Compte rendu TP3 Remote Procedural Call avec python

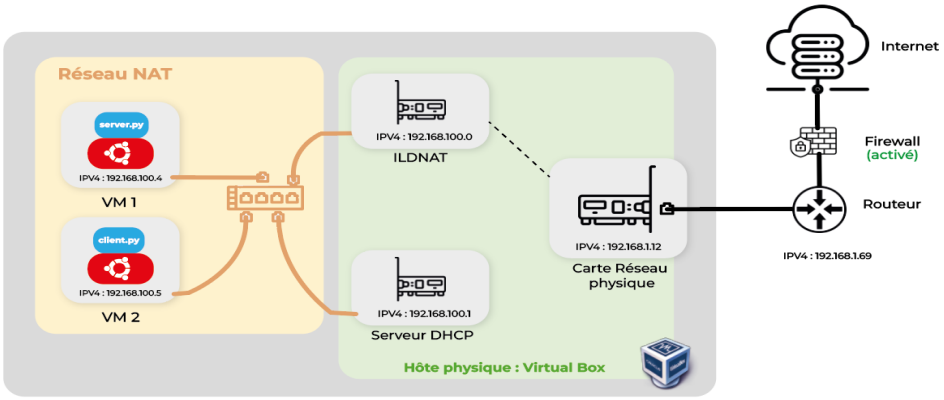
Nom : Ibrahim Lahlou

0. Programmation de Socket

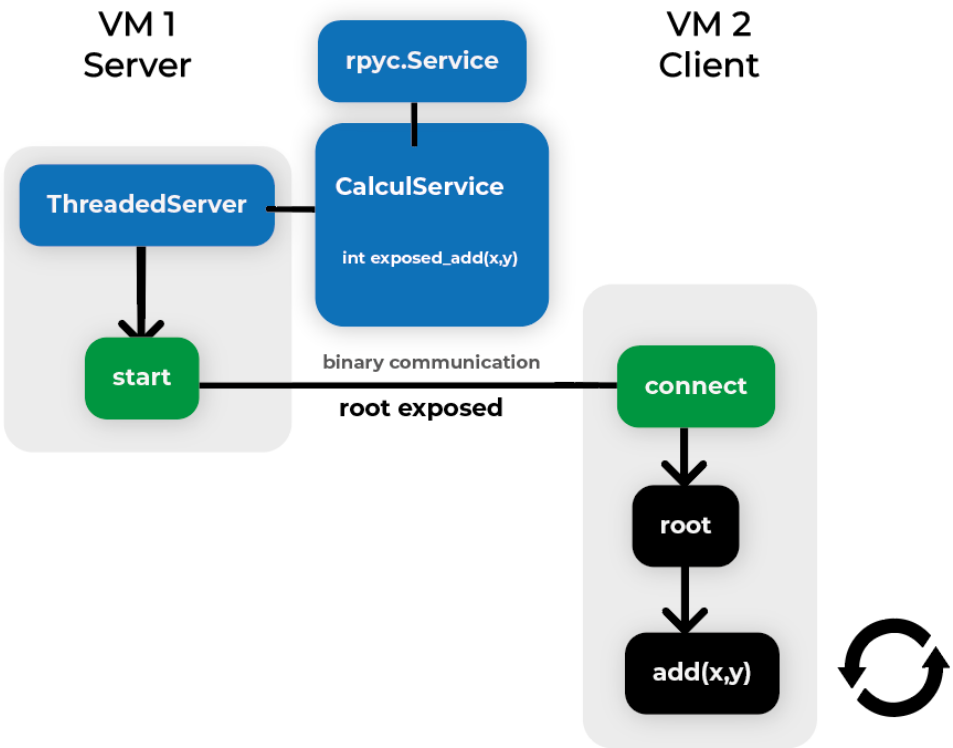
Création d'un programme communicant via avec python entre les deux machine VM1 et VM2 des instances d'ubuntu server avec l'appel des procédure en utilisant d'une part le rpyc

1 . Architecture réseau de cette manipulation

- **Réseau NAT**

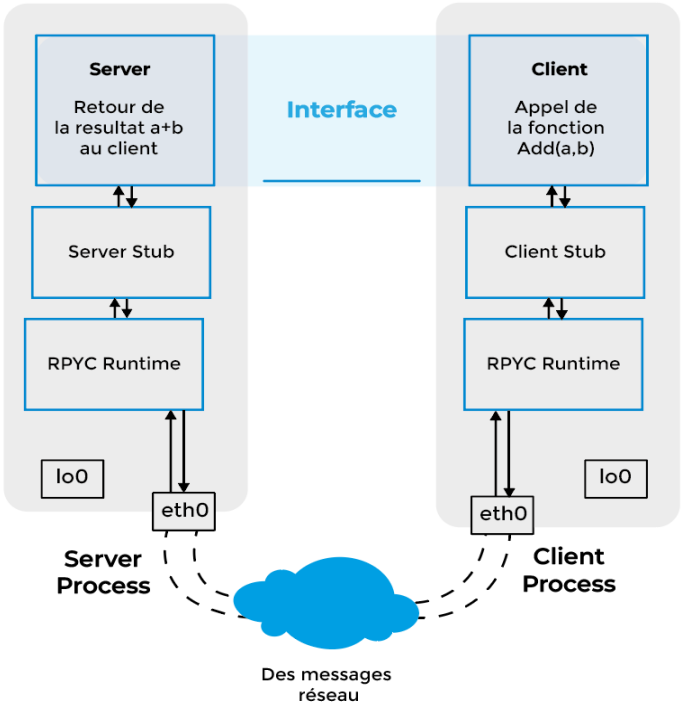


2. Architecture Client-Server



3. Programmation

1. C'est quoi rpyc ?
   C'est une bibliothèque Python qui permet de faire des appels de procédures à distance entre un client Python et un serveur Python, en utilisant un protocole de communication.



L'idée est de permettre à un client Python d'appeler des fonctions ou des méthodes d'objets situés sur un serveur Python, comme s'ils étaient locaux, sans avoir à se soucier des détails de la communication réseau.

2. Comment fonctionne t'il ?
   Le fonctionnement de RPYC sans contrat repose sur un modèle de programmation appelé "réflexion" (ou "reflection" en anglais). Contrairement aux mécanismes RPC réguliers tels que ONC RPC,RPyC est transparent, symétrique et ne nécessite pas de langages de définition ou de décoration spéciaux
   Dans ce modèle, le client Python peut appeler des méthodes sur un objet distant sans avoir connaissance de l'interface complète de cet objet à l'avance.
   En d'autres termes, le client n'a pas besoin d'un contrat formel ou d'une définition d'interface stricte pour communiquer avec le serveur.

3. Remarque :
   Cependant, cela signifie que le client doit connaître les noms des méthodes et des objets sur le serveur, ainsi que les arguments et les valeurs de retour associés à ces méthodes.

3.1. Programmation coté client

```
  GNU nano 6.2                                client_rpc.py
# encoding : utf-8
# author : Ibrahim Lahlou
# code : client_rpc.py

import rpyc

conn = rpyc.connect("192.168.100.4",7000)

print("                              TP3 RPC client")
print("                                                                                ")
print(" We will calculate the sum of two number  a + b =")

while True :
        a=int(input("a = "))
        b=int(input("b = "))
        result = conn.root.add(a,b)
        print("a+b =",result)
        print("")
        c=input("exit (y/n) :")
        if c=='y':
                break
        print("")
```

3.2 Programmation coté serveur

```
  GNU nano 6.2                                server_rpc.py
# encoding : utf-8
# author : Ibrahim Lahlou
# code : server_rpc.py

import rpyc
from rpyc.utils.server import ThreadedServer


class CalculService(rpyc.Service):
        def exposed_add(self,x,y):
                return x+y
        def exposed_mul(self,x,y):
                return x*y

if __name__ == "__main__":
        server=ThreadedServer(CalculService,hostname="0.0.0.0",port=7000)
        server.start()
```

5. Éxecution du Programme

0. Verification du port

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

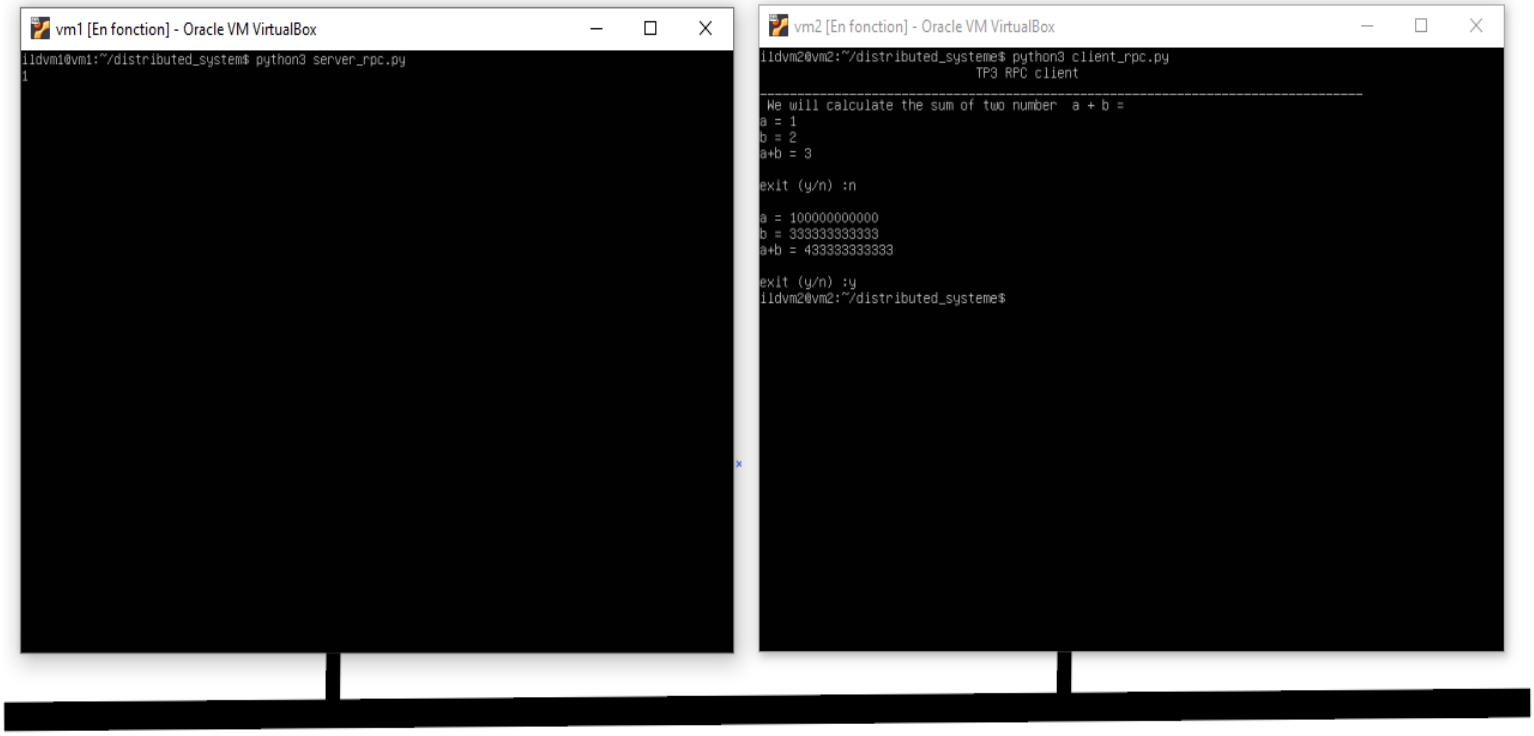
```
ildvm1@vm1:~$ netstat -an

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                    20251
ildvm1@vm1:~$ sudo lsof -i :7999
```

Le port n'est pas occupé , on peut donc l'utilisez pour la communication client-server

sinon on pourra utilisez la commande kill

```
sudo kill [identifiant du packet]
```

1. Lancement du Server
2. Lancement du client
3. Lecture des paramètre
4. Traitement et reception de la valeur de retour



On pourra aussi utiliser rpyc_classic.py pour connecter dans la meme machine les deux programme, il faudra just lancer le server en arrière plan par exemple avec nohup rpyc_classic.py &