Practical 3

Aim: Interact with a block chain network. Execute transactions and requests against a block chain network by creating an app to test the network and its rules.

Installing dependencies

```
sudo apt install make
```

Install GEth (Go Ethereum)

Geth is a CLI with some resources to connect you to the Ethereum network. It will be used to start our private network in the local environment.

To install Geth in Ubuntu/Debian follow the following steps:

```
git clone -b release/1.11 --depth 1
https://github.com/ethereum/go-ethereum.git
```

```
→ web3 git clone -b release/1.11 --depth 1 https://github.com/ethereum/go-ethereum.git
Cloning into 'go-ethereum'...
remote: Enumerating objects: 2161, done.
remote: Counting objects: 100% (2161/2161), done.
remote: Compressing objects: 100% (1820/1820), done.
remote: Total 2161 (delta 240), reused 1659 (delta 214), pack-reused 0
Receiving objects: 100% (2161/2161), 13.10 MiB | 1.22 MiB/s, done.
Resolving deltas: 100% (240/240), done.
→ web3
```

```
cd go-ethereum
make geth
sudo ln -f ./build/bin/geth /usr/local/bin
```

```
→ web3 cd go-ethereum
→ go-ethereum git:(release/1.11) make geth
env GO111MODULE=on go run build/ci.go install ./cmd/geth
go: downloading github.com/cespare/cp v0.1.0
go: downloading golang.org/x/crypto v0.1.0
go: downloading github.com/Azure/azure-sdk-for-go/sdk/storage/azblob v0.3.0
go: downloading github.com/Azure/azure-sdk-for-go/sdk/azcore v0.21.1
go: downloading github.com/Azure/azure-sdk-for-go/sdk/internal v0.8.3
go: downloading golang.org/x/net v0.8.0
go: downloading golang.org/x/text v0.8.0
```

```
→ go-ethereum git:(release/1.11) sudo ln -sf ./build/bin/geth /usr/local/bin
→ go-ethereum git:(release/1.11)
```

Verifying installation

```
→ go-ethereum git:(release/1.11) geth -v
geth version 1.11.6-stable-ea9e62ca
→ go-ethereum git:(release/1.11)
```

The Genesis Block

All blockchains start with the genesis block.This block defines the initial configuration to a blockchain. The configuration to genesis block is defined in *genesis.json* file.

So, let's create a folder inside the web3 folder to start the private network and create the *genesis.json* file.

```
cd .. && mkdir blockchain
cd blockchain
touch genesis.json
```

```
→ go-ethereum git:(release/1.11) cd .. && mkdir blockchain
→ web3 cd blockchain
→ blockchain touch genesis.json
→ blockchain
```

Config for genesis.json

```
{
  "config": {
      "chainId": 6969,
      "homesteadBlock": 0,
      "eip150Block": 0,
      "eip155Block": 0,
      "eip158Block": 0,
      "byzantiumBlock": 0,
      "constantinopleBlock": 0,
      "petersburgBlock": 0,
      "ethash": {}
  },
  "difficulty": "4",
  "gasLimit": "8000000",
  "alloc": {}
}
```

Start Database (The Bootnode)

The bootnode is the first node created when the blockchain is started. To start the database to first node execute:

```
geth init --datadir node1 genesis.json
```

The folder *node1* will be created with the database to bootnode.

The terminal result must be something like this:

```
→ blockchain geth init --datadir node1 genesis.json
INFO [02-14|20:13:50.125] Maximum peer count                       ETH=50 LES=0 total=50
INFO [02-14|20:13:50.127] Smartcard socket not found, disabling    err="stat /run/pcscd/pcs
INFO [02-14|20:13:50.133] Set global gas cap                       cap=50,000,000
INFO [02-14|20:13:50.135] Using leveldb as the backing database
INFO [02-14|20:13:50.136] Allocated cache and file handles         database=/home/hackerman
INFO [02-14|20:13:50.144] Using LevelDB as the backing database
INFO [02-14|20:13:50.162] Opened ancient database                  database=/home/hackerman
INFO [02-14|20:13:50.162] Writing custom genesis block
INFO [02-14|20:13:50.163] Freezer shutting down
INFO [02-14|20:13:50.163] Successfully wrote genesis state         database=chaindata hash=
INFO [02-14|20:13:50.163] Using leveldb as the backing database
INFO [02-14|20:13:50.163] Allocated cache and file handles         database=/home/hackerman
INFO [02-14|20:13:50.168] Using LevelDB as the backing database
INFO [02-14|20:13:50.183] Opened ancient database                  database=/home/hackerman
INFO [02-14|20:13:50.183] Writing custom genesis block
INFO [02-14|20:13:50.184] Successfully wrote genesis state         database=lightchaindata
→ blockchain
```

Start Node

```
geth --datadir node1 --networkid 6969 --http
--allow-insecure-unlock --nodiscover
--rpc.enabledeprecatedpersonal --http.corsdomain "*"
```

- This command will start the node in the private network with id 6969.
- The flag *--http* is used to enable **Web App Access**, we will use this in next post to connect Metamask with that private network.
- The flag *--allow-insecure-unlock* is used to permit execute transfers without a web application, we will use this in tests to this network.
- The flag *--nodiscover* is used to prevent node from trying to connect to others automatically.

The terminal result must be something like this:

```
WARN [02-14|20:14:12.788] Failed to load snapshot                  err="missing or corrupted snapshot"
INFO [02-14|20:14:12.788] Rebuilding state snapshot
INFO [02-14|20:14:12.788] Resuming state snapshot generation       root=56e81f..63b421 accounts=0 slots=0 storage=0.00B dangling=0 elapsed="26
INFO [02-14|20:14:12.788] Regenerated local transaction journal    transactions=0 accounts=0
INFO [02-14|20:14:12.788] Generated state snapshot                 accounts=0 slots=0 storage=0.00B dangling=0 elapsed="695.882µs"
INFO [02-14|20:14:12.789] Gasprice oracle is ignoring threshold set threshold=2
WARN [02-14|20:14:12.789] Error reading unclean shutdown markers   error="leveldb: not found"
WARN [02-14|20:14:12.789] Engine API enabled                       protocol=eth
WARN [02-14|20:14:12.789] Engine API started but chain not configured for merge yet
INFO [02-14|20:14:12.789] Starting peer-to-peer node               instance=Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0
INFO [02-14|20:14:12.803] IPC endpoint opened                      url=/home/hackerman/web3/blockchain/node1/geth.ipc
INFO [02-14|20:14:12.803] New local node record                    seq=1,707,921,852,799 id=d8fd370aa42d6411 ip=127.0.0.1 udp=0 tcp=30303
INFO [02-14|20:14:12.803] Started P2P networking                   self="enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665
90511@127.0.0.1:30303?discport=0"
INFO [02-14|20:14:12.803] Generated JWT secret                     path=/home/hackerman/web3/blockchain/node1/geth/jwtsecret
INFO [02-14|20:14:12.804] HTTP server started                      endpoint=127.0.0.1:8545 auth=false prefix= cors= vhosts=localhost
INFO [02-14|20:14:12.805] WebSocket enabled                        url=ws://127.0.0.1:8551
INFO [02-14|20:14:12.805] HTTP server started                      endpoint=127.0.0.1:8551 auth=true  prefix= cors=localhost vhosts=localhost
```

Done!! The blockchain in the private network is running.

Let's go do some tests to verify this. Open another terminal window in the same folder *my-blockchain* and execute the following command:

```
geth attach node1/geth.ipc
```

This opens an interactive javascript terminal to execute some tasks to this node. The terminal result should be something like this:

```
→ blockchain geth attach node1/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0
at block: 0 (Thu Jan 01 1970 05:30:00 GMT+0530 (IST))
 datadir: /home/hackerman/web3/blockchain/node1
 modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d or type exit
> 
```

```
> admin.nodeInfo
{
  enode: "enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665755487c5830ece35961b3887ac0bc3ecc5056cdfe
466cd7ff552d15e6b47bd90511@127.0.0.1:30303?discport=0",
  enr: "enr:-Jy4QLLNs0fuNR76iB7Jh91tSnUrPBvEurMiUC3NY6uO7cOFGTWqlmIXj93bkbwtvhNpKSWPQzcez98wkA2jQ3cpZcqGAY2oE0l_g2V0aMfG
hL3NduuAgmlkgnY0gmlwhH8AAAGJc2VjcDI1NmsxoQNK-abWszs8gmCrXYsQZLC39kNOV5FIb7bzpJBUIWZXVYRzbmFwwIN0Y3CCdl8",
  id: "d8fd370aa42d6411bccbb36efe2f140fed669ce42278392d09d73584a983e1e8",
  ip: "127.0.0.1",
  listenAddr: "[::]:30303",
  name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
  ports: {
    discovery: 0,
    listener: 30303
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 0,
        chainId: 6969,
        constantinopleBlock: 0,
        eip150Block: 0,
        eip155Block: 0,
        eip158Block: 0,
        ethash: {},
        homesteadBlock: 0,
        petersburgBlock: 0
      },
      difficulty: 4,
      genesis: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      head: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      network: 6969
    },
    snap: {}
  }
}
```

Adding member peers

Let's start the second node to our blockchain. To do this execute the following commands in new terminal inside the same directory:

```
geth init --datadir node2 genesis.json
```

```
→ blockchain geth init --datadir node2 genesis.json
INFO [02-14|20:19:56.795] Maximum peer count                       ETH=50 LES=0 total=50
INFO [02-14|20:19:56.796] Smartcard socket not found, disabling    err="stat /run/pcscd/pcscd.comm: n
o such file or directory"
INFO [02-14|20:19:56.799] Set global gas cap                       cap=50,000,000
INFO [02-14|20:19:56.801] Using leveldb as the backing database
INFO [02-14|20:19:56.801] Allocated cache and file handles         database=/home/hackerman/web3/bloc
```

```
geth --datadir node2 --networkid 6969 --port 42069 --authrpc.port
8552 --rpc.enabledeprecatedpersonal --http.corsdomain "*"
```

The commands are very similar to the previous ones with some minor differences:

- The folder to database in this case is *node2*.
- The flags *--http* and *--allow-insecure-unlock* are not necessary.
- The flag *--port* it's necessary to differentiate this node from the previous one. The first node is running on default port **30303**.

Open a new javascript terminal to second node created and verify the informations executing:

```
geth attach node2/geth.ipc
```

```
> admin.nodeInfo
{
  enode: "enode://a69479424a095bcfc6189a5f74b27cb1ec0e2b401f1ab4dd0e9410e63530fd83e26003e900e2b994f0644966d2891b7d05c366
d680142cc4923c72eecb922498@103.127.252.29:42069?discport=60626",
  enr: "enr:-Ku4QGmS8AgHtZTiGwh7uGBCYNksu2OM0CagmiRqKwOppXAHCHih3VPTF2U1Vg0mfNLSG6DgH5n9inYx8isGrf11wPeGAY2oGkIog2V0aMfG
hL3NduuAgmlkgnY0gmlwhGd__B2Jc2VjcDI1NmsxoQKmlHlCSglbz8YYml90snyx7A4rQB8atN0OlBDmNTD9g4RzbmFwwIN0Y3CCpFWDdWRwguzShHVkcDaC
pFU",
  id: "7571846621b25325453365506996d754d445a6a408d368da6901702177aa213b",
  ip: "103.127.252.29",
  listenAddr: "[::]:42069",
  name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
  ports: {
    discovery: 60626,
    listener: 42069
  },
  protocols: {
    eth: {
      config: {
        byzantiumBlock: 0,
        chainId: 6969,
        constantinopleBlock: 0,
        eip150Block: 0,
        eip155Block: 0,
        eip158Block: 0,
        ethash: {},
        homesteadBlock: 0,
        petersburgBlock: 0
      },
      difficulty: 4,
      genesis: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      head: "0xac4d9a5f75e2d6ce6831373d57279fe25c60cbfe82da5b3adfe620d7fe2ca582",
      network: 6969
    },
    snap: {}
  }
}
>
```

Connecting Peers

Now, let's create the peer-to-peer connection between the two nodes in blockchain.

Copy the **enode** from the *nodeInfo* of the **second** node, then execute the following command in javascript terminal of the **first** node

```
admin.addPeer("<enode value of second")
```

```
> admin.addPeer("enode://a69479424a095bcfc6189a5f74b27cb1ec0e2b401f1ab4dd0e9410e63530fd83e26003e900e2b994f0644966d2891b7d05c366d680142cc4923c72eecb922498@103.127.252.29:42069?discport=60626")
true
>
```

Copy the **enode** from the *nodeInfo* of the **first** node, then execute the following command in javascript terminal of the **second** node

```
admin.addPeer("<enode value of first")
```

```
> admin.addPeer("enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665755487c5830ece35961b3887ac0bc3ecc5056cdfe466cd7ff552d15e6b47bd90511@127.0.0.1:30303?discport=0")
true
```

Node 1:

```
> admin.peers
[{
    caps: ["eth/66", "eth/67", "eth/68", "snap/1"],
    enode: "enode://a69479424a095bcfc6189a5f74b27cb1ec0e2b401f1ab4dd0e9410e63530fd83e26003e900e2b994f0644966d2891b7d05c366d680142cc4923c72eecb922498@127.0.0.1:49398",
    id: "7571846621b25325453365506996d754d445a6a408d368da6901702177aa213b",
    name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
    network: {
      inbound: true,
      localAddress: "127.0.0.1:30303",
      remoteAddress: "127.0.0.1:49398",
      static: false,
      trusted: false
    },
    protocols: {
      eth: {
        version: 68
      },
      snap: {
        version: 1
      }
    }
}]
>
```

Node: 2

```
> admin.peers
[{
    caps: ["eth/66", "eth/67", "eth/68", "snap/1"],
    enode: "enode://4af9a6d6b33b3c8260ab5d8b1064b0b7f6434e5791486fb6f3a4905421665755487c5830ece35961b3887ac0bc3ecc5056cdfe466cd7ff552d15e6b47bd90511@127.0.0.1:30303?discport=0",
    id: "d8fd370aa42d6411bccbb36efe2f140fed669ce42278392d09d73584a983e1e8",
    name: "Geth/v1.11.6-stable-ea9e62ca/linux-amd64/go1.22.0",
    network: {
      inbound: false,
      localAddress: "127.0.0.1:49398",
      remoteAddress: "127.0.0.1:30303",
      static: true,
      trusted: false
    },
    protocols: {
      eth: {
        version: 68
      },
      snap: {
        version: 1
      }
    }
}]
```

Done!! Your blockchain is created with multiple nodes running and connected. Let's create accounts and start mining to see data updated in both nodes.

Mining

To mine blocks it's necessary to have a base account. Let's do this. In the javascript terminal of the **first** node, execute the command to create a new account and define the password required to this account, the public address from the created account should be presented.

```
personal.newAccount()
miner.setEtherbase(personal.listAccounts[0])
miner.start()
# wait for 1-2 mins before stopping(when eth.blockNumber is not 0)
miner.stop()
eth.blockNumber
eth.getBalance("<address from the above newAccount command>")
```

```
> personal.newAccount()
Passphrase:
Repeat passphrase:
"0x3c664a12f8c3afb2126437d75ae09f74f4a2e01e"
```

```
> miner.setEtherbase(personal.listAccounts[0])
true
> miner.start()
null
> miner.stop()
null
>
```

```
> eth.blockNumber
45
> eth.getBalance("0x3c664a12f8c3afb2126437d75ae09f74f4a2e01e")
9400000000000000000
>
```

You can verify the **blockNumber** also in javascript terminal of the **second** node after some seconds, the value must be the same presented in the first node. This means that in fact the nodes are connected and are updated.

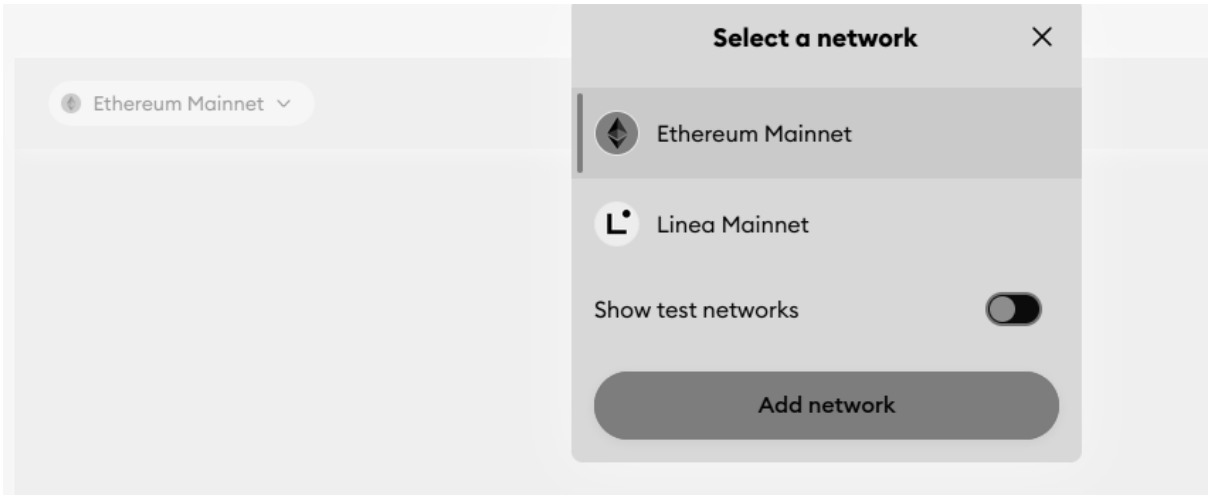
Terminal of Node 2

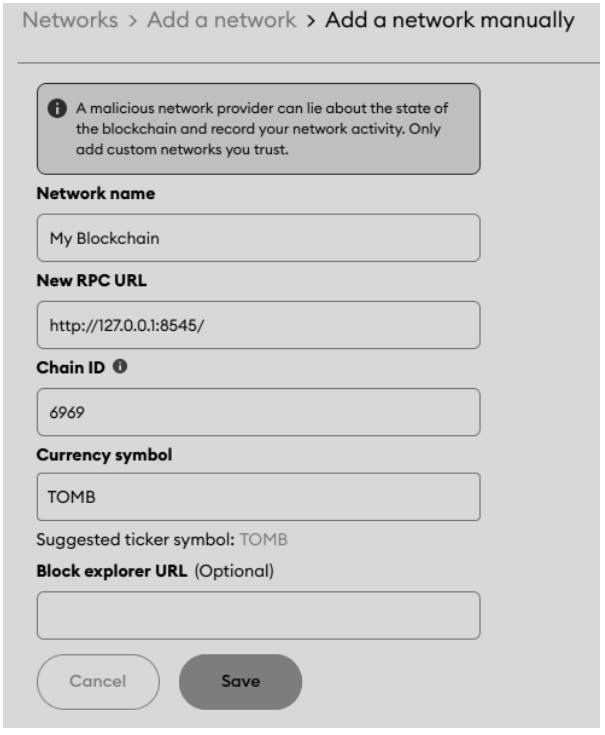
```
> eth.blockNumber
52
>
```

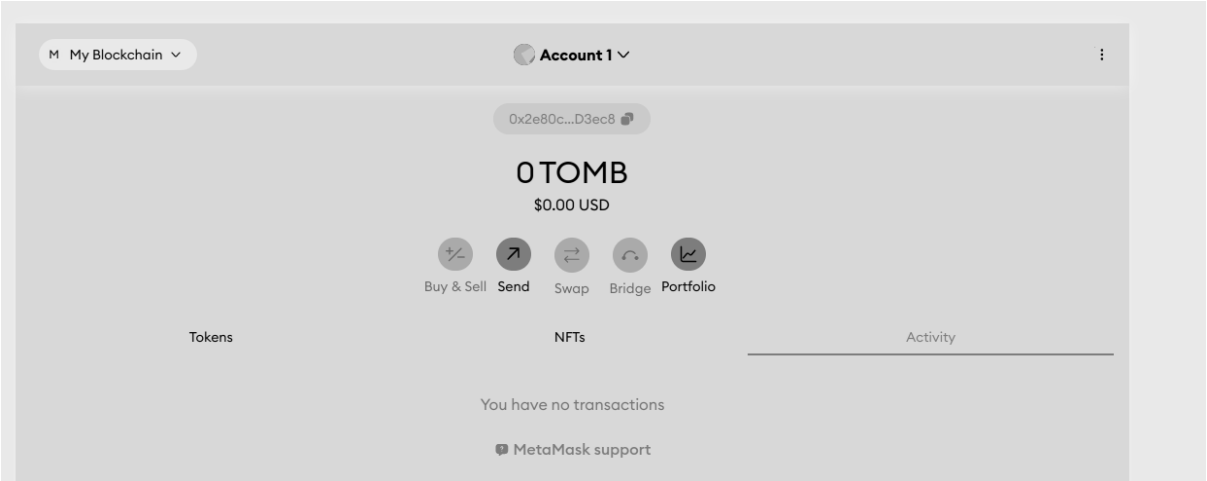
Install metamask and create a new account

Add My Blockchain Network

Let's add our Blockchain Network to MetaMask. Click on *Ethereum Mainnet* and select *Add New Network* option.





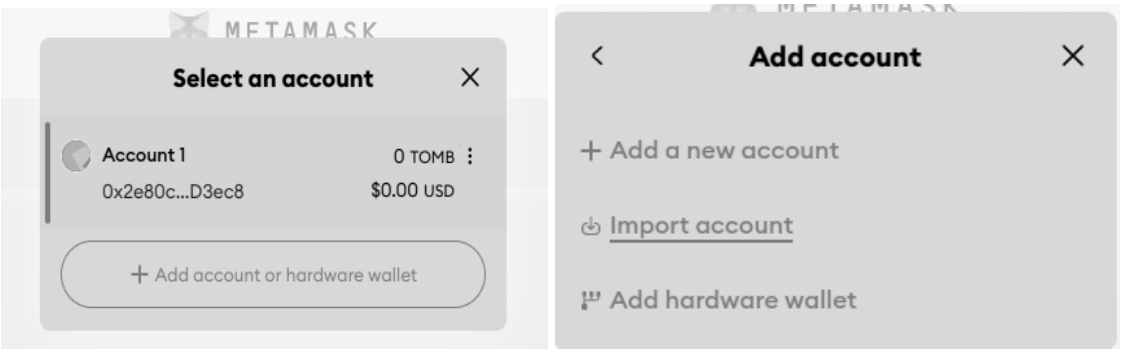


Importing accounts using json file

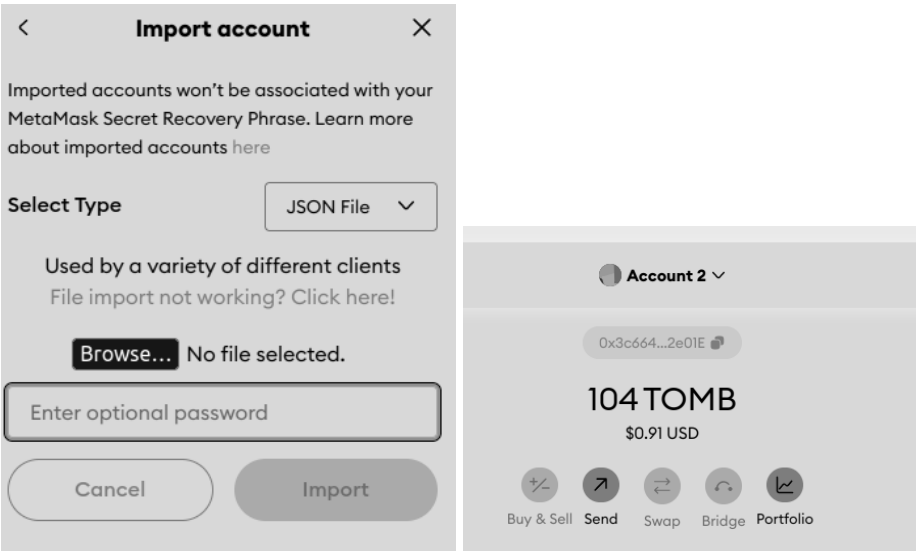
The *account json file* can be found in database of the *Node1* in your blockchain, the path to this file contains the account address, something like this:

```
node1/keystore/UTC--2022-03-17T20-40-36.929865867Z--bd3156b239e2bb8d073406e67eba59a651be18f0
```

Click on Account > Import account



Select json file (Takes upto 1-2 mins)



Transfer Funds

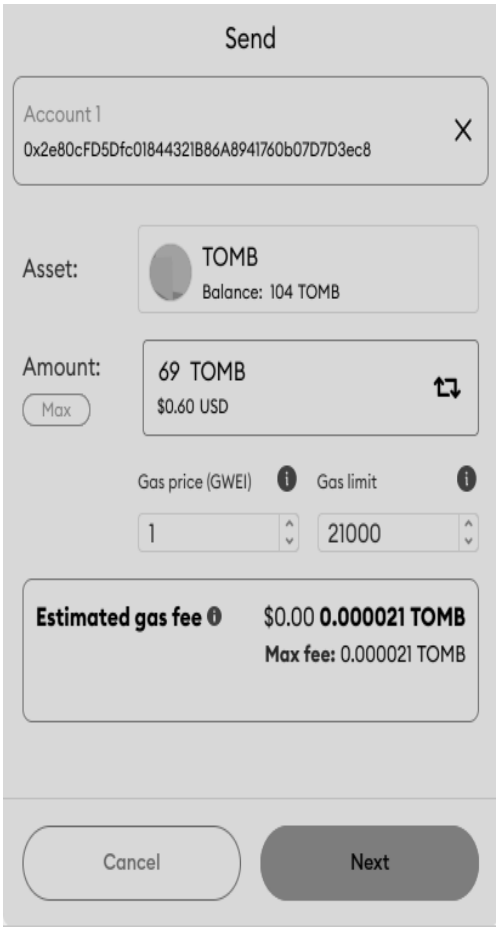
Let's transfer some value from *Account 2* to *Account 1* to validate the structure and finish this post.

First, go to *Account 1* in MetaMask and copy the *Account Public Address* and back to *Account 2*. In my case the public address of the *Account 1* is:

0x2e80cFD5Dfc01844321B86A8941760b07D7D3ec8.

In *Account 2* click on *"Send"* button, paste the public address of the *Account 1* and type the value to transfer. Click on *"Next"* button and then click on *"Confirm"* button.

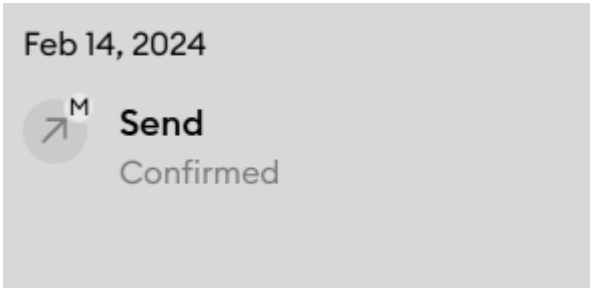
The transaction will be with status *"Pending"*, waiting for a block to be mined for validation.



Now, it's necessary to start mining to validate that transaction. Let's do this in our blockchain using Geth in Javascript Console from *Node1*. Type the command:

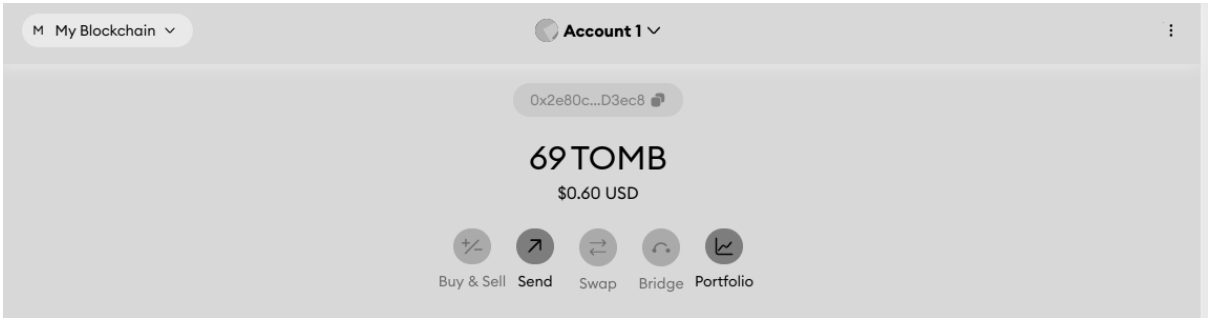
miner.start()

Wait for some seconds and verify your account in MetaMask again.



Obs: If you want, you can stop the mining in *Node1* after the transaction validation. Type command to stop the mining:

miner.stop()



The transaction was validated and the *Account 1* received the **69 TOMB**.