

Private Ethereum Network Configuration with IoT Devices

This document will describe how to set up a private Ethereum blockchain that will be composed of computers (miners) and several Raspberry PI 3 devices (nodes). The objective is to instruct how to setup a development environment to learn about blockchain principles, then develop and test your own smart contracts before deploying them to the main chain.

Through series of tutorials, you could integrate a RPi with a Smart Contract demo deployed on a private chain. The Smart Contract will be used to check if a user has enough tokens to use a service. The RPi will be used to visualize the status of the contract.

NB: “private Ethereum blockchain” in this document refers to a private instance of the Ethereum implementation, instead of “private blockchain” conception which are called permissioned blockchains.

Chapter 1: Environmental preliminary

Before installing Ethereum client and related tools, basic development environment is required to install several preliminary libs and tools.

A) Install Docker Engine and Docker Compose. For docker installation, please refer to <https://docs.docker.com/install/>

1) Install Docker:

```
$curl -sSL https://get.docker.com | sh
```

2) Adding your user to the "docker" group with something like:

```
$sudo usermod -aG docker pi
```

3) Install docker compose:

```
$sudo pip install docker-compose
```

B) Since Ethereum use Go as development language, install Go development environment is needed.

We do not suggest use: `sudo apt-get install golang` to install golang environment, please download go source package and install by command. All golang package can be found on <https://golang.org/dl/>

1) First of all, download go package and extract to */usr/local*

#For Raspberry pi:

```
$wget https://storage.googleapis.com/golang/go1.9.3.linux-armv6l.tar.gz
```

```
$sudo tar -C /usr/local -xzf go1.9.3.linux-armv6l.tar.gz
```

2) Install golang package from source code

#For Ubuntu

```
$wget https://storage.googleapis.com/golang/go1.9.3.linux-amd64.tar.gz
```

```
$sudo tar -C /usr/local -xzf go1.9.3.linux-amd64.tar.gz
```

3) Configure GOROOT and append to PATH

```
$sudo echo 'export GOROOT=/usr/local/go' >> ~/.bashrc  
$sudo echo 'export PATH=$PATH:$GOROOT/bin' >> ~/.bashrc
```

C) Reboot & Test by executing following command:

```
$go version  
$docker -v  
$docker-compose -v  
$pip -V  
$git --version
```

Chapter 2: Install Ethereum client on Raspberry PI

In this chapter, we will describe the steps to install Ethereum client on Raspberry PI 3 and transform RPi into an Ethereum node. Owing to low computation capacity, the RPi is intended to act as a node which does not carry out mining task and connected to private Ethereum blockchain.

Step 1 - Connect to your RPI

To connect remotely to your RPi through SSH, you have to know the IP address of your RPi device. Log on the RPi to install Ethereum client geth.

Step 2 - Install Geth from source code

As a first thing we start by updating our software:

```
$sudo apt-get update
```

- 1) download from official ethereum repository the Geth client

```
$git clone https://github.com/ethereum/go-ethereum.git
```

- 2) Ether go-ethereum:

```
$cd go-ethereum
```

- 3) build geth:

```
$make geth
```

This will take several minutes to be done.

- 4) Add go-ethereum executable bin path in ~/.bashrc:

```
$export GOETHEREUM=@full_local_path/go-ethereum/build
```

```
$export PATH=$PATH:$GOETHEREUM/bin
```

- 5) Check version:

```
$geth version
```

Step 3 - Run Geth

Start Geth using the following command:

```
$geth
```

Then press CTRL-C to stop the Ethereum server. (We do not synchronize with the live chain)

Summary

At this stage, Ethereum is installed on RPi and able to synchronize with the live chain (main net). Here are blockchain environment on local:

The default blockchain data (chaindata) is located: ~/.ethereum/chaindata

The accounts will be stored in a wallet located in this folder: ~/.ethereum/keystore

You can repeat these steps to setup Ethereum client on additional RPis.

Chapter 3: Install Ethereum client on Computer

In this chapter, we will describe the steps to install Ethereum client on computer and transform RPi into an Ethereum node. Installing Geth on a computer is quite straightforward. The installation instructions are available for different target platforms, including Linux, Windows and Mac. You can find them right: <https://github.com/ethereum/go-ethereum/wiki/Building-Ethereum>. We suggest install from geth source code.

Step 1 - Install Geth from source code

As a first thing we start by updating our software:

```
$sudo apt-get update
```

- 6) download from official ethereum repository the Geth client

```
$git clone https://github.com/ethereum/go-ethereum.git
```

- 7) Ether go-ethereum:

```
$cd go-ethereum
```

- 8) build geth:

```
$make geth
```

This will take several minutes to be done.

- 9) Add go-ethereum executable bin path in ~/.bashrc:

```
$export GOETHEREUM=@full_local_path/go-ethereum/build
```

```
$export PATH=$PATH:$GOETHEREUM/bin
```

- 10) Check version:

```
$geth version
```

Step 2 - Run Geth

Start Geth using the following command:

```
$geth
```

Then press CTRL-C to stop the Ethereum server. (We do not synchronize with the live chain)

Summary

At this stage, Ethereum is installed on your computer and able to synchronize with the live chain (mainnet).

The blockchain (chaindata) is located right here:

Mac: ~/Library/Ethereum/chaindata

Windows: %APPDATA%\Ethereum

Linux: ~/.ethereum

Your accounts will be stored in a wallet located in this folder:

Mac: ~/Library/Ethereum/chaindata/keystore

Windows: %APPDATA%\Ethereum\keystore

Linux: ~/.ethereum/keystore

Now your computer become an Ethereum node with geth client.

Chapter 4: Configure miners on the private chain

Private chain also needs miners 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. Note that this ether will only be usable within our private

blockchain. Unfortunately, the Raspberry pi is not powerful enough to be used as a miner. Thus the miners are only deployed on computer. In Ethereum network, at least two miners should be configured to maintain the network.

Before describing the steps to setup the miners, it is important to understand the requirements for each node to join the same private blockchain:

- Each node will use a distinct data directory to store the database and the wallet.
- Each node must initialize a blockchain based on the same genesis file.
- Each node must join the same network id different from the one reserved by Ethereum (0 to 3 are already reserved).
- The port numbers must be different if different nodes are installed on the same computer.

All the operations must be performed from your computer.

Step 1 - Create the datadir folder

When running a private blockchain, it is highly recommended to use a specific folder to store the data (database and wallet) of the private blockchain without impacting the folders used to store the data coming from the public blockchain.

From your computer, create the folder that will host your first miner:

```
$mkdir ~/MyChains/miner1
```

Repeat the operation for the second miner:

```
$mkdir ~/MyChains/miner2
```

Step 2 - Create the Genesis file

Each blockchain starts with a genesis block that is used to initialize the blockchain and defines the terms and conditions to join the network. Our genesis block is called “genesis.json” and is stored under “~/MyChains” folder.

Create a text file under ~/ MyChains, called genesis.json, with the following content:

```
{
  "nonce": "0x00000000000000042",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "difficulty": "0x400",
  "alloc": {},
  "coinbase": "0x000000000000000000000000000000000000000000000000",
  "timestamp": "0x00",
  "parentHash": "0x000000000000000000000000000000000000000000000000",
  "extraData": "0x436861696e536b696c6c732047656e6573697320426c66636b",
  "gasLimit": "0xffffffff",
  "config": {
    "chainId": 42,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
  }
}
```

Comment:

Update (24-May-2017): Since Geth 1.6, the genesis.json file has to provide a section “config” identifying the

network id (chainId) of your private network. Based on our tutorial, we will use the network Id “42”.

Update (25-Sep-2017): A comma was missing after the gasLimit. We have also converted the extraData string with its hex value to avoid an error during the init phase.

Among the parameters, we have the following ones:

- *difficulty*: if the value is low, the transactions will be quickly processed within our private blockchain.
- *gasLimit*: define the limit of Gas expenditure per block. The gasLimit is set to the maximum to avoid being limited to our tests.

Step 3 - Initialize the private blockchain

It's time to initialize the private blockchain with the genesis block. This operation will create the initial database stored under the data directory dedicated to each miner. You could refer detail command in script “MyChains/init_miners.sh”.

Step 3.1 - Initialize miner1

In /MyChains folder, then type the following command to create the blockchain for the first miner:

```
$geth --datadir ./miner1 init genesis.json
```

After executing command, The logs provide the following information:

- you need a default account
- the blockchain has been successfully created

If you list the content of the miner1 folder, you will notice the following subfolders:

- *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.

Step 3.1 - Initialize miner2

In /MyChains folder, Repeat the same operation to initialize the second miner by specifying its own target folder (~/MyChains/miner2):

```
$geth --datadir ./miner2 init genesis.json
```

Step 4 - Create accounts

Each node has to at least one account to join the blockchain network.

Step 4.1 – New account for miner1

The first created account will work as the default account that will be used to run the node. This account will receive all ethers created by the miner in the private blockchain. These ethers will serve to test our solutions by paying the gas required to process each transaction.

To create the default account for the miner1, type the following command. Keep the password in a safe place:

```
$geth --datadir ~/MyChains/miner1 account new
```

You could repeat above command to add an additional account for testing purpose.

List ~/MyChains /miner1 to check account data:

```
$ls -al ~/MyChains/miner1/keystore
```

To list all accounts of your node, use the following command:

```
$geth --datadir ~/MyChains/miner1 account list
```

```
Account #0: {3d40fad73c91aed74ffbc1f09f4cde7cce533671} keystore:///home/ronghua/Desktop/Github/Blockchain_dev/MyChains/miner1/keystore/UTC--2018-01-25T20-43-46.651303605Z--3d40fad73c91aed74ffbc1f09f4cde7cce533671
Account #1: {62c7db36889cf94288f8bc567f3d32a4aa8f90a0} keystore:///home/ronghua/Desktop/Github/Blockchain_dev/MyChains/miner1/keystore/UTC--2018-01-25T20-45-16.199431544Z--62c7db36889cf94288f8bc567f3d32a4aa8f90a0
```

Step 4.2 – New account for miner1

Repeat the same operation to create the default account for the second miner. The difference lies in the target folder (~/.MyChains/miner2).

New the default account of miner2:

```
$geth --datadir ~/.MyChains/miner2 account new
```

An additional account of miner2:

```
$geth --datadir ~/.MyChains/miner2 account new
```

To list all accounts of your node, use the following command:

```
$geth --datadir ~/.MyChains/miner1 account list
```

```
Account #0: {13b126c92d0fc254c4a8bf414b60953d3fdcd9cb} keystore:///home/ronghua/Desktop/Github/Blockchain_dev/MyChains/miner2/keystore/UTC--2018-01-25T20-45-48.051110307Z--13b126c92d0fc254c4a8bf414b60953d3fdcd9cb
Account #1: {2315f8459dad537954e1b61064f3fc870b6bcf14} keystore:///home/ronghua/Desktop/Github/Blockchain_dev/MyChains/miner2/keystore/UTC--2018-01-25T20-45-57.363030088Z--2315f8459dad537954e1b61064f3fc870b6bcf14
```

Step 5 - Prepare the miners

We are ready to start the miners from our computer and to mine some ethers that will reward our default accounts.

Step 5.1 - Miner1: setup

To start miner1 without input password, a file that will contain the password for default account should be created

Create a password.sec file under ~/.MyChains/miner1/ that contains the password you configured for the first account on miner1, in plain text.

To start the miner1, we will require to run the following command:

```
$ geth --identity "miner1" --networkid 42
--datadir "~/.Desktop/Github/Blockchain_dev/MyChains/miner1"
--nodiscover --mine --rpc --rpcport "8042" --port "30303"
--unlock 0 --password ~/.Desktop/Github/Blockchain_dev/MyChains/miner1/password.sec
--ipcpath "~/.ethereum/geth.ipc"
```

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/pipeline

We suggest store the Geth command into a runnable script. In our example, this script is called “startminer1.sh” and is located here: ~/.MyChains/miner1.

What’s the meaning of the parameter “ipcpath”?

If you want to manage your private blockchain via the Mist browser, you will need to create the “geth.ipc” file in the default Ethereum folder. Mist will detect that your private network is running and will connect to it rather than starting its own instance of Geth.

Step 5.2 - Miner1: start

Make the script runnable:

```
$chmod +x ./miner1/startminer1.sh
```

Start the miner1:

```
$/miner1/startminer1.sh
```

You will notice that the server and the mining process start. The default account will receive ethers mined by the node.

Step 5.3 - Miner1: JavaScript console

You can manage your miner using the Geth Javascript console. For example, you can start and stop mining from the console or send transactions.

This console needs to be attached to a running instance of Geth. Open a new terminal session and type “geth attach”. If you want to start or stop the mining process, proceed as below:

```
Welcome to the Geth JavaScript console!

instance: Geth/miner1/v1.7.3-stable/linux-amd64/go1.9.3
coinbase: 0x3d40fad73c91aed74ffbc1f09f4cde7cce533671
at block: 43325 (Thu, 31 May 2018 14:38:04 EDT)
 datadir: /home/ronghua/Desktop/Github/Blockchain_dev/MyChains/miner1
 modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> miner.start(2)
null
> miner.stop()
true
> []
```

You will notice that “geth attach” didn’t need any additional parameters. The reason is that the “geth.ipc” file is generated into the Ethereum default directory.

More commands for the JavaScript console are described right there:

<https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console>

To quit the attached console, simply use the “exit” command.

Step 5.4 – Miner2: setup

To prepare the second miner, the instructions are similar to above described for the miner1 with some minor changes.

First, create the “password.sec” file under ~/MyChains/miner2 that will contain the password of the default account of your miner2.

Create the script “startminer2.sh” that will be stored at the following location: ~/MyChains/miner2.

```
#!/bin/bash

geth --identity "miner2" --networkid 42 --datadir "~/Desktop/Github/Blockchain_dev/MyChains/miner2" --nodiscover --mine --rpc --rpcport "8043" --port "30304" --unlock 0 --password ~/Desktop/Github/Blockchain_dev/MyChains/miner2/password.sec
```

The difference from the first miner is about the following parameters:

- *identity*: a specific name used to identify our miner
- *datadir*: the folder related to the second miner
- *rpcport*: a specific HTTP-RPC port number used to reach the miner2
- *port*: a specific network port number used to reach the miner2
- *password*: path of the file containing the password of the default account on the miner2

You will notice that we have the same networkid (42). This will be useful later. We are not providing the “ipcpath” parameter. This means that the “geth.ipc” will be created in the data directory of the miner2.

Step 5.5 – Miner2: start

Make the script runnable:

```
$chmod +x ./miner2/startminer2.sh
```

Start the miner1:

```
$/miner2/startminer2.sh
```

You will notice that the server and the mining process start. The default account will receive ethers mined by the node.

Step 5.6 – Miner2: JavaScript console

Unlike miner1, we will have to attach the console to the running instance of Geth because its “geth.ipc” file has been generated in the “datadir” folder and not in the default Ethereum folder.

To attach the console, you can attach it by mentioning the path to the IPC-RPC interface (*geth.ipc*):

```
$geth attach ipc:./miner2/geth.ipc
```

```
Welcome to the Geth JavaScript console!

instance: Geth/miner2/v1.7.3-stable/linux-amd64/go1.9.3
coinbase: 0x13b126c92d0fc254c4a8bf414b60953d3fdcd9cb
at block: 43325 (Thu, 31 May 2018 14:38:04 EDT)
  datadir: /home/ronghua/Desktop/Github/Blockchain_dev/MyChains/miner2
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:
1.0 web3:1.0
```

Now you can manage the mining process by using JavaScript console.

Step 6 – Star mining and check the balance

You can use javascript console to start mining task to earn eth. Running after a while, check the balance of miners.

Get the default account with “eth.coinbase”:

```
> eth.coinbase
"0x13b126c92d0fc254c4a8bf414b60953d3fdcd9cb"
```

List the accounts with “eth.accounts”:

```
> eth.accounts
["0x13b126c92d0fc254c4a8bf414b60953d3fdcd9cb", "0x2315f8459dad537954e1b61064f3fc870b6bcf14"]
```

Check balances of account with “eth.getBalance(<account id>)”, and use command: “web3.fromWei(<balances>)” to get the balance in ether.

```
> web3.fromWei(eth.getBalance(eth.coinbase))
73335.184656174
> web3.fromWei(eth.getBalance(eth.accounts[1]))
7.999622
```

You could use “miner.stop()” to stop mining task.

Chapter 5: Pair the miners

Remember that a blockchain is a peer-to-peer network. This means that our miners need to see each other to propagate transactions. In addition, the discovery protocol is not working on a private blockchain. This means that we have to configure each node to specify the identity and the location of its peers. This chapter instructs how to pair our miners.\

Step 1 – Stop miners

First ensure that your miners are stopped. You can either press ^C on the miner's console or use "miner.stop()" in javascript console to stop mining task.

Step 2 – Get IP address

Get the IP address of your computer running miners

\$ifconfig eth0

Replace the interface according to your network settings:

- Eth0: wired/ethernet
- wifi0: wireless

Step 3 – Get Node info from miners

Retrieve the node information of miner1 in geth console by using "admin.nodeInfo.enode":

```
> admin.nodeInfo.enode
"enode://79f2d7e0a76a1a16505778b06a2b2eae4466f6efaf3f46f3087cde9b148afd74de9f2cbaa9d2e7bb8a71189dc380b618168b76adf852239bcd913101e6847381@[::]:30303?discport=0"
```

Log on geth console of miner2 and Retrieve the node information by using "admin.nodeInfo.enode":

```
> admin.nodeInfo.enode
"enode://6b56c8fce33c0ddddd84c44e9d2a7c7847a33b228cf01656b6910f6c0b6e3ca9cc20322eac801232d6836c2f7a3ce536c2306a50715f20e0808db054dfde4450@[::]:30304?discport=0"
```

Step 4 – Pair the nodes

Here, we illustrate how to define permanent static nodes stored in a file called "static-nodes.json". This file will contain the node information of our miners.

Based on our environment, we will have the following content:

```
[
"enode://79f2d7e0a76a1a16505778b06a2b2eae4466f6efaf3f46f3087cde9b148afd74de9f2cbaa9d2e7bb8a71189dc380b618168b76adf852239bcd913101e6847381@128.226.76.181:30303",
"enode://6b56c8fce33c0ddddd84c44e9d2a7c7847a33b228cf01656b6910f6c0b6e3ca9cc20322eac801232d6836c2f7a3ce536c2306a50715f20e0808db054dfde4450@128.226.76.181:30304",
]
```

You will notice that we have replaced the placeholder [::] with the IP address of our computer. The last part (?discport=0) has been removed.

Based on our example, the file "static-nodes.json" must be stored under the following location:

- ~/MyChains/miner1
- ~/MyChains/miner2

When the miner starts, the Geth process will process the file automatically.

Step 5 – Restart your miners

Stop and start the miners to ensure that they will properly reload the "static-nodes.json" file.

Step 6 – Check the synchronization process

Open the Geth console linked to the miner1 and use "admin.peers" to check which nodes are paired to miner1:

```
caps: ["eth/62", "eth/63"],
id: "6b56c8fce33c0ddddd84c44e9d2a7c7847a33b228cf01656b6910f6c0b6e3ca9cc20322eac801232d6836c2f7a3ce536c2306a50715f20e0808db054dfde4450",
name: "Geth/miner2/v1.7.3-stable/linux-amd64/go1.9.3",
network: {
  localAddress: "128.226.76.181:34312",
  remoteAddress: "128.226.76.181:30304"
},
protocols: {
  eth: {
    difficulty: 174987432726,
    head: "0xdf1974109e989c8fb2ee89bb318318828e45e755f9bbfc2c24f93d63b26eca",
    version: 63
  }
}, {
```

We can see that our node is paired to miner #2 identified by its IP address and its port number (30304).

Then Open the Geth console linked to the miner2 and check peers status:

```
caps: ["eth/62", "eth/63"],
id: "79f2d7e0a76a1a16505778b06a2b2eae466f6efaf3f46f3087cde9b148afd74de9f2cbaa9d2e7bb8a71189dc380b6181e68b76adf852239bcd913101e6847381",
name: "Geth/miner1/v1.7.3-stable/linux-amd64/go1.9.3",
network: {
  localAddress: "128.226.76.181:30304",
  remoteAddress: "128.226.76.181:34312"
},
protocols: {
  eth: {
    difficulty: 174957432725,
    head: "0xd1974109e989c8fb2ee99bb318318828e45e755f9bbfc2c24f93d63b26eca",
    version: 63
  }
}
```

We can see that our node is paired to miner #1 identified by its IP address and its port number (34312).

Step 7 – Validate the synchronization

Let's validate the synchronization process by sending some ethers between accounts defined on each miner.

We are going to send 10 ethers between the following accounts:

miner1(eth.coinbase) -> miner2(eth.accounts[1])

eth.coinbase is the default account that receive the rewards for the mining process. In the miner1, eth.coinbase is the same as **eth.accounts[0]**.

Step 7.1 – Send ethers from Miner1 to Miner 2

Let's check the balances of eth.accounts[1] on miner2

```
> web3.fromWei(eth.getBalance(eth.accounts[1]))
7.999622
```

From the Geth console linked to the miner1, send 10 ethers from the default account to the address of the account[1]:

```
> eth.sendTransaction({from: eth.coinbase, to: "0x2315f8459dad537954e1b61064f3fc870b6bcf14", value: web3.toWei(10, "ether")})
"0x8a72923f86b22052137de6dad58a88412a75559f5f9f37109ae7774a407b9b8b"
```

From the miner2, check whether transferring 10 ethers succeed or not:

```
> web3.fromWei(eth.getBalance(eth.accounts[1]))
17.999622
```

We found that 10 ethers have been added to account[1] of miner2.

Repeat above step to verify transaction that send some ethers from miner2 to miner1.

Finally, synchronization between miners is successful.

Chapter 6: Synchronize the Raspberry PI with the Private Blockchain

In this chapter, we are going to move forward with the setup of the Raspberry PI and pair it with the miners.

Step 1 – Create the datadir folder

Log on RPi using SSH and Create this folder with the following command:

```
$mkdir ~/MyChains/node
```

Step 2 – Transfer the genesis file

From your computer, upload the genesis.json file to the RPi under path: ~/MyChains

Step 3 – Initialize the node

It's time to initialize the private blockchain on the RPi with the genesis block.

```
$cd ~/MyChains/
```

```
$geth --datadir ./node init genesis.json
```

The private blockchain data will reside in the folder “~/MyChains/node”.

Step 4 – Create accounts

Create an initial account that will be used to run the node. To create the default account, type the following command. Keep the password in a safe place:

```
$geth --datadir ./node account new
```

Add an additional account for testing purposes:

```
$geth --datadir ./node account new
```

To list all accounts available on your node, use the following command:

```
$geth --datadir ./node/ account list
```

```
Account #0: {aa09c6d65908e54bf695748812c51d8f2ceea0f5} keystore:///home/pi/Desktop/Github/Blockchain_dev/MyChains/node/keystore/UTC--2018-01-25T23-06-06.300658891Z--aa09c6d65908e54bf695748812c51d8f2ceea0f5
Account #1: {950d8eb4825c597534027638c862496ea0d7cf43} keystore:///home/pi/Desktop/Github/Blockchain_dev/MyChains/node/keystore/UTC--2018-01-25T23-06-23.002653917Z--950d8eb4825c597534027638c862496ea0d7cf43
```

Step 5 – Prepare the node

We are ready to start the node from our RPi. To start the node, we will require to run the following command:

```
$geth --identity "node1" --fast --networkid 42
--datadir /home/pi/Desktop/Github/Blockchain_dev/MyChains/node
--nodiscover --rpc --rpcport "8042" --port "30303"
--unlock 0
--password "/home/pi/Desktop/Github/Blockchain_dev/MyChains/node/password.sec"
--ipcpath /home/pi/.ethereum/geth.ipc
```

The meaning of the main parameters is the following:

- *identity*: name of our node
- *fast*: fast syncing of the database (more details here)
- *networkid*: this network contains an arbitrary value that will be used to identify all nodes of the network. This value must be different from 0 to 3 (used by the live chains)
- *datadir*: folder where is stored in our private blockchain
- *rpc* and *rpcport*: enabling HTTP-RPC server and giving its listening port number
- *port*: network listening port number
- *nodiscover*: disable the discovery mechanism (we will pair our nodes later)
- *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

We recommend you to store the Geth command into a runnable script. In our example, this script is called “startnode.sh” and is located here: ~/MyChains/node

```
#!/bin/bash

geth --identity "node1" --fast --networkid 42 --datadir /home/pi/Desktop/Github/Blockchain_dev/MyChains/node --nodiscover --rpc --rpcport "8042" --port "30303"
--unlock 0 --password "/home/pi/Desktop/Github/Blockchain_dev/MyChains/node/password.sec" --ipcpath /home/pi/.ethereum/geth.ipc
```

Step 6 – Start the node

Make the script runnable:

```
$chmod +x startnode.sh
```

start the node:

```
./startnode.sh
```

You will notice that the node service starts.

Step 7 – JavaScript console

You can manage your node using the Geth Javascript console.

To use this console from your RPi, open a second SSH session attached to your running instance of Geth. Open a new terminal session and type “geth attach”.

Step 8 – Synchronise the blockchain

In this step, we are going to link the RPi to the private blockchain already synchronized on our miners.

Step 8.1 – Get Node info

Start the node and log on JavaScript console, then Retrieve the node information:

```
> admin.nodeInfo.enode
"enode://d77dcbca70dedf06506fb42342dd115885a70ceeff9097552e5f10970e4ae1caec38bf0b6b7452158b6ae59a4f68082df51c3b6fb1b0fb66b7a5a2a94b042135@[:]:30303?discport=0"
```

Step 8.2 – Update the file “static-nodes.json”

The file “static-nodes.json” created in chapter 5 has to be updated by adding the information of the node deployed on the RPi.

This file is on your computer under one of the following locations:

- ~/MyChains/miner1
- ~/MyChains/miner2

Based on our environment, we will have the following content (adjust the values according to your environment):

```
[
  "enode://79f2d7e0a76a1a16505778b06a2b2eae4466f6efaf3f46f3087cde9b148afd74de9f2cbaa9d2e7bb8a71189dc380b618168b76adf852239bcd913101e6847381@128.226.76.181:30303",
  "enode://6b56c8fce33c0ddddd84c44e9d2a7c7847a33b228cf01656b6910f6c0b6e3ca9cc20322eac801232d6836c2f7a3ce536c2306a50715f20e0808db054dfde4450@128.226.76.181:30304",
  "enode://d77dcbca70dedf06506fb42342dd115885a70ceeff9097552e5f10970e4ae1caec38bf0b6b7452158b6ae59a4f68082df51c3b6fb1b0fb66b7a5a2a94b042135@128.226.79.127:30303"
]
```

The first two entries are related to miner1 and miner2. The last row identified the node deployed on the RPi (with its IP address and port number).

Make sure your IP addresses are still up-to-date as they tend to change on a local network.

This new version of “static-nodes.json” must be stored under the following locations:

- [miner1] ~/MyChains/miner1
- [miner2] ~/MyChains/miner2
- [RPi] ~/MyChains/node

Stop and start each node of your blockchain: miner1, miner2 and RPi

Step 9 – Check the synchronization process

Open the Geth console linked to the miner1 and check the paired RPi nodes to miner1:

```
caps: ["eth/62", "eth/63"],
id: "d77dcbca70dedf06506fb42342dd115885a70ceeff9097552e5f10970e4ae1caec38bf0b6b7452158b6ae59a4f68082df51c3b6fb1b0fb66b7a5a2a94b042135",
name: "Geth/node1/v1.7.3-stable/linux-arm/gol.9.3",
network: {
  localAddress: "128.226.76.181:30303",
  remoteAddress: "128.226.79.127:51346"
},
protocols: {
  eth: {
    difficulty: 175016342244,
    head: "0xc69bfd59dd1e256e4d1a6e37f4e597e530b72f1d8e4d48b22ceb47818bfc722",
    version: 63
  }
}
```

Open the Geth console linked to the miner2 and check the paired RPi nodes to miner2:

```
caps: ["eth/62", "eth/63"],
id: "d77dcbca70dedf06506fb42342dd115885a70ceeff9097552e5f10970e4ae1caec38bf0b6b7452158b6ae59af68082df51c3b6fb1b0fb66b7a5a2a94b042135",
name: "Geth/node1/v1.7.3-stable/linux-arm/gol.9.3",
network: {
  localAddress: "128.226.76.181:30304",
  remoteAddress: "128.226.79.127:48094"
},
protocols: {
  eth: {
    difficulty: 174981014966,
    head: "0xf45af8fd509d3f0a52797f52c3337b0a47ee6025df93f108b422a030fce19d74",
    version: 63
  }
}
```

Open the Geth console linked to the node1 and check the paired miner1 and miner2 to RPi node:

```
caps: ["eth/62", "eth/63"],
id: "6b56c8fce33c0ddddd84c44e9d2a7c7847a33b228cf01656b6910f6c0b6e3ca9cc20322eac801232d6836c2f7a3ce536c2306a50715f20e0808db054dfde4450",
name: "Geth/miner2/v1.7.3-stable/linux-amd64/gol.9.3",
network: {
  localAddress: "128.226.79.127:48094",
  remoteAddress: "128.226.76.181:30304"
},
protocols: {
  eth: {
    difficulty: 175016342244,
    head: "0xc69bfd59dd1e256e4d1a6e37f4e597e530b72f1d864d48b22ceb47818bffc722",
    version: 63
  }
}, {
  caps: ["eth/62", "eth/63"],
  id: "79f2d7e0a76a1a16505778b06a2b2eae4466f6efaf3f46f3087cde9b148afd74de9f2cbaa9d2e7bb8a71189dc380b618168b76adf852239bcd913101e6847381",
  name: "Geth/miner1/v1.7.3-stable/linux-amd64/gol.9.3",
  network: {
    localAddress: "128.226.79.127:51346",
    remoteAddress: "128.226.76.181:30303"
  },
  protocols: {
    eth: {
      difficulty: 174981014966,
      head: "0xf45af8fd509d3f0a52797f52c3337b0a47ee6025df93f108b422a030fce19d74",
      version: 63
    }
  }
}
```

We can see that RPi nodes are paired with the miner #1: miner #2.

Step 10 – Validate the synchronization

Let's validate the synchronization process by sending some ethers between accounts defined among each miner or node. Repeat step 7 in chapter 5 to test transactions after starting mining task on two miners.

Finally, You have synchronized your miners and the RPi.

Chapter 7: Create and deploy a Smart Contract

In this chapter, we will discuss how to create and deploy a Smart Contract on the private blockchain.

We are going to create a Smart Contract called SmartToken. The source codes are located in Blockchain_dev/Projects/SmartToken/.

This contract will hold an associative array of addresses and integers. Each address will be associated with a number of tokens that expresses if the user identified by this address is able to use a specific service (tokens > 0) or not (tokens == 0).

As you can see, this example is simplistic. But it will be useful to understand the basic principles of Smart Contracts.

Step 1 – Install Truffle

Truffle is a development framework for Ethereum. There are many others, but we will use Truffle in this tutorial, to develop and deploy our smart contract.

To install Truffle, run the following command:

```
$ sudo npm install -g truffle
```

Step 2 – Create the project

From your computer, create the folder that will host your project:

```
$ mkdir ~/Projects/SmartToken
```

Then, use Truffle to initiate your project:

```
$ cd ~/Projects/SmartToken
```

```
$ truffle init
```

You should obtain the following tree structure::

```
|__ contracts
| |__ ConvertLib.sol
| |__ MetaCoin.sol
| |__ Migrations.sol
|__ migrations
| |__ 1_initial_migration.js
| |__ 2_deploy_contracts.js
|__ test
| |__ metacoin.js
| |__ TestMetacoin.sol
|__ truffle.js
```

This initial project comes with a sample project (MetaCoin).

Open the project with your favorite text editor (Atom, Sublime, etc.).

Step 3 – Create the contract

In the “contracts” directory, create a file named “SmartToken.sol” and paste the following code:

```
1  pragma solidity ^0.4.18;
2
3  contract SmartToken {
4      mapping(address => uint) tokens;
5
6      event OnValueChanged(address indexed _from, uint _value);
7
8      function depositToken(address recipient, uint value) returns (bool success) {
9          tokens[recipient] += value;
10         OnValueChanged(recipient, tokens[recipient]);
11         return true;
12     }
13     function withdrawToken(address recipient, uint value) returns (bool success) {
14         if ((tokens[recipient] - value) < 0) {
15             tokens[recipient] = 0;
16         } else {
17             tokens[recipient] -= value;
18         }
19         OnValueChanged(recipient, tokens[recipient]);
20         return true;
21     }
22     function getTokens(address recipient) constant returns (uint value) {
23         return tokens[recipient];
24     }
25 }
```

You can refer to source code in:

https://github.com/samuelxu999/Blockchain_dev/blob/master/Projects/SmartToken/contracts/SmartToken.sol

Here are description of functions in contract:

- *depositToken*: add some tokens to a specific address
- *withdrawToken*: withdraw some token from a specific address
- *getTokens*: retrieve the number of tokens available for a specific address

Step 4 – Prepare for deployment

Before deploying the Smart Contract, you have to adjust several files in your project.

Step 4.1 – Adapt deployment file

Replace the content of “migrations/2_deploy_contracts.js” with the following content in order to deploy our “SmartToken” Smart Contract:

```
1  var SmartToken = artifacts.require("./SmartToken.sol");
2
3  module.exports = function(deployer) {
4      deployer.deploy(SmartToken);
5  };
```

Step 4.2 – Adapt network settings

The file named “truffle.js” contains network settings used to identify your deployment platform. We will deploy the Smart Contract onto one of our miners. Let’s change the file to adjust the port number to fit our environment:

```
1  module.exports = {
2      // See <http://truffleframework.com/docs/advanced/configuration>
3      // to customize your Truffle configuration!
4      networks: {
5          development: {
6              host: "localhost",
7              port: 8042,
8              network_id: "42", // Match any network id
9              gas: 4712388
10         }
11     }
12 };
```

Step 5 – Deploy the contract

Before proceeding, start your miners to ensure that your Smart Contract will be properly mined and deployed on your private blockchain.

Step 5.1 – Start your miners in 2 different tabs

\$~/MyChains/miner1/startminer1.sh

\$~/MyChains/miner1/startminer2.sh

Step 5.2 – Compile and deploy your contract

Compile contract

\$truffle compile

Deploy your contract to your blockchain:

\$truffle migrate --reset

The “--reset” command line argument is used here to force truffle to deploy the contract, even if has already been deployed before, for repeatability’s sake. And that’s it. Now our SmartToken contract is deployed onto our private Ethereum chain and it is ready to receive calls.

Step 6 – Interact with the contract

You can interact with your contract through the Geth console, the Mist browser or a client application.

In this tutorial, we will show how to use Mist to “watch” the contract and interact with it.

Step 6.1 – Identify the contract

Before using Mist, you have to retrieve two elements about your deployed contract:

- its address
- its ABI (Application Binary Interface)

This information can be retrieved through the Truffle console in this way:

\$truffle console

Then use “SmartToken.address” and “JSON.stringify(SmartToken.abi)” command retrieve contract address and ABI.

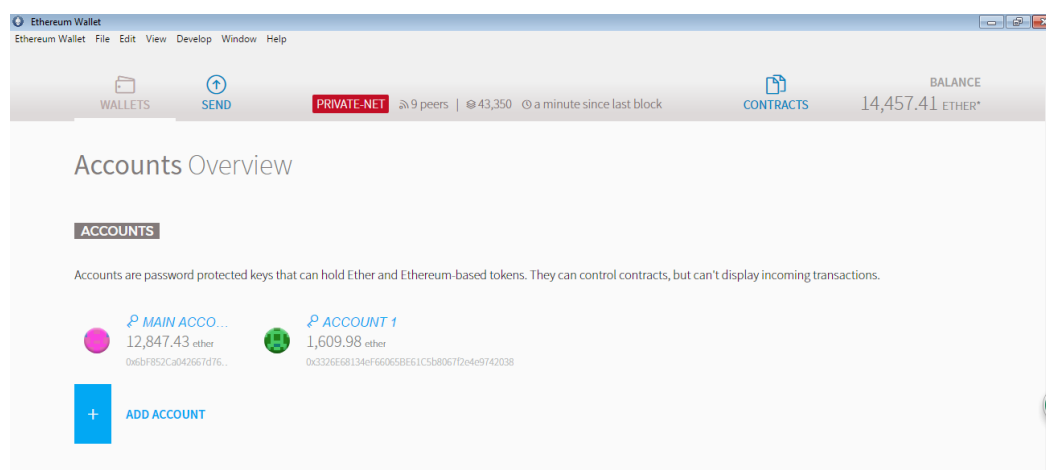
```
truffle(development)> SmartToken.address
'0x1987802daa5798cfb5e881c109c4d448b6e4125b'
truffle(development)> JSON.stringify(SmartToken.abi)
'[{{"constant":false,"inputs":[{"name":"recipient","type":"address"},{"name":"value","type":"uint256"}],"name":"depositToken","outputs":[{"name":"success","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"constant":true,"inputs":[{"name":"recipient","type":"address"}],"name":"getTokens","outputs":[{"name":"value","type":"uint256"}],"payable":false,"stateMutability":"view","type":"function"},{"constant":false,"inputs":[{"name":"recipient","type":"address"},{"name":"value","type":"uint256"}],"name":"withdrawToken","outputs":[{"name":"success","type":"bool"}],"payable":false,"stateMutability":"nonpayable","type":"function"},{"anonymous":false,"inputs":[{"indexed":true,"name":"_from","type":"address"},{"indexed":false,"name":"_value","type":"uint256"}],"name":"OnValueChanged","type":"event"}]'
```

Step 6.2 – Watch the contract on Mist

You can download the latest version of Mist for your system here: <https://github.com/ethereum/mist/releases/>.

Mist is a special kind of browser, a distributed application browser, with a built in Ethereum wallet system that can connect to any network.

Start Mist. As it detects that an IPC file in the default location is being used, it connects to your private network. Once the user interface appears, click the Ethereum Wallet tab on the left, and click the CONTRACTS button in the upper right corner:



Select “WATCH CONTRACT” and fill the form:

Give an arbitrary name for the contract, like “SmartToken”


Enter the contract address retrieved from the Truffle console (SmartToken.address)

Enter the ABI (without enclosing quotes) retrieved from the Truffle console

(JSON.stringify(SmartToken.abi))

Watch contract

CONTRACT ADDRESS

 0x1987802daa5798cfb5e881c109c4d448b6e4125B

CONTRACT NAME

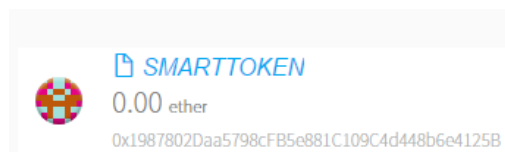
SmartToken

JSON INTERFACE

```
{
  "name": "SmartToken",
  "type": "contract",
  "name": "withdrawToken",
  "outputs": [
    {
      "name": "success",
      "type": "bool"
    }
  ],
  "payable": false,
  "stateMutability": "nonpayable",
  "type": "function",
  "anonymous": false,
  "inputs": [
    {
      "indexed": true,
      "name": "_from",
      "type": "address"
    },
    {
      "indexed": false,
      "name": "_value",
      "type": "uint256"
    }
  ],
  "name": "OnValueChanged",
  "type": "event"
}
```

CANCEL OK

The contract is now visible on Mist:



And you can interact with it:

WALLETS

SEND

PRIVATE-NET


9 peers

43,371

2 minutes since last block

CONTRACTS

14,457.41 ETH*



SmartToken

0x1987802Daa5798cfB5e881C109C4d448b6e4125B

0.00 ETH*

Transfer Ether & Tokens

Copy address

Show QR-Code

Show Interface

HIDE CONTRACT INFO

READ FROM CONTRACT

WRITE TO CONTRACT

Get tokens

Recipient - address

0x123456...

Value

0

Select function

Pick A Function

Pick A Function

Deposit Token

Withdraw Token

Step 6.3 – Test the contract

To test the contract, pick the default address created on the RPi (you will understand why in the next part).

In our tutorial, this address (eth.coinbase) is: “0xaa09c6d65908e54bf695748812c51d8f2ceea0f5”.


To check your contract, get the number of tokens of this contract:

 SMARTTOKEN

READ FROM CONTRACT

Get tokens

Recipient - address


 0xaa09c6d65908e54bf695748812c51d8f2ceea0f5

Value

14

The function “getTokens” is a constant. You can use it without paying any fees.

Now, test transaction by depositing 1 token to this address. Choose the account that will execute the function and pay the fees:


 SMARTTOKEN 0.00 ETHER*

READ FROM CONTRACT

WRITE TO CONTRACT

Get tokens

Recipient - address

 0xaa09c6d65908e54bf695748812c51d8f2ceea0f5


Value

15

Select function

Deposit Token


Recipient - address

 0xaa09c6d65908e54bf695748812c51d8f2ceea0f5

Value - 256 bits unsigned integer

1

Execute from


 Main Account (Etherbase) - 12,847.43 ETH

EXECUTE

When the block is mined, you should see that the value has changed to 15.

Proceed with the withdraw function to ensure that your contract works as expected.

You can also watch the events triggered by the contract:

 SMARTTOKEN 0.00 ETHER*

Filter events

Jun 15	On Value Changed	from: 0xaa09c6d65908e54bf695748812c51d8f2ceea0f5 value: 15	7 of 12 Confirmations
Mar 6	On Value Changed	from: 0x950d8eb4825c597534027638c862496ea0d7cf43 value: 56	
Mar 6	On Value Changed	from: 0x950d8eb4825c597534027638c862496ea0d7cf43 value: 58	
Mar 6	On Value Changed	from: 0x950d8eb4825c597534027638c862496ea0d7cf43 value: 60	
Mar 6	On Value Changed	from: 0x950d8eb4825c597534027638c862496ea0d7cf43 value: 62	

Chapter 8: Summary

Until now, you have created, compiled and deployed a Smart Contract. We also demonstrate how to use Mist to interact with Smart Contract.

To sum it up:

- We installed the Go Ethereum client on both our laptop and our Raspberry Pi
- We created our very own private Ethereum instance
- We installed 2 mining nodes on the laptop, and a user node on the Raspberry Pi
- We made sure all nodes were synchronized
- And finally we deployed a smart contract to our private instance.