Creating your own

Ethereum Private Network

A Hands-On Tutorial

Arnab Mukherjee

Department of Computer Science & Engineering
Indian Institute of Technology Patna

What is Ethereum?

- **Blockchain Platform:** Ethereum is a decentralized, open-source blockchain platform designed to host the cryptocurrency and execute smart contracts.
- **Turing-Complete Virtual Machine:** Ethereum features the Ethereum Virtual Machine (EVM), which is a Turing-complete, bytecode-execution environment. The EVM ensures code execution consistency across all nodes on the Ethereum network.
- **Ether (ETH) and Gas:** Ether (ETH) is the native cryptocurrency of the Ethereum platform. It is the driving currency for carrying out financial activities and executing smart contracts.

What is an Ethereum Client?

- It is a piece of software that enables a computer to connect to the **Ethereum network**.
- Works with other nodes (or clients) to **validate, execute, and propagate** transactions and smart contracts on the blockchain.
- They also play a critical role in **reaching consensus** on the **state** of the blockchain, through **consensus algorithms** like Proof-of-Work, Proof-of-Statke or Proof-of-Authority.

Popular Ethereum Clients

1. **Geth** (Go-Ethereum)**:**
   Geth is one of the original Ethereum clients written in the **Go** programming language. It is known for its **performance and robustness**.

2. **Parity** (OpenEthereum)**:**
   Parity, also known as OpenEthereum, is written in the **Rust** programming language. It is known for its **efficiency and security features**.

3. **Hyperledger Besu** (formerly Pantheon)**:**
   Besu is an Ethereum client developed by the **Hyperledger Foundation**, implemented in **Java**. It is designed to be **highly modular and enterprise-ready** and is particularly well-suited for **business applications** and consortium chains.

The Clique PoA Consensus

- Designed for **permissioned or private** Ethereum networks.
- Validators nodes **validate transactions** and **create new blocks**.
- Consensus works in **epochs** where a set of **signers** (validators) take turns proposing and validating blocks.
- Transactions in a Clique PoA network are confirmed quickly.
- Clique PoA relies on a **controlled set of validators**, making it more suitable for private networks with known participants.

Outline of this Tutorial

1. Installation of Geth on the host OS.
2. Creating the Genesis Config file `genesis.json`.
3. Initiating Ethereum **Node-1**.
4. Initiating Ethereum **Node-2** and Connecting it with **Node-1**.
5. Ether Transfer from and to Accounts:
    1. From an Account to an Account on the **same Node**.
    2. From an Account on **Node-1** to an Account on **Node-2**.
6. Connecting the Network Node to **Metamask Wallet**.

Geth Installation

- **Windows**
  1. Download the Installer from https://geth.ethereum.org/downloads
  2. Run the Installer.
  3. Verify Installation by running `geth version`.
- **Linux**
  1. Download the Geth Binaries from https://geth.ethereum.org/downloads
  2. Extract the Tarball (.tar.gz) archive using `tar -xzvf geth.tar.gz`
  3. Add the Path to the geth executable to the PATH environment variable.

The Genesis File (Part-1: Chain Config)

```
"config": {
    "chainId": 824032,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "berlinBlock": 0,
    "clique": {
        "period": 5,
        "epoch": 30000
    }
}
```

The Genesis File (Part-2: Consensus & Accounts)

```
"difficulty": "1",
"gasLimit": "8000000",
"extradata": "0x00...00232C6611680b5993432Fbf35fb94005cAa3AAbd300..00",
"alloc": {
    "232C6611680b5993432Fbf35fb94005cAa3AAbd3": {
        "balance": "300000000000000000000000000000000000000000000000000"
    },
    "f41c74c9ae680c1aa78f42e5647a62f353b7bdde": {
        "balance": "400000000000000000000000000000000000000000000000000"
    }
}
```

Node-1 Setup

1. Create a Directory for Node-1 named `node1`.
2. Create a New Account for Node-1, which will act as the **Signer** account:

```
geth account new --datadir node1
```

3. **Copy the address** of the generated account (as shown in the terminal).
4. Modify the genesis.json to include the **address as the Signer Account**.
   - In the `extradata` field, add **0x**
   - Then add **32 zero bytes**, i.e., 0000....0000 after 0x
   - Then add **the address** of the account generated without the 0x
   - Then add **65 more zero bytes** after the address

Node-1 Setup (contd.)

5. Initialize the **Geth Database** for Node-1 using the Genesis Config file.

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

6. Start the Node-1 using:

```
geth --datadir node1 --networkid UniqueNetworkID --http \
--allow-insecure-unlock --unlock 0xSignerAccountAddress --mine \
--miner.etherbase 0xSignerAccountAddress
```

7. Start the **Javascript Console** for Node-1:

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

Node-1 Console Usage

1. To check the balance of the signer Account, use the following command in the **JS Console**:

```
eth.getBalance("0xSignerAccountAddress")
```

2. Get the **Node Record of Node-1** by executing the following in the **JS Console**:

```
admin.nodeInfo.enr
```

   This shows a string starting with `enr:`, which will be later useful for **connecting Node-2** with **Node-1**.

Node-2 Setup

1. Create a Directory for Node-2 named `node2`.
2. Initialize the **Geth Database** for Node-2 using the Genesis Config file.

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

3. Start the Node-2 using:

```
geth --datadir node2 --networkid UniqueNetworkID \
--port 30306 --authrpc.port 8553 \
--bootnodes "enr:ENRofNode-1"
```

4. Start the **Javascript Console** for Node-2:

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

Node-2 Setup (contd.)

5. Create an Ethereum Account on Node-2

```
geth account new --datadir node2
```

6. Copy the account address and check it's balance.

```
eth.accounts
```

This should show the newly created account's address in the list.

```
eth.getBalance('0xNode2Acc')
```

At this point this account does not have any Ether.

Geth Console Commands

1. Check for Other Peers:

```
admin.peers
```

   this returns an array of all the peers the current node has discovered and connected to.

2. Check for Accounts associated with the current node:

```
eth.accounts
```

   This returns the list of accounts the current node has access to locally.

Transfer Ether for accounts on Same Node

1. On the node's JS Console execute:

```
eth.sendTransaction({from: '0xNodeAcc1', to: '0xNodeAcc2', value: 5000})
```

   Executing this command returns a **hexadecimal string** which is the **Transaction ID**, if the transaction is **successful**, else it prints the **stack trace of the error**.

2. On the node's JS Console check the balance of the account where teh funds are transferred:

```
eth.getBalance("0xNodeAcc2")
```

Transfer Ether from Node-1's to Node-2's account

1. On the Node-1's JS Console execute:

```
eth.sendTransaction({from: '0xNode1Acc', to: '0xNode2Acc', value: 5000})
```

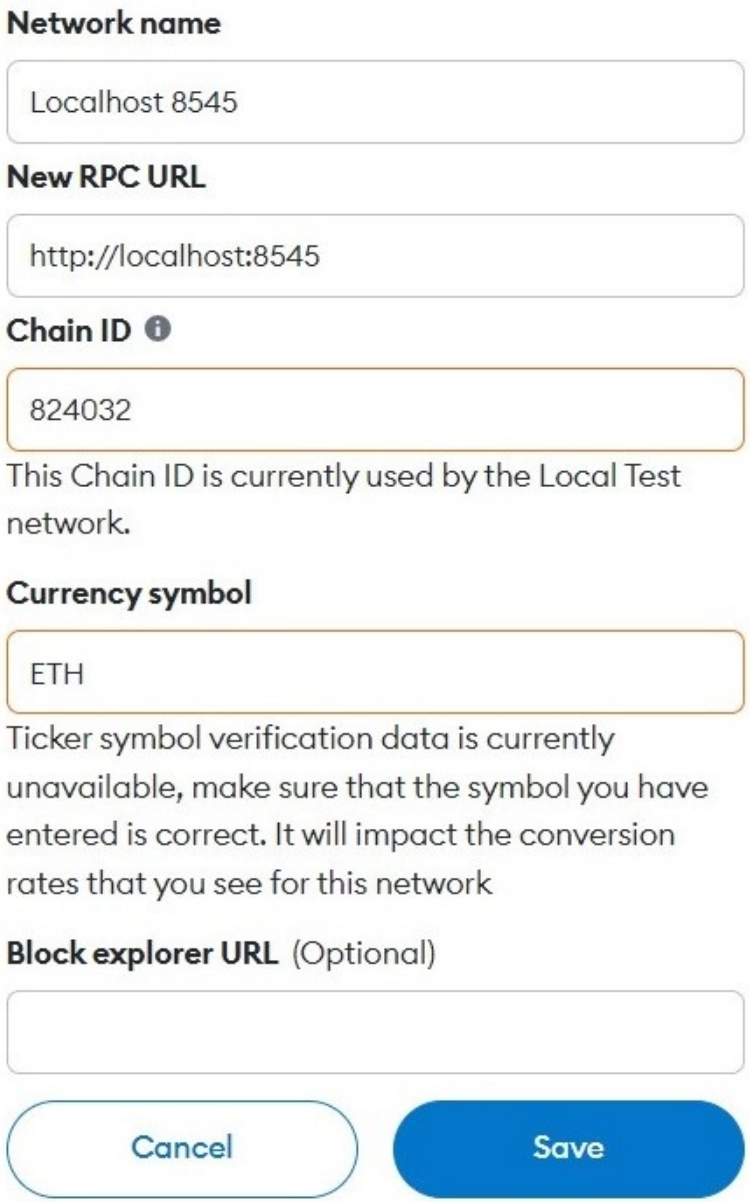
   Executing this command returns a **hexadecimal string** which is the **Transaction ID**, if the transaction is **successful**, else it prints the **stack trace of the error**.

2. On the Node-2's JS Console check the balance of the account where teh funds are transferred:

```
eth.getBalance("0xNode2Acc")
```

Connecting to Metamask

1. Open Metamask.
2. Click on Add Network.
3. Enter the Information as follows:



Import the Account into Metamask

1. Click on Accounts.
2. Click **Import Account**.
3. **Select** Type as `JSON`.
4. In `node1/keystore` a file starting `UTC--` contains the encrypted JSON of the account information. **Select** that file.
5. For **Password**, enter the password you set during the creation of the account.
6. Hit **Import**, and wait for a few minutes (the screen might freeze, due to a bug in Metamask) to import the account.
7. Now you can view the Account on Metamask.

Some Technical Details

- **Ethereum Node Record** (ENR):
  - It is a **discovery protocol** used in Ethereum to **find and connect** to peers on the network.
  - It is a **self-signed** record that contains information about a node, allowing it to announce itself to the network.
    - **Node ID**: This is a unique identifier for the node.
    - **IP Address and Port**: Where others can connect to this node.
    - **UDP Port**: For discovery protocol messages.
    - **Transport Public Key**: For secure communication between nodes.
    - **Signature**: Signed by the private key corresponding to the public key in the record.

Some Technical Details (contd.)

- **enode**:
  - It is an identifier used in Ethereum to represent a node on the network.
  - combination of the node's **Ethereum address**, the **IP address** and **port** that the node is listening on.

    ```
    enode://<node_id>@<ip_address>:<port>
    ```

  - The node's Ethereum address (`<node_id>`) is the last 20 bytes of its public key's hash.

Explaining the Genesis Config

- What were these `"...Block": 0`?

```
  "homesteadBlock": 0,
  "eip150Block": 0,
  "eip155Block": 0,
  "eip158Block": 0,
  "byzantiumBlock": 0,
  "constantinopleBlock": 0,
  "petersburgBlock": 0,
  "istanbulBlock": 0,
  "berlinBlock": 0,
```

  - They define various protocol upgrade blocks.
  - We specify the block numbers at which these protocol upgrades are activated when creating a new Ethereum network.

Explaining the Genesis Config (contd.)

- What were the `epoch` and `period` for `clique`?

```
"clique": {
    "period": 5,
    "epoch": 30000
}
```

  - **Epoch**: A unit of time during which a fixed set of signers take turns for creating blocks.
  - **Period**: A duration of time within which a certain number of blocks must be created. It defines the time allocated for validators to take their turns in creating blocks.

Explaining the Genesis Config (contd.)

- What was the `extradata` field in `genesis.json`?
  - Is used to specify the Ethereum address of the **initial validator(s)** who will participate in block creation.
  - These validators are often referred to as the **"sealers"**.
  - Establishes which accounts have the **authority to participate** in block sealing and block proposal from the very beginning.
  - Additionally, the it can contain other **optional data**, like:
    - such as identifying the network.
    - providing additional information about the genesis block.
    - or indicating a specific network configuration.

Conclusion

In this tutorial, we:

1. Learned about Ethereum and Ethereum Clients.
2. Learned how to design a Private Ethereum Network.
3. Wrote the Genesis Config for the Network.
4. Orchestrated our own Private Ethereum Network with 2 nodes.
5. Transferred funds from one Ethereum Account to another.
6. Connected the Network Node to Metamask Wallet.

Next Steps

- Try Orchestrating a Geth Network on **seperate machines**.
- Learn more about **Geth and Hyperledger Besu**, and how we can design **performant and scalable** Private Ethereum Networks for **Business Applications**.
- Explore ways to **Dockerize** and create **clusters** of Ethereum nodes.
- Build your **own personal project** on blockchain using these technologies.

References

- Go Ethereum Documentation: https://geth.ethereum.org/
- Hyperledger Besu Documentation: https://besu.hyperledger.org/
- GitHub Repo of this Tutorial: https://github.com/mukherjeearnab/ethereum-tutorial

Thank you!