

# Compte rendu TP1 Mode Connecté

Nom : Ibrahim Lahlou | IDSCC4

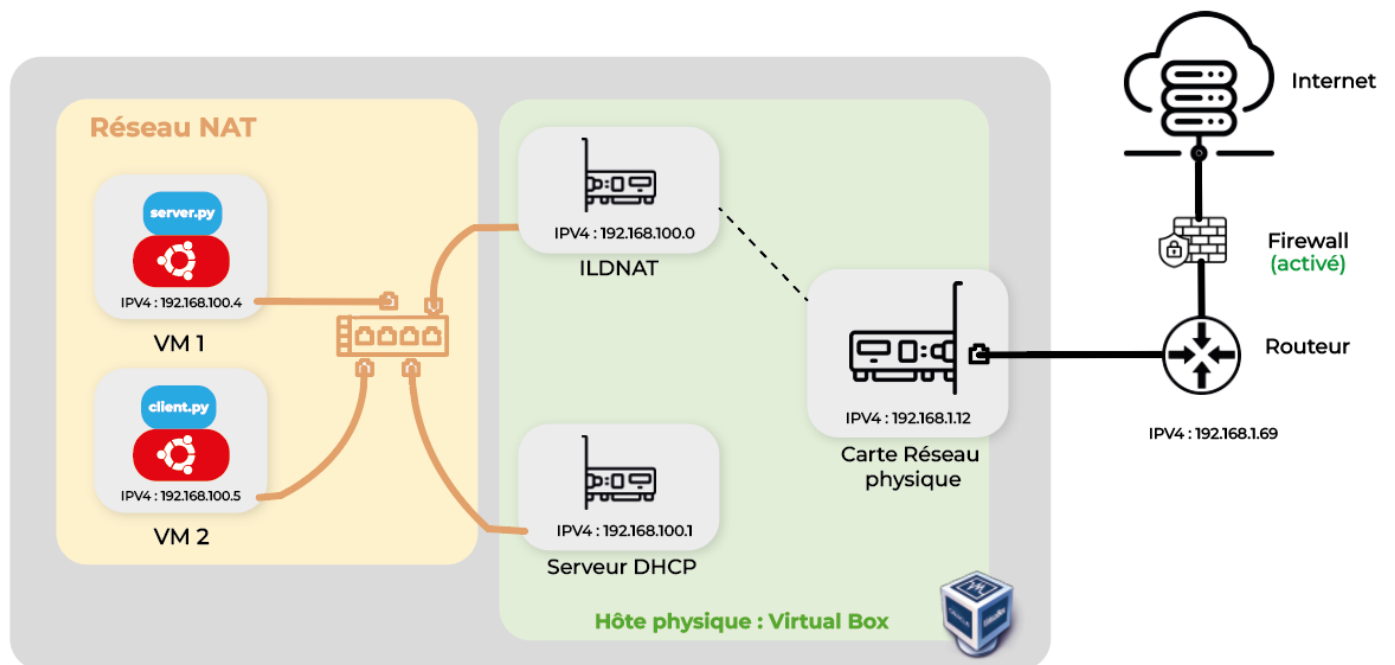
## 0. Programmation de Socket

Création d'un programme communiquant des sockets avec python entre les deux machine VM1 et VM2 des instances d'Ubuntu server

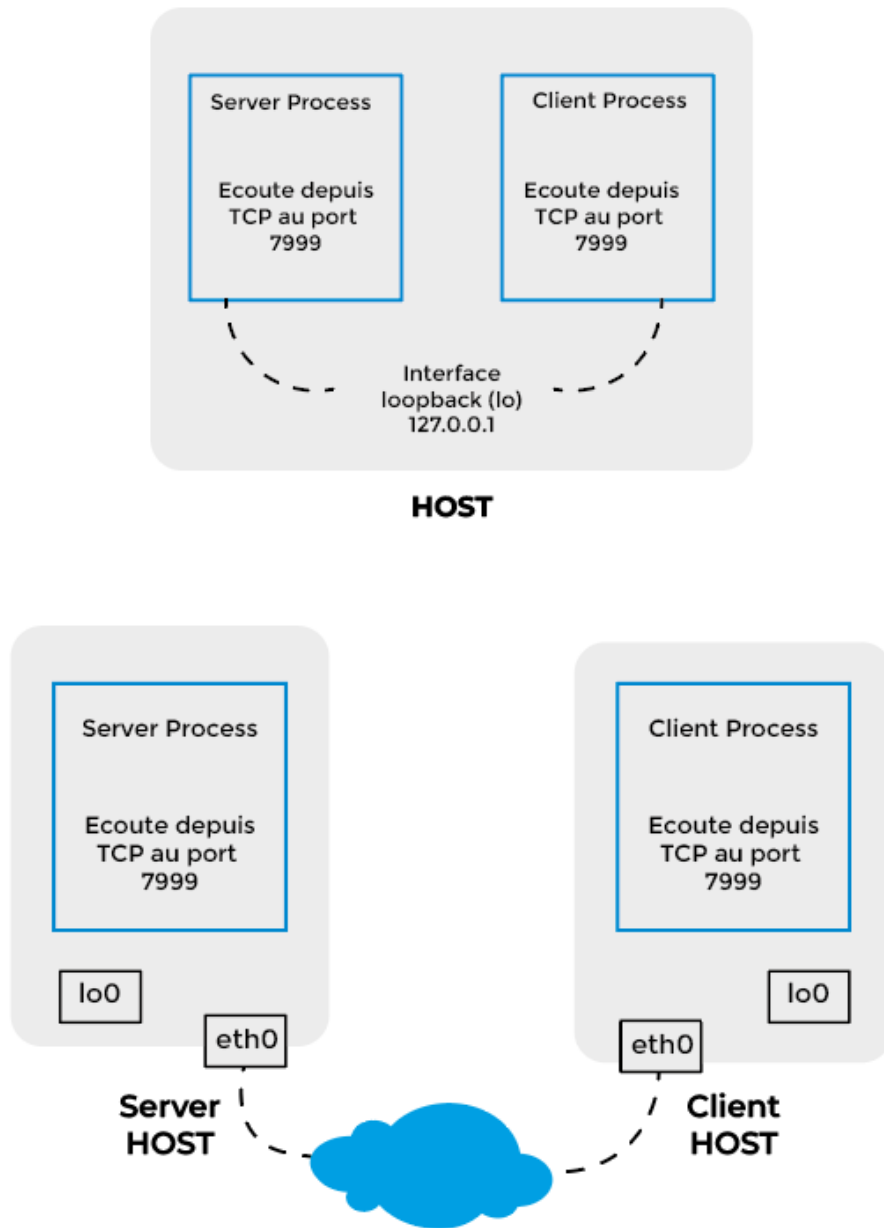
.

## 1. Architecture réseau de cette manipulation

Réseau NAT



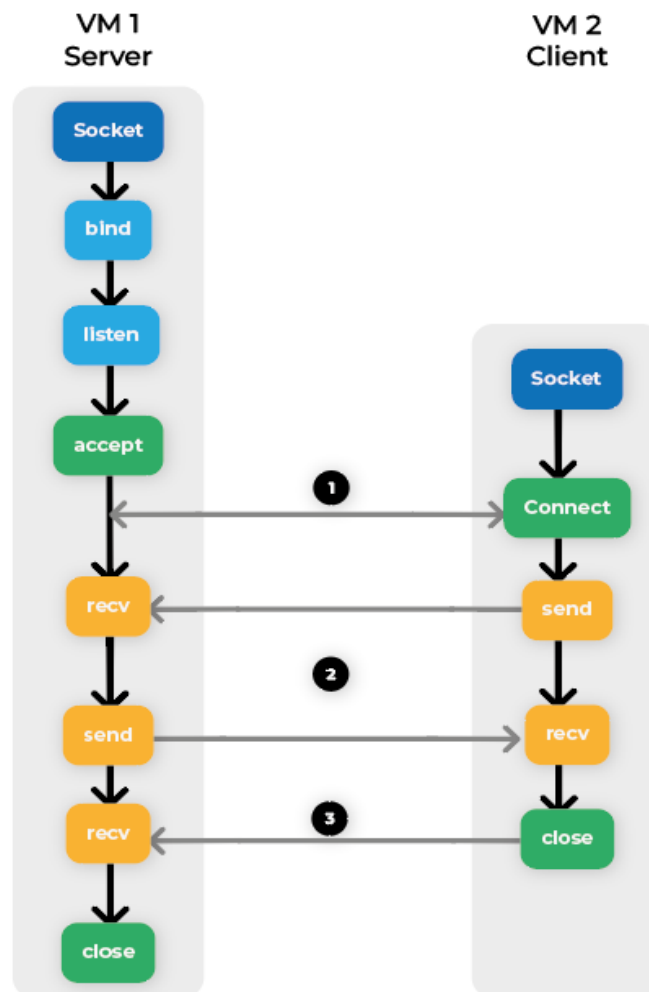
## Les modèles d'architecture de communication



Dans ce tp on utilisera l'interface réseau ethernet "eth0" pour communication entre sockets .

## 2. Architecture Client-Server

Dans le module socket de python



1. Etablissement de connection (3-way handshake) liaison à trois voies
2. Transfère de donnée entre client et server
3. Fermeture de la communication entre client et server

Au mode Connecté , on spécifie un socket de flux. pour une communication fiable

## 3. Programmation

Le module socket de Python fournit une interface à l'API des sockets de Berkeley.

Python fournit une API pratique et cohérente qui mappe directement aux appels système, leurs homologues C.

Voir la documentation de python-socket :

[Programmation de sockets en Python \(Guide\) - Real Python](#)

## 3.1. Programmation coté client

```
GNU nano 6.2 client.py
# encoding : utf-8
# author : Ibrahim Lahliou
# code : client.py
import socket as skt

host='192.168.100.4'
port=7999

print('""',end="\n\n")
print('"" TP1 CLI')
print('""')
with skt.socket() as s:
    s.connect((host,port))
    while True :
        msg=input('client@vm2:7999$ ')
        s.sendall(msg.encode())
        data = s.recv(1024).decode()
        print("from server",data)
        if msg == 'exit':
            break
    s.close()
```

[ Read 23 lines ]

## 3.2 Programmation coté serveur

```
GNU nano 6.2 server.py
# encoding : utf-8
# author : Ibrahim Lahliou
# code : server.py
import socket as skt

port= 7999
print('""',end="\n\n")
print('"" TP1 Server ""')
print('""')
with skt.socket(skt.AF_INET,skt.SOCK_STREAM) as s:
    s.bind(('',port))
    s.listen()
    print('Listening ...')
    conn,addr=s.accept()
    with conn:
        print('Connected by :', addr)
        while True :
            data=conn.recv(1024)
            print("from client",repr(data.decode()))
            msg=input("server@vm1:7999$ ")
            conn.send(msg.encode())
            if msg == "exit":
                break
            if not data:
                break
            conn.sendall(data)
    s.close()
```

[ Read 29 lines ]

**SOCK\_STREAM** : Spécifie un socket de flux. Doit être fourni lorsqu'un client ou un serveur basé sur une connexion de streaming est nécessaire et sera utilisé pour une communication fiable , utilisant le protocole TCP

## 4. Exécution du Programme

### 0. Verification du port

`netstat -an`  
`sudo lsof -i :7999`

```
lldvm1@vm1:~$ netstat -an

```

unix	3	[ ]	STREAM	CONNECTED	20363	/run/systemd/journal/stdout
unix	3	[ ]	DGRAM	CONNECTED	19719	
unix	3	[ ]	DGRAM	CONNECTED	18040	
unix	3	[ ]	DGRAM	CONNECTED	21709	
unix	3	[ ]	STREAM	CONNECTED	20322	
unix	2	[ ]	DGRAM	CONNECTED	19096	
unix	2	[ ]	DGRAM	CONNECTED	20251	

```
lldvm1@vm1:~$ sudo lsof -i :7999
lldvm1@vm1:~$
```

Le port n'est pas occupé , on peut donc l'utiliser pour la communication client-server  
sinon on pourra utiliser la commande kill

`sudo kill 12345`

### 1. Exécution du server

Server :

```
lldvm1@vm1:~$ python3 server.py
```

```
TP1 Server
-----
Listening ...
```

le server attends le client pour qu'il se connecte ...

### 2. Exécution du client

Client :

```
lldvm2@vm2:~$ python3 client.py
```

```
TP1 CLI
-----
client@vm2:7999$
```

le client se connecte au server

Server :

```
TP1 Server
-----
Listening ...
Connected by : ('192.168.100.5', 49240)
_
```

le server accepte la connexion du client

### 3. Transfère de donn  depuis le client

Client :

```
TP1 CLI
-----
client@vm2:7999$ Hello vm1
_
```

Envoie de donn  , le client attends le retour du server

Server :

```
TP1 Server
-----
Listening ...
Connected by : ('192.168.100.5', 36222)
from client 'Hello vm1'
server@vm1:7999$ _
```

R ception de donn  , le server peut r pondre

### 4. Transf re de donn  depuis le server

Server :

```
TP1 Server
-----
Listening ...
Connected by : ('192.168.100.5', 36222)
from client 'Hello vm1'
server@vm1:7999$ Hello ! ,What's up VM2
_
```

Envoie de donn  , le server attends le retour du server

Client :

```
TP1 CLI
-----
client@vm2:7999$ Hello vm1
from server Hello ! ,What's up VM2
client@vm2:7999$ _
```

R ception de donn  , le client peut r pondre

### 5. Extinction de la session

Client :

```
TP1 CLI
-----
client@vm2:7999$ Hello vm1
from server Hello ! ,What's up VM2
client@vm2:7999$ exit
from server Hello vm1
ildvm2@vm2:~$ _
```

Le client mets fin   la communication