Geth — “……if you have experience with web development and are interested in building frontends for dapps, you should experiment with Geth.”

Eth — “…If you want added security by running two different implementations in parallel or are serious about GPU mining, then the C++ “Eth” client is for you.”

Pyethapp — “ …If you are a Python developer that wants to build decentralized apps or are interested in Ethereum for research or an academic purpose, this is a great client”

**Note:**

Pyethapp is officially deprecated. If you want to run a mainnet node, we recommend using either Geth or Parity. If you want to use Python to develop tooling or interact with the Ethereum ecosystem, we recommend that you shift you attention to Trinity which is based on py-evm.

Pyethapp will no longer receive updates.

**Geth**

Geth is a command line interface (CLI) tool that communicates with the Ethereum Network and acts as the a link between your computer, its hardware and the rest of the ethereum nodes or network computers.

If a block is mined by another node, your Geth program will pick it up and then pass on the new information onto your GPU or CPU to update the blockchain. With Geth, you can

* mine ether (ether is the cryptocurrency fueling the Ethereum network)
* transfer funds between addresses
* create smart contracts and send transactions to them

**Geth Command Help:**

**> eth.**

eth.\_requestManager eth.getBlockUncleCount eth.getWork

eth.accounts eth.getCode eth.hashrate

eth.blockNumber eth.getCoinbase eth.iban

eth.call eth.getCompilers eth.icapNamereg

eth.coinbase eth.getGasPrice eth.isSyncing

eth.compile eth.getHashrate eth.mining

eth.constructor eth.getMining eth.namereg

eth.contract eth.getPendingTransactions eth.pendingTransactions

eth.defaultAccount eth.getProtocolVersion eth.protocolVersion

eth.defaultBlock eth.getRawTransaction eth.resend

eth.estimateGas eth.getRawTransactionFromBlock eth.sendIBANTransaction

eth.filter eth.getStorageAt eth.sendRawTransaction

eth.gasPrice eth.getSyncing eth.sendTransaction

eth.getAccounts eth.getTransaction eth.sign

eth.getBalance eth.getTransactionCount eth.signTransaction

eth.getBlock eth.getTransactionFromBlock eth.submitTransaction

eth.getBlockNumber eth.getTransactionReceipt eth.submitWork

eth.getBlockTransactionCount eth.getUncle eth.syncing

**> personal**

personal.\_requestManager personal.getListWallets personal.newAccount

personal.constructor personal.importRawKey personal.sendTransaction

personal.deriveAccount personal.listAccounts personal.sign

personal.ecRecover personal.listWallets personal.unlockAccount

personal.getListAccounts personal.lockAccount

**> admin**

admin.addPeer admin.importChain admin.startRPC

admin.constructor admin.isPrototypeOf admin.startWS

admin.datadir admin.nodeInfo admin.stopRPC

admin.exportChain admin.peers admin.stopWS

admin.getDatadir admin.propertyIsEnumerable admin.toLocaleString

admin.getNodeInfo admin.removePeer admin.toString

admin.getPeers admin.sleep admin.valueOf

admin.hasOwnProperty admin.sleepBlocks

**> miner**

miner.constructor miner.setEtherbase miner.toLocaleString

miner.getHashrate miner.setExtra miner.toString

miner.hasOwnProperty miner.setGasPrice miner.valueOf

miner.isPrototypeOf miner.start

miner.propertyIsEnumerable miner.stop

***Genesis.json:***

*{*

***"config":*** *{*

***"chainId":*** *987,*

***"homesteadBlock":*** *0,*

***"eip155Block":*** *0,*

***"eip150Block":*** *0,*

***"eip158Block":*** *0*

*},*

***"difficulty":*** *"0x400",*

***"gasLimit":*** *"0x8000000",*

***"alloc":*** *{}*

*}*

**config:** the config block defines the settings for our custom chain and has certain attributes to create a private Blockchain

**chainId:** identifies our blockchain, the main Ethereum chain has its own ID, but we will set it to a unique value for our private chain

**homesteadBlock:** Homestead is the second major version of the Ethereum platform and is the first production release of Ethereum. It includes several protocol changes. Since we are already on homestead version, this attribute is 0.

**eip155Block/eip158Block:** Homestead version was released with a few backward-incompatible protocol changes, and therefore requires a hard fork. These protocol changes/improvements proposed through a process Ethereum Improvement Proposals (EIPs). Our chain however won’t be hard-forking for these changes, so leave as 0

**difficulty:** This value is used to control the Block generation time of a Blockchain. The higher the difficulty, the statistically more calculations a Miner must perform to discover a valid block. On our test network, we will keep this value low to avoid waiting during tests, since the generation of a valid Block is required to execute a transaction on the Blockchain.

**gasLimit:** this value specifies current chain-wide limit of 'Gas' expenditure per block. Gas is Ethereum's fuel that is spent during transactions. We will mark this value high enough in our case to avoid being limited during tests.

**alloc:** This is where you can create your wallet and pre fill it with fake ether. For this post however, we will mine our ether locally quickly so we don’t use this option.

**Step to Create Multi Node Private Blokchain Network**

1. To create your first node, open a new terminal window and navigate to your project folder, and type in the following command:

**$ geth --datadir blkchain1 init genesis.json**

**--**datadir we specify the name of the directory where the local copy of the blockchain will be stored on the node.

**Note:** If you already run init then it will not create genesis block again, you need to delete from root folder.

1. start Node 1 using Geth. Within your terminal window, navigate to your project folder (where you saved your genesis.json file) and type in the following:

**$ geth --datadir blkchain1 --nodiscover --networkid 1234 console**

**--datadir blkchain1:**

specifies the data directory of the blockchain. If you do not specify this, it will default to the main Ethereum blockchain.

**--nodiscover:**

disables the peer discovery mechanism and enables manual peer addition.

**--networkid 1234:**

identity of your Ethereum network, other peer nodes will also have the same network identifier. The network id can be any random integer value.

1. Check node information in geth Console:

**> admin.nodeInfo**

**Output:** Screenshot of running node

**Note the following:**

* by default, the listener port is 30303
* the enode id is your node id
* the discovery port is ‘0’, since we set it to --nodiscover.
* To get a list of all admin commands, type in “admin.” and press <tab>

**Remember to access private network**

* **geth -- init “E://Private\_Blockchain//Chain//myGenesis.json -- datadir “E://Private\_Blockchain //Data”**
* **geth --datadir "E://Private\_Blockchain//Data" --networkid 416 console**
* Next, open another terminal and connect with existing running node by typing
* **geth attach "//./pipe/geth.ipc" (**<https://github.com/ethereum/go-ethereum/issues/17298>**)**
* It will open javascript terminal.

1. Set up Account: (where the mined ethers will be collected)

**> personal.newAccount()**

To get a list of all the accounts:

**> personal.listAccounts**

**Node 1 is now ready!**

**Keep the terminal window (with the Geth Console for Node1) open and running.**

**We will open a new terminal window to setup Node 2.**

1. **Set up the second node (Node 2)**

The process will be similar to setting up Node1.

Open a new terminal window and navigate to the project folder that contains the **genesis.json** file.

Initialize the new node with the following command:

$ **geth --datadir blkchain2 init genesis.json**

**Note**: Since we want this node to be part of the same blockchain, we use the same genesis block.

This will create a new node whose data will be stored in a new directory called **blkchain2**(this will contain the local copy of the blockchain database for the node 2).

To get the node 2 up and running with a Geth console, type in the following:

$ **geth --datadir blkchain2 --nodiscover --networkid 1234 --port 30304 console**

This will start up Node 2 and bring up the Geth console connected to Node 2.

**Note the following differences for Node 2:**

* port number 30304
* The default port 30303 is already being used by the first node, so if we do not specify a separate port it will throw an error.
* we give the same networkid as we did for node 1. This is important, since we want both the nodes to be part of the network
* we specify that the — -datadir for node2 will be in blkchain2

**Run the admin.nodeInfo command in Geth Console for Node2:**

> admin.nodeInfo

Output Screenshot:

**Notice:**

**Compare the enode id by running the same admin.nodeInfo command in the Terminal window running the Geth console for Node1.**

Screenshot of Node-1

Screenshot of Node-2

1. **Set up Account for node-2:**

personal.newAccount() in the Geth console for Node2

**Node 2 is now ready!**

**Keep open both the terminal windows, one running the Geth Console (Node1) and the second running the Geth Console (Node2), side by side.**

**In the next step to connect Node1 and Node2 and create the blockchain network!**

1. **Connect the node 1 and node 2 (**by adding one of the nodes to the other as a “peer”**)**

Run the following command on the Geth Consoles of both Node1 and Node2:

**> admin.peers**

**Notice that both return an empty array, since the nodes are not connected to any other nodes.**

1. **Add Node1 as a peer to Node2**
2. Run the **admin.nodeInfo** command on Node1
3. Copy the enode id for Node 1:

"enode://549468e6d00e135128af33e03a6d27b0ee5fda7fbd0154b2e83fe68afdfda869eb6ace6ccaefe84ed7a5b804529dcef49f0b5d64be97da87b1e28ddecfca227a@[::]:30303?discport=0"

1. In the Geth Console for Node2, add the Node1 as a peer to Node2:

> admin.addPeer("enode://549468e6d00e135128af33e03a6d27b0ee5fda7fbd0154b2e83fe68afdfda869eb6ace6ccaefe84ed7a5b804529dcef49f0b5d64be97da87b1e28ddecfca227a@[ ::]:30303?discport=0")

1. Run the **admin.peers** command in either of the Geth Console you will see the other Node listed as a peer!
2. **Verify that you have a connected blockchain and a distributed database**

**Requirements:**

* Two terminal windows open
* One running Geth Console for Node1, and the second running Get Console for Node2
* Node 1 saves its local copy of the blockchain in the blkchain1 folder
* Node 2 saves its local copy of the blockchain in the blkchain2 folder
* Also, in the previous step you connected the two nodes using admin.addPeer()

1. **if the new blocks are mined and added in Node 1, this new block should propagate over the blockchain network, and the local blockchain copy of node2 should update automatically.**

**In the Geth Console(Node1) let’s begin mining:**

> miner.start(1)

**To check how many blocks have been mined at Node1 at any point, run the following in the Geth Console (Node1):**

> eth.blockNumber

This will return the number of the current block mined (or the blockchain height).

You will notice that the block height in Node 2 matches with the block height in Node 1! This verifies that the Blocks that were mined in Node 1 were propagated over the Blockchain to Node 2.

To kick it up a notch, you can verify the data in the actual blocks on Node 1 and Node 2.

Wait until about 10 blocks have been mined and then run the following command in Geth **Console(Node1) and Geth Console(Node2):**

> eth.getBlock(3)

This command displays the contents of Block 3 from the blockchain.

Notice that the data displayed in Geth Console (Node1) and Geth Console(Node2) is identical, indicating that the Block3 that was mined by Node1 was propagated over the blockchain network and is also part of the local blockchain data in Node 2.

1. **Sending Ether from one account to other**

If you only created one account, then create a second account. Confirm the balance of this account running

eth.getBalance(‘address’) then send some ether using

eth.sendTransaction({from:”address”, to:”address”, value: web3.toWei(amount, "ether")})

e.g. **eth.sendTransaction({from:eth.accounts[0], to:eth.accounts[1], value: web3.toWei(4, "ether")})**

**Note:**

If you try running the example, you will get an authentication error. Before you transfer ether from an account, for obvious reasons you need to unlock the account first.

To unlock the sender account in this scenario, we will use the personal library.

Run the command below:

**personal.unlockAccount(eth.accounts[0], "password")**

**Attempt to send ether again and check the balance of both accounts.**

The last command should return a hash of the transaction. In my case it returned **“0xf94ece7d159bcc750862b92396e0d98c537593bfa2ecb3a21ec49b0d439592eb”.**

**Note:**

You would expect eth.getBalance(eth.accounts[1]) to be 4, right? Well, it will return 0 because the transaction has not been added to the blockchain yet.

Also, if you check the balance of the sender, you will notice that hasn’t changed either.

In order for this transaction to take effect, i.e. the account balances reflect the new values, the transaction needs to be added to a block and the block to the blockchain i.e. mined. So, lets mine.

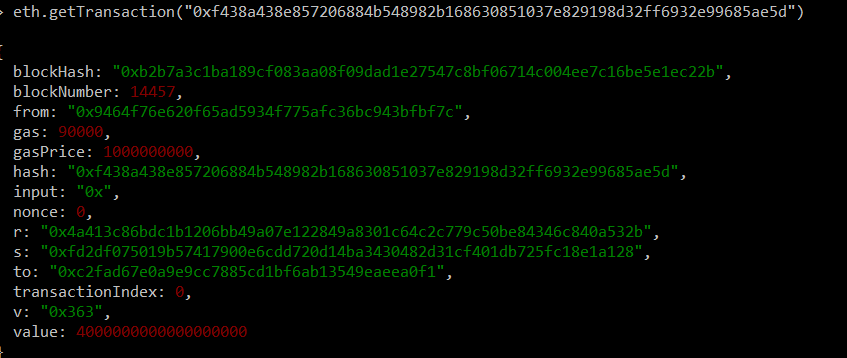
**run miner.start() and then miner.stop()**

If you look at the console output, you will notice the first block mined has one transaction. If you check the balance of the sender and receiver, they should both show the expected values.

1. **check the details of the transaction by running**

eth.getTransaction("transactionId")

e.g. eth.getTransaction("0xf94ece7d159bcc750862b92396e0d98c537593bfa2ecb3a21ec49b0d439592eb")



1. Y**ou can get more information about the block by running.**

eth.getBlock("blockAddress")

e.g. eth.getBlock("0x1420da4e3dc647c651135b6d2f617e7046fc748c0321c567cec5e92df352ff54")

**Summarization of Blockhain Based Private Network**

* Exit out of the geth console if you still have it running then restart but this time around provide a networkid

**geth --datadir ./datadir --networkid 2018 console**

* The last step just restarted the old blockchain with a networkid.
* To add a new node, run the command below in a new terminal

**geth --datadir ./datadir\_new init init.json**

**Two important things to note here.**

1. You must use the same init.json file
2. You must use a different datadir folder. The data dir is effectively the database/local copy of the blockchain and each node needs to have its own copy.

* Now run the new node, ensure you use the same network id.
* You also need to specify port. The default port is already used by the first node.
* You need to add **nodiscover**to ensure that the node is discoverable, it is not required but it’s good practice.

**geth --datadir ./datadir\_new --networkid 2018 --port 30306 --nodiscover console**

* in the console of the of second node (you can use either), run admin.nodeInfo you should get something similar to this
* copy the value of the enode property and in the console of the first node run

admin.addPeer(enode://f7aa5b604056ff77dc561034f12874586b44b4a00e92355e7f750cfb43717ef1d0092f208b08661b209656d09540b11f6d0c6667a611674f7a75b718424d0c9a@[::]:30306?discport=0")

* This function returns true if successful but that doesn’t mean the node was added successfully. To confirm run admin.peers and you should see the details of the node you just added
* if you start mining on the first node by running miner.start(1) you’ll notice the block depth/number increase on the second node.

Note:

**Restart the network By**

geth --datadir Data --networkid 416 --nodiscover --rpc --rpcapi "db,personal,eth,net,web3,debug" --rpccorsdomain=”\*” --rpcaddr="localhost/IP address" --rpcport 8545 console

**Set up Network Analytics:**

<https://medium.com/coinmonks/how-to-build-ethereum-dashboard-and-to-monitor-your-ethereum-network-status-9f1941beac08>

**1. Execute ethereum private/public node**

If you are unfamiliar with how to do this, please refer to my previous article for a step-by-step guide. To execute Ethereum private node the following command:

*geth --networkid 4224 --datadir "E:\privateChain\GethPrivateChain" --nodiscover --rpc --rpcport "8545" --port "30303" --rpccorsdomain "\*" --nat "any" --rpcapi eth,web3,personal,net --unlock 0 --password password.sec*

**2. Download and Install Tools**

To clone and install eth-netsates run the following command:

C:\> git clone https://github.com/cubedro/eth-netstats

C:\> cd eth-netstats

C:\eth-netstats> npm install

C:\eth-netstats> npm install -g grunt-cli

C:\eth-netstats> grunt

**3. To clone and install eth-net-intelligence-api run the following command:**

C:\> git clone https://github.com/cubedro/eth-net-intelligence-api

C:\> cd eth-net-intelligence-api

C:\eth-net-intelligence-api> npm install

C:\eth-net-intelligence-api> npm install -g pm2

**3. Configure the Node Monitoring App**

We will need to modify the app.json file located in the eth-net-intelligence-api directory:

[

{

"name" : "Private\_BT",

"script" : "app.js",

"log\_date\_format" : "YYYY-MM-DD HH:mm Z",

"merge\_logs" : false,

"watch" : false,

"max\_restarts" : 10,

"exec\_interpreter" : "node",

"exec\_mode" : "fork\_mode",

"env":

{

"NODE\_ENV" : "production",

"RPC\_HOST" : "localhost",

"RPC\_PORT" : "8545",

"LISTENING\_PORT" : "30303",

"INSTANCE\_NAME" : "Geth/v1.8.23-stable-c9427004/windows-amd64/go1.11.5",

"CONTACT\_DETAILS" : "",

"WS\_SERVER" : "http://localhost:3000",

"WS\_SECRET" : "mysecret",

"VERBOSITY" : 2

}

}

]

There are few configurations that need to be changed in this file.

* "name": The name that will appear for your node app in the pm2 process manager (see next section for more details on pm2)
* "RPC\_HOST": The IP address for the machine hosting your running gethinstance
* "RPC\_PORT": The RPC port that your geth instance is running on
* "LISTENING\_PORT": The network listening port for your geth instance
* "INSTANCE\_NAME": The name that will appear on the dashboard in your browser for this particular node
* "WS\_SERVER": The server address for the frontend UI (i.e. eth-netstats) running in background
* "WS\_SECRET": The secret used to allow the frontend to connect eth-net-intelligence-api. You choose any value to put here.

In app.json, we have to modify three parameters, namely INSTANCE\_NAME, WS\_SERVER andWS\_SECRET.

You can get INSTANCE\_NAME from private chain information.

WS\_SERVER will point eth-netstats so we use eth-netstats' default url : http://localhost:3000

We set WS\_SECRET = "mysecret"

**5. Start The Node App**

4.1 We execute the backend tool ( eth-net-intelligence-api) run the following command:

C:\eth-net-intelligence-api> pm2 start app.json

**5.2 execute the front-end tool (eth-netstats ) run the following command:**

C:\eth-netstats> set WS\_SECRET=mysecret&& npm start

**5.3 Issues:**

You might encounter two issues- No action and wrong auth

solution:

My solution is restart eth-net-intelligence-api again

C:\eth-net-intelligence-api> pm2 start app.json

or

C:\eth-net-intelligence-api> pm2 restart <App name>

**6. Start the Frontend**

visit http://localhost:3000 in your web browser. If you notice that no node are listed in the monitoring dashboard. This is because no instance of geth is currently running or your eth-net-intelligence-api does not connect to the node.

**Sending ether from geth to Metamask account**

Open geth in new terminal and type this command.

> geth attach ipc:/\*\*path-to-geth.ipc\*\*

This opens javascript console.

Send ether from geth account to Metamask with the below command,

eth.sendTransaction({from:eth.accounts[0], \*\*\*metamask account\*\*\*,value: web3.toWei(4, “ether”)})

and start mining so that the transaction gets processed.

Miner.start(1)

The transferred ether gets reflected in your Metamask account after some time(longer than expected).

**Importing geth account to Metamask**

Now your geth and Metamask are connected to localhost:8545. You need to import the account in this localhost network.

You can import geth account to your Metamask.

1. Click on Metamask extension, open the account menu in the top right and click on Import Account.

2. You can choose to import using Private Key and JSON File.

3. If you choose to use Json File type then you need to import KeyFile stored in your Keystore folder where your geth data is present.

4. Choose the KeyFile and enter the password you entered when you created this account in geth.

5. Click on Import button.