Table of Contents

[Setup resource group and nodes 3](#_Toc516646687)

[Network Security Group setup 4](#_Toc516646688)

[Setup Geth 4](#_Toc516646689)

[Create workspace for your test network 5](#_Toc516646690)

[Create wallets 5](#_Toc516646691)

[Create Genesis block 6](#_Toc516646692)

[Initialize your nodes 8](#_Toc516646693)

[Create a boot node 8](#_Toc516646694)

[Get Nodes live 9](#_Toc516646695)

[Validate that Boot nodes is getting Pings from miners 9](#_Toc516646696)

[Validate that both miners/validators are mining/sealing 10](#_Toc516646697)

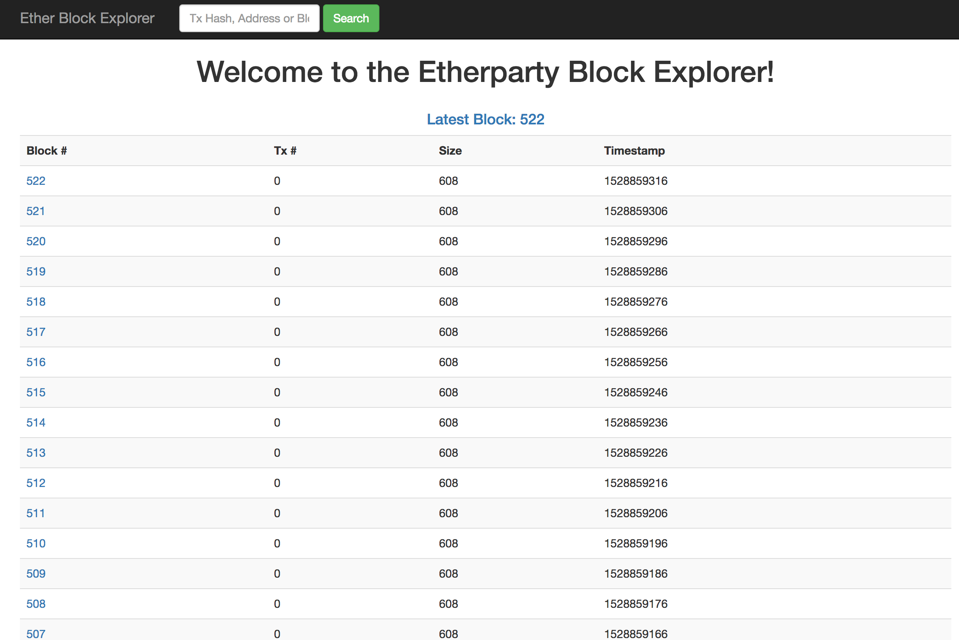
[Recap 10](#_Toc516646698)

[Run a voting to add a miner 11](#_Toc516646699)

[Setup Metamask – Fund and test transaction 12](#_Toc516646700)

[Setup Netstats 13](#_Toc516646701)

[Setup Block explorer 13](#_Toc516646702)

[ 14](#_Toc516646703)

[IOT Benchmark 14](#_Toc516646704)

[Notary App 14](#_Toc516646705)

[Replicated Bootnodes 14](#_Toc516646706)

[Setup secure network – design pattern 14](#_Toc516646707)

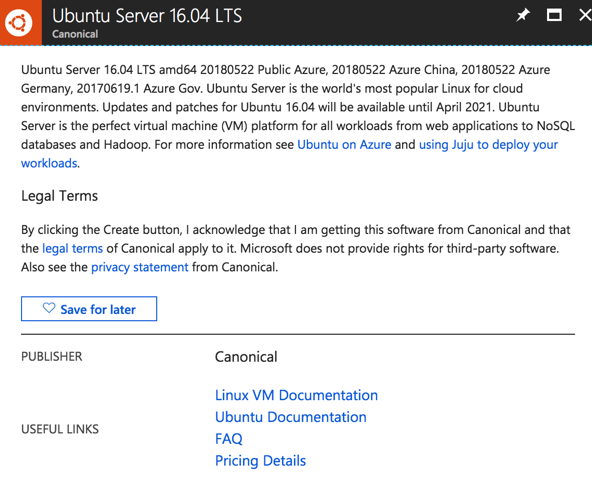
[Load balance transactions for network 14](#_Toc516646708)

[Azure Automation for shut-down of services etc. 15](#_Toc516646709)

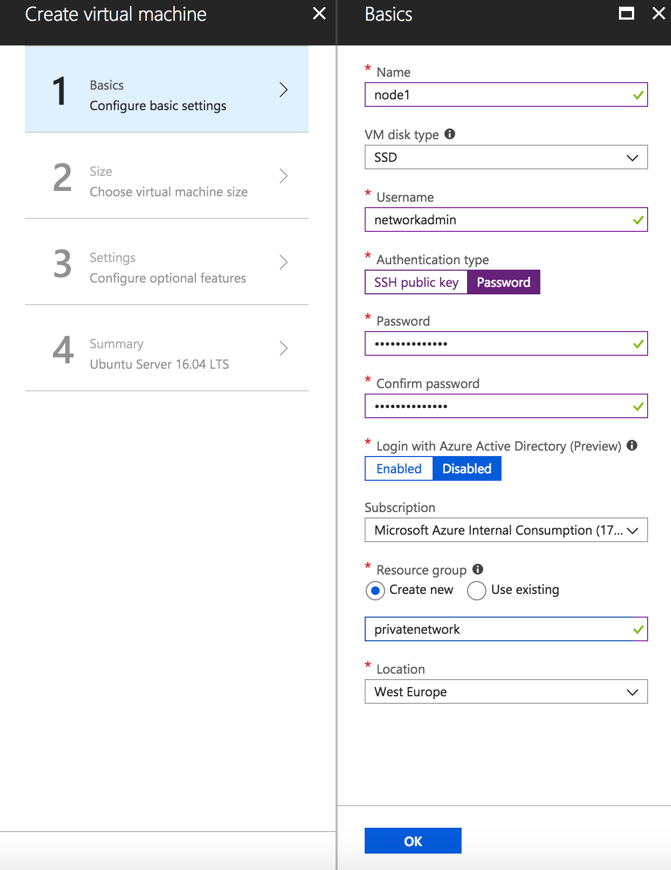
[References: 15](#_Toc516646710)

# Setup resource group and nodes

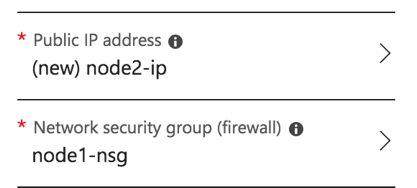
For private network we are going to use Ubuntu (however setting up a network on any other distribution shall work fine).



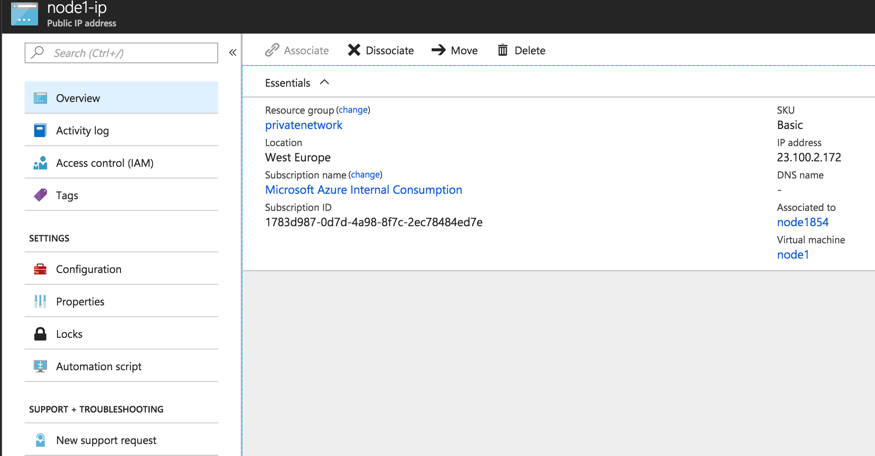
Provision 2 nodes/vm, we name them node1 and node2. Make a note of Username as you will use it to ssh to vm.



For sake of simplicity for firewall handling use same network security group for both nodes. In this instance for node2 the same network security group provisioned for node1 is re-used. By default with every vm provisioning a new network security group is provisioned.



Go to resource group and make a note of public ip addresses from list of resources.



# Network Security Group setup

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Security rule** | **Port** | **Protocol** | **Source** | **Destination** | **Purpose** |
| Inbound | 22 | TCP | Any | Any | For ssh |
| Inbound | 30310-30312 (port range for boot nodes) | UDP | Any (Better use CIDR or specific IP addresses of network) | Any (Better use the IP address of **boot node host(s)**) | Boot node |
| Inbound | 40310-40311 (range of ports used by nodes/miners/tx-nodes) | TCP | Any (or better use CIDR or specific IP addresses of network) | Any (or **better use the IP addresses of network**) | For p2p communication (check via **admin.peers**) |
| Inbound | 8501-8502 (range of RPC ports used by nodes) | TCP | Any (or better use CIDR or specific IP addresses of network) | Any (or **better use the IP addresses of network**) | Used for Metamask, or e.g.  geth attach 'http://IP:RPC\_PORT' |
| Inbound | 8000 | TCP | Any (or any machine with static ip from where block explorer needs to be accessed) | IP address of vm running explorer | Used for block explorer  'http://IP:8000 |
| Inbound | 3000 | TCP | Any (or any machine with static ip from where block Ethstats needs to be accessed) | IP address of vm running eth-netstats server | Used for WebSockets api server, used for eth-netstats |

# Setup Geth

Ssh to your nodes and install geth on both nodes.

ssh \_your\_username@\_node\_1\_2\_public\_ip\_address

Use following install instructions

|  |  |
| --- | --- |
| **Ubuntu** | **Mac** |
| sudo apt-get install software-properties-common  sudo add-apt-repository -y ppa:ethereum/ethereum  sudo apt-get update  sudo apt-get install ethereum | brew tap ethereum/ethereum  brew install ethereum  Or build from source  xcode-select --install  git clone https://github.com/ethereum/go-ethereum  brew install go  cd go-ethereum  make geth |

In addition browse source code

git clone <https://github.com/ethereum/go-ethereum>

cd go-ethereum/

Verify geth installation

geth version

You should see something like this

Version: 1.8.11-stable

Git Commit: dea1ce052a10cd7d401a5c04f83f371a06fe293c

Architecture: amd64

Protocol Versions: [63 62]

Network Id: 1

Go Version: go1.10

Operating System: linux

GOPATH=

GOROOT=/usr/lib/go-1.10

## Create workspace for your test network

mkdir my-ethereum-network

cd my-ethereum-network

On node 1 create two peers in your workspace

mkdir node1 node2

On node 1 create two additional peers

mkdir node3 node4

## Create wallets

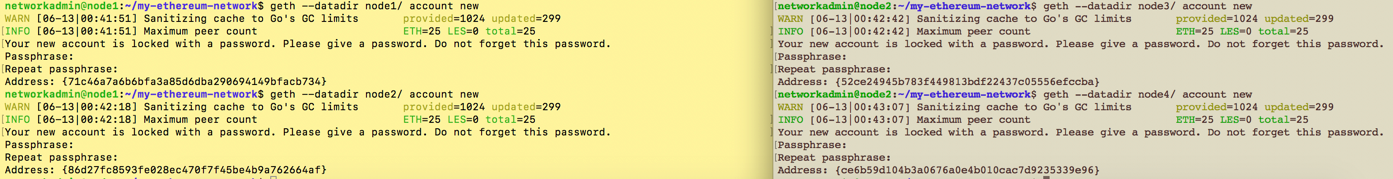
The wallets (or node accounts) hold a key-pair (private-public key pair) required interacting with blockchain network.

There are 2 key types of nodes

1. Boot nodes
2. Peers/nodes
   1. Minding nodes or Voters or Sealers
   2. Transaction nodes (non-miners / non-voters etc)

We need create accounts/wallets for our nodes that we want to create a Blockchain network with.

geth --datadir node**1-4**/ account new



Save generated accounts (e.g. in above case **52ce24945b783f449813bdf22437c05556efccba**)

Validate that keystores are generated

ls node**1-4**/keystore/



Save password and accounts for your ease.

echo 'node1: \_generated\_account\_address' >> accounts

echo 'your\_account\_pwd' >> password.txt

## Create Genesis block

Genesis file/block is used to initialize a chain. This is also used to join an existing network.

For that, you will need accounts for purposes of approving them as sealers and pre-funding them.

puppeth

+-----------------------------------------------------------+

| Welcome to puppeth, your Ethereum private network manager |

| |

| This tool lets you create a new Ethereum network down to |

| the genesis block, bootnodes, miners and ethstats servers |

| without the hassle that it would normally entail. |

| |

| Puppeth uses SSH to dial in to remote servers, and builds |

| its network components out of Docker containers using the |

| docker-compose toolset. |

+-----------------------------------------------------------+

Please specify a network name to administer (no spaces or hyphens, please)

> mynetwork

Sweet, you can set this via --network=mynetwork next time!

INFO [06-07|07:47:41] Administering Ethereum network name=mynetwork

WARN [06-07|07:47:41] No previous configurations found path=/Users/haseeb/.puppeth/mynetwork

What would you like to do? (default = stats)

1. Show network stats

2. Configure new genesis

3. Track new remote server

4. Deploy network components

> 2

Which consensus engine to use? (default = clique)

1. Ethash - proof-of-work

2. Clique - proof-of-authority

> 2

How many seconds should blocks take? (default = 15)

> 10

Which accounts are allowed to seal? (mandatory at least one)

> 0x4db50f170ad86be76293efc6e825d450f85d00b6

> 0xf84363c5b0f84aa178ca2f4d075baeb41d155d87

> 0x

Which accounts should be pre-funded? (advisable at least one)

> 0x4db50f170ad86be76293efc6e825d450f85d00b6

> 0xf84363c5b0f84aa178ca2f4d075baeb41d155d87

> 0x

Specify your chain/network ID if you want an explicit one (default = random)

> 1500

INFO [06-07|07:49:01] Configured new genesis block

What would you like to do? (default = stats)

1. Show network stats

2. Manage existing genesis

3. Track new remote server

4. Deploy network components

> 2

1. Modify existing fork rules

2. Export genesis configuration

3. Remove genesis configuration

> 2

Which file to save the genesis into? (default = mynetwork.json)

>

INFO [06-07|07:49:22] Exported existing genesis block

What would you like to do? (default = stats)

1. Show network stats

2. Manage existing genesis

3. Track new remote server

4. Deploy network components

> ^C

Explore your genesis block and try to understand it

haseeb$ less mynetwork.json | more

{

"config": {

"chainId": 1500,

"homesteadBlock": 1,

"eip150Block": 2,

"eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",

"eip155Block": 3,

"eip158Block": 3,

"byzantiumBlock": 4,

"clique": {

"period": 10,

"epoch": 30000

}

},

"nonce": "0x0",

….

….

"4db50f170ad86be76293efc6e825d450f85d00b6": {

"balance": "0x200000000000000000000000000000000000000000000000000000000000000"

},

"f84363c5b0f84aa178ca2f4d075baeb41d155d87": {

"balance": "0x200000000000000000000000000000000000000000000000000000000000000"

}

},

"number": "0x0",

"gasUsed": "0x0",

"parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000"

}

Copy your generated genesis block to node

scp mynetwork.json node2\_user\_name@node2\_ip:/home/networkadmin/my-ethereum-network

## Initialize your nodes

Initialize your nodes with generated genesis block. Remember anytime you change your genesis block you must re-initialize your nodes.

geth --datadir node1-4/ init mynetwork.json

## Create a boot node

Boot node helps node discovery for network. You can create your network without a boot node but then adding a node to network might require manual addition throught management api. Please see reference, the desired API will be admin.addPeer(…..).

bootnode -genkey boot.key

bootnode -nodekey boot.key -verbosity 9 -addr :30310

INFO [06-07|08:04:30] UDP listener up self=enode://f7693502b46789545f6b48d6ca46f8cec79f075a0a467e9e18e0bc729c7eacc7f6e9d3fd27797f4aabc1497069478dbf589222d8bb5194b0d913feec1e4af377@[::]:30310

Save generated enode for latter reference. Enode uniquely refers to a node, in this instance our created boot node. Also note we started boot node on port 30310, we can start it on any other port as well (avoid commonly used ports though)

## Get Nodes live

geth --datadir node1/ --syncmode 'full' --port 40310 --rpc --rpcaddr '0.0.0.0' --rpcport 8501 --rpcapi 'personal,db,eth,net,web3,txpool,miner' --bootnodes 'enode://f7693502b46789545f6b48d6ca46f8cec79f075a0a467e9e18e0bc729c7eacc7f6e9d3fd27797f4aabc1497069478dbf589222d8bb5194b0d913feec1e4af377@127.0.0.1:30310' --networkid 1500 --gasprice '1' -unlock '0x4db50f170ad86be76293efc6e825d450f85d00b6' --password node1/password.txt --mine console

**Note:**

Change --bootnodes enode to your boot node’s enode.

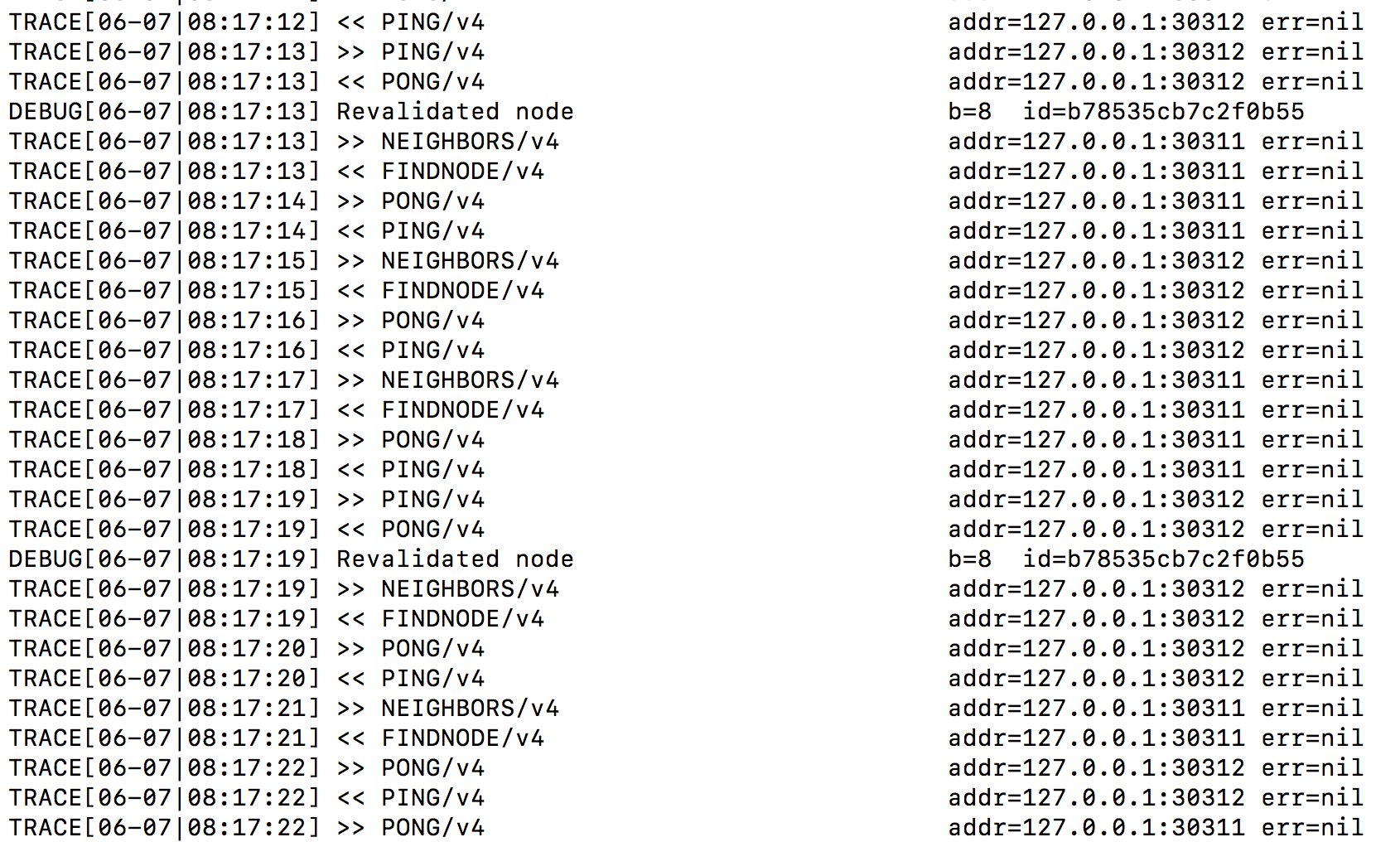
Change unlock address to your node’s address

Use the same networkid you choose for your genesis block

Similarly get node 2 live. Make sure to not create any port conflict (if running on same machine; we will use a different port as we have 2 nodes each on each vm)

## Validate that Boot nodes is getting Pings from miners

Check your boot node and see PING/PONG from both nodes. You can identify them by ports (if you used same machine or by IP if both nodes are on different machines)



## Validate that both miners/validators are mining/sealing



## Recap

Both accounts for both miners were approved as signers in genesis block. That’s why they could start mining upon start. No voting was needed for them to be approved.

Try out some management API.

1. **Interaction with console:** As we started nodes with console, you can interact with nodes directly.
2. **Interaction through IPC:** Another way is to attach to node’s IPC to get access to console

$ geth attach node1/geth.ipc

* Try this out – this has to be done locally from same machine

1. Interaction through RPC:

$ geth attach 'http://IP:RPC\_PORT'

Please note that we allowed RPC for nodes, please go back to command and see –-rpc and in --rpcapi we allowed following APIs for a node 'personal,db,eth,net,web3,txpool,miner

e.g. try out following commands through any of the above

> admin.peers

e.g. Check out signers/sealers/miners in network (remember we used clique as consensus algorithm)

> clique.getSigners()

["0x4db50f170ad86be76293efc6e825d450f85d00b6", "0xf84363c5b0f84aa178ca2f4d075baeb41d155d87"]

Also note that we instructed nodes upon starting to mine using --mine. You can use API’s like miner:start() or miner.stop() etc.

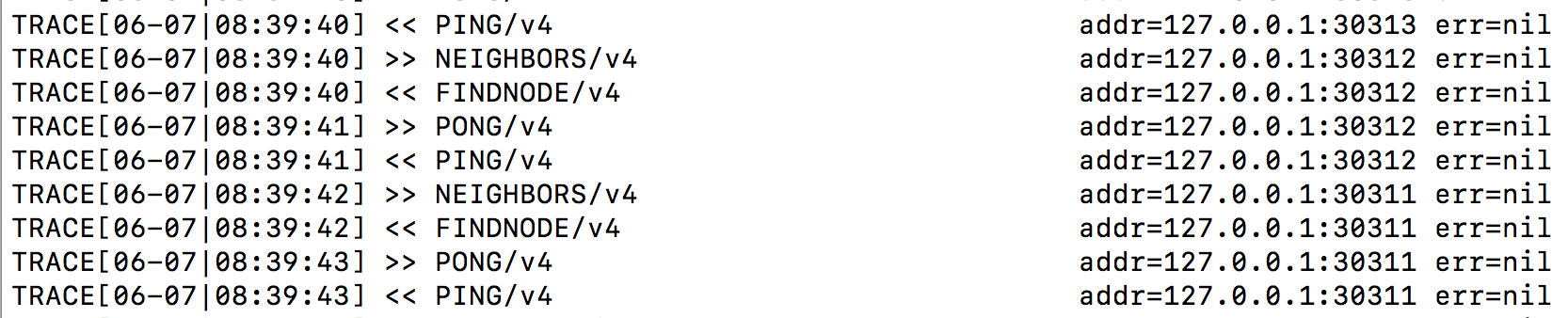
## Run a voting to add a miner

Start node3 by following similar steps

1. Creating a datadir for node
2. Creating an account for node
3. Initializing the node with same gensis block
4. Making node live with same boot node, networkid. (again be careful of ports you use if you are using the same machine)

For ease, start the node with --mine and console. i.e. start a console and instruct it to start mining right away.

First check that boot node is getting PINGS from new node. (note, all ports listed here – from ports you can identify if using same IP for all nodes)



Check all 3 nodes how does a peer count look like using

> admin.peers

Now verify signers in network (all 3 nodes, including node 3 should have the already pre-approved 2 signers from node1 and node2) i.e. node3 even though instructed to mine can’t mine and is not approved.

> clique.getSigners()

["0x4db50f170ad86be76293efc6e825d450f85d00b6", "0xf84363c5b0f84aa178ca2f4d075baeb41d155d87"]

On node3 you should see something like

INFO [06-07|08:43:26] Commit new mining work number=171 txs=0 uncles=0 elapsed=216.97µs

WARN [06-07|08:43:26] Block sealing failed err=unauthorized

INFO [06-07|08:43:36] Imported new chain segment blocks=1 txs=0 mgas=0.000 elapsed=459.692µs mgasps=0.000 number=171 hash=dba37f…783825 cache=0.00B

INFO [06-07|08:43:36] Commit new mining work number=172 txs=0 uncles=0 elapsed=121.55µs

WARN [06-07|08:43:36] Block sealing failed err=unauthorized

INFO [06-07|08:43:46] Imported new chain segment blocks=1 txs=0 mgas=0.000 elapsed=655.961µs mgasps=0.000 number=172 hash=a68d12…4b26bb cache=0.00B

INFO [06-07|08:43:46] Commit new mining work number=173 txs=0 uncles=0 elapsed=235.461µs

WARN [06-07|08:43:46] Block sealing failed err=unauthorized

Now propose/vote node3 to be a signer/sealer from node1 and node2 (note that to approve a signer, majority needs to vote). Propose the account for node3.

> clique.propose("0x3d8e07c1f21a74d18c87849abbc2ee6da50fb0ed", true)

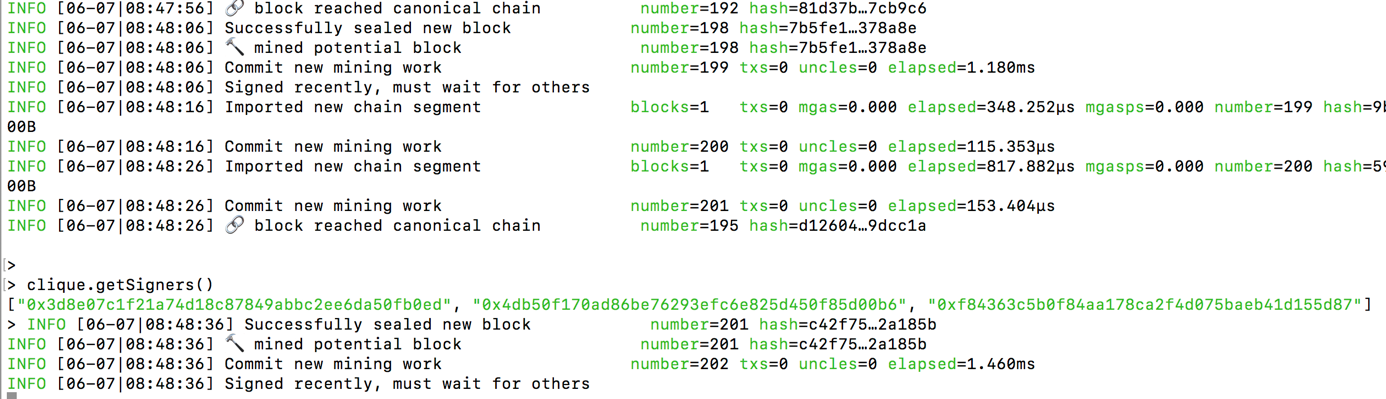
null

Now you should be able to see node3 signing/committing blocks. All nodes should show node3’s account as a valid signers as well.

> clique.getSigners()

["0x3d8e07c1f21a74d18c87849abbc2ee6da50fb0ed", "0x4db50f170ad86be76293efc6e825d450f85d00b6", "0xf84363c5b0f84aa178ca2f4d075baeb41d155d87"]

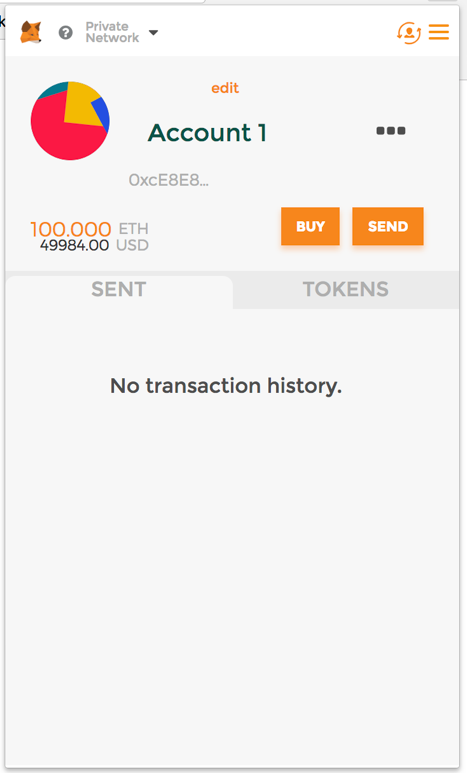
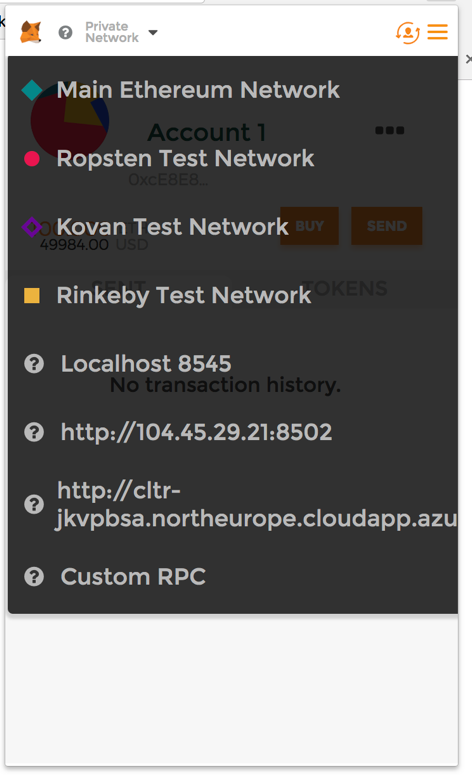
Node 3 should show something like this



# Setup Metamask – Fund and test transaction

Install Metamask chrome plugin. And connect to your private network, through any of the nodes. Please note that **rpcaddr** should allow connection from any source, and **rpccorsdomain** shall allow any external application as well, like this

--rpcaddr '0.0.0.0' --rpcport 8502 --rpccorsdomain "\*"



Copy the address from newly created account. It should initially show balance as 0. Now goto console of any node and use pre-funded account to new address and send transaction.

var tx = {from: "0x\_\_funded\_account", to: "0x\_\_account\_to\_fund", value: web3.toWei(100, "ether")}

personal.sendTransaction(tx, “\_your\_passphrase\_”)

# Setup Netstats

# Setup Block explorer

sudo apt install npm

sudo apt-get install nodejs

sudo npm install bower -g

git clone <https://github.com/etherparty/explorer>

npm start

Edit explorer/app/app.js file to point to transaction node



Visit <http://_public_ip_address:8000/> to access block explorer.

# 

# IOT Benchmark

# Notary App

# Replicated Bootnodes

# Setup secure network – design pattern

# Load balance transactions for network

# Azure Automation for shut-down of services etc.

# References:

1. Try out [API’s](https://github.com/ethereum/go-ethereum/wiki/Management-APIs) yourself
2. Try out [Metamask](https://github.com/cubedro/eth-netstats) for wallet creation. (here you might have an issue with RPC ports – make sure those are open).
3. Install [ethstats](https://github.com/cubedro/eth-netstats) to have a view like [this](https://ethstats.net/) for your network.
4. Install [block explorer](https://github.com/carsenk/explorer) to have a visual way to explore your blocks.
5. Explore [remix](https://remix.ethereum.org/) (online solidity compiler), try out some smart contracts.
6. Deploy [smart contracts through remix](https://codeburst.io/build-your-first-ethereum-smart-contract-with-solidity-tutorial-94171d6b1c4b) or try transactions between accounts.