The Chinese University of Hong Kong

Department of Information Engineering

FTEC5520– Applied Blockchain & Cryptocurrency

Lab1 Exercise

Ethereum Infrastructure Setup & Practice – Part2

Task 6 A simple Smart Contract demo

Smart Contract usually need to use transaction of blockchain to realize the contract which means the account should have some ethers at least. However, mining in blockchain network usually cost a long time, to make sure everyone can take this step even without finishing the previous, we recommend you use an Ethereum development tool working as blockchain simulator named Gnache. Here we will use the CLI version.

To initialize a new blockchain network, we have to create a new folder to hold it instead of using the ~/blockchain again.

```
cd && mkdir voting_dapp
```

```
cd voting_dapp
```

Install nodejs in this directory:

```
curl -fsL https://deb.nodesource.com/setup_12.x | sudo -E bash -
```

```
sudo apt-get install -y nodejs
```

Use node -v and npm -v to verfiy the installation:

```
ubuntu@ip-172-31-21-150:~$ node -v
v12.11.1
ubuntu@ip-172-31-21-150:~$ npm -v
6.11.3
ubuntu@ip-172-31-21-150:~$
```

Download the ganache-cli in ~/voting_dapp in current directory:

```
npm install ganache-cli
```

Run the ganache-cli command to start a blockchain automatically even without genesis:

```
node_modules/.bin/ganache-cli
```

```
ubuntu@ip-172-31-21-150:~/voting_dapp$ node_modules/.bin/ganache-cli
Ganache CLI v6.7.0 (ganache-core: 2.8.0)

Available Accounts
==================
(0) 0x5eC7798B39BdB46627CCC8fdE761d513F5151442 (100 ETH)
(1) 0xBFe1aA2cBE59512Eb23D281F1a78345110506800 (100 ETH)
(2) 0x2BD66b2b40F2E35BEa786D423Bd6F11E7a3D1F58 (100 ETH)
(3) 0x08949Db9d53875Cf74D5a7F581b88357ca46c0b3 (100 ETH)
(4) 0x23882F49952c237A597B990b4A94506015dc5D9a (100 ETH)
(5) 0x0E707edd67B8D7a55c3eaa1c99EcB55eA36628C2 (100 ETH)
(6) 0xCC79e8e85163789aD928b01211367E5040611223 (100 ETH)
(7) 0xD0B9Ea0268ce7b71C1649a53e06EFbdd499A484B (100 ETH)
(8) 0xFdB4c4C9ef3Cd1E1FD13bC1898d3145e42609fb1 (100 ETH)
(9) 0xE8E283Ea08a7E76Dbf76d20A39A36f6eB1EF42bc (100 ETH)

Private Keys
==================
(0) 0x5059a02b78fd0ae5d933de7ce980dd4dab9e2bb8a7e3b81d4cdde6c0c24c1540
(1) 0x16cbce729f936556d90b0a39c4a56cfbbb46ba787abe99c5f5f0adc935731295
(2) 0x5c82c46637c00579cd0350e5eb2e3e9114237bcbc544793a05ab5049a230ea77
(3) 0x4e5a187940aee61d2bda4249bdcc41a1c70d55a8f514871a7edbbaf8776cf047
(4) 0xb1a52b845635596da9e965f3e828dbe798b9538d711a4a0ccb372b5cdc8d31b4
(5) 0x2aae2fd6ab39cb7a30e181de7591353ae9660623aa904588c6dfd65b7f56d0e2
(6) 0x1ba2a5d7488b7e7263999e63a6e640706966aa56447106627e8c4dbe7b3cbcaa
(7) 0xad0e471a24b4d6f0e0f8911e83eee1b875ff668c3c09db99885f93b8a49f6c53
(8) 0x4125e16902bd51fd16df0139cf4d92ecb55f807170d7f49355dadb0a4d975522
(9) 0x1d5c0614fbc888048bfcc1f3f7e222c585246fd484edd36b75c166e8fff8f02b

HD Wallet
==================
Mnemonic:      census observe rich must mother work fog possible goose flag envelope simple
Base HD Path:  m/44'/60'/0'/0/{account_index}

Gas Price
==================
20000000000

Gas Limit
==================
6721975

Listening on 127.0.0.1:8545
```

If you get the return, that means you have got 10 accounts with 100 ether each generated automatically and a blockchain initialized. The accounts are equally to these you created in the previous steps.

Please leave the ganache-cli running and **open a new terminal** connected to these Linux VM and do below steps.

Task 6.1 Development of Smart Contract

We are going to use the **Solidity** programming language especially designed for Ethereum to write a simple sample contract. If you are familiar with Object-Oriented Programming (OOP) such as C++, Java or Python, learning to write solidity contracts should be a breeze.

We have given you the source code of the simple contract (think of contract as a class in your favorite OOP language) called **Voting** with a constructor which initializes an array of candidates. It includes 2 methods: one to return the total votes a candidate has received and another method to increment vote count for a candidate.

Note: The constructor is invoked once and only once when you deploy the contract to the blockchain. Unlike in the web world where every deployment of your code overwrites the old code, deployed code in the blockchain is immutable, i.e. if you update your contract and deploy again, the old contract will still be in the blockchain untouched along with all the data stored in it, the new deployment will create a new instance of the contract.

1. Write Solidity source code:

   To simplify the procedure, we don't introduce any details about Solidity grmamer here, but only give you a sample code for fast demo. For more details about solidity please refer to Solidity doc, a popular Solidity IDE Remix, more tutroials on YouTube

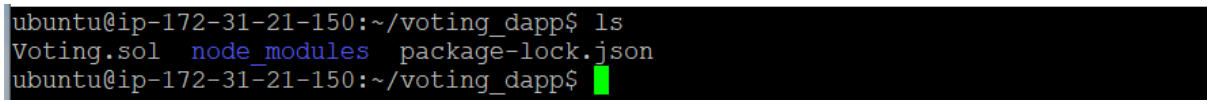
   Below is the Voting contract code with comment explanation. Please create a .sol file named **Voting.sol** using these commands and copy the contract code to the Voting.sol file in your Linux blockchain directory

   ```
   touch Voting.sol
   ```

   ```
   vim Voting.sol
   ```

   Press "i" to enter the editor mode and copy the source code above to Voting.sol

   Then press ":wq" to exit.


```
pragma solidity >=0.4.0 <0.6.0;
// We have to specify what version of compiler this code will compile with

contract Voting {
  /* mapping field below is equivalent to an associative array or hash.
  The key of the mapping is candidate name stored as type bytes32 and value is
  an unsigned integer to store the vote count
  */

  mapping (bytes32 => uint256) public votesReceived;

  /* Solidity doesn't let you pass in an array of strings in the constructor (yet).
  We will use an array of bytes32 instead to store the list of candidates
  */

  bytes32[] public candidateList;

  /* This is the constructor which will be called once when you
  deploy the contract to the blockchain. When we deploy the contract,
  we will pass an array of candidates who will be contesting in the election
  */
  constructor(bytes32[] memory candidateNames) public {
    candidateList = candidateNames;
  }

  // This function returns the total votes a candidate has received so far
  function totalVotesFor(bytes32 candidate) view public returns (uint256) {
    require(validCandidate(candidate));
    return votesReceived[candidate];
  }

  // This function increments the vote count for the specified candidate. This
  // is equivalent to casting a vote
  function voteForCandidate(bytes32 candidate) public {
    require(validCandidate(candidate));
    votesReceived[candidate] += 1;
  }

  function validCandidate(bytes32 candidate) view public returns (bool) {
    for(uint i = 0; i < candidateList.length; i++) {
      if (candidateList[i] == candidate) {
        return true;
      }
    }
    return false;
  }
}
~
```

2. Compile the source code:

   To deploy the Voting contract, we need to compile the source code firstly.

   Install the CLI solidity compiler called **solc**

   `npm install solc`

   Install the web3 for further interaction.

   `npm install web3`

   Run the solc to compile Voting.sol from source code to **.bin** file and **.abi** file.

```
node_modules/.bin/solcjs --bin --abi Voting.sol
```

Show the compiled results of Voting.sol using ls and your will see two new files:

```
ls
```

```
ubuntu@ip-172-31-21-150:~/voting_dapp$ ls
Voting.sol  node_modules  package-lock.json
ubuntu@ip-172-31-21-150:~/voting_dapp$ node_modules/.bin/solcjs --bin --abi Voting.sol
ubuntu@ip-172-31-21-150:~/voting_dapp$ ls
Voting.sol  Voting_sol_Voting.abi  Voting_sol_Voting.bin  node_modules  package-lock.json
ubuntu@ip-172-31-21-150:~/voting_dapp$
```

When you compile the code successfully using the command above, the compiler outputs 2 files that are important to understand:

- **Voting_sol_Voting.bin**: This is the bytecode you get when the source code in Voting.sol is compiled. This is the code which will be deployed to the blockchain.
- **Voting_sol_Voting.abi**: This is an interface or template of the contract (called abi) which tells the contract user what methods are available in the contract. Whenever you interact with the contract in the future, you will need this abi definition. You can read more details about **ABI** here

Task 6.2 Deployment of Smart Contract

web3js is a library which lets you interact with the blockchain through **RPC**. We will use that library to deploy our application and interact with it.

First, run the commands below in your terminal to get into the node console.

```
node
```

```
ubuntu@ip-172-31-21-150:~/voting_dapp$ node
Welcome to Node.js v12.11.1.
Type ".help" for more information.
>
```

Note: All the code snippets below need to be typed in the node console.

Initialize the web3 object.

```
Web3 = require('web3')
```

```
> Web3 = require('web3')
[Function: Web3] {
  version: '1.2.1',
  utils: {
    _fireError: [Function: _fireError],
    _jsonInterfaceMethodToString: [Function: _jsonInterfaceMethodToString],
    _flattenTypes: [Function: _flattenTypes],
    randomHex: [Function: randomHex],
    _: [Function: _] {
```

...

```
  modules: {
    Eth: [Function: Eth] { givenProvider: null, providers: [Object] },
    Net: [Function: Net] { givenProvider: null, providers: [Object] },
    Personal: [Function: Personal] { givenProvider: null, providers: [Object] },
    Shh: [Function: Shh] { givenProvider: null, providers: [Object] },
    Bzz: [Function: Bzz] { givenProvider: null }
  },
  givenProvider: null,
  providers: {
    WebsocketProvider: [Function: WebsocketProvider],
    HttpProvider: [Function: HttpProvider],
    IpcProvider: [Function: IpcProvider]
  }
}
> 
```

```
web3 = new Web3("http://localhost:8545")
```

```
> web3 = new Web3("http://localhost:8545")
Web3 {
  currentProvider: [Getter/Setter],
  _requestManager: RequestManager {
    provider: HttpProvider {
      host: 'http://localhost:8545',
      httpAgent: [Agent],
      timeout: 0,
      headers: undefined,
      connected: false
    },
    providers: {
      WebsocketProvider: [Function: WebsocketProvider],
      HttpProvider: [Function: HttpProvider],
      IpcProvider: [Function: IpcProvider]
    },
    subscriptions: {}
  },
  givenProvider: null,
  providers: {
    WebsocketProvider: [Function: WebsocketProvider],
    HttpProvider: [Function: HttpProvider],
    IpcProvider: [Function: IpcProvider]
  },
  _provider: HttpProvider {
    host: 'http://localhost:8545',
```

……

```
    unsubscribe: [Function: send] {
      method: [Method],
      request: [Function: bound ],
      call: 'shh_unsubscribe'
    }
  },
  bzz: Bzz {
    givenProvider: null,
    currentProvider: null,
    isAvailable: [Function],
    upload: [Function],
    download: [Function]
  }
}
> 
```

To make sure web3 object is initialized and can communicate with the blockchain, let's query all the accounts in the blockchain. You should see a result with many accounts like below:

```
web3.eth.getAccounts(console.log)
```

```
> web3.eth.getAccounts(console.log)
Promise { <pending> }
> null [
  '0x5eC7798B39BdB46627CCC8fdE761d513F5151442',
  '0xBFe1aA2cBE59512Eb23D281F1a78345110506800',
  '0x2BD66b2b40F2E35BEa786D423Bd6F11E7a3D1F58',
  '0x08949Db9d53875Cf74D5a7F581b88357ca46c0b3',
  '0x23882F49952c237A597B990b4A94506015dc5D9a',
  '0x0E707edd67B8D7a55c3eaa1c99EcB55eA36628C2',
  '0xCC79e8e85163789aD928b01211367E5040611223',
  '0xD0B9Ea0268ce7b71C1649a53e06EFbdd499A484B',
  '0xFdB4c4C9ef3Cd1E1FD13bC1898d3145e42609fb1',
  '0xE8E283Ea08a7E76Dbf76d20A39A36f6eB1EF42bc'
]
```

To load the bytecode(bin) and Application Binary Interface (abi) from the file system into a string like below:

```
bytecode = fs.readFileSync('Voting_sol_Voting.bin').toString()
```

```
> bytecode = fs.readFileSync('Voting_sol_Voting.bin').toString()
'6080604052348015610011576000600fd5b506040516104c23803806104c2833981810160405260208110156100355
760006000fd5b810190808051604051939291908464010000000082111561005657600060006000fd5b83820191506020820
18581111561006d5760006000fd5b82518660208202830111640100000000821117156100 8b5760006000fd5b8083526
020830192505050908051906020019060200280838360005b838110156100c35780820151818401525b6020810190506
100a7565b505050509050019604052602001505050 5b80600160005090805190602001906100ed9291906100f5565b505
b50610176565b828054828255906000526020600020908101928215610 13a579160200282015b828111561013957825
18260005090600019169055916020019190600101906101155565b5b509050610147919061014b565b5090565b6101739
190610155565b8082111561016f576000818150600090555060010161015565b5090565b90565b61033d80610185600
0396000f3fe6080604052348015610011576000600fd5b506004361061005c5760003560e01c80632f265cf71461006
2578063392e6678146100a9578063 7021939f146100f4578063b13c744b1461013b578063cc9ab267146101865761005
c565b60006000fd5b6100936004803603602081101561007957600060 00fd5b810190808035600019169060200190929
1905050506101b9565b604051808281526020019150506040518091 0390f35b6100da600480360360208110156100c05
760006000fd5b810190808035600019169060200190929190505050 61020456 5b6040518082151515158152602001915
0506040518091 0390f35b610125600480360360208110156101 0b5760006000fd5b810190808035600019169060200019
0929190505050610275565b604051808281526020019150506040518091 0390f35b610168600480360360208110156101
1525760006000fd5b810190808035906020019092919050505061 0290565b604051808260001916600019168152602 00
191505060405180910390f35b6101b760048036036020811015610 19d5760006000fd5b810190808035600019169060 2
00190929190505050610 2b8565b005b60006101ca8261020463ffffffff16565b15156101d65760006000fd5b6000600
0506000836000191660001916815260200190815260200160002060005054905061 01ff565b919050565b60006000600
090505b600160005080549050811015610266578260001916600160005082815481101515610 23457fe5b90600052602
0600020900160005b505460001916141561025857600191505061027056 5b5b8080600101915050610 20d565b5060009
050610270565b919050565b60006000506020528060005260406000206000915090505481565b6001600050818154811
015156102a257fe5b906000526020600020900160005b915090505481565b6102c78161020463ffffffff16565b15156
102d35760006000fd5b600160006000506000836000191660001916815260200190815260200160002060008282825 05
401925050819090905550 5b5056fea265627a7a723158206f5d0dcbfdcab51725af16cf7c8989956dee672e8ebe8584d
827d990e9e0c11564736f6c634300050c0032'
>
```

```
abi=JSON.parse(fs.readFileSync('Voting_sol_Voting.abi').toString
())
```

```
> abi = JSON.parse(fs.readFileSync('Voting_sol_Voting.abi').toString())
[
  {
    inputs: [ [Object] ],
    payable: false,
    stateMutability: 'nonpayable',
    type: 'constructor'
  },
  {
    constant: true,
    inputs: [ [Object] ],
    name: 'candidateList',
    outputs: [ [Object] ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  },
  {
    constant: true,
    inputs: [ [Object] ],
    name: 'totalVotesFor',
    outputs: [ [Object] ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  },
  {
    constant: true,
    inputs: [ [Object] ],
    name: 'validCandidate',
    outputs: [ [Object] ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  },
  {
    constant: false,
    inputs: [ [Object] ],
    name: 'voteForCandidate',
    outputs: [],
    payable: false,
    stateMutability: 'nonpayable',
    type: 'function'
  },
  {
    constant: true,
    inputs: [ [Object] ],
    name: 'votesReceived',
    outputs: [ [Object] ],
    payable: false,
    stateMutability: 'view',
    type: 'function'
  }
]
>
```

Now, let's deploy the Voting contract.

1. Create a contract object name **deployedContract** which is used to deploy and ini tiate contracts in blockchain

   ```
   deployedContract = new web3.eth.Contract(abi)
   ```

```
> deployedContract = new web3.eth.Contract(abi)
Contract {
  currentProvider: [Getter/Setter],
  _requestManager: RequestManager {
    provider: HttpProvider {
      host: 'http://localhost:8545',
      httpAgent: [Agent],
      timeout: 0,
      headers: undefined,
      connected: true
    },
    providers: {
      WebsocketProvider: [Function: WebsocketProvider],
      HttpProvider: [Function: HttpProvider],
      IpcProvider: [Function: IpcProvider]
    },
    subscriptions: {}
  },
  givenProvider: null,
```

.....

```
      constant: false,
      inputs: [Array],
      name: 'voteForCandidate',
      outputs: [],
      payable: false,
      stateMutability: 'nonpayable',
      type: 'function',
      signature: '0xcc9ab267'
    },
    {
      constant: true,
      inputs: [Array],
      name: 'votesReceived',
      outputs: [Array],
      payable: false,
      stateMutability: 'view',
      type: 'function',
      signature: '0x7021939f'
    }
  ]
}
> 
```

2. Create a list of three candidates:

```
listOfCandidates = ['Rama', 'Nick', 'Jose']
```

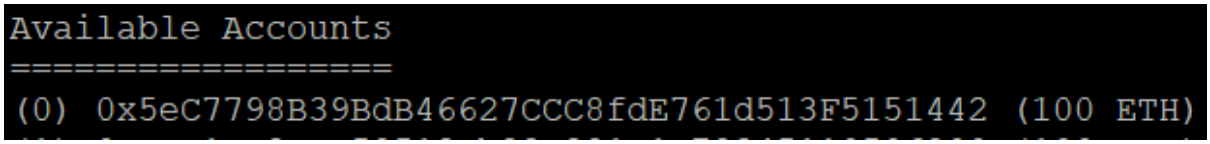
```
SyntaxError: Unexpected token '}'
> listOfCandidates = ['Rama', 'Nick', 'Jose']
[ 'Rama', 'Nick', 'Jose' ]
> 
```

3. Use deploy method with many parameters to deploy smart contract:

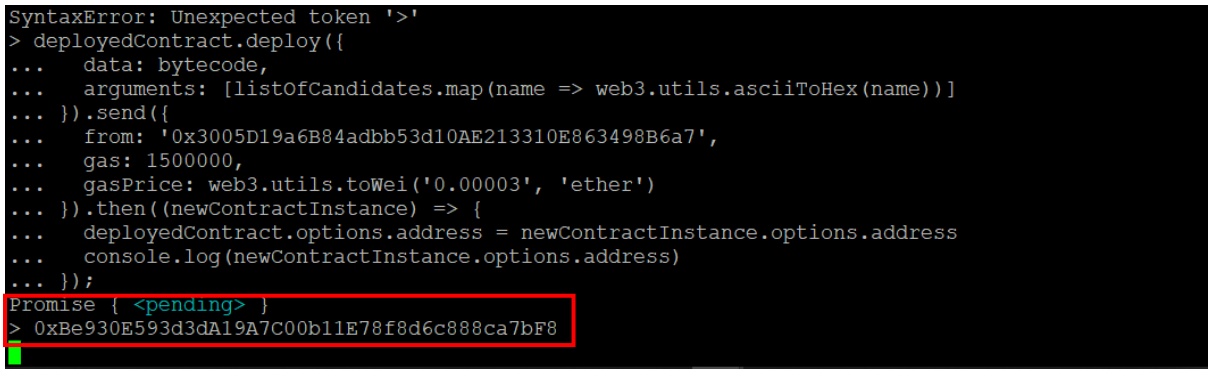
```
deployedContract.deploy({
        data: bytecode,
        arguments: [listOfCandidates.map(name =>
              web3.utils.asciiToHex(name))]
        }).send({
        from: '0xaC49bcD8A5f5288Da8DEFA7CeeDDe6702E745112',
        gas: 1500000,
        gasPrice: web3.utils.toWei('0.00003', 'ether')
```

```
}).then((newContractInstance) => {
deployedContract.options.address = newContractInstanc
e.options.address
console.log(newContractInstance.options.address)
});
```

Note: Here go back to the terminal running ganache-cli and copy one account address Use this account address to replace the yellow highlight part in above code.

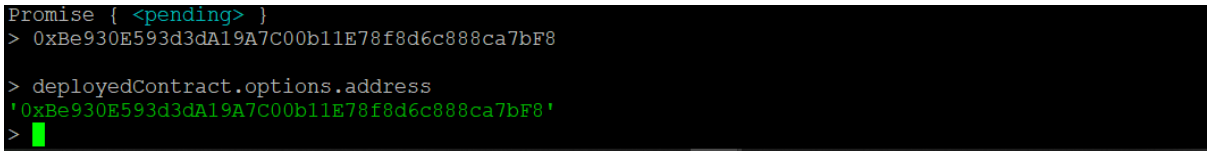


After a short time pending, you'll get a HASH value return which is also your contract address and means your smart contract has been deployed.



4. To verify it again, use this command and you will the same HASH value:

```
deployedContract.options.address
```



We use the web3 deploy function along with send to deploy the contract to the blockch ain. Let's see what are all the arguments we pass to deploy and send functions:

- **data:** This is the compiled bytecode which we deploy to the blockchain.
- **arguments**: These are the arguments we pass to the constructor of the contract. In our case we pass an array of candidate names. Note that we have to explicitly convert string to bytes32, that's why we call **web3.utils.asciiTo**Hex on each candidate name (using map function).
- **from**: The blockchain keep track of who deployed the contract. In this case, we are just picking the first account we get back from calling web3.eth.getAccounts to be the owner of this contract (who will deploy it to the blockchain). Remember that web3.eth.getAccounts returns an array of 10 test accounts ganache created when we started the test blockchain. In the live blockchain, you can't just use any

account. You own that account and unlock it before transacting. You are asked for a passphrase while creating an account and that is what you use to prove your ownership of that account. Ganache by default unlocks all the 10 accounts for convenience.

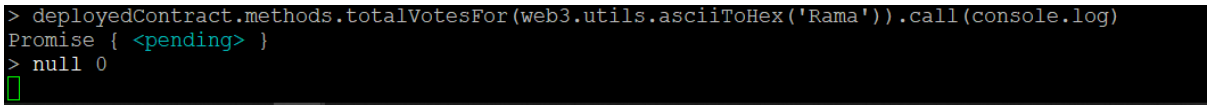
- **gas**: It costs money to interact with the blockchain. This money goes to miners who do all the work to include your code in the blockchain. You specify how much money you are willing to pay to get your code included in the blockchain and you do that by setting the value of 'gas'. The ether balance in your 'from' account will be used to buy gas. The price of gas is set by the network.
- **gasPrice**: Each unit of gas has a price associated with it. That is set in the gasPrice field.

We have now deployed the contract and have an instance of the contract (variable **deployedContract** above) which we can use to interact with the contract. There are hundreds of thousands of contracts deployed on the blockchain.

Task 6.3 Interact with the deployed contract in the NodeJS console:

1. To see the voting result of each candidate. Here we use "Rama" as example:

```
deployedContract.methods.totalVotesFor(web3.utils.asciiToHex('Ram
a')).call(console.log)
```



2. Vote for "Rama" using transaction:

```
deployedContract.methods.voteForCandidate(web3.utils.asciiToHex('
Rama')).send({from: 'YOUR ACCOUNT ADDRESS'}).then((f) =>
console.log(f))
```

```
> deployedContract.methods.voteForCandidate(web3.utils.asciiToHex('Rama')).send({from: '0x3005D1
9a6B84adbb53d10AE213310E863498B6a7'}).then((f) => console.log(f))
Promise { <pending> }
> {
  transactionHash: '0xf619c496ba87f928de08ad8b65cf76bcc60736666c2058558b503e7fcc870e29',
  transactionIndex: 0,
  blockHash: '0x7dcedff33169afd082704396e6fa7e2e977d98c0597fe9427d9c4c77cc9a4800',
  blockNumber: 2,
  from: '0x3005d19a6b84adbb53d10ae213310e863498b6a7',
  to: '0xbe930e593d3da19a7c00b11e78f8d6c888ca7bf8',
  gasUsed: 43184,
  cumulativeGasUsed: 43184,
  contractAddress: null,
  status: true,
  logsBloom: '0x00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000',
  v: '0x1c',
  r: '0x6997d19bcf8cf907a8b032de0aff8e50ab2776403b7481220bce3b6d5783e261',
  s: '0x53ebc3e596708314ffe8414fcc3e4330ddf1644094fbee7ce31935c4c308bead',
  events: {}
}

>
```

After a short time pending, you will get transaction block with a unique **transactionHash.**

3. Check the voting result of "Rama" again.

```
deployedContract.methods.totalVotesFor(web3.utils.asciiToHex('Ram
a')).call(console.log)
```

```
> deployedContract.methods.totalVotesFor(web3.utils.asciiToHex('Rama')).call(console.log)
Promise { <pending> }
> null 1
```

4. You can also see the transaction from the ganache-cli window:

```
Listening on 127.0.0.1:8545
eth_sendTransaction

  Transaction: 0x9edb7b38f9ab7875cc527341acd79327fcaf7189bf44ae4d38a65b1a2f0225f6
  Contract created: 0xbe930e593d3da19a7c00b11e78f8d6c888ca7bf8
  Gas usage: 377737
  Block Number: 1
  Block Time: Sun Oct 13 2019 09:12:10 GMT+0000 (Coordinated Universal Time)

eth_getTransactionReceipt
eth_getCode
eth_gasPrice
eth_sendTransaction

  Transaction: 0xf619c496ba87f928de08ad8b65cf76bcc60736666c2058558b503e7fcc870e29
  Gas usage: 43184
  Block Number: 2
  Block Time: Sun Oct 13 2019 09:14:33 GMT+0000 (Coordinated Universal Time)

eth_getTransactionReceipt
eth_call
```

You can observe that when you deployed the contract, there was the first block created. Next, when you vote, there will be a new block counted as the 2nd.

Question 3 – If you want to deploy the voting smart contract on the blockchain you crated in tast1~5, what should you do? If you want to deploy your smart contract on other types of Eth networks like main net, ***Rinkeby****,* ***modern test net****, etc, what should you do? (1 mark)*

If you can see the vote count of "Rama" increment from 0 to 1 and above result, you have successfully created your first smart contract. Congratulations!

Summary: Though the app we developed & deployed in this lab does looks simple and raw even without any GUI. If you want to explore more about how to build a GUI Web dApp and even with more nodes, please refer to this article: Two-Node Setup of a Private Ethereum on AWS with Contract Deployment, Full Stack Hello World Voting Ethereum Dapp Tutorial

Question 4 – Please list the main steps of multi-node Setup of a Private Ethereum on AWS with Contract Deployment by reading the below blogs.

1) How to setup a multi-node (**more than two nodes**) Ethereum blockchain network?
   Reference: https://blockgeeks.com/two-node-setup-of-a-private-ethereum/
2) How to deploy the sample Voting smart contract on it?
   Reference: https://blockgeeks.com/two-node-setup-of-a-private-ethereum-on-aws-with-contract-deployment-part-2/

Reference

a) **How to set up Ethereum blockchain:**
https://arctouch.com/blog/how-to-set-up-ethereum-blockchain/
b) **Setup private Ethereum blockchain in AWS**
https://medium.com/nxtplus/setup-private-ethereum-blockchain-in-aws-amazon-web-services-or-gcp-google-cloud-platform-abfaae779f6a
c) **Install Ethereum on Ubuntu 16.04**
https://retrospect.blog/install-ethereum-on-ubuntu-16-4-xenial/
d) **Setup private Ethereum blockchain and deploy your 1st Solidity Smart Contract**
https://medium.com/blockchainbistro/set-up-a-private-ethereum-blockchain-and-deploy-your-first-solidity-smart-contract-on-the-caa8334c343d
e) **Setting up and running a private Ethereum blockchain on ubuntu**
https://steemit.com/ethereum/@nphacker/setting-up-and-running-a-private-ethereum-blockchain-on-ubuntu
f) **Setup a fully synced blockchain node**
https://www.freecodecamp.org/news/ethereum-69-how-to-set-up-a-fully-synced-blockchain-node-in-10-mins-f6318d7aad40/
g) **Setupting up the Go Ethereum geth version in Ubuntu Linux**
https://medium.com/@priyalwalpita/setting-up-the-go-ethereum-geth-environment-in-ubuntu-linux-67c09706b42
h) **Two-nodes setup of private Ethereum on AWS with contract deployment**
https://blockgeeks.com/two-node-setup-of-a-private-ethereum/
https://blockgeeks.com/two-node-setup-of-a-private-ethereum-on-aws-with-contract-deployment-part-2/
i) **Aws blockchain template**
https://docs.aws.amazon.com/blockchain-templates/latest/developerguide/blockchain-templates-ethereum.html
j) **Ethereum for web developers**
https://medium.com/@mvmurthy/ethereum-for-web-developers-890be23d1d0c
k) **Full Stack Hello World Voting dApp tutorial part2** https://medium.com/@mvmurthy/full-stack-hello-world-voting-ethereum-dapp-tutorial-part-1-40d2d0d807c2