Healthcare Application using Ethereum Blockchainhain Installation Guide

P. V. Vineeth Kumar, Student of
Electrical Engineering
IIT Hyderabad

Dr.GVV Sharma, Faculty in
Department of Electrical Engineering
IIT Hyderabad

***Abstract*—This manual provides a step-by-step procedure for deploying a web application into the private Ethereum blockchain network by installing Ethereum geth client and interacting with the Ethereum blockchain network. It also discuss smart contracts and their deployment.**

I. INTRODUCTION

This case study explores the use of blockchain enabled technology to achieve distributed and trustworthy transaction handling between two ethereum nodes with a User Interface as the medium.

II. SYSTEM CONFIGURATION

A. Smart Contract Platform

The framework discussed here is based on the Ethereum Smart Contract Platform. At its simplest, Ethereum is an open software platform based on blockchain technology that enables developers to build and deploy decentralized applications.
Like Bitcoin, Ethereum is a distributed public blockchain network. Although there are some significant technical differences between the two, the most important distinction to note is that Bitcoin and Ethereum differ substantially in purpose and capability. Bitcoin offers one particular application of blockchain technology, a peer to peer electronic cash system that enables online Bitcoin payments. While the Bitcoin blockchain is used to track ownership of digital currency (bitcoins), the Ethereum blockchain focuses on running the programming code of any decentralized application.

B. What is Smart Contract?

Smart contract is just a phrase used to describe computer code that can facilitate the exchange of money, content, property, shares, or anything of value. When running on the blockchain a smart contract becomes like a self-operating computer program that automatically executes when specific conditions are met. Because smart contracts run on the blockchain, they run exactly as programmed without any possibility of censorship, downtime, fraud or third party interference.

III. HARDWARE AND SOFTWARE

We considered a case with two personal computers. One of the PC is a Vendor Node and another one is the passive node. We can use both of them as miners or we can restrict one node as non-miner.

On each device, a geth client(a command line interface implemented in the Go language was installed to transform them into an Ethereum node. With the geth clients, we created an ethereum account for each node and configured these networks to form a private blockchain network.In this network, Vendor Node plays the role of miner and the Passive Node is the lightweight ethereum node that play the function of sending transactions.

A. INSTALLING ETHEREUM CLIENT

The setup of the Vendor Node and Passive Node requires the following hardware:

- A Personal Computer with atleast 1 TB hard disk
- A LAN cable

Open the terminal and type the following commands to install an Ethereum client(geth-Go Ethereum)

- $ sudo apt-get install software-properties-common
- $ sudo add-apt-repository -y ppa:ethereum/ethereum
- $ sudo apt-get update

- $ sudo apt-get install ethereum

  Check the version of Geth:
  pi$ geth version

B. SET UP A PRIVATE CHAIN:MINER NODES

Our private chain needs miners in order to validate and propagate blocks of transactions within the blockchain. Miners will also be used to generate ether to pay for the gas required to process transactions on the Ethereum blockchain.
The requirements for each node to join the same private blockchain:

1) Each node will use a distinct data directory to store the database and the wallet.
2) Each node must initialize a blockchain based on the same genesis file.
3) Each node must join the same network id.
4) The port numbers must be different if different nodes are installed on the same computer.

- Create the datadir folder
  computer $ mkdir ∼/VendorNode/ miner

- Create the genesis file
  The Genesis Block is also known as Block Zero or Block 0. It is the ancestor that every Blockchain networks block can trace its origin back to. In simpler words, a genesis block is the first block of the Blockchain. It defines initial parameters of the Blockchain like level of difficulty, consensus algorithm etc. to mine blocks. It is the only block in the Blockchain that doesnt refer to previous block.

  Create a text file under ∼**/VendorNode**, called **genesis.json**, with the content shown in Fig:1

```
{
 "nonce": "0x0000000000000042",
 "mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "difficulty": "0x400",
 "alloc": {},
 "coinbase": "0x0000000000000000000000000000000000000000",
 "timestamp": "0x00",
 "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "extraData": "0x436861696e536b696c6c732047656e6573697320426c6f636b",
 "gasLimit": "0xffffffff",
 "config": {
    "chainId": 42,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
 }
}
```

Fig. 1. genesis.json

1) **Initialize miner**
   Use the following commands to create the blockchain for the Vendor Node miner.
   Move to the folder 'VendorNode' just created in the terminal. Then type the following command.
   geth –datadir ∼/VendorNode/miner init genesis.json

   If you list the content of the miner folder, you will notice the following sub-folders:

   - **geth**: contains the database of your private blockchain (chaindata)
   - **keystore**: location of your wallet used to store the accounts that you will create on this node.

2) **Creating Accounts**
   The account will receive all ethers created by the miner in the private blockchain. Type the following command to create a new account
   geth –datadir ∼/VendorNode/miner account new

   To list all accounts of your node, use the following command:
   geth –datadir ∼/VendorNode/miner account list

3) **Prepare the Miner Nodes**
   Miner: setup
   Lets start by creating a file that will contain the password for our default account, which is the first account we created for each miner. Create a password.sec file under ∼/VendorNode/miner/ that contains the password you configured for the first account on miner, in plain text.
   To start the miner, we will require to run the following command:

   geth –identity "miner" –networkid

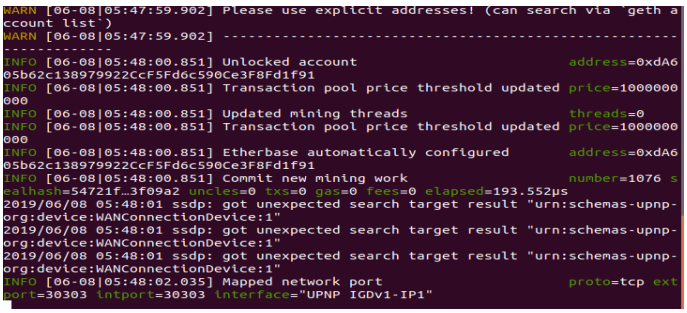

Fig. 2. Starting miner1

42 –datadir ”∼/VendorNode/miner” –nodiscover –mine –rpc –rpcport ”8001” –port ”30303” –unlock 0 –password ∼/VendorNode/miner/password.sec –ipcpath ”∼/.ethereum/geth.ipc”

You can observe the terminal looks like as shown in Fig.2

The meaning of the main parameters is the following:

- **identity:** name of our node
- **networkid:** this network identifier is an arbitrary value that will be used to pair all nodes of the same network. This value must be different from 0 to 3 (already used by the live chains)
- **datadir:** folder where our private blockchain stores its data
- **rpc and rpcport:** enabling HTTP-RPC server and giving its listening port number
- **port:** network listening port number, on which nodes connect to one another to spread new transactions and blocks
- **nodiscover:** disable the discovery mechanism (we will pair our nodes later)
- **mine:** mine ethers and transactions
- **unlock:** id of the default account
- **password:** path to the file containing the password of the default account
- **ipcpath:** path where to store the filename for IPC socket/pipe

To start **Mist** browser in private mode, after downloading mist version, run the following command.
./mist –networkid 42 –rpc ∼/.ethereum/geth.ipc

Note that the ipcpath is the path where you saved your mist application.

4) **Miner’s JavaScript console**
Manage your miner using the javascript console.
This console needs to be attached to a running instance of Geth. Open a new terminal session and type:
geth attach ∼/.ethereum/geth.ipc
This ipc path is shown in the terminal after you run the miner.

Repeat the same procedure for the Passive Node on the same or other PC.
If you are using the same PC, neglect the ipcpath for the second miner node. It will automatically take **./miner/geth.ipc** as the default ipc path.

5) Check Balances of the created accounts
All these functions are performed in the javascript console.
   - Get the default account with **eth.coinbase**
   - List the accounts with **eth.accounts**:
   - To start mining process : **miner.start()**
   - To stop mining process : **miner.stop()**
   - To view balance:
     eth.getBalance(eth.accounts[0])
   - To send ethers from one account to another:
     eth.sendTransaction(from: eth.accounts[0], to: eth.accounts[1], value: web3.toWei(10, ”ether”))

C. SYNCHRONISING THE CHAIN

Here we have to synchronise the network we have created.This is important if we want the transaction to be reflected in the nodes involved.Else they are similar to separate entities.

1) Obtain the ip adrresses of the Vendor and Passive nodes.(Depending on whether its wired or wireless).
2) Start the nodes(both Vendor and the Passive nodes in laptop.Obtain the node information by opening their **javascript console** and typing the following command:

admin.nodeInfo.enode

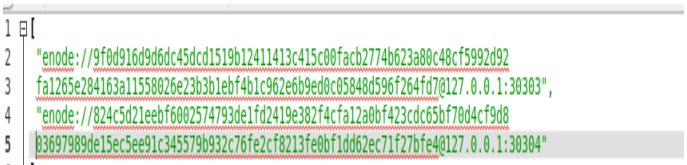

Fig. 3. static-nodes.json

3) Create a file **static-nodes.json**.It contains the node information you just extracted.Replace the placeholder [:] with the IP address of our computer. The last part (?discport=0) has been removed.Each node information is separated by a comma character which can be observed in Fig: 3

   It must be stored under
   1. ∼/VendorNode/miner
   2. ∼/PassiveNode/miner

   All folders should contain the same file.

4) Restart the nodes to start the synchronising process.You can see a line in your terminal **Block synchronising started**.
5) Checking the synchronisation Check from PassiveNode:
   Open the Javascript console. Type **admin.peers** in the console and observe the results.
   We can see that our node is paired to VendorNode miner identified by its IP address and its port number(30303). Similary try this on other node, to check if its connected.

6) Try sending ethers from one node to another to understand the working.
   eth.sendTransaction({from: eth.accounts[0], to: "0xae3ab39b3ebc425289dad620aece197 a4a3f8940", value: web3.toWei(2, "ether")})

   If you have more than one account, you have to unlock those accounts to be able to send transactions to it.

   personal.unlockAccount(eth.accounts[1], 'type your password')

D. CREATING AND DEPLOYIND SMART CONTRACTS

We create a simple contract called 'MedRec'. This contract will hold an associative array of unique integers and Patient objects. Here, the unique integer we considered is Aadhar Number.

1) Installing Truffle A developmental framework for Ethereum, we use truffle to create and deploy smart contract.
   npm install -g truffle
   Note: Install **node-js** and **npm** before the installation of **truffle** framework if you didn't installed earlier.
2) Installing Node.js from NodeSource repository
   - sudo apt-get install curl
   - $ curl -sL https://deb.nodesource.com /setup_10.x | sudo bash -
   - $ sudo apt-get update
   - $ sudo apt install nodejs

   Check the version of node-js and npm:
   pi$ node -v
   pi$ npm -v

3) Create the directory that will host your project.
   mkdir ∼/VendorNode/Projects/healthcare
   Move to this directory using cd command and type:
   truffle init
4) Create a file 'MedRec.sol' in the **contracts** directory and paste the code shown in the Fig:4

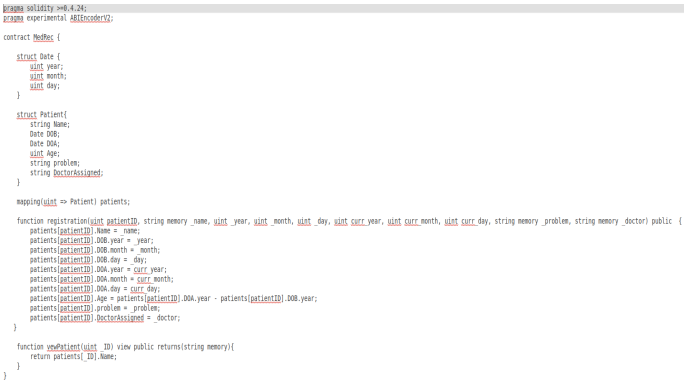

Fig. 4. MedRec.sol

5) Go to the **migrations** directory and create a file **2_deploy_contracts.js** with the content shown in Fig:5

```
var MedRec = artifacts.require("./MedRec.sol");

module.exports = function(deployer) {
   deployer.deploy(MedRec);
};
```

Fig. 5. 2_deploy_contract.js

6) create a file **truffle-config.js** or edit if already present and set the network conditions as shown in Fig:6

```
module.exports = {
   networks: {
    development: {
        host: 'localhost',
        port: 8001,
        network_id: "*",
        gas: 4600000
    }
   },
   solc: {
     optimizer: {
        enabled: true,
        runs: 200
     }
   }
}
```

Fig. 6. truffle-config.js

7) Start your miners before proceeding to the deploying process.
8) **$ cd ∼/VendorNode/Projects/healthcare**
   $ truffle compile
   $ truffle migrate –reset
   Now our SmartToken contract is deployed onto our private Ethereum chain and it is ready to receive calls.
9) Identifying your Contract
   We have to retrieve two elements about our deployed contract:its address and its ABI (Application Binary Interface).
10) Open truffle console: **truffle console**
11) Obtain contract address: **MedRec.address**
12) obtain ABI: **JSON.stringify(MedRec.abi)**
13) Move the **∼/Mistt** and start mist using the command:
    ./mist –networkid 42 –rpc ∼/.ethereum/geth.ipc Make sure it opens in your private network and not to the main-net.
14) Open **Watch Contract**. Fill in the smart contract address and ABI. Now the smart contract is available on Mist and you can easily interact with it.
15) You can copy the smart contract address and ABI on the mist browser running on the PassiveNode.

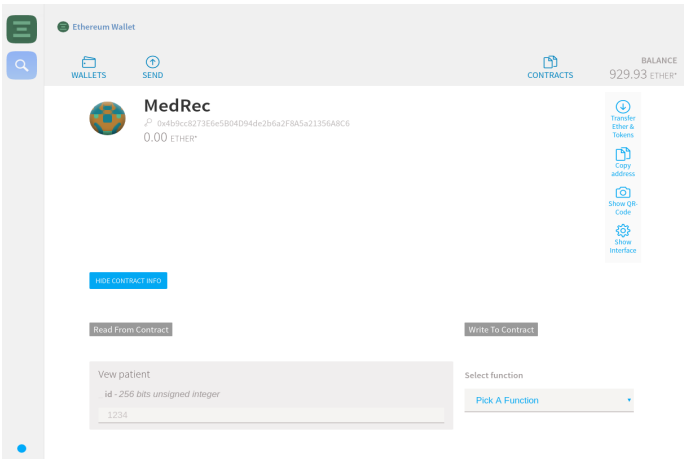

Fig. 7. Mist Browser Screen

E. CREATING THE CLIENT-SIDE CODE

1) Create a file **package.json** under healthcare directory with the code shown in Fig:7

```
{
  "name": "healthcare",
  "version": "1.0.0",
  "description": "Blockchain Healthcare Powered By Ethereum",
  "main": "truffle-config.js",
  "directories": {
    "test": "test"
  },
  "scripts": {
    "dev": "lite-server",
    "test": "echo \"Error: no test specified\" && sexit 1"
  },
  "author": "vineeth",
  "license": "ISC",
  "devDependencies": {
    "bootstrap": "4.1.3",
    "chai": "^4.1.2",
    "chai-as-promised": "^7.1.1",
    "chai-bignumber": "^2.0.2",
    "lite-server": "^2.3.0",
    "nodemon": "^1.17.3",
    "truffle": "5.0.2",
    "truffle-contract": "3.0.6"
  }
}
```

Fig. 8. package.json

2) **$ npm install**
3) Create files **src/index.html, src/app.js**
4) Create a file named **bs-config.json**
   We are using **lite-server** to serve all of the project files for the client side. We'll need to tell **lite-server** where all these files are located. We can do this by updating the browsersync configuration for **lite-server** inside the **bs-config.json** file. Paste this configuration into your project file:

```
{
  "server": {
    "baseDir": [
      "./src",
      "./build/contracts"
    ],
    "routes": {
      "/vendor": "./node_modules"
    }
  }
}
```

Fig. 9. bs-config.json

This configuration tells lite-server to expose all the files in the src and build/contracts directories to the root of our web server. It also adds an alias for any files in the node_modules directory to appear in the vendor route. This will allow us to pull in any project dependencies like bootstrap into the client side with the vendor route, which we'll see momentarily.

5) Paste the following code from github in index.html
6) Paste the following code from github in app.js
   - **loadWeb3():** web3.js is a JavaScript library that allows our client-side application to talk to the blockchain. We configure web3 here. This is default web3 configuration specified by Metamask.
   - **loadContract()** This is where we load the smart contract data from the blockchain. We create a JavaScript representation of the smart conract wit the Truffle Contract library. Then we load the smart contract data with web3.
   - **register()** This is where we retrieve the data from the webpage and send it to the solidity function **registration(),** which will allow us to write the data to the blockchain.
7) **$ npm run dev**
   Now the webpage is loaded. But, you will see errors in the browser console.

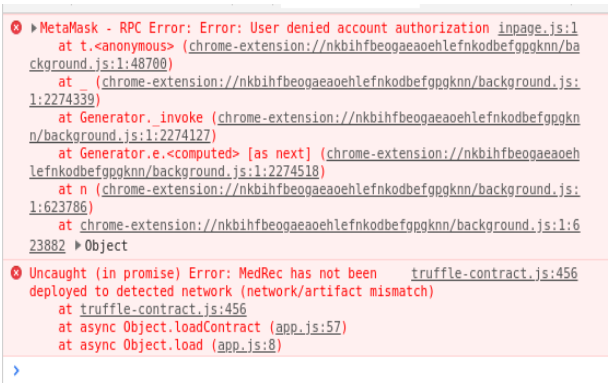

Fig. 10. Errors in the browser console

That's because, you need to connect to one of the accounts that you have created before with the metamask extension. You can get the metamask extension from the link: **Metamask**

8) Now, we need to connect to the account and the hosted network **(http://127.0.0.1:8001)** with the metamask, where, 127.0.0.1 is the **localhost** domain and **8001** is the port that we configured in **truffle-config.json** file.
9) The private key to the account can be obtained by using the python code as shown in the

figure Fig. 10.

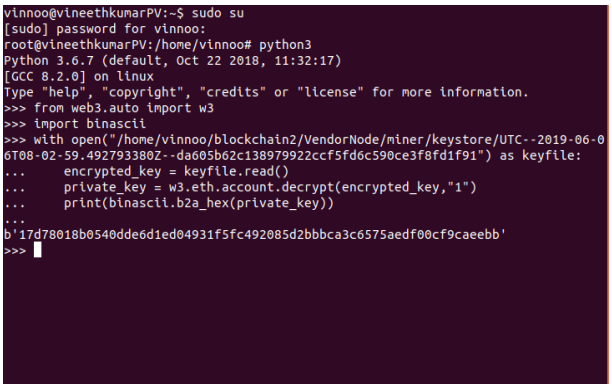

Fig. 11. Private Key Extraction

Now, create a custom rpc with new rpcurl as **(http://127.0.0.1:8001)**

10) Reload the browser to see the account loaded at top right corner of the page and the errors in the browser console subside.

F. WORKING

1) Enter the details on the webpage and click on submit button. Then, you will receive a pop-up, asking for confirmation.
2) Click confirm.
3) Start the mining process for the transaction to get confirmed.
4) After confirmation, you can then confirm the process using **Mist** browser.
   ./mist –networkid 42 –rpc ∼/.ethereum/geth.ipc
5) Enter the Aadhar number you entered before, in the VewPatiet function. It will return the Name of the patient.

G. CONCLUSIONS

You can test your applications using Mist browser by deploying your own Dapps(Smart contracts) on your private ethereum nodes.

REFERENCES

[1] Mastering Blockchain: Distributed ledger Technology, decentralization and smart contracts (1^{st} Edition) by Imran Bashir
[2] http://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/.
[3] https://bitfalls.com/2018/03/25/run-private-ethereum-blockchain/.
[4] https://github.com/trufflesuite/truffle/issues/305