目录

[1 Network Premissioning 1](#_Toc20281)

[1.1 Network 1](#_Toc27867)

[1.1.1 How to obtain Bytecode? 2](#_Toc24694)

[1.1.2 Test if network authorization works 5](#_Toc27662)

[1.2 Transaction 5](#_Toc29242)

[1.3 Validator Set 6](#_Toc4170)

[1.3.1 How to upgrade smartcontract(smartcontract is not immutable) 6](#_Toc25103)

# 1 Network Premissioning

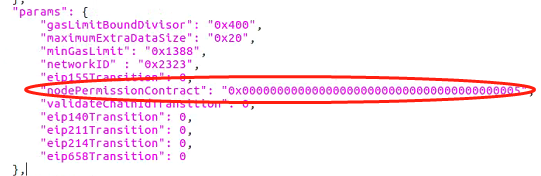
OpenEthereum provides a number of features which enable the network participants to permission different aspects of a blockchain. Often conflated as simply “permissioned blockchains” we introduce permissions on a number of different layers:

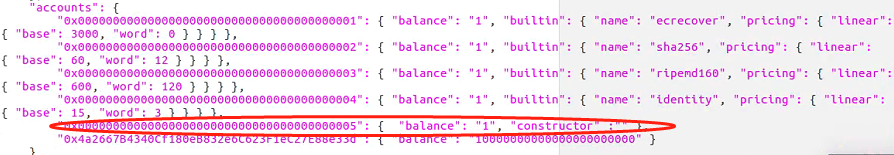
* [Network](https://openethereum.github.io/Permissioning" \l "network)
* [Transaction type](https://openethereum.github.io/Permissioning" \l "transaction-type)
* [Validator set](https://openethereum.github.io/Permissioning" \l "validator-set)
* [Gas price](https://openethereum.github.io/Permissioning" \l "gas-price)
* [Private transactions](https://openethereum.github.io/Private-Transactions)

Each user can have different permissions on each layer. All permissioning is based on blockchain accounts, which means that permissions always correspond to an address.

## 1.1 Network

Permissions on this layer determine which nodes can connect to the network and interact with others





Add nodePermissionContract to the params

And add build API to constructor

### 1.1.1 How to obtain Bytecode?

1, remix

<https://remix.ethereum.org/>

**Install remixd:**

npm -i remixd

**Init truffle:**

mkdir perm\_contract

cd perm\_contract

Truffle init

Ls

Cd contracts

**Start Remixd:**



Go to remix plugin and click remixd



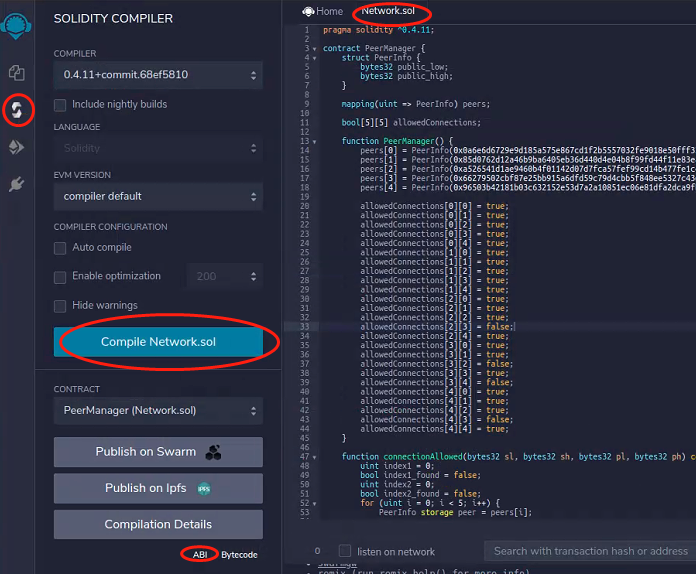
**Start a 5 node network and obtain their encodes**

The EVM architecture allows 32 bytes words. As enodes addresses are 64 bytes long, they need to be cut in two parts for a smart contract to handle them. For example the enode enode://841015562d43c8037b127ee2a89f861d39beb468fecab72ad4bf369d3db8a01a5adeee0e0422cb021acea7ffeb0516db9e1211510ad353dc353b8c52165003c8 would be represented using :

* sl: 0x841015562d43c8037b127ee2a89f861d39beb468fecab72ad4bf369d3db8a01a
* sh: 0x5adeee0e0422cb021acea7ffeb0516db9e1211510ad353dc353b8c52165003c8

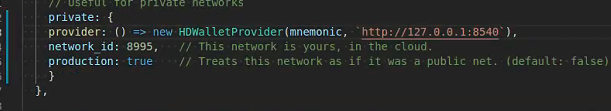
****

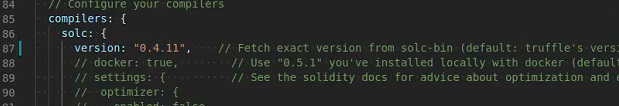
**Obtain Bytecode:**



1, truffle

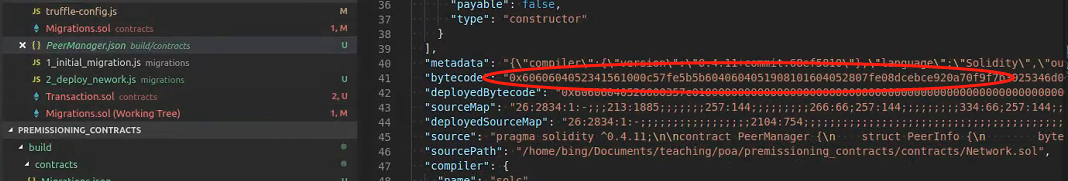






Truffle compile

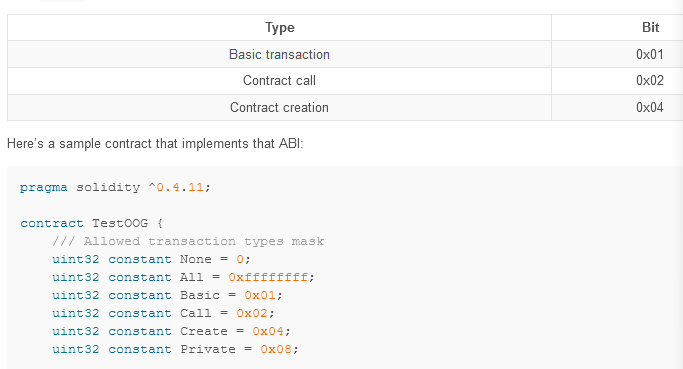
Go to build folder, then open compiled peermanager.json to obtain the bytecode



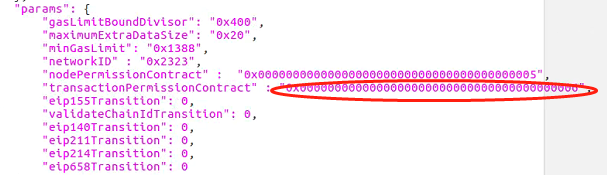
### 1.1.2 Test if network authorization works

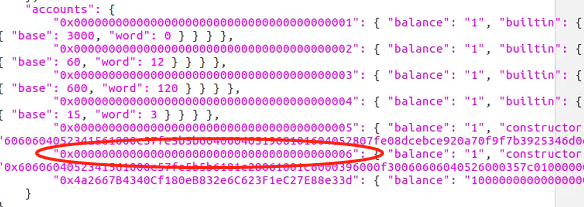
Restart the 5 node network

## 1.2 Transaction



Compile and put the bytecode and address to chain spec





**Test:**

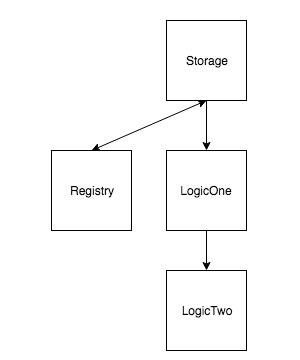
Truffle migrate --reset

Truffle migrate --network private t test if a account can deploy contract

transfer

## 1.3 Validator Set

### 1.3.1 How to upgrade smartcontract(smartcontract is not immutable)



Code:

<https://github.com/onebit256/Upgradable-Proxy-Smart-Contract>

Call functions in Registry to invoke functions in LogicOne / Two

**Code Explain:**

Function() fallback function, when someone send ETH to this contract without providing data or someone try to call function doesn’t Exit

Solidity Assembly language

**Simple illustration of upgradable smart contracts**

Truffle migrate --network private

Truffle console

Let regi = await Registry.at(Registry.address);

regi .setLogicContract(LogicOne.address)

Let log1 = await LogicOne.at(Registry.address);

Log1.setValue(2)

Homework：

1. Set Current Logic contract in Registry

Registry.at(Registry.address).setLogicContract(LogicOne.address)

1. Check logic\_contract address in Registry

Registry.at(Registry.address).logic\_contract()

1. Update Registry storage from LogicOne

LogicOne.at(Registry.address).setVal(2)

// Check value: value should be 4

LogicOne.at(Registry.address).val()

// check owner val

Registry.at(Registry.address).owner()

1. Change logic layer to LogicTwo

Registry.at(Registry.address).setLogicContract(LogicTwo.address)

1. Set LogicTwo new value

LogicTwo.at(Registry.address).setVal(2)

// check value: value should be 6

LogicTwo.at(Registry.address).val()

1. LogicOne should still be able to set the val

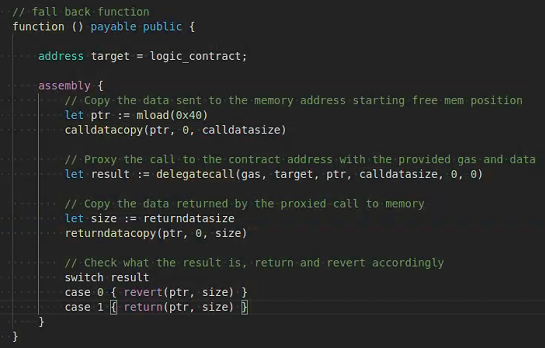
LogicOne.at(Registry.address).setVal(2)

// check value: value should be 6. WHY?

LogicOne.at(Registry.address).val()

**Comparing this example with Validator Set**

<https://github.com/onebit256/kovan-validator-set>



Delegatecall: call other function



Create an instance of the logic contract in proxy contract

Set ValidatorSet by Contract

