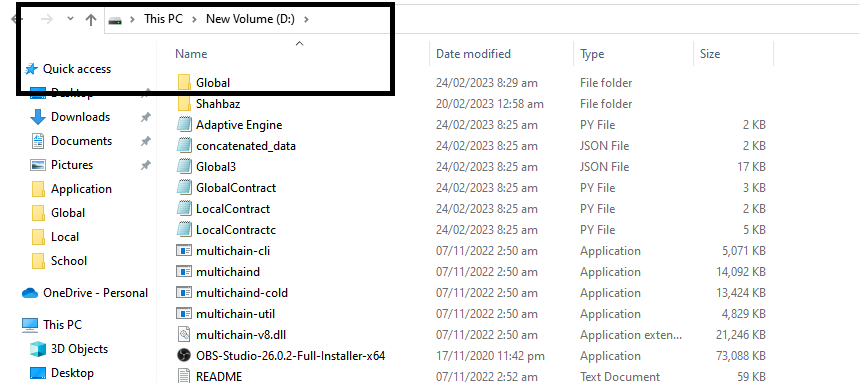
**Wallet Lab Solution**

**Step-1**

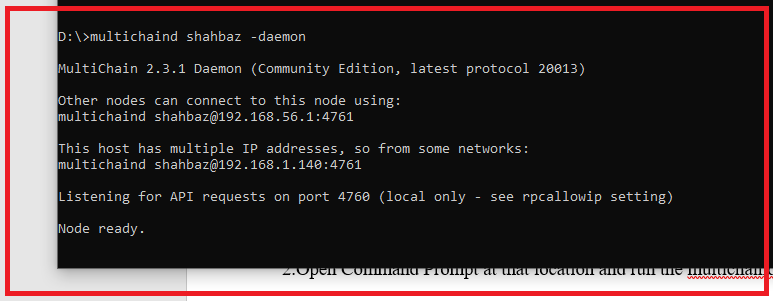
Run Multichain Server

1.Start the Node where you install mutichain



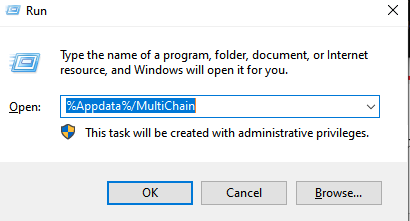
Highlighted is my Location

2.Open Command Prompt at that location and run the multichaind server nodes

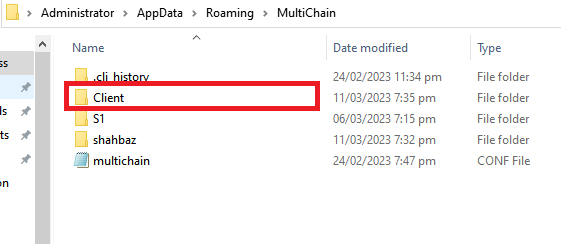


3.To run the client on the same node do the below step

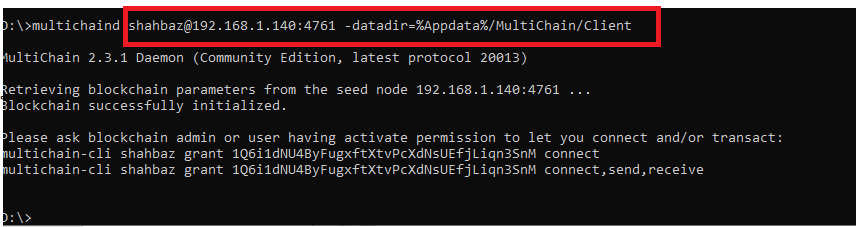
* Run the following command at the run box



* Create the empty folder in which we connect the server node



* To start client node run the new command prompt at the same location where you start server node



[shahbaz@192.168.1.140:4761](mailto:shahbaz@192.168.1.140:4761) represent the server address

-datadir=%Appdata%/Multichain/Client is the location where server copies the chain to this node

* Open New terminal to perform the further administrative task such as fetching data, granting or etc to client and server node
* Note inorder to perform these task the calling of server and client is different

Such as

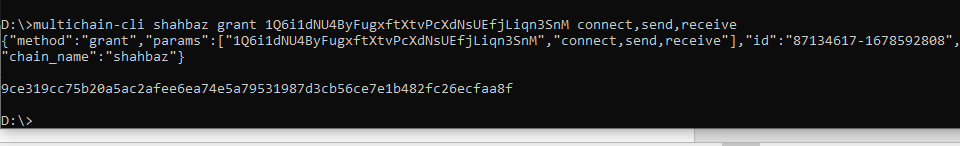
For Server

multichain-cli shahbaz

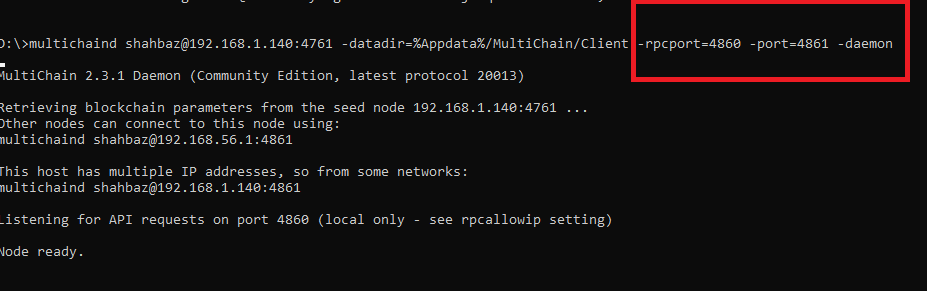
For Client



* Start New terminal for administrative work , first granting the client address



Now Start the Client Node



-rpcport=4680 –port=4861 –daemon

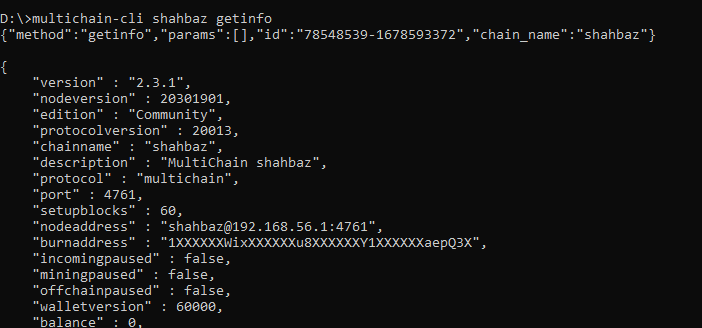
The highlighted command is provide a TCP connection to different port for the client node

1. **Run the command getinfo inorder to get the chain information of client node and server node**

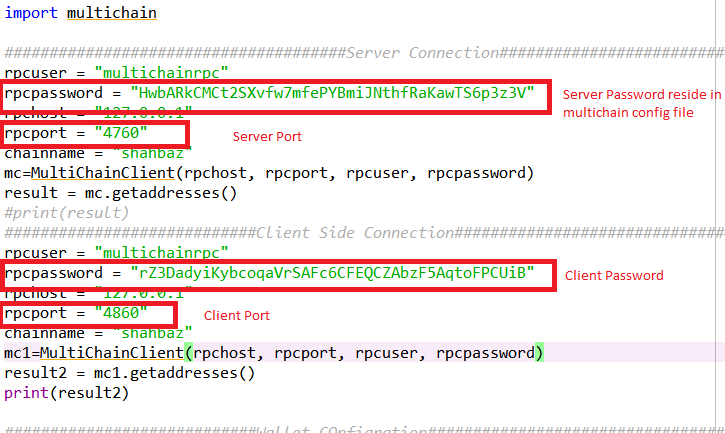
**For Client**



**For Server**



**Now Connect the client and Server with Python Code**

****

Import python

###############Server Connection################

rpcuser = "multichainrpc"

rpcpassword = "HwbARkCMCt2SXvfw7mfePYBmiJNthfRaKawTS6p3z3V"

rpchost = "127.0.0.1"

rpcport = "4760"

chainname = "shahbaz"

mc=MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc.getaddresses()

#print(result)

##########Client Side Connection####################

rpcuser = "multichainrpc"

rpcpassword = "rZ3DadyiKybcoqaVrSAFc6CFEQCZAbzF5AqtoFPCUiB"

rpchost = "127.0.0.1"

rpcport = "4860"

chainname = "shahbaz"

mc1=MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result2 = mc1.getaddresses()

print(result2)

**Python FullCode For Wallet Transaction**

import multichain

######################################Server Connection############################

rpcuser = "multichainrpc"

rpcpassword = "HwbARkCMCt2SXvfw7mfePYBmiJNthfRaKawTS6p3z3V"

rpchost = "127.0.0.1"

rpcport = "4760"

chainname = "shahbaz"

mc=MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc.getaddresses()

#print(result)

############################Client Side Connection###############################

rpcuser = "multichainrpc"

rpcpassword = "rZ3DadyiKybcoqaVrSAFc6CFEQCZAbzF5AqtoFPCUiB"

rpchost = "127.0.0.1"

rpcport = "4860"

chainname = "shahbaz"

mc1=MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result2 = mc1.getaddresses()

#print(result2)

############################Wallet COnfigration#################################

"""

getaddresses true

Choose any address with "ismine" : true, which means that this node’s

wallet contains the private key for the address.

"""

result = mc.getaddresses(True)

pub1=result[0]['pubkey']

"""

Now run the same getaddresses true command on the second server,

again choosing an address with "ismine" : true.

"""

#r=mc1.getnewaddress()

result2 = mc1.getaddresses(True)

pub2=result2[0]['pubkey']

#print("",pub1,pub2)

###########################Create Walletinformation Create on ServerSide###########################

"""

addmultisigaddress 2 '["0...", "0..."]'

"""

address = mc.addmultisigaddress(2, [pub1, pub2])

print(address)

"""

The response should be empty. Now run the same command on the other server,

to add the address to the wallet and start tracking its balance:

addmultisigaddress 2 '["0...", "0..."]'

"""

address1 = mc1.addmultisigaddress(2, [pub1, pub2])

print(address1)

"""

Issuing an asset to the multisig address

For most blockchain actions, a multisig address requires its own

permissions, independent of the permissions of the individual regular

addresses that were combined to create it (more details here).

Let’s grant these permissions on the first server:

grant 4... receive,send

The txid of the grant transaction should be displayed in the response. Now let’s issue a new asset directly to the multisig address:

"""

txid1 = mc.grant(address, 'receive,send') # global permission

#txid12 = mc1.grant(address, 'send,receive') # global permission

"""

The txid of the grant transaction should be displayed in the response.

Now let’s issue a new asset directly to the multisig address:

issue 4... asset9 10000 0.01

And now let’s check the multisig address has received the

funds successfully:

getaddressbalances 4... 0

A balance of 10000 units of asset9 should be displayed.

"""

txid2 = mc.issue(address, {'name' : 'Walletx'}, 1000, 0.01)

result32 = mc.getaddressbalances(address, 0)

print("Blance",result32)

###########################This particular address send and recevieammount on Serverside###

""""

Spending funds from the multisig address

Still on the first server, let’s create a new regular address to receive

some funds from the multisig:

getnewaddress

Copy and paste the new address here:

1...

Now let’s grant this address receive permissions, so it can be sent some funds:

grant 1... receive

"""

txid = mc.grant(result[1]['address'], 'send,receive') # global permission

print("Address Two ",result[1]['address'])

print(address)

print(result[1]['address'])

"""

Now we begin the process of building the transaction which sends

funds from the multisig address to this new address.

Because this is a 2-of-2 multisig, the process will require a

signature from both servers. Let’s begin on the first server:

createrawsendfrom 4... '{"1...":{"asset9":500}}' '[]' sign

"""

result69 = mc.createrawsendfrom(address, {result[1]['address']:{"Walletx":90}}, [], 'sign')

#print(result69)

Signature2=result69['hex']

print(Signature2)

"""

The response should contain a complete field with value false, along with a large hexadecimal blob in the hex field. This hexadecimal blob is the raw transaction,

which has been partially signed, and should be copied to the clipboard.

Now switch to the second server and run the following,

pasting the raw transaction from the clipboard where shown:

"""

"""

signrawtransaction [paste-hex-blob]

The response should contain a complete field with value true,

along with an even larger hexadecimal blob in the hex field.

This means that the transaction has enough signatures to be valid,

and is ready for broadcasting to the blockchain.

"""

result123 = mc1.signrawtransaction(result69['hex'])

print(result123)

"""

Copy the new hexadecimal blob, and run:

sendrawtransaction [paste-bigger-hex-blob]

"""

txid = mc.sendrawtransaction(result123['hex'])

print(txid)

"""

getaddressbalances 4... 0

And on the first server only, check the new address’s balance

(including unconfirmed transactions):

getaddressbalances 1... 0

"""

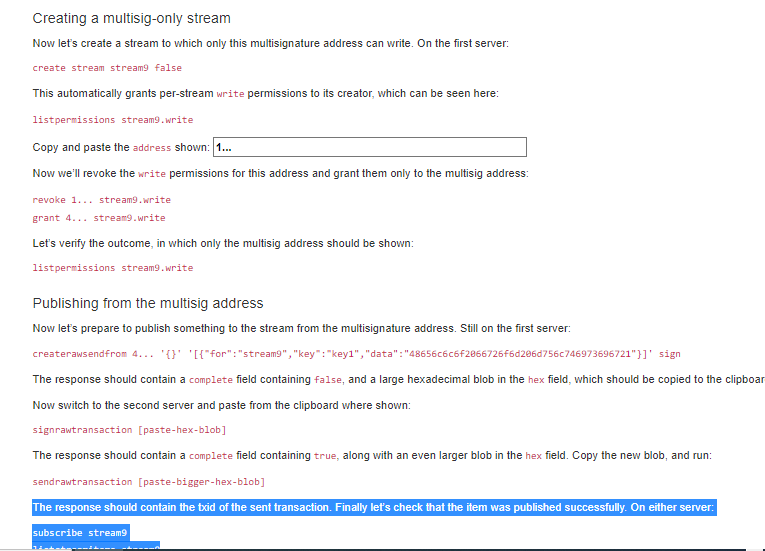
result32 = mc.getaddressbalances(address, 0)

print("Blance",result32)

result32 = mc1.getaddressbalances(address, 0)

print("Blance",result32)

**For Creating Multisignature Stream Follow the Following steps from the Experiment**



**Full Code**

import multichain

######################################Server Connection############################

rpcuser = "multichainrpc"

rpcpassword = "HwbARkCMCt2SXvfw7mfePYBmiJNthfRaKawTS6p3z3V"

rpchost = "127.0.0.1"

rpcport = "4760"

chainname = "shahbaz"

mc=MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc.getaddresses()

#print(result)

############################Client Side Connection###############################

rpcuser = "multichainrpc"

rpcpassword = "rZ3DadyiKybcoqaVrSAFc6CFEQCZAbzF5AqtoFPCUiB"

rpchost = "127.0.0.1"

rpcport = "4860"

chainname = "shahbaz"

mc1=MultiChainClient(rpchost, rpcport, rpcuser, rpcpassword)

result = mc1.getaddresses()

#print(result2)

############################Wallet COnfigration#################################

"""

getaddresses true

Choose any address with "ismine" : true, which means that this node’s

wallet contains the private key for the address.

"""

result = mc.getaddresses(True)

pub1=result[0]['pubkey']

"""

Now run the same getaddresses true command on the second server,

again choosing an address with "ismine" : true.

"""

#r=mc1.getnewaddress()

result2 = mc1.getaddresses(True)

pub2=result2[0]['pubkey']

#print("",pub1,pub2)

###########################Create Walletinformation Create on ServerSide###########################

"""

addmultisigaddress 2 '["0...", "0..."]'

"""

address = mc.addmultisigaddress(2, [pub1, pub2])

print(address)

"""

The response should be empty. Now run the same command on the other server,

to add the address to the wallet and start tracking its balance:

addmultisigaddress 2 '["0...", "0..."]'

"""

address1 = mc1.addmultisigaddress(2, [pub1, pub2])

print(address1)

"""

Creating a multisig-only stream

Now let’s create a stream to which only this multisignature address

can write. On the first server:

create stream stream9 false

This automatically grants per-stream write permissions to its creator,

which can be seen here:

listpermissions stream9.write

"""

txid = mc.create('stream', 'shahbaz1', False) # open to all to write

re1 = mc.listpermissions('shahbaz1.write') # all permissions

#print(result[0]['address'])

"""

Now we’ll revoke the write permissions for this address and

grant them only to the multisig address:

revoke 1... stream9.write

grant 4... stream9.write

"""

#txid = mc.revoke(result[0]['address'],'shahbaz1.write')

#print(txid)

txid = mc.grant(address, 'shahbaz1.write') # per-entity permission

print(txid)

"""

Publishing from the multisig address

Now let’s prepare to publish something to the stream from the

multisignature address. Still on the first server:

createrawsendfrom 4... '{}' '[{"for":"stream9","key":"key1","data":"48656c6c6f2066726f6d206d756c746973696721"}]' sign

The response should contain a complete field containing

false, and a large hexadecimal blob in the hex field,

which should be copied to the clipboard.

"""

re4 = mc.createrawsendfrom(address,{result[1]['address']: 0}, [{'for' : 'shahbaz1', 'key' : 'key1', 'data' : 'a1b2c3d4'}], 'sign')

print(re4)

"""

Now switch to the second server and paste from the clipboard where shown:

signrawtransaction [paste-hex-blob]

"""

Signature2=re4['hex']

print(Signature2)

result123 = mc1.signrawtransaction(re4['hex'])

print(result123)

"""

The response should contain a complete field containing true, along with an

even larger blob in the hex field. Copy the new blob, and run:

"""

"""

sendrawtransaction [paste-bigger-hex-blob]

"""

txid = mc.sendrawtransaction(result123['hex'])

print(txid)

"""

The response should contain the txid of the sent transaction.

Finally let’s check that the item was published successfully.

On either server:

subscribe stream9

liststreamitems stream9

"""

Server=mc.subscribe('shahbaz1')

Client=mc1.subscribe('shahbaz1')

result = mc.liststreamitems('shahbaz1')

print(result)

"""

The response should contain the txid of the sent transaction.

Finally let’s check that the item was published successfully.

On either server:

subscribe stream9

liststreamitems stream9

You should see the item listed with the multisig address shown

in the publishers field, as well as the key and data entered above.

"""