Seminario 5

Creación de aplicaciones Cliente/Servidor

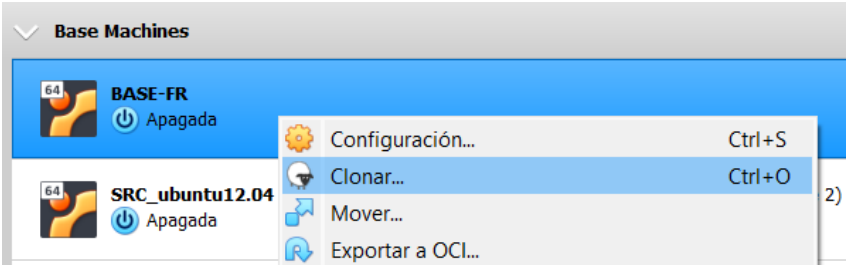
1. Preparación del seminario

Para la correcta realización del seminario de Cliente Servidor se recomienda preparar con antelación las máquinas virtuales que se serán utilizadas.

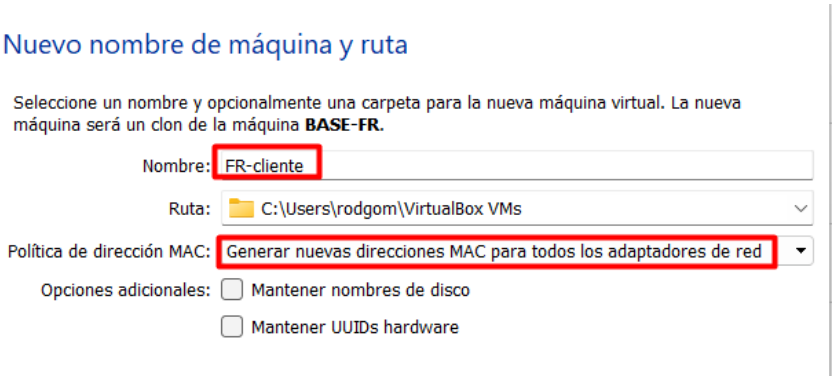
1.1. Clonación enlazada de la máquina base de seminarios

Se han de realizar dos clonaciones enlazadas de la máquina de seminarios de la asignatura Fundamentos de Redes disponible en PRADO.

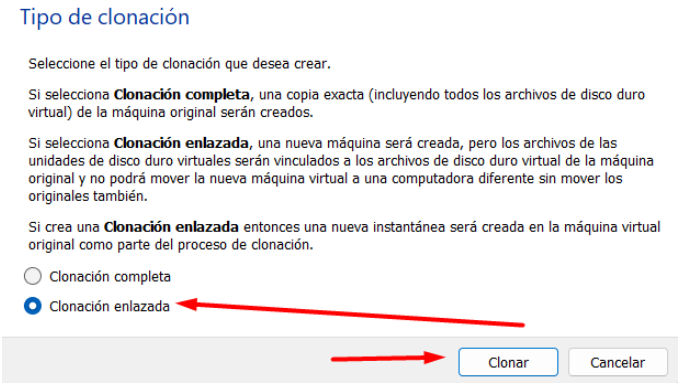
Para esto, se hace clic derecho sobre la máquina ya importada en Virtualbox y se escoge la opción Clonar.



Se puede cambiar el nuevo nombre de la máquina por (FR-cliente en la primera clonación y FR-Servidor en la segunda). Es muy importante que en este punto se seleccione la opción generar nuevas direcciones MAC para todos los adaptadores de red. Elegido esto se clica sobre Siguiente.



Por último, se escoge la opción clonación enlazada y se clica sobre Clonar.



Nota importante: Realizad las dos clonaciones clicando sobre el botón derecho desde la máquina original importada. Esa máquina será ahora la máquina base de estas dos clonaciones y no debe encenderse para minimizar los conflictos que esto pudiera generar. Sólo se utilizarán las máquinas clonadas.

1.2. Conexión en red de ambas clonaciones

Para conectar en red estas dos máquinas virtuales se procederá a cambiar la IP de la máquina servidora. Para ello, arranque la máquina servidora y abra una terminal (Ctrl + Alt + T).

Edite con permisos de superusuario el fichero de configuración de red `/etc/netplan/01-network-manager-all.yaml` puede hacerlo con nano con el comando siguiente o con el editor de textos de su preferencia (vim, gedit, etc.):

```
sudo nano /etc/netplan/01-network-manager-all.yaml
```

Cambie la IP de la interfaz `enp0s9` por la dirección 33.1.1.1:

```
# datos
enp0s9:
  dhcp4: no
  addresses: [33.1.1.1/24]
  #gateway4:
```

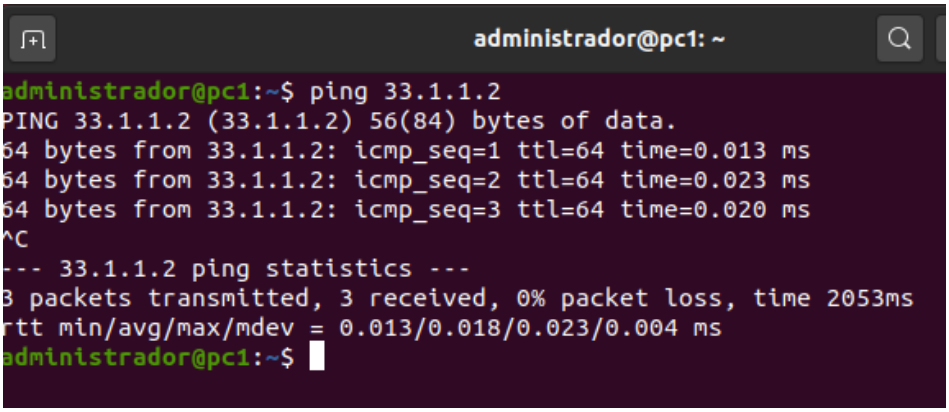
Reinicie la configuración de red en el servidor con el comando: `sudo netplan apply`

Compruebe que el cambio se ha realizado con éxito con el comando `ip -c a`:

```
administrador@pc1:~$ ip -c a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue s
t qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s9: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 q
oup default qlen 1000
    link/ether 08:00:27:19:36:e4 brd ff:ff:ff:ff:ff:ff
    inet 33.1.1.1/24 brd 33.1.1.255 scope global enp0s9
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:fe19:36e4/64 scope link
       valid_lft forever preferred_lft forever
```

Haga un ping entre ambas máquinas para asegurarse de que hay conectividad. Desde el servidor ejecute: `ping 33.1.1.2`

Compruebe que el ping es exitoso:



Pulse Ctrl + C para cortar el envío de mensajes ping.

2. Códigos del cliente y servidor en UDP

Prepare los códigos tanto de este apartado como del siguiente para poder seguir la clase de seminarios con mayor fluidez. Los conceptos teóricos asociados serán repasados en la clase de seminarios.

Cread el archivo `clienteUPD.py` que contenga el siguiente código:

```
import socket
s_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s_client.sendto(b'Hola clase', ('33.1.1.1',12345))
s_client.close()
```

Cread el archivo `servidorUPD.py` que contenga el siguiente código:

```
import socket
s_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s_server.bind(('',12345))
data, clientaddr = s_server.recvfrom(4096)
s_server.close()
```

Estudie cada una de las líneas de código que componen ambos programas. Puede consultar para esto la siguiente documentación: https://docs.python.org/3/library/socket.html

Desde el servidor ejecute el programa con `python3 servidorUDP.py`
Desde el cliente ejecute el programa con `python3 clienteUDP.py`

Reto. ¿Qué habría que incluir en el código para que el servidor pudiera responder al cliente "Bienvenido a clase"?

3. Códigos del cliente y servidor en TCP

Cread el archivo `servidorTCP.py` que contenga el siguiente código:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind(("", 9999))
s.listen(1)
sc, addr = s.accept()
while True:
    recibido = sc.recv(1024)
    if recibido == "close":
        break
    print str(addr[0]) + " dice: ", recibido.decode()
    sc.send(recibido)
print "Adios."
sc.close()
s.close()
```

Cread el archivo `clienteTCP.py` que contenga el siguiente código:

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("33.1.1.1", 9999))
while True:
    mensaje = input("Mensaje a enviar >> ")
    s.send(mensaje.encode())
    if mensaje == "close":
        break
print "Adios."
s.close()
```

Estudie cada una de las líneas de código que componen ambos programas. Puede consultar para esto la siguiente documentación: https://docs.python.org/3/library/socket.html

Desde el servidor ejecute el programa con `python3 servidorTCP.py`
Desde el cliente ejecute el programa con `python3 clienteTCP.py`
Debe escribir el mensaje a enviar al servidor.

¿Cuál es el mensaje que ha de escribir para que el programa del cliente finalice?

Reto. ¿Cómo podría descubrir si tiene lanzado el programa servidor en su máquina? ¿Cómo podría saber las conexiones que hay establecidas con su programa servidor? Pista: para ambas cuestiones ha de usar el comando `ss` busque con qué argumentos.

Para profundizar

Para estudiar en mayor profundidad los conceptos de cliente/servidor utilizando TCP o UDP se recomienda estudiar el capítulo 2 del libro Computer Networking [1].

Para ver cómo programar socket concurrentes con Python se recomienda el tutorial [2].

[1] James, Kurose, y Ross Keith. Computer Networking: A Top-Down Approach. Boston Munich, 2016. Capítulo 2.
[2] GeeksforGeeks. «Socket Programming with Multi-Threading in Python», 30 de septiembre de 2017. https://www.geeksforgeeks.org/socket-programming-multi-threading-python/.