Creating a private Ethereum Blockchain with PLCnext Technology

Introduction

This tutorial creates a private Ethereum blockchain consisting of a server running on an Xubuntu PC, and a remote node running on PLCnext.

The tutorial demonstrates:

- The transfer of Ether between accounts on the server and on the PLC.
- The implementation of a simple Smart Contract on the Ethereum blockchain.

The Xubuntu server is a miner, which (a) validates and propagates blocks of transactions within the blockchain, and (b) generates Ether to pay for transactions on the private blockchain.

Target Group

PLCnext Early Adopters.

Requirements

PLCnext Technology AXC F 2152/A SAMPLE hardware with firmware Build 4739.

PC running Xubuntu.

Internet connection

Procedure

This procedure closely follows the tutorial in reference (1), which includes more detailed explanations of why some steps in this process are necessary.

Set PLC IP address, default gateway and DNS server

Connect the PLC to the Internet. This can be done with the following two commands (change the IP address shown below to the address of your internet gateway):

```
# route add default gateway 192.168.1.1
```

```
# echo nameserver 8.8.8.8 > /etc/resolv.conf
```

The settings can be tested by pinging a well-known URL:

```
# ping www.google.com
```

Set the correct time on the PLC

Geth will not work correctly if the system clock is wrong. You can set the correct date & time with with the `date` command. Luckily, the PLC also includes an ntp service that will keep the clock on the PLC synchronised.

Edit the file /etc/ntp.conf to add the ntp server(s) you want to use. In my case, the ntp servers are in Ireland:

```
   :
server 0.ie.pool.ntp.org
server 1.ie.pool.ntp.org
server 2.ie.pool.ntp.org
server 3.ie.pool.ntp.org
   :
```

Install geth on the PLC

Browse to the Geth download site – https://geth.ethereum.org/downloads/

Download the latest stable Geth release for Linux ARMv7.

Copy the download to the PLC using a command like:

```
$ scp geth* root@192.168.1.10:/home/root
```

On the PLC, extract the archive and copy the resulting executable to /usr/local/bin, then delete the extracted directory:

```
# tar zxvf geth-linux-arm7-1.6.7-ab5646c5.tar.gz
# cd geth-linux-arm7-1.6.7-ab5646c5
# sudo cp geth /usr/local/bin
# cd ..
# rm –R geth*
```

You may need to add /usr/local/bin to the PATH in the PLC.

Check the version of Geth:

```
# geth version
```

```
Geth
Version: 1.6.7-stable
Git Commit: ab5646c532292b51e319f290afccf6a44f874372
Architecture: arm
Protocol Versions: [63 62]
Network Id: 1
Go Version: go1.8.3
Operating System: linux
GOPATH=
GOROOT=/usr/local/go
```

If you see version information like the above, then Geth has been installed correctly.

Start Geth on the PLC:

```
# geth
```

The PLC will start synchonising data with the live Ethereum chain. Press CTRL-C to stop Geth.

The blockchain data is located in the directory ~/.ethereum/geth/chaindata

Your accounts will be stored in a wallet located in the directory ~/.ethereum/keystore

Install Geth on the private Blockchain server (Linux)

First, install Homebrew (package manager) on the private Blockchain server (Linux)

```
$ sudo apt install linuxbrew-wrapper
$ brew update
```

Now install Geth:

```
$ brew tap ethereum/ethereum
$ brew install ethereum
```

Create a symbolic link to Geth:

```
$ ln -s ~/.linuxbrew/Cellular/ethereum/1.6.7.bin/geth /usr/local/bin/geth
```

Check the version of Geth:

```
$ geth version
```

Make sure the version of Geth is the same on the server and on the PLC.

Start Geth on the server:

```
$ geth
```

Geth now starts synchronising with the main chain, which will take several days. Press CTRL-C to stop Geth.

The blockchain data is now located in the directory ~/.ethereum/geth/chaindata

Your accounts will be stored in a wallet located in the directory ~/.ethereum/keystore

Create two accounts on the blockchain server

The two accounts on the Blockchain server will be named “miner1” and “miner2”.

Create data directories on the blockchain server (PC):

```
$ mkdir -p ~/blockchain/miner1
$ mkdir -p ~/blockchain/miner2
```

Create a blockchain Genesis file called genesis.json in the ~/blockchain directory, with the following content:

```
{
 "nonce": "0x0000000000000042",
 "mixhash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "difficulty": "0x400",
```

```
 "alloc": {},
 "coinbase": "0x0000000000000000000000000000000000000000",
 "timestamp": "0x00",
 "parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
 "extraData": "0x4d792047656e6573697320426c6f636b",
 "gasLimit": "0xffffffff",
 "config": {
    "chainId": 42,
    "homesteadBlock": 0,
    "eip155Block": 0,
    "eip158Block": 0
 }
}
```

Note that the parameter “extraData” in the genesis file is a hexadecimal encoded string – in my case, the string that is encoded is “My Genesis Block”.

Initialise the private blockchain for both miners:

```
$ cd ~/blockchain
$ geth --datadir ~/blockchain/miner1 init genesis.json
$ geth --datadir ~/blockchain/miner2 init genesis.json
```

Create the default accounts for miner1 and miner2. Remember the passwords that you use here!

```
$ geth --datadir ~/blockchain/miner1 account new
$ geth --datadir ~/blockchain/miner2 account new
```

List the account details for each miner:

```
$ geth --datadir ~/blockchain/miner1 account list
$ geth --datadir ~/blockchain/miner2 account list
```

Create a text file called password.sec in the ~/blockchain/miner1 directory that contains the Password (in plain text) that you entered when creating the account for miner1.

Create a text file called password.sec in the ~/blockchain/miner2 directory that contains the Password (in plain text) that you entered when creating the account for miner2.

Create a script file called startminer1.sh in the ~/blockchain/miner1 directory, with the following content:

```
#!/bin/bash

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

Make the script runnable:

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

Create a script file called startminer2.sh in the ~/blockchain/miner2 directory, with the following content:

```
#!/bin/bash

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

Make the script runnable:

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

You can test the scripts by running them in separate consoles:

From the directory ~/blockchain/miner1:

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

From the directory ~/blockchain/miner2:

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

You are now mining Ether on the private blockchain!

Check the balance on accounts on both nodes

With the miners running, attach a geth console to miner1:

```
$ geth attach
```

List the accounts associated with miner1:

```
> eth.accounts
```

Check the balance on the default account:

```
> eth.getBalance(eth.coinbase)
```

Note that it's possible to have multiple accounts associated with a node, but in this case we only created one account for each node.

Check the balance on the default account:

```
> eth.getBalance(eth.coinbase)
```

The result is a very large integer because the balance is reported in Wei (the smallest unit of currency). To report the balance in ether:

```
> web3.fromWei(eth.getBalance(eth.coinbase))
```

Now, from a separate terminal window, attach a geth console to miner2:

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

Check the balance on the default account for miner2:

```
> web3.fromWei(eth.getBalance(eth.coinbase))
```

Synchronise nodes on the private blockchain

We can only transfer ether between nodes that have been synchronised on the same private blockchain.

With both nodes running, attach a geth console to miner1:

```
$ geth attach
```

Get the node information for miner1:

```
> admin.nodeInfo.enode
```

```
"enode://b8863bf7c8bb13c3afc459d5bf6e664ed4200f50b86aebf5c70d205d32dd77cf2a
888b8adf4a8e55ab13e8ab5ad7ec93b7027e73ca70f87af5b425197712d272@[::]:30303?d
iscport=0"
```

Now, from a separate terminal window, attach a geth console to miner2:

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

Get the node information for miner2:

```
> admin.nodeInfo.enode
```

```
"enode://41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d
9cfdd091b81464b9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7@[::]:30304?d
iscport=0"
```

Now stop both miners by pressing CTRL-C in the terminal where the miners were started.

To pair the nodes:

Create a file called static-nodes.json, containing the node information for both nodes. For example:

```
[
"enode://b8863bf7c8bb13c3afc459d5bf6e664ed4200f50b86aebf5c70d205d32dd77cf2a
888b8adf4a8e55ab13e8ab5ad7ec93b7027e73ca70f87af5b425197712d272@192.168.1.39
:30303",
"enode://41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d
9cfdd091b81464b9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7@192.168.1.39
:30304"
]
```

Notes:

- The placeholder [::] has been replaced with the IP address of the network interface of our PC.
- The last part (?discport=0) has been removed.
- The node information is separated by a comma.

Put a copy of the static-nodes.json file in each of the following directories:

```
~/blockchain/miner1
~/blockchain/miner2
```

Now restart both nodes, attach a geth console to each node, and from each console check which other nodes are paired to this node:

```
> admin.peers
```

```
[{
 caps: ["eth/62", "eth/63"],
 id:
"41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d9cfdd091
b81464b9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7",
 name: "Geth/miner2/v1.5.5-stable-ff07d548/darwin/go1.7.4",
 network: {
 localAddress: "192.168.1.25:59153",
remoteAddress: "192.168.1.25:30304"
 },
 protocols: {
 eth: {
 difficulty: 96831214,
 head:
"0x0f8a3318a47429aee7a442cd1c3258eab7427b7fa1cb3c2a3e4bdf70ed6d8cf8",
 version: 63
 }
 }
}]
```

Move ether between accounts on different nodes of the blockchain server

After the two miners on the server have been running for a while, you should have mined enough ether to start moving ether between accounts on different nodes.

With the miners running, attach geth consoles to both mining nodes and check the balance on both default accounts.

Send approximately half the ether from the default account on miner1 to the default account on miner2. From the geth console attached to miner1, replace the account id in the following command with the id of the default account in miner2, and change the value “500” to a suitable amount of ether:

```
> eth.sendTransaction({from: eth.coinbase, to:
"0x3e3753727dd6d965c0c696ea5619b8050ca89a49", value: web3.toWei(500,
"ether")})
```

Now check the balance on both accounts to make sure the ether was transferred.

Connect the PLC to the private blockchain

On the PLC, create a directory to hold data for the private blockchain:

```
# mkdir -p ~/blockchain/node
```

Copy the genesis.json file to the ~/blockchain directory on the PLC.

Initialise the PLC node:

```
# cd ~/blockchain
# geth --datadir ~/blockchain/node init genesis.json
```

Create the default account. Remember the passwords that you use here!

```
# geth --datadir ~/blockchain/node account new
```

List the account details:

```
# geth --datadir ~/blockchain/node account list
```

Create a text file called password.sec in the ~/blockchain/node directory that contains the Password (in plain text) that you entered when creating the default account.

Create a script file called startnode.sh in the ~/blockchain/node directory, with the following content:

```
#!/bin/bash

geth --identity "node" --fast --networkid 42 --datadir ~/blockchain/node --
nodiscover --rpc --rpcport "8042" --port "30303" --unlock 0 --password
~/blockchain/node/password.sec --ipcpath ~/.ethereum/geth.ipc
```

Make the script runnable:

```
# chmod +x startnode.sh
```

From the directory ~/blockchain/node:

```
# ./startnode.sh
```

Attach a geth console:

```
$ geth attach
```

List the accounts:

```
> eth.accounts
```

Stop mining on this node, since the PLC doesn’t have enough resources to mine ether:

```
> miner.stop()
```

Check the balance on the default account:

```
> eth.getBalance(eth.coinbase)
```

Get the node information:

```
> admin.nodeInfo.enode
```

```
"enode://b8863bf7c8bb13c3afc459d5bf6e664ed4200f50b86aebf5c70d205d32dd77cf2a
888b8adf4a8e55ab13e8ab5ad7ec93b7027e73ca70f87af5b425197712d272@[::]:30303?d
iscport=0"
```

Stop the node on the PLC by pressing CTRL-C in the terminal where the node was started.

On the blockchain server (PC), edit the static-nodes.json file to add the node information from the PLC. For example:

```
[
"enode://b8863bf7c8bb13c3afc459d5bf6e664ed4200f50b86aebf5c70d205d32dd77cf2a
888b8adf4a8e55ab13e8ab5ad7ec93b7027e73ca70f87af5b425197712d272@192.168.1.25
:30303",
"enode://41be9d79ebe23b59f21cbaf5b584bec5760d448ff6f57ca65ada89c36e7a05f20d
9cfdd091b81464b9c2f0601555c29c3d2d88c9b8ab39b05c0e505dc297ebb7@192.168.1.25
:30304",
"enode://c52b3349d899e1f8ea67c2d01df35c3a40532dec41174460b777bab020079e1a54
6313552b91d5f61adb86ed4e74dd80c0ced70c91d658ef0e4f05969a1cf78e@192.168.1.10
:30303"
]
```

Put a copy of the static-nodes.json file in each of the following directories:

On the PC:

```
~/blockchain/miner1
~/blockchain/miner2
```

On the PLC:

```
~/blockchain/node
```

Now restart all nodes on the PC and the PLC.

Attach a geth console to the node on the PLC and stop mining by issuing the miner.stop() command.

Check that all the nodes have synchronised correctly using the following geth command on all nodes:

```
> admin.peers
```

You should see that each node is paired with the other two nodes.

Now try sending ether from miner1 on the PC to the default account on the PLC node. For example, on the console attached to miner1:

```
> eth.sendTransaction({from: eth.coinbase, to:
"0xfa919b49ef34a821fb4cadfdfa5cc6593cb46fe1", value: web3.toWei(2,
"ether")})
```

Check that the ether was transferred successfully to the PLC.

Create and deploy a Smart Contract to the private blockchain

This can be done by following Part 6 of the tutorial in reference (1), found here:

http://chainskills.com/2017/04/03/create-and-deploy-a-smart-contract-66/

This requires “truffle” to be installed on the PC via npm, and also the Mist browser from https://github.com/ethereum/mist/releases/

If you get an error when attempting to compile the contract using truffle, upgrade node.js to version 8:

```
curl -sL https://deb.nodesource.com/setup_8.x | sudo -E bash -
sudo apt-get install -y nodejs
```

Further work

Integrate the Smart Contract on the private blockchain with a Node-Red flow on the PLC. Use this information to interact with the I/O on the PLC via OPC UA.

References

1. “Create a private Ethereum blockchain with IoT devices” by Said Eloudrhiri. http://chainskills.com/2017/02/24/create-a-private-ethereum-blockchain-with-iot-devices-16/