



**L-Università  
ta' Malta**

# **DLT5400 - DLT Implementation and Internals**

Assignment P-3 (geth)

**Balaganapathy Vanagiri Selvakumar**

Centre for Distributed Ledger Technologies

*Prof. Joshua Ellul*

May 31, 2023

## 1 Q3.2 - Private Network using geth client

### 1.1 Introduction

Geth is a popular Go implementation of the Ethereum blockchain client. It allows users to connect to and interact with the Ethereum network. In this report, we will outline the steps to download and build Geth from the source and run a private network.

### 1.2 Pre-Requisites

Before downloading Geth, it is necessary to install some prerequisites. These include *Go*, *Git*, and *make*. Instructions for installing Go, Git, and make can be found on their respective websites. Make sure that you install versions that are suitable for your machine.

### 1.3 Geth

Once the prerequisites are installed, we can download and build Geth from source. This is done by running the following commands:

```
git clone https://github.com/ethereum/go-ethereum.git
cd go-ethereum
make geth
```

The first command clones the Geth repository from GitHub, and the second command changes the directory to the cloned repository and builds the Geth binary using make.

### 1.4 Accounts

This is a small step that can be used to generate an Ethereum account, that can be used in further steps.

```
geth --datadir data account new
```

You can create multiple accounts using the above command. Follow the instructions on the terminal to generate your public key and keystore file. Save the password that was used in a text file named *password.txt*.

```

(base) bala_s@Balaganapathys-MacBook-Pro privateGeth % geth --datadir node1 account new
INFO [05-15|20:07:24.1921 Maximum peer count ETH=50 LES=0 total=50
Your new account is locked with a password. Please give a password. Do not forget this
password.
Password:
Repeat password:

Your new key was generated

Public address of the key: OxeAA6d98a3658c01BcfaABA807f38d22D573Fe4dg
Path of the secret key file: node1/keystore/UTC--2023-05-15T18-07-45.230924000Z--
eaa6d98a3658c01bcfaaba807f38d22d573fe4d9

- You can share your public address with anyone. Others need it to interact with you.
- You must NEVER share the secret key with anyone! The key controls access to your funds!
- You must BACKUP your key file! Without the key, it's impossible to access account funds!
- You must REMEMBER your password! Without the password, it's impossible to decrypt the key!

```

Figure 1: New Account Output

## 1.5 Genesis Block

The next step is to create a Genesis block. A Genesis block is the first block of a blockchain and defines the initial state of the blockchain. We can create a Genesis block by creating a JSON file with the following content:

```

{
  "config": {
    "chainId": 12345,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "muirGlacierBlock": 0,
    "berlinBlock": 0,
    "londonBlock": 0,
    "arrowGlacierBlock": 0,
    "grayGlacierBlock": 0,
    "clique": {
      "period": 5,
      "epoch": 30000
    }
  },
  "difficulty": "1",
  "gasLimit": "8000000000",

```

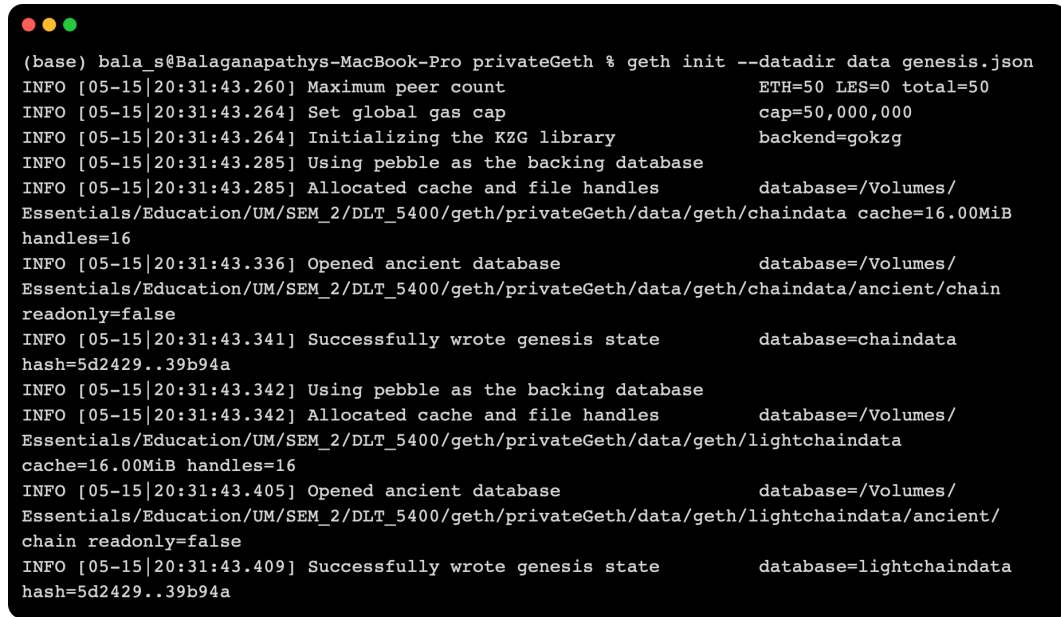
```

    "extradata": "...",
    "alloc": {
      "0b70e444E1834Fd39E948c970DB94aCe309ACC29": { "balance": "500000" },
      "4C520630E1feDec204EB9278d6a5Dfd8968126fB": { "balance": "500000" }
    }
  }
}

```

The above file, *genesis.json* states the default values and configurations of the private chain that is being created. We create a Proof of Authority(PoA) chain with a clique consensus mechanism. We also initialize the user balance with some amount of ethers pre-loaded. Now we can initialize our private blockchain network with the above configurations using the following command

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



```

(base) bala_s@Balaganapathys-MacBook-Pro privateGeth % geth init --datadir data genesis.json
INFO [05-15|20:31:43.260] Maximum peer count                      ETH=50 LES=0 total=50
INFO [05-15|20:31:43.264] Set global gas cap                      cap=50,000,000
INFO [05-15|20:31:43.264] Initializing the KZG library             backend=gokzg
INFO [05-15|20:31:43.285] Using pebble as the backing database
INFO [05-15|20:31:43.285] Allocated cache and file handles         database=/Volumes/
Essentials/Education/UM/SEM_2/DLT_5400/geth/privateGeth/data/geth/chaindata cache=16.00MiB
handles=16
INFO [05-15|20:31:43.336] Opened ancient database                 database=/Volumes/
Essentials/Education/UM/SEM_2/DLT_5400/geth/privateGeth/data/geth/chaindata/ancient/chain
readonly=false
INFO [05-15|20:31:43.341] Successfully wrote genesis state         database=chaindata
hash=5d2429..39b94a
INFO [05-15|20:31:43.342] Using pebble as the backing database
INFO [05-15|20:31:43.342] Allocated cache and file handles         database=/Volumes/
Essentials/Education/UM/SEM_2/DLT_5400/geth/privateGeth/data/geth/lightchaindata
cache=16.00MiB handles=16
INFO [05-15|20:31:43.405] Opened ancient database                 database=/Volumes/
Essentials/Education/UM/SEM_2/DLT_5400/geth/privateGeth/data/geth/lightchaindata/ancient/
chain readonly=false
INFO [05-15|20:31:43.409] Successfully wrote genesis state         database=lightchaindata
hash=5d2429..39b94a

```

Figure 2: Genesis Block Created

## 1.6 Running your nodes

### 1.6.1 Bootstrap Node

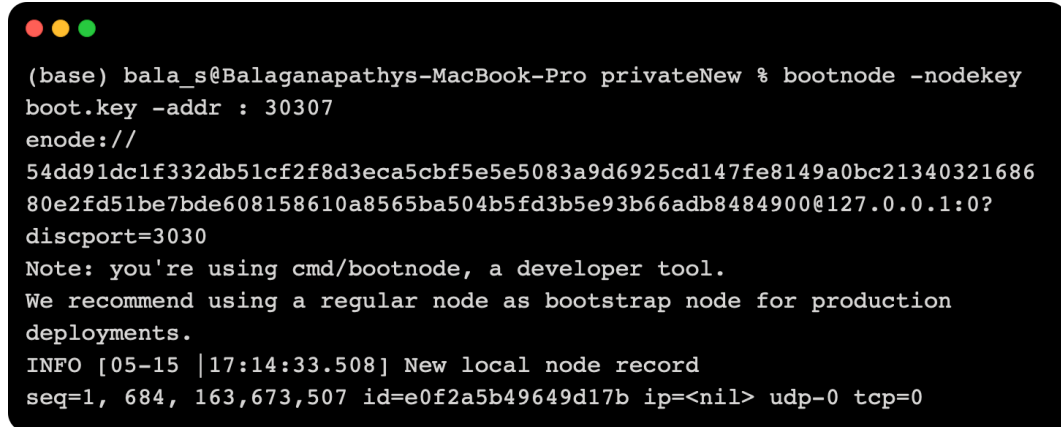
A bootstrap node can be considered as an entry point for all the other nodes to engage in peer-to-peer discovery and join the network. Any node can be opted for the bootstrap node, as we are creating a private network for learning purposes we use a utility tool named *bootnode*. This tool helps us to create a bootstrap node easily and can be used out of the box. In order to create a bootnode we require a key that can be generated by the below command.

```
bootnode -genkey boot.key
```

Now we create a bootnode using the key that was created

```
bootnode -nodekey boot.key -addr :30305
```

You can use any port number to your preference, but it is advised to use the same.



```
(base) bala_s@Balaganapathys-MacBook-Pro privateNew % bootnode -nodekey
boot.key -addr : 30307
enode://
54dd91dc1f332db51cf2f8d3eca5cbf5e5e5083a9d6925cd147fe8149a0bc21340321686
80e2fd51be7bde608158610a8565ba504b5fd3b5e93b66adb8484900@127.0.0.1:0?
discport=3030
Note: you're using cmd/bootnode, a developer tool.
We recommend using a regular node as bootstrap node for production
deployments.
INFO [05-15 |17:14:33.508] New local node record
seq=1, 684, 163,673,507 id=e0f2a5b49649d17b ip=<nil> udp=0 tcp=0
```

Figure 3: Bootnode Creation Output

A bootstrap node can be a private node or a cloud instance in a production network. Now we can proceed with running our node.

### 1.6.2 Member Node

In a separate terminal start the node by using the below command.

```
./geth --datadir node1 --port 30308 --bootnodes "... " --networkid 12345
--unlock 0x4C520630E1feDec204EB9278d6a5Dfd8968126fB --password
node1/password.txt --authrpc.port 8552
```

Replace `".."` in the `--bootnodes` parameter with the enode output that was generated by the bootnode tool as shown in Figure 3. We also pass our public key and the path to our `password.txt` file for the respective account that was created in the subsection Accounts. You can create several nodes by changing the values of the field `--datadir`, `--port`, `unlock`, `password`, `authrpc.port` respectively. The node must be initialized the same way we initialized the above node `node1` using the genesis file.

Figure 4: Running node1

```
INFO [05-15|17:21:53.997] Rebuilding state snapshot
INFO [05-15|17:21:53.997] Resuming state snapshot generation
INFO [05-15|17:21:53.998] Generated state snapshot
INFO [05-15|17:21:53.998] Regenerated local transaction journal
INFO [05-15|17:21:53.998] Gasprice oracle is ignoring threshold set
WARN [05-15|17:21:53.998] Error reading unclean shutdown markers
INFO [05-15|17:21:53.998] Engine API enabled
WARN [05-15|17:21:53.998] Engine API started but chain not configured for merge yet
INFO [05-15|17:21:53.998] Stored checkpoint snapshot to disk
INFO [05-15|17:21:53.998] Starting peer-to-peer node
INFO [05-15|17:21:54.019] New local node record
INFO [05-15|17:21:54.020] Starting P2P networking
496a9afde14a1ab281f73c05f149a1f681bf8e0f365cb9b@127.0.0.1:30308
INFO [05-15|17:21:54.020] IPC endpoint opened
INFO [05-15|17:21:54.020] Generated JWT secret
INFO [05-15|17:21:54.024] WebSocket enabled
INFO [05-15|17:21:54.024] HTTP server started
INFO [05-15|17:21:54.503] Unlocked account
INFO [05-15|17:22:04.065] Looking for peers
INFO [05-15|17:22:14.107] Looking for peers
INFO [05-15|17:22:24.150] Looking for peers
INFO [05-15|17:22:34.191] Looking for peers
INFO [05-15|17:22:44.224] Looking for peers
INFO [05-15|17:22:54.270] Looking for peers
INFO [05-15|17:23:04.320] Looking for peers
INFO [05-15|17:23:14.356] Looking for peers
INFO [05-15|17:23:24.391] Looking for peers
INFO [05-15|17:23:34.432] Looking for peers
INFO [05-15|17:23:44.461] Looking for peers
INFO [05-15|17:23:54.492] Looking for peers
INFO [05-15|17:24:04.528] Looking for peers
INFO [05-15|17:24:14.565] Looking for peers
INFO [05-15|17:24:24.610] Looking for peers
INFO [05-15|17:24:34.661] Looking for peers
INFO [05-15|17:24:44.698] Looking for peers
INFO [05-15|17:24:54.734] Looking for peers
INFO [05-15|17:25:04.771] Looking for peers
INFO [05-15|17:25:14.806] Looking for peers
INFO [05-15|17:25:24.837] Looking for peers
INFO [05-15|17:25:34.885] Looking for peers
INFO [05-15|17:25:44.940] Looking for peers
INFO [05-15|17:25:54.985] Looking for peers
INFO [05-15|17:26:05.029] Looking for peers
INFO [05-15|17:26:15.064] Looking for peers
```

Figure 5: Node connected with a peer

We can now attach a javascript console to either one of the nodes to query network properties and perform actions.

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

By executing the above command, we now attach a console to our node *node1*

```
(base) bala_s@balaganapathys-MacBook-Pro: privatenew % geth attach node1/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.11.7-unstable-dfdd804c-20230511/darwin-arm64/go1.20.4
at block: 0 (Thu Jan 01 1970 01:00:00 GMT+0100 (CET))
datadir: /Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privateNew/node1
modules: admin:1.0 cli:1.0 engine:1.0 eth: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
> net.peerCount
1
> admin.peers
[{"caps": ["eth/66", "eth/67", "eth/68", "snap/1"],
  enode: "enode://681e7a53b614820d1c5f3929de8109db1fa30712842909baa4bf8b462426fe04bf428ee25f1ba0e96496aa6def4a1ab2817e305ff49a1f681b8fe0365b9b9@127.0.0.1:30308",
  id: "ae6424d779a0caf5c783d6a3f99dd9f718da87df3e1c56f25bf38689ba71133",
  name: "Geth/v1.11.7-unstable-dfdd804c-20230511/darwin-arm64/go1.20.4",
  network: {
    inbound: false,
    localAddress: "127.0.0.1:54570",
    remoteAddress: "127.0.0.1:30308",
    static: false,
    trusted: false
  },
  protocols: {
    eth: {
      version: 68
    },
    snap: {
      version: 1
    }
  }
}]
> eth.getBalance(eth.accounts[0])
500000
> []
```

Figure 6: JavaScript console attached to a node

In the above Figure 6, you can see the JS console that can be used to do basic network queries, like fetching the number of peers attached, getting the account balance for a particular address, and fetching peer details.

## 2 Q3.3 - Demo of geth networks with PoA, PoS & PoW consensus

### 2.1 PoW Demo

#### 2.1.1 Pre-requisites

As Ethereum has moved to proof of stake consensus, it is not possible for us to run a PoW ethereum network. In order to do that we will be using an older version of geth. We create an older version of geth binary by performing the same tasks listed in sub-section 1.3, but with a different version of the geth repository.

```
git clone https://github.com/ethereum/go-ethereum.git
```

The above command clones the github repository of geth version 1.9, which will help us build geth binary which will support ethash or PoW(Mining).

#### 2.1.2 Genesis Block

We need to configure our blockchain using our *genesis.json* file as mentioned in sub-section 1.5. Here we will be using a different genesis file in order to run a network with PoW consensus.

```
{
  "alloc": {},
  "config": {
    "chainId": 2019,
    "homesteadBlock": 0,
    "DAOForkBlock": 0,
    "DAOForkSupport": true,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0
  },
  "difficulty": "0x400",
  "gasLimit": "0x989680",
  "nonce": "0x0000000000000042",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x0000000000000000000000000000000000000000",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "extraData": "",
  "timestamp": "0x00"
}
```



### 2.1.3 Start PoW Network

We can start our network by performing the below command.

```
./geth --datadir data --networkid 2019 console
```

Make sure that the genesis block has been created and the directory files and chain id is correct.

```
INFO [05-24|17:31:54.123] Maximum peer count          ETH=50 LES=0 total=50
INFO [05-24|17:31:54.137] Set global gas cap          cap=25000000
INFO [05-24|17:31:54.137] Allocated trie memory caches  clean=256.00MiB dirty=256.00MiB
INFO [05-24|17:31:54.138] Allocated cache and file handles  database=/Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privatePOW/data/geth/chaindata cache=512.00MiB handles=5120
INFO [05-24|17:31:54.179] Opened ancient database      database=/Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privatePOW/data/geth/chaindata/ancient
INFO [05-24|17:31:54.180] Initialised chain configuration  config={ChainID: 2019 Homestead: 0 DAO: 0 DAOsupport: true EIP150: 0 EIP155: 0 EIP158: 0 Byzantium: <nil> Const
antinoople: <nil> Petersburg: <nil> Istanbul: <nil> Muir Glacier: <nil> Volo V2: <nil> Engine: unknown}
INFO [05-24|17:31:54.180] Disk storage enabled for ethash caches  dir=/Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privatePOW/data/geth/ethash count=3
INFO [05-24|17:31:54.180] Disk storage enabled for ethash DAGs  dir=/Users/bala_s/Library/Ethash count=2
INFO [05-24|17:31:54.181] Initialising Ethereum protocol  version=[65 64 63] network=2019 divergence=8
INFO [05-24|17:31:54.183] Loaded most recent local header  number=512 hash="65c74b...1055bc" td=76115912 age=1d20h31m
INFO [05-24|17:31:54.183] Loaded most recent local full block  number=512 hash="65c74b...1055bc" td=76115912 age=1d20h31m
INFO [05-24|17:31:54.183] Loaded most recent local fast block  number=512 hash="65c74b...1055bc" td=76115912 age=1d20h31m
INFO [05-24|17:31:54.185] Loaded local transaction journal  transactions=0 dropped=0
INFO [05-24|17:31:54.185] Regenerated local transaction journal transactions=0 accounts=0
WARN [05-24|17:31:54.185] Switch sync mode from fast sync to full sync
INFO [05-24|17:31:54.186] Starting peer-to-peer node     instance=Geth/v1.9.25-stable-e7872729/darwin-arm64/go1.20.4
INFO [05-24|17:31:54.219] New local node record          ser=2 id=86159a6861471f0b ip=127.0.0.1 udp=38303 tcp=38303
INFO [05-24|17:31:54.219] Started P2P networking         self=enode://edec3af09cabed58ab643b19319fa3696efd8e8f5c8ee7290912c2ed3b162aca0465d8ed8f73a5ce53305152f48a72b07d
04e377a2885ee817247389dc0f913@127.0.0.1:30303
INFO [05-24|17:31:54.221] IPC endpoint opened            url=/Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privatePOW/data/geth.ipc
WARN [05-24|17:31:54.246] Served eth_coinbase            reqid=3 tx="15.25µs" err="etherbase must be explicitly specified"
Welcome to the Geth JavaScript console!

Instance: Geth/v1.9.25-stable-e7872729/darwin-arm64/go1.20.4
at block: 512 (Mon May 22 2023 21:00:15 GMT+0200 (CEST))
datadir: /Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privatePOW/data
modules: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

To exit, press ctrl-d
> INFO [05-24|17:32:04.226] Looking for peers              peercount=0 tried=112 static=0
INFO [05-24|17:32:14.227] Looking for peers              peercount=0 tried=157 static=0
```

Figure 7: PoW Based private network started

Here we can see that the blockchain has started and we have peer discovery and a javascript console for us to interact with the blockchain. In order for us to start mining we need to set an account as etherbase to receive the miner rewards. You can use the Accounts that was created in sub-section 1.4 or generate new accounts using below commands.

```
personal.newAccount('yourpassword')
miner.setEtherbase("<AddressGenerated>")
```

The above commands will create a new account and set the account as the etherbase.

```
> personal.newAccount('yourpassword')
INFO [05-24|18:03:22.358] Your new key was generated    address=0x48e46fd703cc5df279864e5e9c348691ecb9bc
WARN [05-24|18:03:22.358] Please backup your key file!  path=/Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/privatePOW/data/keystore/UTC--2023-05-24T16-03-21.4138
060002--48e46fd703cc5df279864e5e9c348691ecb9bc
WARN [05-24|18:03:22.358] Please remember your password!
0x48e46fd703cc5df279864e5e9c348691ecb9bc
> miner.setEtherbase("0x48e46fd703cc5df279864e5e9c348691ecb9bc")
true
```

Figure 8: Etherbase account

In the above Figure 8, we can see that the etherbase has been set. Now we can start our miner and run our network's consensus.

```
miner.start()
```

```

> miner.start()
INFO [05-24|18:12:32.842] Updated mining threads          threads=10
INFO [05-24|18:12:32.842] Transaction pool price threshold updated price=10000000000
null
> INFO [05-24|18:12:32.845] Commit new mining work
INFO [05-24|18:12:33.551] Successfully sealed new block
INFO [05-24|18:12:33.551] ^ mined potential block
INFO [05-24|18:12:33.551] Commit new mining work
INFO [05-24|18:12:33.645] Successfully sealed new block
INFO [05-24|18:12:33.645] ^ mined potential block
INFO [05-24|18:12:33.646] Commit new mining work
INFO [05-24|18:12:33.820] Successfully sealed new block
INFO [05-24|18:12:33.820] ^ mined potential block
INFO [05-24|18:12:33.820] Mining too far in the future
> INFO [05-24|18:12:35.821] Commit new mining work
INFO [05-24|18:12:35.856] Successfully sealed new block
INFO [05-24|18:12:35.856] ^ mined potential block
INFO [05-24|18:12:35.857] Commit new mining work
INFO [05-24|18:12:36.073] Successfully sealed new block
INFO [05-24|18:12:36.073] ^ mined potential block
INFO [05-24|18:12:36.073] Commit new mining work
> INFO [05-24|18:12:36.490] Successfully sealed new block
INFO [05-24|18:12:36.490] ^ mined potential block
INFO [05-24|18:12:36.490] Mining too far in the future
INFO [05-24|18:12:38.492] Commit new mining work
INFO [05-24|18:12:38.986] Successfully sealed new block
INFO [05-24|18:12:38.986] ^ mined potential block
INFO [05-24|18:12:38.987] Commit new mining work
INFO [05-24|18:12:39.120] Successfully sealed new block
INFO [05-24|18:12:39.120] ^ block reached canonical chain
INFO [05-24|18:12:39.120] ^ mined potential block
INFO [05-24|18:12:39.120] Commit new mining work
INFO [05-24|18:12:39.353] Successfully sealed new block
INFO [05-24|18:12:39.353] ^ block reached canonical chain
INFO [05-24|18:12:39.353] ^ mined potential block
INFO [05-24|18:12:39.353] Mining too far in the future
INFO [05-24|18:12:41.355] Commit new mining work
INFO [05-24|18:12:41.604] Successfully sealed new block
INFO [05-24|18:12:41.604] ^ block reached canonical chain
INFO [05-24|18:12:41.604] ^ mined potential block
INFO [05-24|18:12:41.604] Commit new mining work
INFO [05-24|18:12:41.605] Successfully sealed new block
INFO [05-24|18:12:41.605] Mining too far in the future
INFO [05-24|18:12:41.605] ^ block reached canonical chain
INFO [05-24|18:12:43.607] Commit new mining work
INFO [05-24|18:12:43.653] Successfully sealed new block
INFO [05-24|18:12:43.653] ^ block reached canonical chain
INFO [05-24|18:12:43.653] ^ mined potential block
INFO [05-24|18:12:43.653] Commit new mining work
INFO [05-24|18:12:44.006] Successfully sealed new block
INFO [05-24|18:12:44.006] ^ block reached canonical chain
INFO [05-24|18:12:44.006] ^ mined potential block
INFO [05-24|18:12:44.006] Commit new mining work
number=513 sealhash="9e55c0_8332b5" uncles=0 txs=0 gas=0 fees=0 elapsed=2.27ms
number=513 sealhash="9e55c0_8332b5" hash="eecd9a_ca356a" elapsed=708.525ms
number=513 hash="eecd9a_ca356a"
number=514 sealhash="6f6683_369bf5" uncles=0 txs=0 gas=0 fees=0 elapsed=78.583µs
number=514 sealhash="6f6683_369bf5" hash="078d53_276b42" elapsed=94.329ms
number=514 hash="078d53_276b42"
number=515 sealhash="9c6f8b_a483d3" uncles=0 txs=0 gas=0 fees=0 elapsed="73.709µs"
number=515 sealhash="9c6f8b_a483d3" hash="8b8665_de4e07" elapsed=174.432ms
number=515 hash="8b8665_de4e07"
wait=2s
number=516 sealhash="d2f25c_d1618b" uncles=0 txs=0 gas=0 fees=0 elapsed=2.001s
number=516 sealhash="d2f25c_d1618b" hash="eec104_b7e285" elapsed=35.119ms
number=516 hash="eec104_b7e285"
number=517 sealhash="1a582f_72b728" uncles=0 txs=0 gas=0 fees=0 elapsed="71.25µs"
number=517 sealhash="1a582f_72b728" hash="8ffe47_95f986" elapsed=216.046ms
number=517 hash="8ffe47_95f986"
number=518 sealhash="848db3_5a709d" uncles=0 txs=0 gas=0 fees=0 elapsed="106.5µs"
number=518 sealhash="848db3_5a709d" hash="788b58_b2a852" elapsed=417.184ms
number=518 hash="788b58_b2a852"
wait=2s
number=519 sealhash="2e4aeb_67adba" uncles=0 txs=0 gas=0 fees=0 elapsed=2.001s
number=519 sealhash="2e4aeb_67adba" hash="78c4a0_d0c714" elapsed=494.672ms
number=519 hash="78c4a0_d0c714"
number=520 sealhash="dda6b_4bb944" uncles=0 txs=0 gas=0 fees=0 elapsed="89.708µs"
number=520 sealhash="dda6b_4bb944" hash="ed95b3_df0126" elapsed=133.098ms
number=520 hash="ed95b3_df0126"
number=521 sealhash="2fbcab_332a03" uncles=0 txs=0 gas=0 fees=0 elapsed="61.708µs"
number=521 sealhash="2fbcab_332a03" hash="0af7ef_b24a2e" elapsed=233.022ms
number=521 hash="0af7ef_b24a2e"
number=522 sealhash="3c0ffe_7c81e2" uncles=0 txs=0 gas=0 fees=0 elapsed=2.001s
number=522 sealhash="3c0ffe_7c81e2" hash="40f97b_c5b67c" elapsed=249.045ms
number=522 hash="40f97b_c5b67c"
number=523 sealhash="b0203e_e6c1c7" uncles=0 txs=0 gas=0 fees=0 elapsed="87.75µs"
number=523 sealhash="b0203e_e6c1c7" hash="2fcd27_e21922" elapsed=1.423ms
number=523 hash="2fcd27_e21922"
wait=2s
number=524 sealhash="a1f15e_ef24fd" uncles=0 txs=0 gas=0 fees=0 elapsed=2.001s
number=524 sealhash="a1f15e_ef24fd" hash="38bf35_645e21" elapsed=45.414ms
number=524 hash="38bf35_645e21"
number=525 sealhash="a1508e_d80924" uncles=0 txs=0 gas=0 fees=0 elapsed="59.541µs"
number=525 sealhash="a1508e_d80924" hash="17c1d2_8ef02d" elapsed=353.532ms
number=525 hash="17c1d2_8ef02d"
number=526 sealhash="8bf8d4_726979" uncles=0 txs=0 gas=0 fees=0 elapsed="72.5µs"
number=526 hash="8bf8d4_726979"

```

Figure 9: PoW Miner running

Here, in Figure 9, we can see that the mining process is active and new blocks are being mined and added to our private blockchain. The following commands can be used to stop the miner and kill the blockchain instance.

```
miner.stop()
exit
```

## 2.2 PoS Demo

### 2.2.1 Pre-requisites

Running a Proof of Stake private network does not make any sense as we need our own beacon chain, validator setup and several nodes to reach consensus. So for the demonstration purposes, we run a node in the ethereum mainnet using *prysm* as consensus client and *geth* as our execution client.

### 2.2.2 Client Setup

We can use the geth binaries that was built on sub-section 1.3. Now to set up a consensus client we need to setup *prysm* and run the same. The below steps can be followed to install and setup prysm.

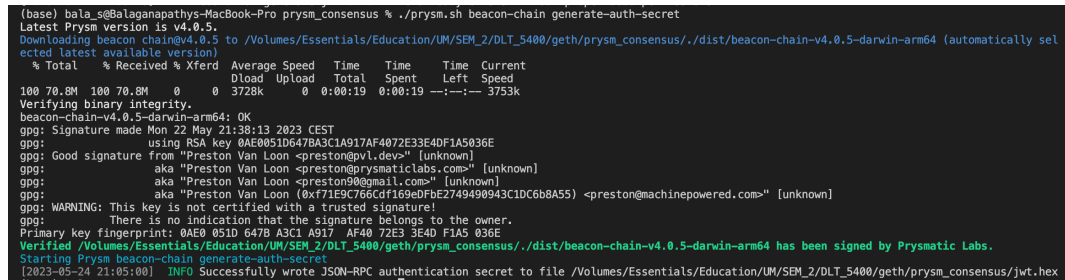
```
mkdir prysm && cd prysm
```

```
curl https://raw.githubusercontent.com/prysmaticlabs/prysm/master/prysm.sh --
output prysm.sh && chmod +x prysm.sh
```

After installing the prysm binaries, we need to generate a JWT secret. Now, you generate this using an online tool like [this](#), or generate through geth or prysm itself.

```
./prysm.sh beacon-chain generate-auth-secret
```

This will generate and store your *jwt.hex* file in a directory and output a path on the console.



```
(base) bala_s@Balaganapathy-MacBook-Pro prysm_consensus % ./prysm.sh beacon-chain generate-auth-secret
Latest Prysm version is v4.0.5.
Downloading beacon chain@v4.0.5 to /Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/prysm_consensus/.dist/beacon-chain-v4.0.5-darwin-arm64 (automatically sel
ected latest available version)
% Total    % Received % Xferd    Average Speed   Time    Time     Time  Current
             Dload  Upload    Total   Spent    Left   Speed
100 70.8M  100 70.8M    0     0  3728k      0  0:00:19  0:00:19 --:--:-- 3753k
Verifying binary integrity.
beacon-chain-v4.0.5-darwin-arm64: OK
gpg: Signature made Mon 22 May 21:38:13 2023 CEST
gpg: using RSA key 0AE0851D647BA3C1A917AF4072E33E4DF1A5836E
gpg: Good signature from "Preston Van Loon <preston@pvl.dev>" [unknown]
gpg: aka "Preston Van Loon <preston@prysmaticlabs.com>" [unknown]
gpg: aka "Preston Van Loon <preston90@gmail.com>" [unknown]
gpg: aka "Preston Van Loon (8x71E9C765cdf169e0PbE2749490943C1DC6b8A55) <preston@machinepowered.com>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Primary key fingerprint: 0AE0 051D 647B A3C1 A917 AF40 72E3 3E4D F1A5 036E
Verified /Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/prysm_consensus/.dist/beacon-chain-v4.0.5-darwin-arm64 has been signed by Prysmatic Labs.
Starting Prysm beacon-chain generate-auth-secret
[2023-05-24 21:08:00] INFO Successfully wrote JSON-RPC authentication secret to file /Volumes/Essentials/Education/UM/SEM_2/DLT_5400/geth/prysm_consensus/jwt.hex
```

Figure 10: JWT secret generation

Now, we can proceed with running our execution client with certain flags that can help the execution client connect with the consensus client.

```
geth --http --http.api eth,net,engine,admin --authrpc.jwtsecret /path/to/jwt.hex
```

Replace `/path/to/jwt.hex` with the path generated on the previous step as shown in Figure 10. Now let's proceed with our consensus client, and run our beacon chain.

```
./prysm.sh beacon-chain --execution-endpoint=http://localhost:8551 --jwt-
secret=path/to/jwt.hex
```

In the below Figures 11 & 12, we can see that the mainnet has started with PoS consensus, and our beacon node is connected with geth. We have an execution client and a consensus client running together. We have successfully managed to run an ethereum node with PoS consensus mechanism

```
(base) bala_s@Balaganapathys-MacBook-Pro geth & geth --http --http.api eth,net,engine,admin --authrpc.jwtsecret prysm_consensus/jwt.hex
INFO [05-24 | 21:12:41.596] Starting Geth on Ethereum mainnet...
INFO [05-24 | 21:12:41.596] Bumping default cache on mainnet
INFO [05-24 | 21:12:41.599] Maximum peer count
INFO [05-24 | 21:12:41.604] Set global gas cap
INFO [05-24 | 21:12:41.604] Initializing the KZG library
INFO [05-24 | 21:12:41.629] Allocated trie memory caches
INFO [05-24 | 21:12:41.630] Using pebble as the backing database
INFO [05-24 | 21:12:41.630] Allocated cache and file handles
INFO [05-24 | 21:12:41.684] Opened ancient database
INFO [05-24 | 21:12:41.687] Initialising Ethereum protocol
INFO [05-24 | 21:12:41.688]
INFO [05-24 | 21:12:41.688]
provided=1024 updated=4096
Eth=50 LE=0 total=50
cap=50,000,000
backends=gokzg
clean=614.00MiB dirty=1024.00MiB
database=/Users/bala_s/Library/Ethereum/geth/chaindata cache=2.00GiB handles=5120
database=/Users/bala_s/Library/Ethereum/geth/chaindata/ancient/chain readOnly=false
network=1 dbversion=8

INFO [05-24 | 21:12:41.689] Chain ID: 1 (mainnet)
INFO [05-24 | 21:12:41.689] Consensus: Beacon (proof-of-stake), merged from Ethash (proof-of-work)
INFO [05-24 | 21:12:41.689]
INFO [05-24 | 21:12:41.689] Pre-Merge hard forks (block based):
INFO [05-24 | 21:12:41.689] - Homestead: #1150000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/homestead.md)
INFO [05-24 | 21:12:41.689] - DAO Fork: #1920000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/dao-fork.md)
INFO [05-24 | 21:12:41.689] - Tangerine Whistle (EIP 150): #2463000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/tangerine-whistle.md)
INFO [05-24 | 21:12:41.689] - Spurious Dragon/1 (EIP 155): #2675000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [05-24 | 21:12:41.689] - Spurious Dragon/2 (EIP 158): #2675000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/spurious-dragon.md)
INFO [05-24 | 21:12:41.689] - Byzantium: #4370000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/byzantium.md)
INFO [05-24 | 21:12:41.689] - Constantinople: #7280000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/constantinople.md)
INFO [05-24 | 21:12:41.689] - Petersburg: #7280000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/petersburg.md)
INFO [05-24 | 21:12:41.689] - Istanbul: #9069000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/istanbul.md)
INFO [05-24 | 21:12:41.689] - Muir Glacier: #9200000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/muir-glacier.md)
INFO [05-24 | 21:12:41.689] - Berlin: #12244000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/berlin.md)
INFO [05-24 | 21:12:41.689] - London: #12965000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/london.md)
INFO [05-24 | 21:12:41.689] - Arrow Glacier: #13773000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/arrow-glacier.md)
INFO [05-24 | 21:12:41.689] - Gray Glacier: #15050000 (https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/gray-glacier.md)
INFO [05-24 | 21:12:41.689] Merge configured:
INFO [05-24 | 21:12:41.689] - Hard-fork specification: https://github.com/ethereum/execution-specs/blob/master/network-upgrades/mainnet-upgrades/paris.md
INFO [05-24 | 21:12:41.689] - Network known to be merged: true
```

Figure 11: PoS Mainnet - Execution client(geth)

```
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x9797dcfb54e7 slot=6431730
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x0f14720b7a46 slot=6431731
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x6bcbceadferd5 slot=6431732
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x015231656af slot=6431733
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0xa70844e11f83 slot=6431734
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x1de5b9293a15 slot=6431735
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x2945694fbf30 slot=6431736
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x62cc187b9b4c slot=6431737
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0xa7060ae9e375 slot=6431738
[2023-05-24 | 21:15:39] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x26232161e53 slot=6431739
[2023-05-24 | 21:15:40] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x2b0b9d192ad1 slot=6431740
[2023-05-24 | 21:15:40] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x7cbe861b8fc1 slot=6431741
[2023-05-24 | 21:15:40] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0xa42dcdb13d85 slot=6431742
[2023-05-24 | 21:15:40] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0xd78d69acc16c slot=6431743
[2023-05-24 | 21:15:40] INFO blockchain: Called fork choice updated with optimistic block finalizedPayloadBlockHash=0xd467760c7132 headPayloadBlockHash=0xd78d69acc16c headSlot=6431743
[2023-05-24 | 21:15:40] INFO initial-sync: Processing block batch of size 61 starting from 0x6f3d36d5... 6431744/6510976 - estimated time remaining 1h24m6s blocks
PerSecond=15.7 peers=12
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x942fca343c4d slot=6431744
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x8c537f2750f2 slot=6431745
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0xb17c06c4612e slot=6431746
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x3127a41ccf8 slot=6431747
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x181255e6c79d slot=6431748
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x641d50cd4f87 slot=6431749
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x9145da4476a8 slot=6431750
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0xfa5d6b23e123 slot=6431751
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x485cd5d6a0bf slot=6431752
[2023-05-24 | 21:15:45] INFO blockchain: Called new payload with optimistic block payloadBlockHash=0x8a277bedab48 slot=6431753
```

Figure 12: PoS Mainnet - Consensus client(prysm)

## 2.3 PoA Demo

PLEASE REFER TO SECTION 1.

As a PoA(Clique) based private network was demonstrated in section 1, we will not repeat the same here. All processes are same as the mentioned section.

We have successfully managed to run an ethereum node in Proof of Work(PoW), Proof of Stake(PoS) and Proof of Authority(PoA) consensus mechanisms. This gives us an idea and clarity on different consensus mechanisms, their setup and implementation.

### 3 Q3.4 - Blacklist Addresses

#### 3.1 Introduction

We are going to use the PoW geth version(v1.9.25) that was used in the Section 2.1. We would need to make some changes in the implementation of geth in order to implement the blacklist to avoid transactions from.

#### 3.2 Changes in Geth

The file named *transaction\_signing.go* in the path */go-ethereum/core/types/* needs to be modified and added with our blacklist implementation.

##### 3.2.1 New Error

We create a new *Error* object after the import block in the file. This error object can be used to flag the error that the sender is blacklisted and invalid.

```

29
30  var (
31      ErrInvalidChainId    = errors.New("invalid chain id for signer")
32      ErrBlacklistedAddress = errors.New("Address Blacklisted, Malicious Sender")
33  )

```

Figure 13: Error for Blacklisted addresses

#### 3.3 Blacklist Functionality

Now we create a new array named *blacklist* and add a new function *addressBlacklist* that will return the boolean value *True* if the address is blacklisted and *False* if absent.

```

66  var blacklist = []common.Address{
67      common.HexToAddress("f5a2f640992744ca1312bfb13e7c566ab663c1e2"),
68      common.HexToAddress("592bea02739197d861571174a133fe2fd78d440c"),
69  }
70
71  func addressBlacklist(address common.Address) bool {
72      for _, blacklistedAddress := range blacklist {
73          if blacklistedAddress == address {
74              return true
75          }
76      }
77      return false
78  }

```

Figure 14: Blacklist Function

### 3.3.1 Implementation

In order for our node to avoid transactions from the addresses present in *blacklist* array, we need to make sure that we check the sender address before we accept and execute a transaction. We modify the code in function *func Sender(signer Signer, tx \*Transaction) (common.Address, error)* on the same file. We add an *if* statement and we use the *addressBlacklist* function to check if our sender address is present in the blacklist. We return the address and error object that was created in the above sub-section 3.2.1. If not blacklisted we proceed with the transaction.

```

87 func Sender(signer Signer, tx *Transaction) (common.Address, error) {
88     if sc := tx.from.Load(); sc != nil {
89         sigCache := sc.(sigCache)
90         // If the signer used to derive from in a previous
91         // call is not the same as used current, invalidate
92         // the cache.
93         if sigCache.signer.Equal(signer) {
94             if addressBlacklist(sigCache.from) {
95                 return sigCache.from, ErrBlacklistedAddress
96             }
97             return sigCache.from, nil
98         }
99     }
100
101     addr, err := signer.Sender(tx)
102     if err != nil {
103         return common.Address{}, err
104     }
105     tx.from.Store(sigCache{signer: signer, from: addr})
106     return addr, nil
107 }

```

Figure 15: Checking if blacklisted

### 3.3.2 Build

Now we can build our modified version of geth and start our modified geth node by using the steps used in 2.1. Start the geth node and attach a JS console to the geth node in a new terminal.

## 3.4 Working

We will now try transacting with valid and invalid sender and see the results of our implementation.

### 3.4.1 Valid Sender

```
> personal.unlockAccount("65a16a338ce89f519e9afdfbfd93744758f14dec")
Unlock account 65a16a338ce89f519e9afdfbfd93744758f14dec
Passphrase:
true
> web3.eth.sendTransaction({
  from: "65a16a338ce89f519e9afdfbfd93744758f14dec",
  to: "c138c3cf7a50ebf1e39287dcfdc8c92db3c97071",
  value: web3.toWei(1, "ether")
})
"0xcc489ce34141cdd01cd80f5e44b044d24dd10a0c6eee1392440da01d681f777f"
```

Figure 16: Transaction from valid sender

### 3.4.2 Invalid Sender

```
> web3.eth.sendTransaction({
  from: "f5a2f640992744ca1312bfb13e7c566ab663c1e2",
  to: "0x3d9c5cf9f17b4a6fd911e950921d4133c0c54ddd",
  value: web3.toWei(1, "ether")
})
Error: invalid sender
    at web3.js:6347:37(47)
    at web3.js:5081:62(37)
    at <eval>:1:25(16)
```

Figure 17: Transaction from invalid sender



## 4 Q3.5 - Proof of Stake implementation

### 4.1 Introduction

Historically, geth was sufficient to run an entire ethereum node including the consensus mechanism i.e. Proof of Work and the execution engine together. But after the merge, you need geth, the execution client, and a consensus client to run the proof of stake consensus algorithm.

#### 4.1.1 Execution client

An execution client is a node that runs the software that is tasked with processing and broadcasting transactions and managing Ethereum's state. They run the computations for each transaction using the Ethereum Virtual Machine to ensure that the rules of the protocol are followed. These nodes running the execution clients together form the execution layer. There are four execution clients that are being used currently such as *geth*, *Besu*, *Nethermind*, and *Erigon*.

#### 4.1.2 Consensus client

These are nodes with the consensus client such as, *Lighthouse*, *Nimbus*, *Prysm*, *Teku*, and *Lodestar*. These clients can be used to facilitate the Proof of Stake consensus for your network and must be connected with your execution client.

### 4.2 Connecting geth with prysm

#### 4.2.1 Theory

In this section, we are going to discuss the theory behind implementing PoS with geth. As we have seen before, geth is an execution client. Geth handles state management, and other EVM and protocol tasks as mentioned in sub-section 4.1.1. In order to achieve consensus our execution client needs to be connected with a consensus client. Prysm is ethereum PoS consensus client written in Go. You can run a beacon node and can also stake 32ETH to run a validator node and gain rewards.

In order to connect your execution client(geth) and your consensus client(prysm), we need to generate a *JWT Secret* (Refer Section 2.2.2 & Figure 10). JSON Web Tokens(JWT) are an open, industry standard RFC 7519 method for representing claims securely between two parties. After successfully generating your jwt secret, now you can start your execution by mentioning a port number to engage in RPC connections. You can use the generated jwt secret to authenticate the RPC connection between your clients.

All consensus clients expose a Beacon API that can be used to query information and status of the client and blocks. You can use a public checkpoints to sync your chain faster and sync your geth node.



### **4.2.2 Practical Implementation**

For this part, we are going to see how to connect an execution client(eth) with a consensus client(prysm) and successfully run an ethereum node. PLEASE REFER TO SECTION 2.2. HERE WE HAVE DEMONSTRATED ON RUNNING AN EXECUTION CLIENT AND CONNECT IT WITH A CONSENSUS CLIENT.