplementation(libs.androidx.junit)  
48     androidTestImplementation(libs.androidx.espresso.core)  
49  
50     implementation("com.squareup.retrofit2:retrofit:2.9.0")  
51     implementation("com.squareup.retrofit2:converter-gson:2.9.0")  
52 }
```

2. Añade el permiso de Internet en el manifiesto (ponlo justo antes de la etiqueta <application>):

```
<uses-permission android:name="android.permission.INTERNET" />
```



```
<?xml version="1.0" encoding="utf-8"?>  
<manifest xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools">  
    <uses-permission android:name="android.permission.INTERNET" />  
    <application  
        android:allowBackup="true"
```

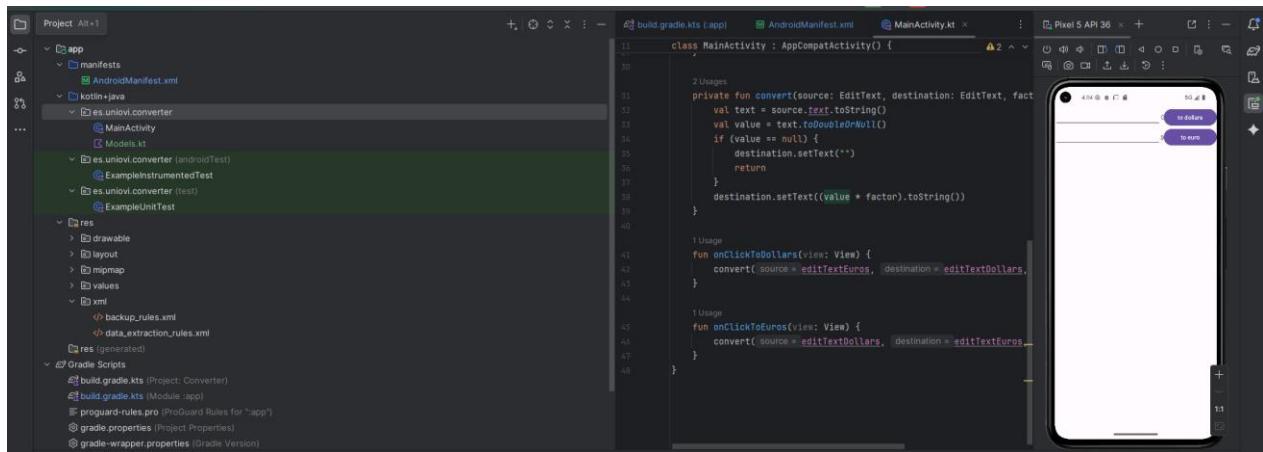
#### EJERCICIO 4.3

Un nuevo archivo, en la misma carpeta en que esté MainActivity.kt llamado Models.kt y añade en él las dos data class que hemos definido antes (Rates y ExchangeRateResponse). Así como la interfaz ExchangeRateApi . Añade en el mismo fichero la clase RetrofitClient que hemos definido.

Resuelve los errores de importación que te salgan (Android Studio te sugerirá las importaciones necesarias). Asegúrate también de que la primera línea declara el paquete correcto (el mismo que el de MainActivity.kt).

Edita MainActivity.kt y quita la llamada a fetchExchangeRate() que habías puesto antes en onCreate() (pues ya vimos que hace que la actividad se aborte). Despues programaremos una versión que evita ese problema.

De momento, vamos a verificar que todo compila sin errores y funciona sin romper, si bien no estamos haciendo uso todavía de nada de lo que hemos programado con Retrofit, y por tanto usará el ratio prefijado de 1.16 que habíamos puesto al principio.



## EJERCICIO 4.4 Y 4.5

Ha llegado el momento de escribir una versión de fetchExchangeRate() que use Retrofit y corutinas para hacer la llamada a la API REST en un hilo aparte, y actualizar la variable euroToDollar cuando se reciba la respuesta, de modo que no rompa la aplicación.

En MainActivity.kt cambia la función fetchExchangeRate() que teníamos por la siguiente:

```
private fun fetchExchangeRate() {
    lifecycleScope.launch {
        try {
            val response = RetrofitClient.api.convert("EUR", "USD", 1.0)
            val exchangeRateResponse = response.body()
            if (!response.isSuccessful || exchangeRateResponse == null) {
                Log.e("MainActivity", "Error al obtener el cambio: ${response.code()}")
            }
            return@launch
        }
        euroToDollar = exchangeRateResponse.rates.USD
        Toast.makeText(
            this@MainActivity,
```

```

        "Cambio actualizado: $euroToDollar",
        Toast.LENGTH_SHORT
    ).show()

    Log.d("MainActivity", "Cambio actualizado: $euroToDollar")

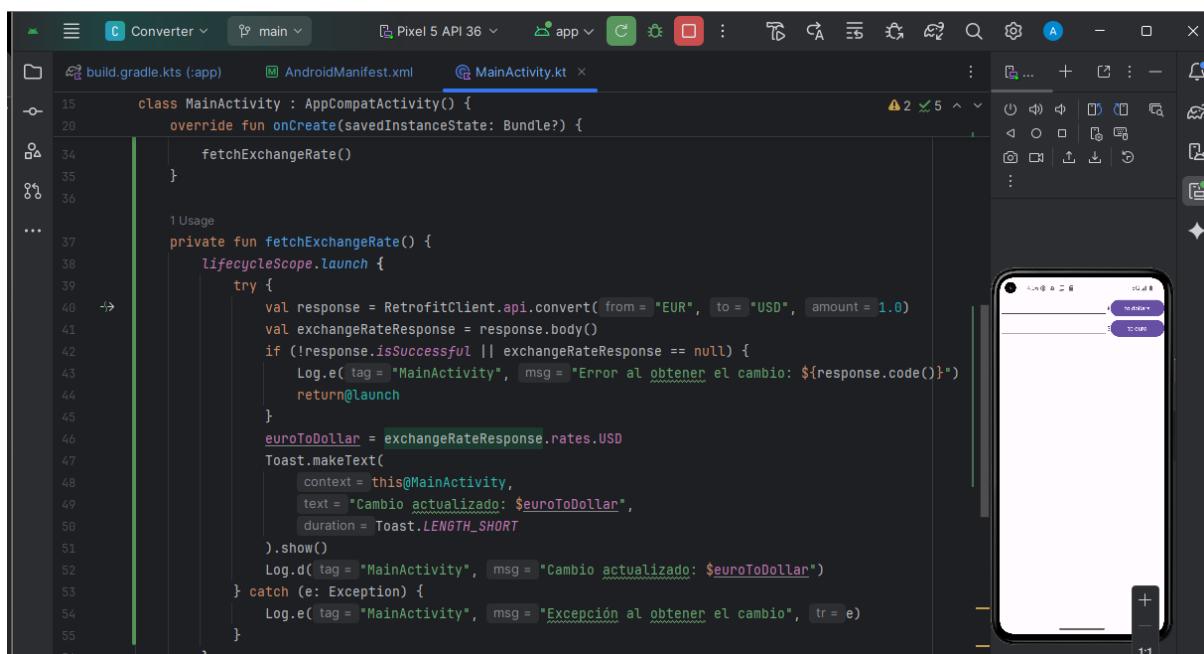
} catch (e: Exception) {

    Log.e("MainActivity", "Excepción al obtener el cambio", e)

}

}

```



## EJERCICIO 5

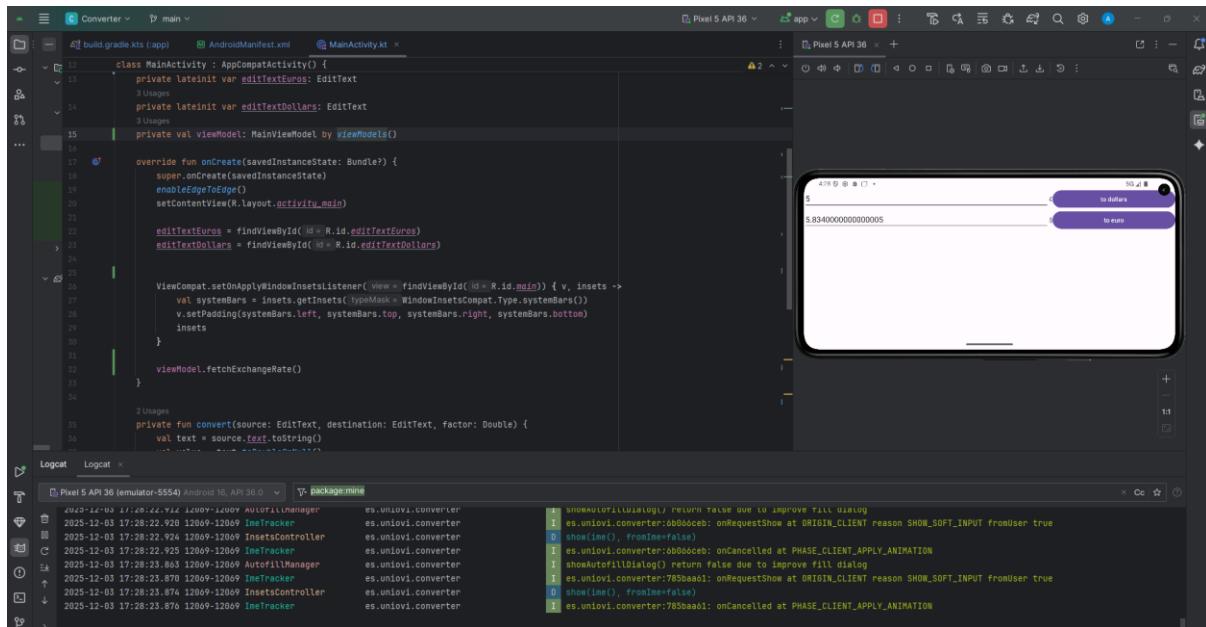
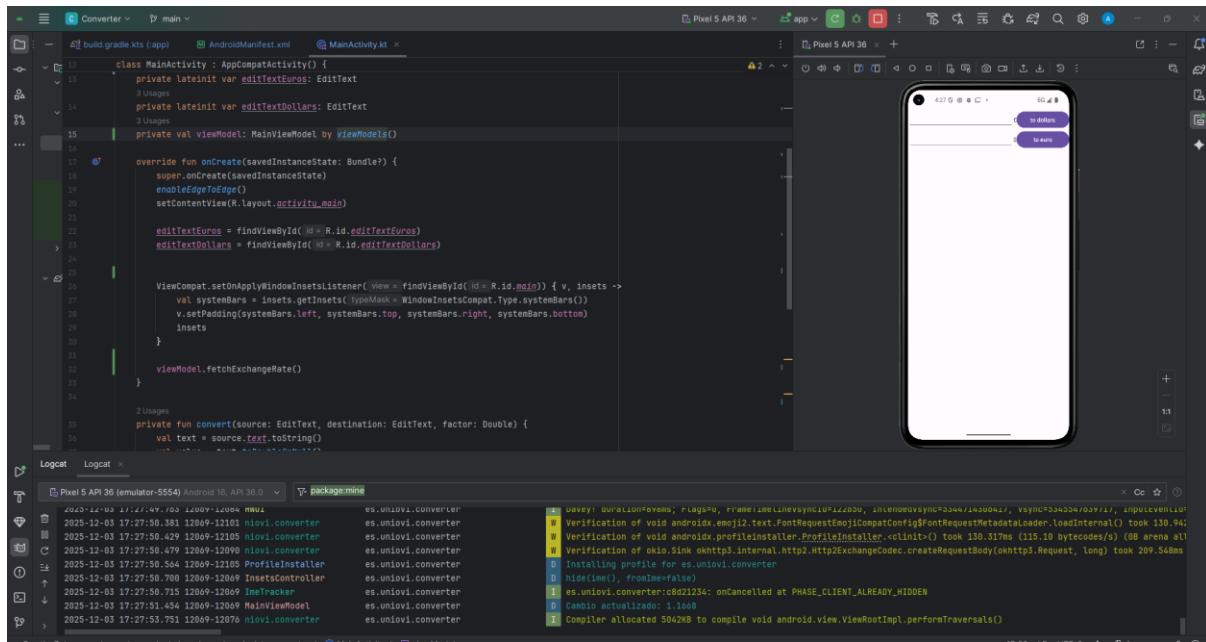
Realiza los cambios explicados, completando el código donde sea necesario para que sea funcional.

Ejecuta la aplicación mientras tienes el logcat abierto. Verás aparecer el mensaje en que se crea el ViewModel y la solicitud de la tasa de cambio. Verifica que los botones funcionan para las conversiones entre euros y dólares.

Ahora gira la pantalla (o el emulador). Verás que los mensajes de creación del ViewModel y de solicitud de tasa de cambio NO vuelven a aparecer. Verifica que los botones siguen funcionando correctamente.

```
49  
50     implementation("com.squareup.retrofit2:retrofit:2.9.0")  
51     implementation("com.squareup.retrofit2:converter-gson:2.9.0")  
52  
53     // ViewModel  
54     implementation("androidx.lifecycle:lifecycle-viewmodel-ktx:2.8.1")  
55     // LiveData (opcional, no lo usaremos de momento)  
56     implementation("androidx.lifecycle:lifecycle-livedata-ktx:2.8.1")  
57     // Activity KTX (para la sintaxis 'by viewModels')  
58     implementation("androidx.activity:activity-ktx:1.9.0")  
59  
60
```

```
build.gradle.kts (app)  AndroidManifest.xml  MainActivity.kt  
12  class MainActivity : AppCompatActivity() {  
13      private lateinit var editTextEuros: EditText  
14      private lateinit var editTextDollars: EditText  
15      private val viewModel: MainViewModel by viewModels()  
16  
17      override fun onCreate(savedInstanceState: Bundle?) {  
18          super.onCreate(savedInstanceState)  
19          enableEdgeToEdge()  
20          setContentView(R.layout.activity_main)  
21  
22          editTextEuros = findViewById(R.id.editTextEuros)  
23          editTextDollars = findViewById(R.id.editTextDollars)  
24  
25          ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main)) { v, insets ->  
26              val systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars())  
27              v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom)  
28              insets  
29          }  
30      }  
31  }  
32
```



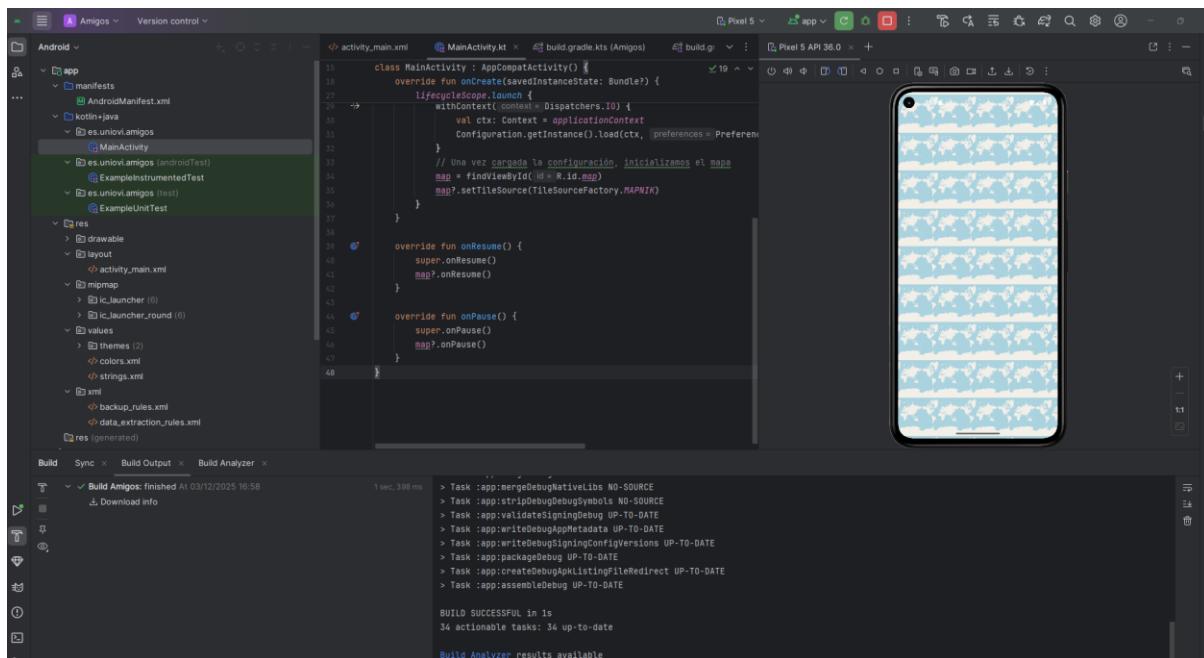
## PRACTICA 7.2

Durante esta practica creamos una aplicación con Kotlin en Android Studio para aprovechar las practicas anteriores de web y enlazar nuestra aplicación basada en un mapa en el que ubicar a nuestros amigos con la base de datos de amigos y la respectiva web. Encontramos problemas con el simulador habiendo veces en las que se ralentizaba mucho y también con las external tools que no solían funcionarnos y teníamos que hacer los cambios de localizaciones de manera manual.

## EJERCICIO 1

- Abre el fichero `res > layout > activity_main.xml`
- En lugar de la vista *Design*, accede a la vista *Code*, donde verás el XML que define el aspecto de la aplicación. Cámbialo todo por este otro:
- **Como ves el layout se compone ahora de un solo elemento de tipo `org.osmdroid.views.MapView` que será donde se mostrará el mapa.**

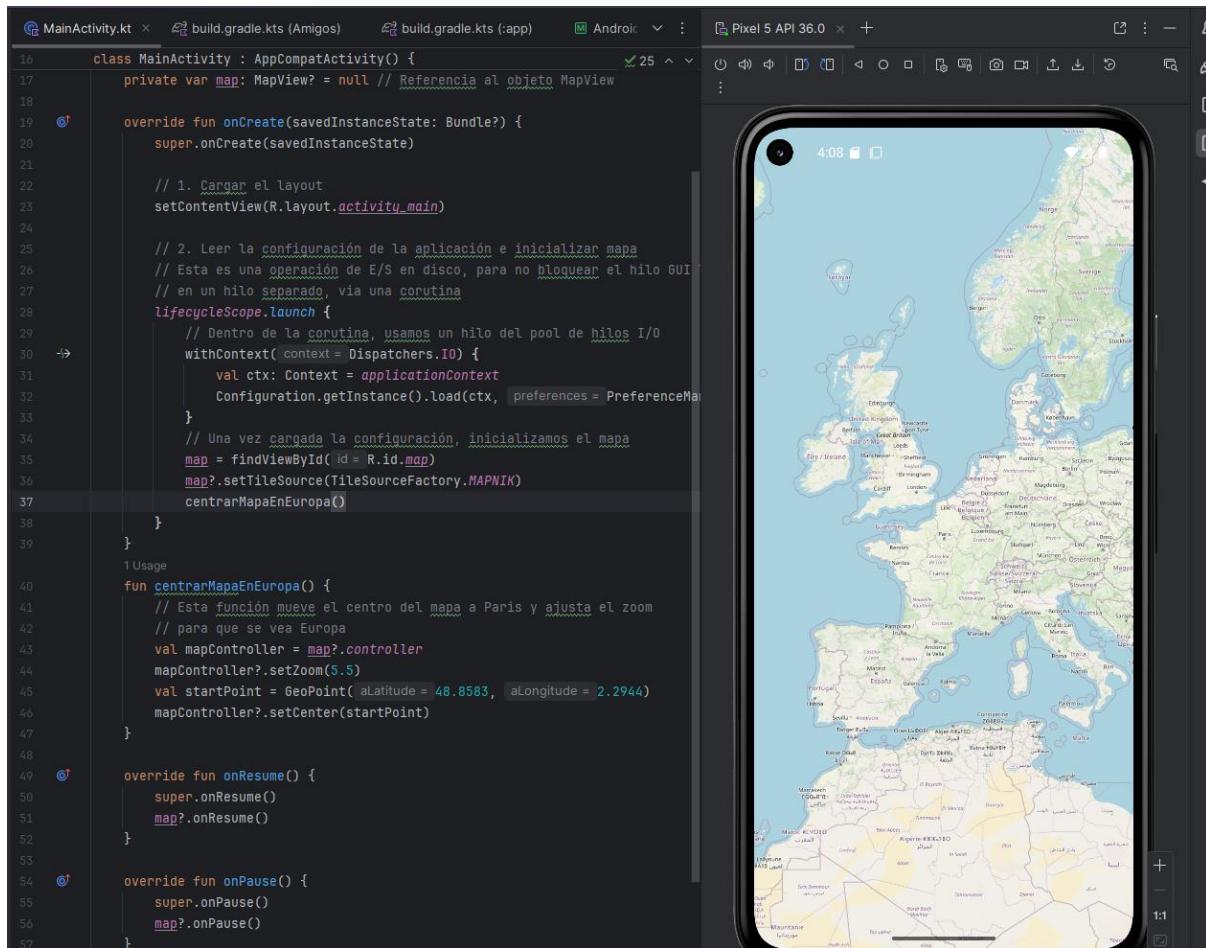
También es necesario cambiar el código de la aplicación. Edita `MainActivity.kt` y reemplaza todo el código que aparece por el siguiente (en Kotlin):



## EJERCICIO 2

Añade el siguiente método dentro de la clase `MainActivity` y resuelve los import que sean necesarios:

Invoca este método desde `onCreate()` (intenta deducir en qué punto de `onCreate()` tienes que insertar la llamada) y ejecuta de nuevo la aplicación. Ahora el mapa inicial debería verse así:



- Instala en Ubuntu ngrok (puedes hacerlo con `sudo snap install ngrok`)
- Ejecuta en la terminal Ubuntu el comando `ngrok http 80`. Puede que necesites configurar una cuenta gratuita en [ngrok.com](https://ngrok.com) y asociar tu cuenta a la instalación local de ngrok mediante un token que te proporcionan en su web. Si es así, sigue las instrucciones que aparecen en la web de ngrok para hacerlo.
- En la pantalla aparecerá una URL del estilo `http://1395db1c124d.ngrok-free.app` (es diferente cada vez que ejecutes ngrok). Esta URL es visible desde cualquier lugar del mundo, y cuando un navegador se conecte a ella, ngrok redirigirá la petición a tu máquina Ubuntu, aunque la tengas en modo NAT y aunque esté ejecutándose detrás de un router NAT.
- Deja esa terminal funcionando (pues si la cierras, la redirección dejará de funcionar también)

```

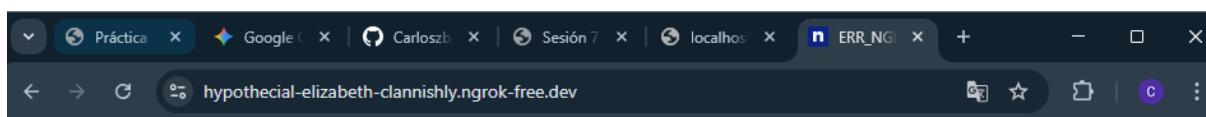
uo289853@is-01: ~
ngrok
(Ctrl+C to quit)

Block threats before they reach your services with new WAF actions - https://ngrok.com/r/waf

Session Status          online
Account                 carlitoszapicobello@gmail.com (Plan: Free)
Update                  update available (version 3.33.1, Ctrl-U to update)
Version                3.33.0
Region                 Europe (eu)
Web Interface          http://127.0.0.1:4040
Forwarding              https://hypothecial-elizabeth-clannishly.ngrok-free.dev -> http://localhost:80

Connections             ttl     opn     rsl     rt5     p50     p90
                        0       0      0.00    0.00    0.00    0.00

```



#### ERR\_NGROK\_8012

Traffic successfully made it to the ngrok agent, but the agent failed to establish a connection to the upstream web service at <http://localhost:80>. The error encountered was:

```
dial tcp 127.0.0.1:80: connect: connection refused
```

[Get help with this error](#)

#### If you're the developer of this page

On the machine where the ngrok agent is running, make sure a service is running on <http://localhost:80>. Try to cURL or open the address in a browser to see that you get the correct response.

Check out the docs to get [help with this error](#).

#### If you're a visitor of this page

Wait a few minutes and [refresh the page](#). If that still doesn't work, please contact the developer of this page for more information.

## EJERCICIO 3

Crea un nuevo fichero Model.kt en la carpeta donde está MainActivity.kt. Asegúrate de poner como primer línea el paquete correcto.

- Crea una data class Amigo para contener la información de un amigo, con atributos para guardar su nombre, latitud y longitud (no necesitas guardar su device).

- Crea una interfaz Amigos ApiService que defina el método getAmigos(), que hará una petición GET a la URL /api/amigos y devolverá una Response<List<Amigo>>
- Crea un object RetrofitClient basándote en el que escribiste en la sesión anterior para el conversor de divisas, pero ahora la URL base será la que te proporciona ngrok, y el campo api hará uso de Amigos ApiService.

Si no hay errores de sintaxis o importación, ya tendríamos lista la parte que resuelve las comunicaciones con el servidor web y el parseo del JSON.

## EJERCICIO 4

Solicitar los permisos en tiempo de ejecución en Android nos dio bastantes problemas. Aunque declaramos los permisos en el Manifiesto, la aplicación no funcionaba hasta que entendimos que en versiones más modernas hay que lanzar un diálogo explícito y gestionar la respuesta del usuario de forma asíncrona.

Crea un fichero MainViewModel.kt en la misma carpeta que MainActivity.kt y Model.kt, que tenga en la primera línea el paquete correcto.

Añade este código dentro y resuelve los import necesarios:

```
class MainViewModel : ViewModel() {

    private var amigosList: List<Amigo>? = null // Por defecto es nula

    init {
        Log.d("MainViewModel", "MainViewModel created")
    }

    fun getAmigosList() {
        viewModelScope.launch {
            // Código que usa retrofit para inicializar amigosList
        }
    }
}
```

Dentro del launch debes escribir código que:

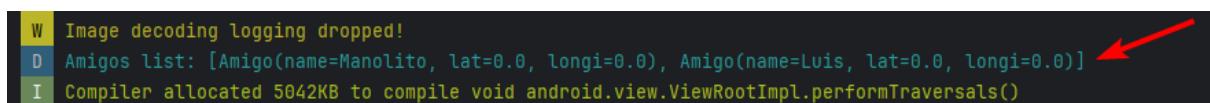
- Declare una variable response que reciba la respuesta de la llamada a RetrofitClient.api.getAmigos()
- Compruebe si response.isSuccessful (imprime por el logcat un mensaje de error si no, y retorna), o asigna a amigosList el cuerpo de la respuesta (response.body()) si hubo éxito.

- Mira si amigosList es nula (imprime un mensaje de error si lo es, y retorna), o imprime por el logcat la lista de amigos recibida si no es nula. Para imprimir la lista desde Kotlin no necesitas iterar por ella, basta algo como Log.d("MainViewModel", "Amigos: \$amigosList")

¡Pero alguien tiene que invocar ese código! Lo haremos desde la Activity y para ello:

- Declara una propiedad viewModel en la clase MainActivity, que obtenga la instancia del MainViewModel asociada a esa Activity (esto es igual que lo que se hizo en la sesión anterior para el conversor de divisas).
- Desde el lugar adecuado de onCreate(), invoca viewModel.getAmigosList() para iniciar la carga de datos.

Ejecuta el programa y observa el logcat. Deberías ver mensajes como el siguiente, que indican que la lista de amigos se ha recibido correctamente:

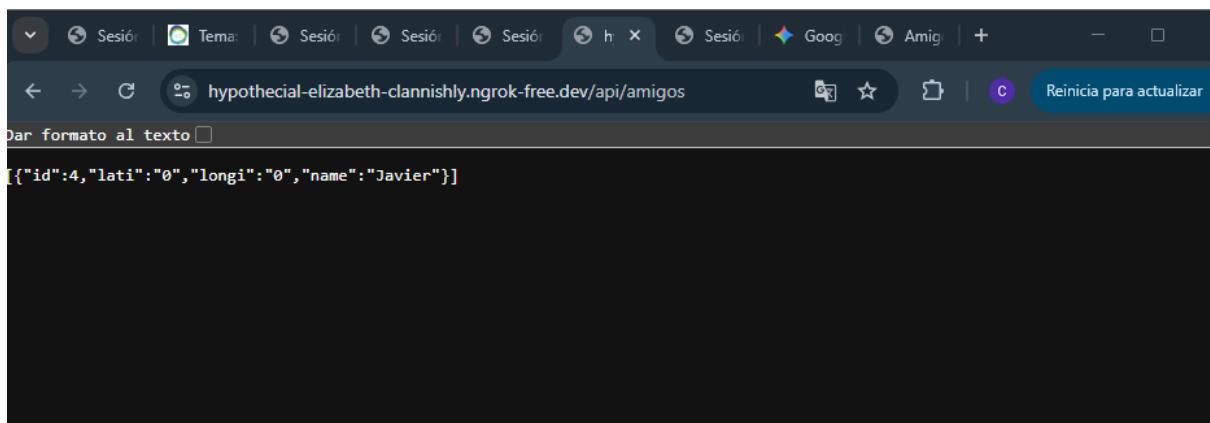
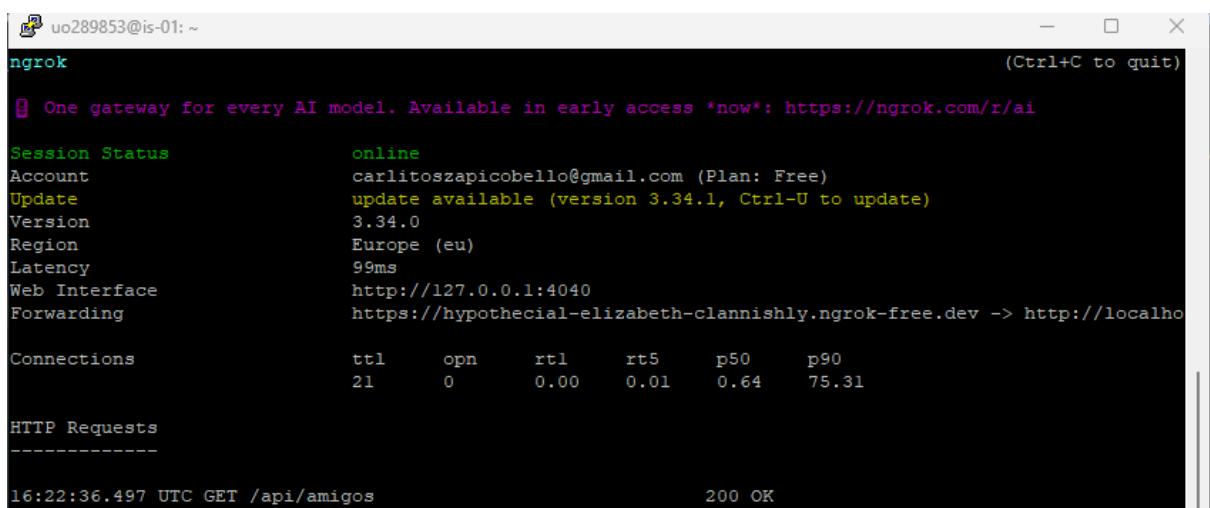


```

W Image decoding logging dropped!
D Amigos list: [Amigo(name=Manolito, lat=0.0, longi=0.0), Amigo(name=Luis, lat=0.0, longi=0.0)]
I Compiler allocated 5042KB to compile void android.view.ViewRootImpl.performTraversals()

```

¿Qué ha aparecido en la terminal de ngrok?

```

uo289853@is-01: ~
ngrok
(Ctrl+C to quit)

Session Status          online
Account                 carlitoszapicobello@gmail.com (Plan: Free)
Update                  update available (version 3.34.1, Ctrl-U to update)
Version                3.34.0
Region                 Europe (eu)
Latency                99ms
Web Interface          http://127.0.0.1:4040
Forwarding              https://hypothecial-elizabeth-clannishly.ngrok-free.dev -> http://localho

Connections             ttl     opn      rsl      rt5      p50      p90
                        21      0       0.00    0.01    0.64    75.31

HTTP Requests
-----
16:22:36.497 UTC GET /api/amigos                         200 OK

```

## EJERCICIO 5

Tuvimos un problema con el mapa, que se quedaba en negro en el emulador. Al principio pensamos que era un error de la librería osmdroid o de la conexión a internet, pero tras investigar nos dimos cuenta de que se quedaba pillado y al encender y apagar la pantalla volvía a la normalidad

**Refactorizar el ViewModel para usar LiveData**

**Modifica tu MainViewModel.kt.**

1. Necesitaremos dos variables para la lista de amigos, siguiendo un patrón de diseño habitual:

- Una privada y mutable (de tipo MutableLiveData), que el ViewModel usará para *actualizar* el valor. Nómbrala \_amigosList.
- Una pública e inmutable (de tipo LiveData), que la Activity usará para *leer* (observar) el valor. Nómbrala amigosList.

Borra tu antigua variable amigosList y añade estas dos:

// Privada: Solo el ViewModel puede modificar esta lista

```
private val _amigosList = MutableLiveData<List<Amigo>>()
```

// Pública: La Activity la "verá", pero no podrá modificarla

```
val amigosList: LiveData<List<Amigo>> = _amigosList
```

2. Modifica tu función getAmigosList(). Dentro de la corutina viewModelScope.launch, en lugar de asignar el resultado a la variable amigosList, ahora debes publicar el valor en el MutableLiveData.

- El método para hacer esto es .setValue() (a menos que estés trabajando desde un hilo en segundo plano, en cuyo caso usarías .postValue(), pero no es el caso).
- Si la respuesta de Retrofit fue exitosa y tiene un body() no nulo, actualiza el LiveData así: \_amigosList.setValue(response.body()).
- Puedes además mostrar por el logcat la lista de amigos recibida, a la que puedes acceder ahora en amigosList.value.

```

class MainViewModel : ViewModel() {
    // Privada: Solo el ViewModel puede modificar esta lista
    2 Usages
    private val _amigosList = MutableLiveData<List<Amigo>>()

    // Pública: La Activity la "verá", pero no podrá modificarla
    1 Usage
    val amigosList: LiveData<List<Amigo>> = _amigosList

    init {
        Log.d( tag = "MainViewModel", msg = "MainViewModel created")
    }

    2 Usages
    fun getAmigosList() {
        viewModelScope.launch {
            try {
                // 1. Llamada a RetrofitClient para obtener amigos
                val response = RetrofitClient.api.getAmigos()

                // 2. Comprobar si la respuesta fue exitosa
                if (response.isSuccessful && response.body() != null) {
                    // PUBLICAR el valor en el MutableLiveData usando .value (equivalente a setValue)
                    _amigosList.value = response.body()

                    // Logcat accediendo al valor a través de la propiedad pública
                    Log.d( tag = "MainViewModel", msg = "Amigos recibidos y publicados: ${amigosList.value}")
                } else {
                    Log.e( tag = "MainViewModel", msg = "Error en la respuesta o cuerpo nulo: ${response.code()}")
                }
            } catch (e: Exception) {
                Log.e( tag = "MainViewModel", msg = "Fallo de conexión: ${e.message}")
            }
        }
    }
}

```

## EJERCICIO 6: Observar el LiveData desde la Activity

Modifica tu MainActivity.kt.

En onCreate(), después de tu código de lifecycleScope.launch que configura el mapa, añade el “observador”. Este es el código que “conecta” la Activity al LiveData del ViewModel:

```

viewModel.amigosList.observe(this) { listaDeAmigos ->

    // Este bloque de código se ejecutará automáticamente

    // cada vez que el ViewModel llame a _amigosList.postValue()

    // Por ahora, solo verificamos que funciona:

    Log.d("MainActivity", "¡Observer notificado! Amigos: $listaDeAmigos")

}

```

Ejecuta la aplicación. Si todo ha ido bien, deberías ver el logcat, además del mensaje del ViewModel cuando recibe la lista, otro emitido desde MainAcivity, que dice “¡Observer notificado!...”

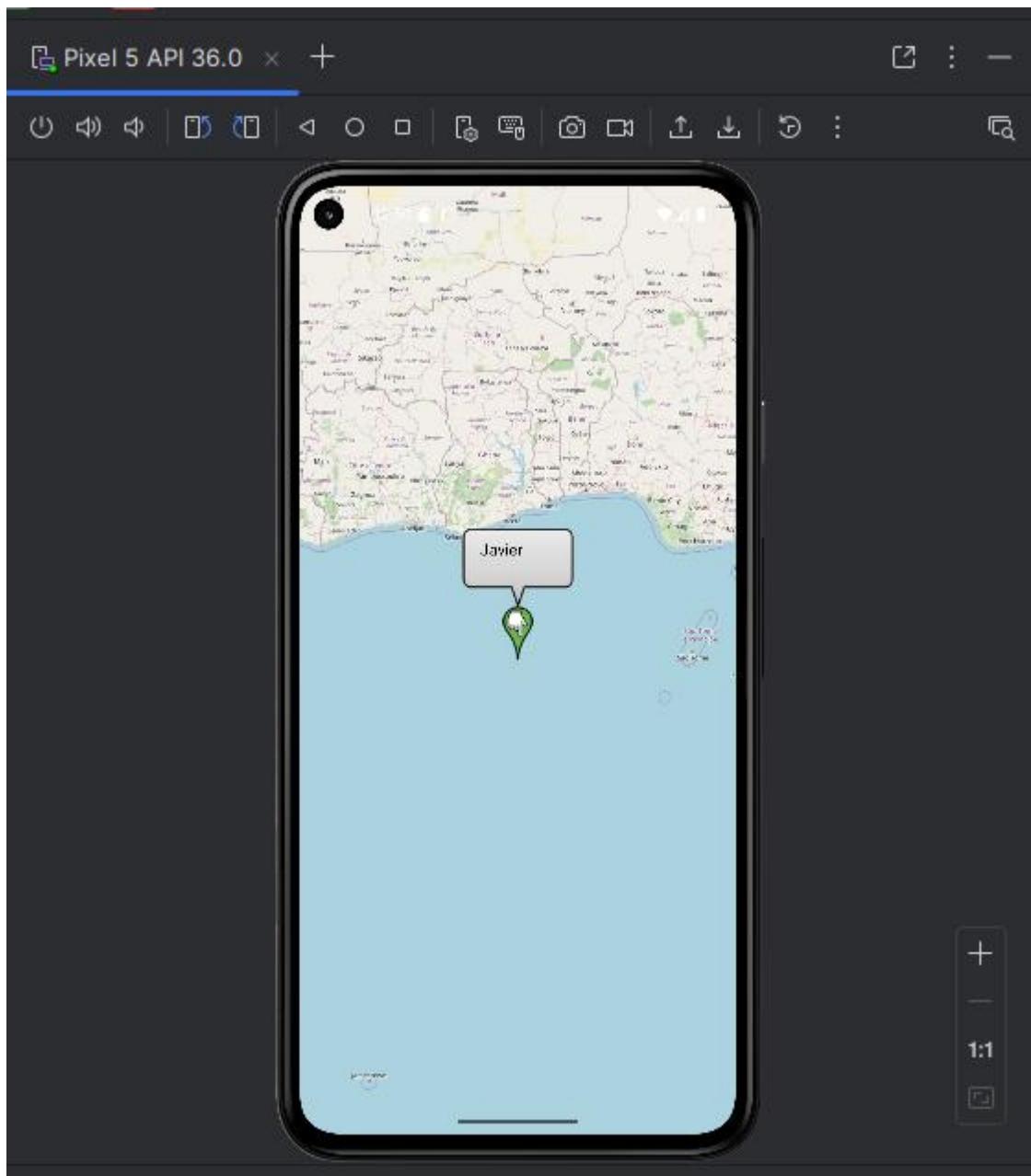
**¡Esto confirma que el bucle MVVM está completo! El ViewModel ha obtenido los datos y se los ha notificado con éxito a la Activity.**

```
2026-01-05 01:27:44.394 10736-10770 HWUI      es.uniovi.amigos
2026-01-03 01:27:44.912 10736-10736 MainActivity    es.uniovi.amigos
2026-01-03 01:27:44.912 10736-10736 MainViewModel   es.uniovi.amigos
2026-01-03 01:27:45.356 10736-10741 s.uniovi.amigos es.uniovi.amigos
2026-01-03 01:27:47.586 10736-10798 ProfileInstaller  es.uniovi.amigos
I Image decoding logging dropped!
D [Observer notificado] Amigos: [Amigo(name=Javier, lati=0.0, longi=0.0)]
D Amigos recibidos y publicados: [Amigo(name=Javier, lati=0.0, longi=0.0)]
I Compiler allocated 5042KB to compile void android.view.ViewRootImpl.performTraversals()
D Installing profile for es.uniovi.amigos
```

**EJERCICIO 7:** Chinhetas visibles Ahora que tenemos la función para añadir una chincheta, podemos escribir la función que pinte todas las chinhetas de la lista de amigos: `paintAmigosList()`. Para ello basta iterar por la lista (`for (amigo in amigos en Kotlin)`, y llamar a `addMarker()` para cada amigo).

Después de ello tienes que invocar `map?.invalidate()` para forzar el repintado del mapa (de lo contrario, no se repintará hasta que el usuario interactúe con él).

Implementa el método `paintAmigosList()` e invócalo desde el observador que habíamos programado en el ejercicio anterior.



## EJERCICIO 8: Añadir Polling al ViewModel

Vamos a modificar MainViewModel.kt para que pida la lista de amigos cada 5 segundos.

1. En el bloque `init { ... }` del ViewModel, llama a una nueva función que vamos a crear, llamada `startPolling()`:
2. `init {`
3.   `Log.d("MainViewModel", "MainViewModel created")`
4.   `startPolling() // Empezamos el polling`
5. Crea la función `startPolling()`:

```

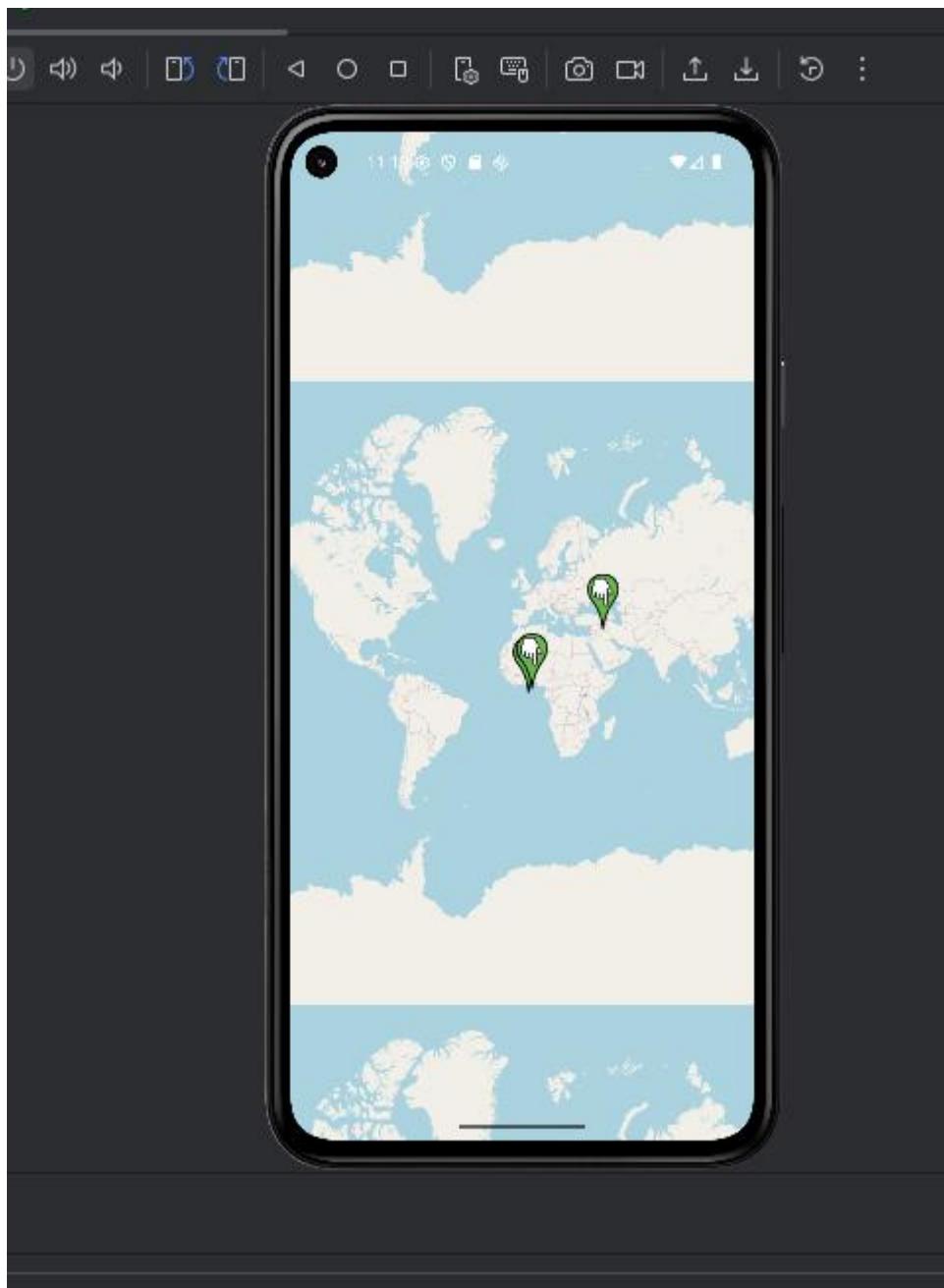
6.    private fun startPolling() {
7.        viewModelScope.launch {
8.            while (true) {
9.                Log.d("Polling", "Timer disparado, pidiendo amigos...")
10.               getAmigosList()
11.               delay(5000)
12.           }
13.       }
}

```

Como ves el mecanismo es conceptualmente simple. Se obtiene la lista del servidor y se duerme 5 segundos, repetido en bucle. El repintado de la lista ocurre automáticamente porque la Activity está observando el LiveData y se le notifica cada vez que el ViewModel actualiza la lista. No obstante, todo el código de este bucle está dentro de un bloque launch, para que se ejecute en una tarea asíncrona. La llamada a delay() es un punto de suspensión, que permite devolver el control al hilo GUI para que haga progresar otras tareas, o atienda eventos de la interfaz de usuario durante esos cinco segundos.

14. Finalmente, en MainActivity.kt, elimina la llamada a viewModel.getAmigosList() de tu onCreate(). Ya no la necesitamos, porque el Timer se encargará de llamarla automáticamente al iniciarse.

Nota. La función paintAmigosList() deberá comenzar por borrar las chinchetas antes pintadas, pues de no hacerlo así cada nueva invocación a esta función pintaría chinchetas adicionales encima de las que ya había. Para borrar las chinchetas previas usa map?.getOverlays()?.clear(); Esta misma situación ocurre con la lista de amigos en getAmigosList(), donde deberás vaciar tu lista con el método clear() antes de añadir todos los amigos de la respuesta, o de lo contrario se añadirían nuevos Amigos en lugar de actualizar los existentes.



Añade al manifiesto los permisos necesarios para acceder a la posición del GPS en caso de que no lo estuvieran ya.

<!-- Se require para obtener la posición -->

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />  
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Ya que no es objetivo de esta práctica entrar en detalle en este asunto, te suministramos directamente el código que puedes copiar y pegar dentro de `MainActivity`. Puedes ponerlo antes de la función `onCreate()`. Cuando el IDE te avise de los import que faltan, resuélvelos. Para `Manifest` debes importar `android.Manifest`.

Y añade una llamada a `checkAndRequestLocationPermissions()` al final de `onCreate()`.

```

2026-01-05 00:36:14.380 18628-18702 ProfileInstaller      es.uniovi.amigos
2026-01-05 00:36:17.310 18628-18628 MainActivity       es.uniovi.amigos
2026-01-05 00:36:17.310 18628-18628 Polling           es.uniovi.amigos
2026-01-05 00:36:17.955 18628-18628 Permissions        es.uniovi.amigos
2026-01-05 00:36:18.021 18628-18628 InsetsController   es.uniovi.amigos

```

```

D Installing profile for es.uniovi.amigos
D ;Observer notificado! Amigos: [Amigo{name=Javier, lati=0.0, longi=0.0}, Amigo{name=tre, lati=34.0, longi=43.0}, Amigo{name=fds, lati=2.0, longi=-4.0}
D Datos recibidos: [Amigo{name=Javier, lati=0.0, longi=0.0}, Amigo{name=tre, lati=34.0, longi=43.0}, Amigo{name=fds, lati=2.0, longi=-4.0}
D Permisos de GPS CONCEDIDO
D hide(im), fromIme=false)

```

## EJERCICIO 9

Lee el código que te acabamos de proporcionar hasta que te hagas una idea clara de lo que hace. El collect del flujo viene a ser como un for para consumir un generador Python. El bloque de código que sigue a collect es ejecutado cada vez que el flujo recibe un nuevo dato. El operador is de Kotlin permite comprobar el tipo de un objeto (similar a instanceof en Java).

Añade el código anterior a tu MainViewModel.kt, y compila para verificar que no hay errores.

Si lo ejecutas ahora, no verás ningún mensaje nuevo en el logcat, porque aún no hemos invocado a startLocationUpdates(). ¿Dónde crees que sería adecuado invocarlo? Piensa que no deberíamos comenzar a escuchar el GPS hasta que el usuario haya concedido los permisos. Entonces ¿dónde pondrías la llamada a startLocationUpdates()?

Hazlo así y verifica que funciona. Deberías ver mensajes en el logcat cada vez que el GPS detecte un cambio de posición (puedes simular cambios de posición en el emulador de Android Studio desde el panel de control del emulador, en la sección “Location”).

```

</> activity_main.xml      MainActivity.kt ×  LocationFlow.kt      MainViewModel.kt      Model.kt      :
24 </> ivity : AppCompatActivity() {
25
26     ar map: MapView? = null // Referencia al objeto MapView
27
28     al viewModel: MainViewModel by viewModels()
29
30     al requestPermissionLauncher =
31         terForActivityResult(
32             contract = ActivityResultContracts.RequestMultiplePermissions()
33             rmissions ->
34                 / Este bloque se ejecuta cuando el usuario responde al diálogo
35                 f (permissions[Manifest.permission.ACCESS_FINE_LOCATION] == true) {
36                     // Permisos concedidos
37                     Log.d( tag = "Permissions", msg = "Permisos de GPS CONCEDIDO")
38                     viewModel.startLocationUpdates()
39
40                 else {
41                     // Permisos denegados
42                     Log.d( tag = "Permissions", msg = "Permisos de GPS DENEGADO")
43                     // (Opcional: Mostrar un Toast o un diálogo explicando por qué
44                     // la función de GPS no funcionará)
45
46             un checkAndRequestLocationPermissions() {
47                 rmissionsToRequest = arrayof(

```

```

2026-01-05 00:55:46.218 19035-19035 HWUI          es.uniovi.amigos
2026-01-05 00:55:46.634 19035-19039 s.uniovi.amigos
2026-01-05 00:55:47.926 19035-19035 GPS           es.uniovi.amigos
2026-01-05 00:55:48.736 19035-19101 ProfileInstaller es.uniovi.amigos
2026-01-05 00:55:51.338 19035-19035 MainActivity    es.uniovi.amigos
2026-01-05 00:55:54.551 19035-19035 MainActivity   es.uniovi.amigos

```

```

W Image decoding logging dropped!
I Compiler allocated 5042KB to compile void android.view.ViewRootImpl.performTraversals()
D Nueva ubicación: 37.421998333333335, -122.084
D Installing profile for es.uniovi.amigos
D ;Observer notificado! Amigos: [Amigo{name=Javier, lati=0.0, longi=0.0}, Amigo{name=tre, lati=34.0, longi=43.0}, Amigo{name=fds, lati=2.0, longi=-4.0}
D ;Observer notificado! Amigos: [Amigo{name=Javier, lati=0.0, longi=0.0}, Amigo{name=tre, lati=34.0, longi=43.0}, Amigo{name=fds, lati=2.0, longi=-4.0}

```

## EJERCICIO 10

Añade a Model.kt, dentro de la interfaz AmigosApiServer, el siguiente método:

```
@PUT("api/amigo/{id}")  
suspend fun updateAmigoPosition(  
    @Path("id") amigold: Int,  
    @Body payload: LocationPayload  
) : Response<Amigo>
```

Como ves, definimos un nuevo *endpoint* PUT /api/amigo/{id}, que recibe en el cuerpo de la petición un objeto LocationPayload. Esta será una data class que debemos definir también en Model.kt, y que contendrá solo los campos lati y longi. Añádela también.

En la ruta aparece {id} entre llaves. Esto significa que esa parte de la ruta se rellenará con un parámetro que se le pasará al método. La anotación @Path("id") indica que el parámetro amigold recibido por la función se usará para llenar esa parte de la ruta.

Para probar si la función funciona, invócala desde el punto apropiado del bloque collect startLocationUpdates(), en el ViewModel, pasando un id fijo, que sepas que existe en el backend (por ejemplo, 1) y las coordenadas recibidas del GPS. No olvides hacer la llamada dentro de un bloque try-catch para capturar posibles excepciones.

Ejecutalo y comprueba que en el backend la posición del amigo con id 1 se actualiza correctamente cada vez que el GPS notifica un cambio de posición (puedes consultarla en la vista HTML de la tabla de amigos). También deberías ver la petición POST en la terminal donde corre ngrok. Además, cuando la aplicación android reciba la lista de coordenadas actualizada, deberás ver moverse la del amigo con id 1 en el mapa.

```

2 Usages
interface Amigos ApiService {
    1 Usage
    @GET( value = "/api/amigos")
    suspend fun getAmigos(): Response<List<Amigo>>

    @PUT( value = "api/amigo/{id}")
    suspend fun updateAmigoPosition(
        @Path( value = "id") amigoId: Int,
        @Body payload: LocationPayload
    ): Response<Amigo>

    1 Usage
    data class LocationPayload(
        val lati: Double,
        val longi: Double
    )
}

```

A screenshot of the Android Studio logcat window. It shows several log entries:
 

- Line 1: 0:01:05:22.100 E/Log: E/Log: 22534 22534 amigos es.uniovi.amigos D Nueva ubicación: 37.421998333333335, -122.084
- Line 2: 5 01:05:23.701 22534-22534 GPS es.uniovi.amigos D Posición actualizada en servidor para ID 4
- Line 3: 5 01:05:24.178 22534-22534 API\_PUT es.uniovi.amigos D Installing profile for es.uniovi.amigos
- Line 4: 5 01:05:24.953 22534-22595 ProfileInstaller

<b>Id</b>	<b>Nombre</b>	<b>Latitud</b>	<b>Longitud</b>	<b>Modificar</b>	<b>Borrar</b>
4	Javier	37.421998333333335	-122.084	<a href="#">Modificar</a>	<a href="#">Borrar</a>

## EJERCICIO 11

En Model.kt, añade a la interfaz AmigosApiServer un nuevo método para obtener un amigo por su nombre. El método se llamará getAmigoByName() y usará el endpoint GET /api/amigo/byName/{name}. Debería devolver un objeto Amigo.

Debes editar la clase Amigo para añadirle un campo id de tipo Int, que represente el id del amigo. Ese valor venía ya en la respuesta JSON, pero al no estar declarado aquí, Retrofit lo ignoraba. Al declararlo, lo mapeará automáticamente desde el JSON al objeto Amigo retornado.

Comprueba que compila sin errores, aunque de momento no lo estaremos usando.

```
5 Usages
data class Amigo(
    val id: Int,
    val name: String,
    val lati: Double,
    val longi: Double
)

3 Usages
interface Amigos ApiService {
    1 Usage
    @GET( value = "/api/amigos")
    suspend fun getAmigos(): Response<List<Amigo>>

    @GET( value = "/api/amigo/byName/{name}")
    suspend fun getAmigoByName(
        @Path( value = "name") name: String
    ): Response<Amigo>

    1 Usage
    @PUT( value = "api/amigo/{id}")
    suspend fun updateAmigoPosition(
        @Path( value = "id") amigoId: Int,
        @Body payload: LocationPayload
    ): Response<Amigo>

    2 Usages
    data class LocationPayload(
        val lati: Double,
        val longi: Double
    )
}
```

## EJERCICIO 12

Implementa lo antes descrito. Recuerda que parte va en `MainActivity.kt` y parte en `MainViewModel.kt`. Inserta una llamada a `askUserName()` al final de `onCreate()` de la Activity, para que el diálogo aparezca al arrancar la aplicación.

Compila y ejecuta. Verás que te pide el nombre, y cuando escribas uno, mira el logcat para verificar que el ViewModel ha recibido el nombre correctamente.

```

1 Usage
private fun askUserName() {
    // 1. Crear el constructor del diálogo
    val builder = AlertDialog.Builder( context = this)
    builder.setTitle("Identificación")
    builder.setMessage("Introduce tu nombre de usuario:")

    // 2. Crear un EditText para que el usuario escriba
    val input = EditText( context = this)
    builder.setView(input)

    // 3. Configurar el botón "OK"
    builder.setPositiveButton( text = "Aceptar") { dialog, which ->
        val name = input.text.toString()
        // 4. ¡LA CONEXIÓN CLAVE!
        // Si el usuario escribió un nombre, se lo pasamos al ViewModel
        if (name.isNotBlank()) {
            viewModel.setUserName(name)
        }
    }

    // 5. Mostrar el diálogo
    builder.show()
}

```

```

2 Usages
💡 private var userName: String? = null

1 Usage
fun setUserName(name: String) {
    userName = name
    Log.d( tag = "MainViewModel", msg = "Nombre de usuario establecido: $userName")
}

```

## EJERCICIO 13

Implementa lo antes descrito en el ViewModel. Recuerda que la llamada a la API REST debe hacerse dentro de un bloque `viewModelScope.launch { ... }` para que no bloquee el hilo GUI. Imprime por el logcat el id obtenido, o los errores en caso de que los haya.

Ejecuta y comprueba que tras introducir el nombre, en el logcat aparece el id correspondiente.

Gira la pantalla. ¿qué ha pasado? ¡Nos pide el nombre otra vez! Piensa por qué ocurre esto (pista, `onCreate()` se ejecutará de nuevo). ¿Se ha perdido el id del usuario? ¿Por qué? ¿Cómo evitamos que nos pida el nombre otra vez al girar la pantalla? (Esto se sale un poco de los objetivos de esta práctica, así que si no lo resuelves no pasa nada)

## EJERCICIO 14

La gestión del userId opcional fue importante. En varias pruebas la aplicación se cerraba inesperadamente porque el GPS intentaba enviar una posición antes de que el servidor hubiera respondido con el ID del usuario.

**Lee el recuadro anterior y modifica la llamada a updateAmigoPosition() para que use userId en lugar del id=1 fijo que habíamos usado antes, protegiendo el acceso a userId con un bloque ?.let**

Prueba a cambiar la posición del GPS a través del emulador y verifica que la aplicación web actualiza la posición (podrás comprobarlo en la tabla de la vista html con un navegador, y también poco después con el movimiento de la chincheta en el móvil, cuando, debido al polling se recuperen de nuevo las posiciones de la base de datos).

```

2     class MainViewModel(application: Application) : AndroidViewModel(application) {
3         fun startLocationUpdates() {
4             viewModelScope.launch {
5                 locationFlow.collect { result ->
6
7                     userId?.let { idNoNulo ->
8                         try {
9                             val payload = Amigos ApiService.LocationPayload(
10                                 lati = location.latitude,
11                                 longi = location.longitude
12                             )
13                             RetrofitClient.api.updateAmigoPosition( amigoid = idNoNulo, payload)
14                         } catch (e: Exception) {
15                             Log.e( tag = "API_PUT", msg = "Fallo de red al actualizar posición: ${e.message}")
16                         }
17                     }
18
19                     } else if (result is LocationResult.PermissionDenied) {
20                         Log.w( tag = "Location", msg = "Permiso de ubicación denegado")
21                     } else if (result is LocationResult.ProviderDisabled) {
22                         Log.w( tag = "Location", msg = "Proveedor GPS desactivado")
23                     }
24                 }
25             }
26         }
27     }

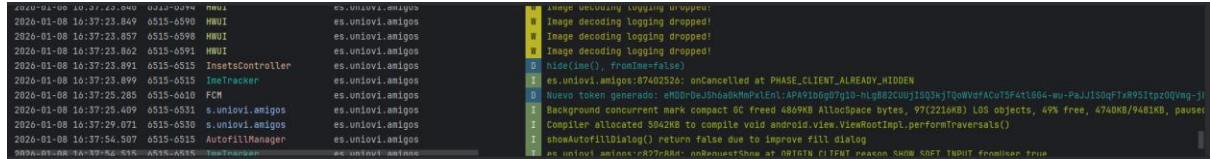
```

## EJERCICIO 15

Implementa lo que se ha descrito más arriba y ejecuta el programa. En principio todo debería seguir funcionando como antes (recuerda tener en marcha el servidor y el túnel ngrok).

Observa en el logcat el mensaje (filtra por el texto “FCM”) y si todo va bien podrás ver el token de dispositivo que se le ha asignado. Es una larga cadena alfanumérica, como por ejemplo “eh-kusa1RlaSss7QN0...tV\_3NI”. Si no lo ves posiblemente se deba a que no es la primera vez que has ejecutado esta aplicación desde que le has añadido soporte FireBase. El token se genera sólo la primera vez que la ejecutas, por lo que sólo esa primera vez se ejecutará el

método `onNewToken()`. Si te ocurre esto no te preocupes, de todas formas es posible obtener el token más adelante, como explicaremos seguidamente.



The screenshot shows a portion of the Android Logcat output. It includes several log entries from the application 'es.uniovi.amigos' and system components like 'Image decoding' and 'Background concurrent mark compact'. One entry specifically mentions a 'Nuevo token generado' (New token generated) with a long hex string. Another entry shows a call to 'onCancelled' at PHASE\_CLIENT\_ALREADY\_HIDDEN. The log ends with a note about 'showAutofillDialog()' returning false due to improve fill dialog.

## EJERCICIO 16

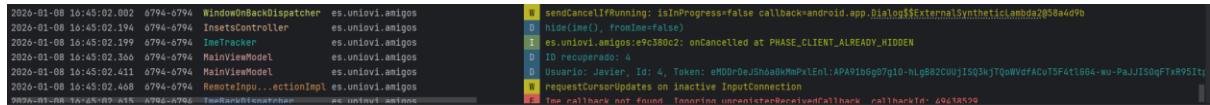
Añade la dependencia antes mencionada a tu `build.gradle.kts` (`Module:app`) y sincroniza el proyecto.

Añade la llamada que obtiene el token en el lugar apropiado (mira la respuesta anterior).

En este punto de la función `setUserName()` tenemos ya tres ítems de información cruciales, y podemos volcarlos todos al logcat con una línea como la siguiente:

```
Log.d("MainViewModel", "Usuario: $userName, Id: $userId, Token: $token")
```

Ejecuta la app y comprueba en el logcat que sale la información correcta.



The screenshot shows the logcat output again, this time with the added log line from the previous step. It shows the application outputting the user information ('Javier'), the user ID ('4'), and the token ('e9c380c2'). The log also includes some background tasks and a note about a 'requestCursorUpdates' call.

## EJERCICIO 17

Implementa lo que se ha descrito más arriba en `Model.kt`. Recuerda definir la data class `DeviceTokenPayload` y la función `updateAmigoDeviceToken()` en la interfaz `AmigosApiServer`.

Compila y comprueba que no hay errores.

Después añade la llamada a esa función en el `ViewModel`, justo después de obtener el token, pasando el `userId` y un nuevo objeto `DeviceTokenPayload` creado con el token obtenido.

Ejecuta el código y comprueba en la terminal donde corre ngrok que se recibe la petición `PUT /api/amigo/{id}` para actualizar el token del usuario. También puedes comprobar en la vista HTML de la tabla amigos que el campo `device` del amigo correspondiente ha sido actualizado con el token, o hacer una petición con la herramienta `httpie` a la lista de amigos para verificar que el token está ahí, en el campo `device` del amigo cuyo nombre has escrito en el teléfono.

```

ngrok
** Free Users: Agents ≤3.19.x stop connecting 2/17/26. Update or upgrade: https://ngrok.com/pricing

Session Status           online
Account                  carlitoszapicobello@gmail.com (Plan: Free)
Version                 3.34.1
Region                  Europe (eu)
Latency                78ms
Web Interface          http://127.0.0.1:4040
Forwarding              https://hypothecial-elizabeth-clannishly.ngrok-free.dev -> http://localhost:80

Connections             ttl     opn      rt1      rt5      p50      p90
                        24       4       0.02     0.02    150.16   1041.48

HTTP Requests
-----
15:49:34.556 UTC GET /api/amigos           200 OK
15:49:38.389 UTC GET /api/amigos           200 OK
15:49:29.359 UTC GET /api/amigos           200 OK
15:49:29.215 UTC GET /api/amigo/byName/Javier 200 OK
15:49:29.555 UTC PUT /api/amigo/4          200 OK
15:49:28.384 UTC GET /api/amigos           200 OK
15:49:24.094 UTC GET /api/amigos           200 OK
15:49:23.386 UTC GET /api/amigos           200 OK
15:49:18.376 UTC GET /api/amigos           200 OK
15:49:18.899 UTC GET /api/amigos           200 OK

```

2026-01-08 16:47:18.291 6941-6941 MainViewModel es.uniovi.amigos
2026-01-08 16:47:18.307 6941-6941 MainViewModel es.uniovi.amigos
2026-01-08 16:47:18.328 6941-6941 WindowOnBackDispatcher es.uniovi.amigos
2026-01-08 16:47:18.932 6941-6941 MainViewModel es.uniovi.amigos
[ 0] ID recuperado: 4
[ 0] Usuario: Javier, Id: 4, Token: eM0DrDeJShsd0KMePx1EnI:APA91b6q07g10-hLgB82CUUjS03kjTQwWdfAcuTSFatt004-wu-PajJIS0qFtxR95
[ 0] sendCancelIfRunning: isInProgress=false callback=android.view.ImeBackAnimationController@1d84f81
[ 0] Token enviado al servidor correctamente

## EJERCICIO 18

Añade a `MainActivity.kt` el código anterior para implementar el receptor de avisos FCM.

Añade en la clase `MyFirebaseMessagingService` el código necesario para crear un `intent` con la acción "updateFromServer" y enviarlo usando `sendBroadcast()`.

Ejecuta el programa y vigila el `logcat` (puedes filtrar por la cadena Aviso). Desde la consola web de Firebase envía un mensaje. Verifica que en el `logcat` aparece la recepción del `intent`. (Nota: si no aparece sigue con la práctica, la consola de FireBase puede tardar en enviar el mensaje).

```
MainViewModel.kt Model.kt AndroidManifest.xml MyFirebaseMessagingService.kt 3 ▾  
1 package es.uniovi.amigos  
2  
3 import android.content.Intent  
4 import android.util.Log  
5 import com.google.firebase.messaging.FirebaseMessagingService  
6 import com.google.firebase.messaging.RemoteMessage  
7  
8 Usage  
8 </>  
9 class MyFirebaseMessagingService : FirebaseMessagingService() {  
10     override fun onMessageReceived(remoteMessage: RemoteMessage) {  
11         super.onMessageReceived(remoteMessage)  
12         Log.d( tag = "FCM", msg = "Mensaje recibido!")  
13  
14         val intent = Intent( action = "updateFromServer")  
15         intent.setPackage(packageName)  
16         sendBroadcast(intent)  
17     }  
18     override fun onNewToken(token: String) {  
19         super.onNewToken(token)  
20         Log.d( tag = "FCM", msg = "Nuevo token generado: $token")  
21     }  
22 }
```

## EJERCICIO 19

**Elimina el código necesario para que ya no se haga *polling* al servidor.**

Añade la llamada a `getAmigosList()` en el punto adecuado, para que se produzca cuando se reciba la notificación.

Ejecuta el programa. Usa la consola web de Firebase para enviar una notificación y vigila la terminal donde tienes abierto el túnel con ngrok. Deberías ver que el cliente hace una nueva petición GET cuando recibe la notificación FCM.

```
    es.uniovi.amigos      I hiddenapi: Accessing hidden method Landroid/os/WorkSource;-->add(I)Z (runtime_flags=0, domain=platform, api_level=1)
    es.uniovi.amigos      I hiddenapi: Accessing hidden method Landroid/os/WorkSource;-->get(I)I (runtime_flags=0, domain=platform, api_level=1)
    es.uniovi.amigos      I hiddenapi: Accessing hidden method Landroid/os/WorkSource;-->getName(I)Ljava/lang/String; (runtime_flags=0, domain=platform, api_level=1)
}
es.uniovi.amigos      W Unable to log event: analytics library is missing
es.uniovi.amigos      W Unable to log event: analytics library is missing
es.uniovi.amigos      D Mensaje recibido!
es.uniovi.amigos      D ¡Aviso de FCM recibido! Actualizando lista...
es.uniovi.amigos      D Lista de amigos actualizada
```

EJERCICIO 20

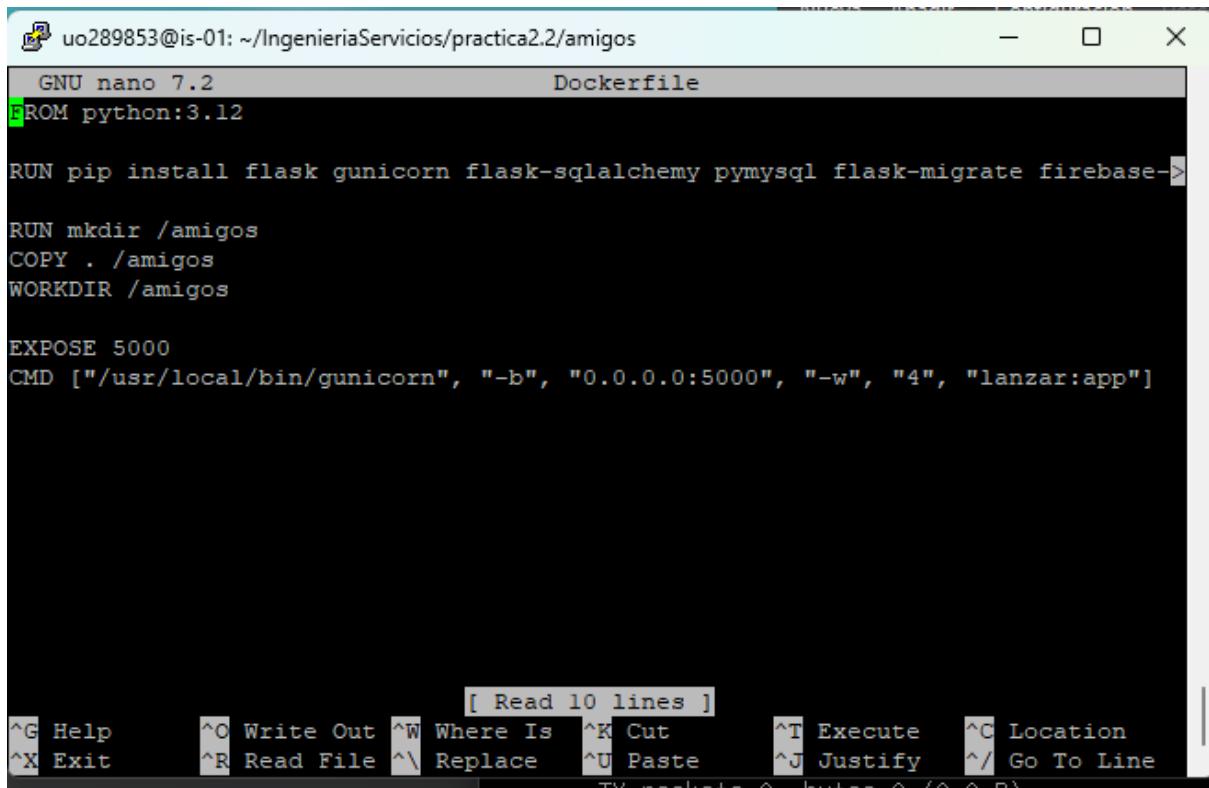
Durante la realización de este ejercicio encontramos los siguientes problemas: Al acabar la práctica tuvimos que volver a generar la base de datos para que se añadieran los nuevos campos y así poder hacer la comprobación del ejercicio pero no fuimos capaces por un problema de conexión al volver a generála, a partir de ahí no fuimos capaces de recuperar la base de datos.

**Implementa todo lo descrito en los puntos anteriores para integrar el envío de notificaciones FCM en el backend Flask.**

**Crea una nueva imagen docker con un tag diferente (por ejemplo `ing_serv_amigos_fcm`) e iníciala. Verifica que no hay errores en el arranque.**

**Prueba a mover la posición de algún amigo (vía web o vía API) y comprueba que el teléfono recibe la notificación FCM y actualiza la chincheta.**

**Comprueba que, si cambias las coordenadas del usuario a través del emulador Android, ésto causa que se reciba un mensaje FCM indicando el cambio.**



The screenshot shows a terminal window titled "u0289853@is-01: ~/IngenieriaServicios/practica2.2/amigos". The window contains a file named "Dockerfile" with the following content:

```
GNU nano 7.2                               Dockerfile
FROM python:3.12

RUN pip install flask gunicorn flask-sqlalchemy pymysql flask-migrate firebase->

RUN mkdir /amigos
COPY . /amigos
WORKDIR /amigos

EXPOSE 5000
CMD ["/usr/local/bin/gunicorn", "-b", "0.0.0.0:5000", "-w", "4", "lanzar:app"]
```

The terminal window has a standard nano editor footer with various keyboard shortcuts for file operations like Help, Exit, Write Out, Read File, etc.

```
uo289853@is-01: ~/IngenieriaServicios/practica2.2/amigos/app
GNU nano 7.2                               models.py

from app import db

class Amigo(db.Model):
    __tablename__ = "amigos"

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(32), unique=True)
    longi = db.Column(db.String(32))
    lati = db.Column(db.String(32))
    device = db.Column(db.String(512))

    def __repr__(self):
        return "<Amigo[{}]: {}>".format(self.id, self.name)

    def to_json(self):
        return {
            'id': self.id,
            'name': self.name,
            'lati': self.lati,
            'longi': self.longi,
            'device': self.device
        }

def get_all_devices():
    amigos = Amigo.query.filter(Amigo.device != None, Amigo.device != "").all()
    return [amigo.device for amigo in amigos]

^G Help          ^O Write Out      ^W Where Is       ^K Cut           ^T Execute       ^C Location
^X Exit         ^R Read File      ^\ Replace        ^U Paste         ^J Justify       ^/ Go To Line
```

## ANEXO

Durante la realización de las prácticas, el porcentaje de trabajo realizado durante este periodo fue igual para todos los miembros del grupo.