Deploy de Smart Contract na rede de testes do Ethereum

Autor: Ramon Rocha Rezende

Deploy de Smart Contract na rede de testes do Ethereum **1**
Introdução 1
Requisitos 1
Criação de conta e sincronização 2
Adquirindo Ether 4
Deploy de smart contract 5
Interação com smart contract 7
Blockchain explorer 8
Referências 10

Introdução

Para fazer deploy de um smart contract na rede de testes do Ethereum precisaremos de uma conta na rede em que faremos o deploy e de um smart contract. Neste caso utilizaremos a rede de testes Rinkeby. Este exemplo foi feito usando Ubuntu 18.04 LTS.

Requisitos

Precisamos de um cliente para interagir com a rede e criar uma conta, para isso utilizaremos o *geth*. O *geth*, ou *Go Ethereum* é uma das três(junto com C++ e Python) implementações originais do protocolo do Ethereum. Ele é escrito em Go, totalmente open source e licenciado sob a GNU LGPL v3.
Um método fácil para instalação do *geth* é a instalação por pacotes binários, para isso basta adicionar o repositório do Ethereum e fazer a instalação por “apt-get”.

sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum

O passo a passo para instalação foi retirado da documentação original do *geth* encontrada em:
https://geth.ethereum.org/install-and-build/Installing-Geth

Além de poder interagir com a rede precisaremos também compilar o nosso smart contract. Os smart contracts do Ethereum são escritos em Solidity e rodam na EVM(Ethereum Virtual Machine), no entanto, primeiro precisamos transformar o código fonte em bytecode. Logo precisaremos do compilador de Solidity para compilar o código fonte e fazer o deploy.
Um método simples de instalação do compilador é a utilização de pacotes binários, para isso basta adicionar o repositório do ethereum e fazer a instalação por “apt-get”.

sudo add-apt-repository ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install solc

O passo a passo para a instalação foi retirado da documentação original do Solidity encontrada em:
https://solidity.readthedocs.io/en/latest/installing-solidity.html#building-from-source

Criação de conta e sincronização

Com o ambiente preparado estamos prontos para começar. O primeiro objetivo é criar uma conta na rede de testes Rinkeby. Existem alguns métodos para isso como por exemplo a utilização de uma extensão de browser chamada Metamask, mas utilizaremos somente o *geth*.
Para criar uma conta com *geth* na rede de testes Rinkeby podemos executar o comando:

geth --rinkeby account new

Podemos ver as contas já criadas com:

geth account list

Ou para ver contas criadas na rede Rinkeby podemos executar:

geth --rinkeby account list

Primeiramente precisamos sincronizar com a rede na qual pretendemos realizar nossos testes, para isso podemos utilizar:

geth --rinkeby --syncmode light

Vamos entender nosso comando, a primeira flag *--rinkeby* indica que vamos nos conectar na rede de testes Rinkeby do Ethereum(o que já era esperado). A flag *--syncmode* indica o modo de sincronização que utilizaremos, neste caso utilizamos o modo *light*, pois ele é o mais rápido e consome menos recursos da máquina, no entanto, ele é um modo **experimental**, e portanto **extremamente instável**. Embora o modo *light* seja suficiente para

nossos testes é recomendada a utilização de algum outro modo de sincronização para melhor estabilidade, como o *fast*.

Feito isso a sincronização será iniciada, é necessário que ela termine para que sejam feitas transações na rede.
A princípio, com o comando executado anteriormente, não foi aberto um servidor http pelo qual nós possamos nos conectar. No entanto temos um IPC socket que é o arquivo *geth.ipc* que só existe enquanto o geth está executando, e é por onde faremos a nossa conexão.
Através da biblioteca web3py podemos interagir com a rede do ethereum. Um exemplo de script em python que utiliza essa biblioteca pode ser o seguinte:

```
from web3 import Web3

infra_url = "/path/to/folder/geth.ipc"
we3 = Web3(Web3.IPCProvider(infra_url))
```

Neste instante já temos uma instância do objeto web3 que provê uma api de interação com o ethereum. Podemos ver se estamos em sincronização com a rede:

```
we3.eth.syncing
```

Caso a sincronização esteja sendo feita, deverá ser impresso na tela algo como:

```
{
  currentBlock: 4795331,
  highestBlock: 4795499,
  knownStates: 0,
  pulledStates: 0,
  startingBlock: 4790723
}
```

Caso contrário, um simples "*False*" será impresso.

Outro modo de criação de conta é utilizar a própria biblioteca Web3py. Podemos fazer isso em nosso python script utilizando:

```
web3 = Web3("Provider here")
web3.geth.personal.newAccount('the-passphrase')
```

Adquirindo Ether

A rede de testes Rinkeby utiliza o protocolo de consenso "proof-of-authority", diferente da rede principal que utiliza o "proof-of-work". Isso torna a rede não totalmente descentralizada, mas como é uma rede de testes isso não é um problema. Sendo assim, não é permitido que

todos os nós da rede possam “minerar” Ether. Como o Ether não tem valor monetário na rede de testes isso evita ataques de *spam*.
No entanto, ainda precisamos de Ether para realizar qualquer tipo de transação de escrita na rede, bem como fazer deploy do nosso smart contract. Para conseguir Ether existem algumas contas que disponibilizam o Ether de forma gratuita. Estas contas são chamadas de *faucets*. O *faucet* da Rinkeby pode ser acessado em:
https://faucet.rinkeb

Se executarmos, dentro do console do *geth*, o comando:

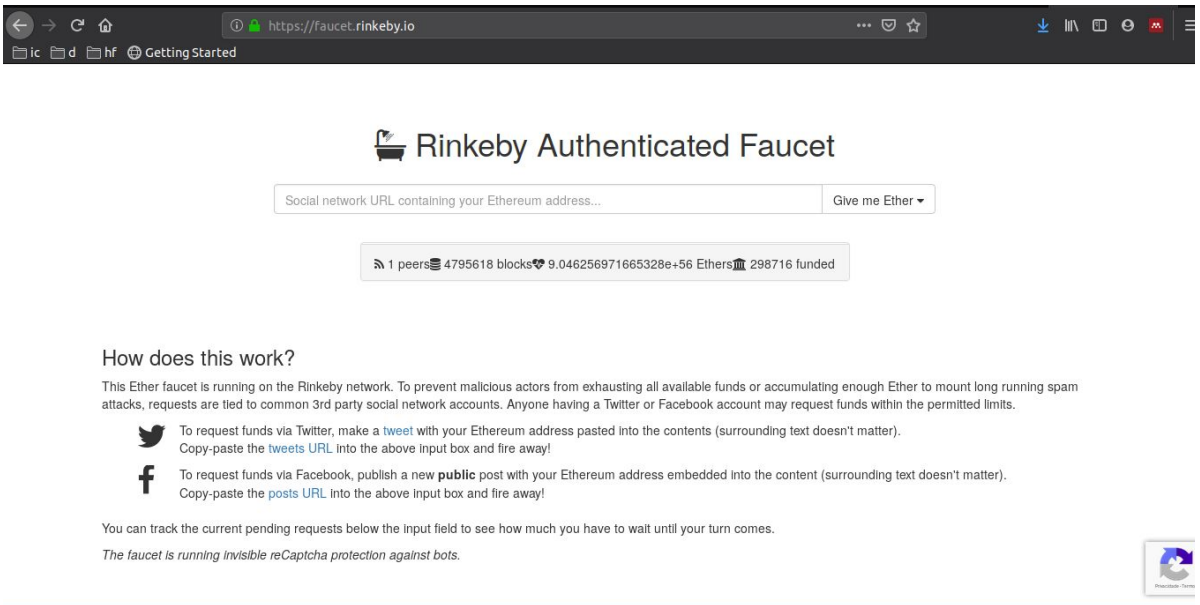
eth.getBalance(eth.accounts[0])

Retornamos a quantidade de Ether associada à primeira conta do nosso vetor de contas. O resultado é um número inteiro que representa a quantidade de *Wei* associada à conta. O *Wei* é a unidade mínima do ethereum, sendo que 1 *Ether* é equivalente a 10^{18} *Wei*. Para uma conta recém criada é esperado que esta função retorne zero.

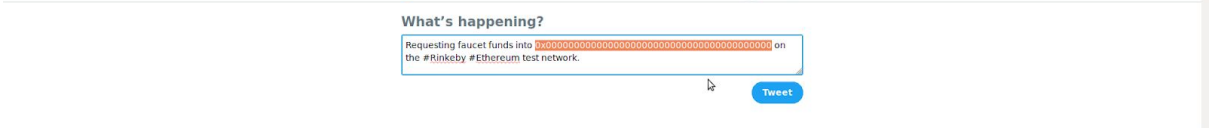
Multiplier	Name
10^{0}	Wei
10^{12}	Szabo
10^{15}	Finney
10^{18}	Ether

Definição de Ether segundo Yellow Paper

Acessando a página do *faucet* podemos seguir as instruções para adquirir alguma quantidade de Ether.



É necessário fazer uma publicação em alguma rede social e colar o link da publicação no box indicado. Clicando em "*tweet*" , você será direcionado para um exemplo de publicação, ela basta alterar o *"0x0000000000000000000000000000000000000000"* para o endereço da sua conta. É possível recuperar o endereço da conta com os comandos passados anteriormente, mas novamente, executar "*eth.accounts*" dentro do console do *geth* recupera as contas criadas.



Após feita a publicação é necessário colar o link dentro do box indicado no site do *faucet* e selecionar a quantidade de Ether desejada, observe que para cada quantidade existe um tempo de espera relacionado para a próxima solicitação.

Uma vez que os fundos foram transferidos a execução da função *eth.getBalance()* deverá retornar um número diferente de zero referente ao valor relacionado a conta.

A partir daí teremos um novo momento chave, onde teremos um terminal aberto rodando a sincronização com a rede, um outro com o console do *geth,* e uma conta desbloqueada e com Ether.

Deploy de smart contract

Para realizar transações na rede de testes é preciso que elas sejam assinadas, para isso precisamos da chave privada da conta que vai realizar a transação. Quando uma conta é

criada é gerado um arquivo de chave criptografado. Devemos descriptografar a chave privada com uma senha que é definida junto com a criação da conta. Esse procedimento pode ser feito em python da seguinte forma:

```
with open('/path/to/folder/keys/keyfile) as keyfile:
        encrypted_key = keyfile.read()
        private_key = web3.eth.account.decrypt(encrypted_key, password)
```

Em ordem de fazer deploy do smart contract precisamos primeiro compilar o mesmo. Para isso podemos abrir um novo terminal, navegar até a pasta onde está o código fonte do smart contract e executar o comando:

```
solc contract_name_file.sol --bin --abi --optimize -o compiled/
```

OBS.: Este comando deve ser executado no console do seu sistema linux, não dentro do console do *geth*

Este comando compila o contrato com nome de arquivo " *contract_name_file.sol*". A flag *--abi* indica que deve ser gerado um arquivo *.abi* com a interface do nosso contrato, a flag *--bin* indiga que deve ser gerado um arquivo *.bin* com o bytecode do nosso contrato, e a flag *--optmize* habilita o otimizador de bytecode. O parâmetro *-o compiled/* indica que todas as saídas vão para a pasta */compiled*(esta pasta deve ser criada previamente). A interface do contrato e o bytecode serão utilizados posteriormente.

A partir daqui temos o necessário para fazer o deploy, só precisamos submeter uma nova transação(esta operação custará Ether, logo é necessário que se tenha fundos em sua conta).

Para fazer o deploy precisamos do bytecode presente no arquivo *.bin* que é gerado na compilação do smart contract.

Um dos modos de fazer deploy do smart contract é criando uma transação pura e passando o bytecode do smart contract no campo de dados, bem como a conta que está realizando a transação, o nonce, a quantidade de gás e o preço do gás. Por exemplo:

```
tx = {
        'nonce': web3.eth.getTransactionCount(myAccount),
        'from': myAccount,
        'data': contract_bytecode,
        'gas': 470000,
        'gasPrice': web3.eth.gasPrice
}
```

E então assinamos a nossa transação com a chave privada:

```
signed_tx = web3.eth.account.signTransaction(tx, private_key)
```

E então enviamos a transação assinada:

tx_hash = web3.eth.sendRawTransaction(signed_tx.rawTransaction)

Se tudo correr corretamente já teremos feito o deploy do smart contract e executando o seguinte comando:

web3.eth.getTransactionReceipt(tx_hash)

Se contrato tiver sido submetido será impressa uma saída como:

```
{
 blockHash:
“0xfed7dcbd5e8c68e17bff9f42cd30d95588674497ae719a04fd6a2ff219bb001
d”,
 blockNumber: 2534930,
 contractAddress: “0xbd3ffb07250634ba413e782002e8f880155007c8”,
 cumulativeGasUsed: 1071323,
 from: “0x1db565576054af728b46ada9814b1452dd2b7e66”,
 gasUsed: 458542,
 logs: [],
 logsBloom: “0x00000...”,
 status: “0x1”,
 to: null,
 transactionHash:
“0x1a341c613c2f03a9bba32be3c8652b2d5a1e93f612308978bbff77ce05ab02c
7”,
 transactionIndex: 4
}
```

Interação com smart contract

Para interagir com um smart contract existente precisamos ter acesso ao seu endereço e à sua interface. Podemos ter acesso à interface do contrato utilizando o arquivo *.abi* criado após a compilação do código fonte.

OBS.: o arquivo *.abi* está em formato JSON e deve ser associado a um JSON object.

Exemplo:

```
[
        {
        "constant": true,
        "inputs": [],
        "name": "getCounter",
        "outputs": [
```

```
        {
                "internalType": "uint256",
                "name": "",
                "type": "uint256"
        }
        ],
        "payable": false,
        "stateMutability": "view",
        "type": "function"
        },
        {
        "constant": false,
        "inputs": [],
        "name": "increment",
        "outputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "function"
        },
        {
        "inputs": [],
        "payable": false,
        "stateMutability": "nonpayable",
        "type": "constructor"
        }
]
```

Para interação com o smart contract também precisamos do endereço da conta que vai fazer a interação. Ele pode ser associado a uma variável do tipo string, como por exemplo:

contract_address = ‘0xFB4Ab7152aFBD6d03BB75Fc41b8369D8DbCa641E’

O endereço do contrato é obtido quando é submetido e também pode ser recuperado através de um blockchain explorer.
Precisamos criar um objeto de contrato passando a nossa interface como parâmetro, podemos fazer isso executando a função:

contract = web3.eth.contract(address = contract_address, abi = contract_abi)

Feito isso precisamos de somente mais um passo para interagir com o contrato. Neste ponto, caso seja feita uma tentativa de execução de funções do contrato(de escrita), provavelmente iremos obter um erro. Isso ocorre porque nossa transação não sabe qual conta utilizar, para resolver isso podemos definir uma conta como padrão:

web3.eth.defaultAccount = myAccount

Feito isso poderemos executar funções do nosso smart contract. Uma função de leitura pode ser executada da forma:

contract.functions.get_measurements().call()

Em *.call()* podemos passar o número identificador do bloco no qual queremos fazer a leitura. O valor padrão neste caso é o '*latest*'.

Note que uma função de leitura não é uma transação pois não escreve nada na rede. Ao contrário das funções de leitura, as operações de escrita na rede são transações e por isso devem ser autenticadas com uma conta que contenha fundos suficientes. Logo, para realizar alguma escrita por meio de funções de contrato precisamos criar uma transação assinada.
Anteriormente criamos uma *rawTransaction*, o que é trabalhoso de se fazer para cada transação. Para não ter que repetir esse processo manualmente podemos utilizar um middleware disponibilizado pela api do web3. Precisamos somente configurar e passar a chave privada e ele irá interceptar as transações, assinar e enviar como *rawTransaction* de forma automática. Podemos configurá-lo por exemplo da seguinte forma:

```
from web3 import Web3
from web3.middleware import geth_poa_middleware
from web3.middleware import construct_sign_and_send_raw_middleware

infra_url = "/path/to/folder/geth.ipc"
we3 = Web3(Web3.IPCProvider(infra_url))
web3.middleware_onion.inject(geth_poa_middleware, layer=0)
web3.middleware_onion.add(construct_sign_and_send_raw_middleware(private_key))
```

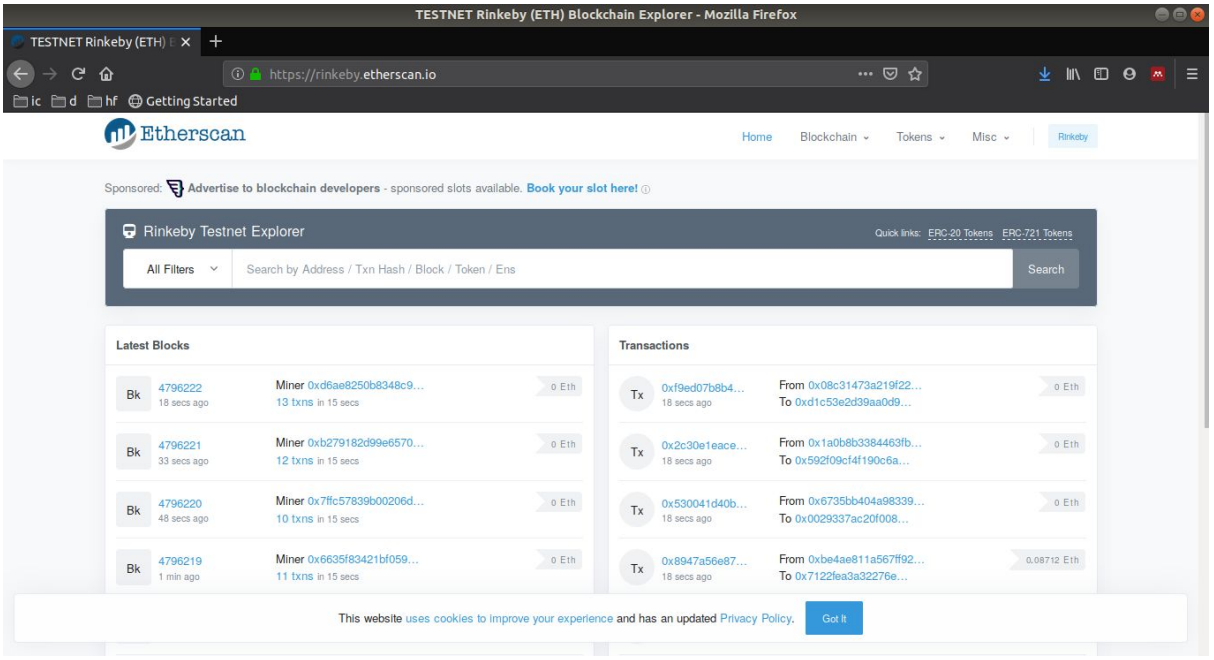
A partir daí podemos chamar as nossas funções do smart contract:

tx_hash = contract.functions.funcaoDeEscrita(data).transact()

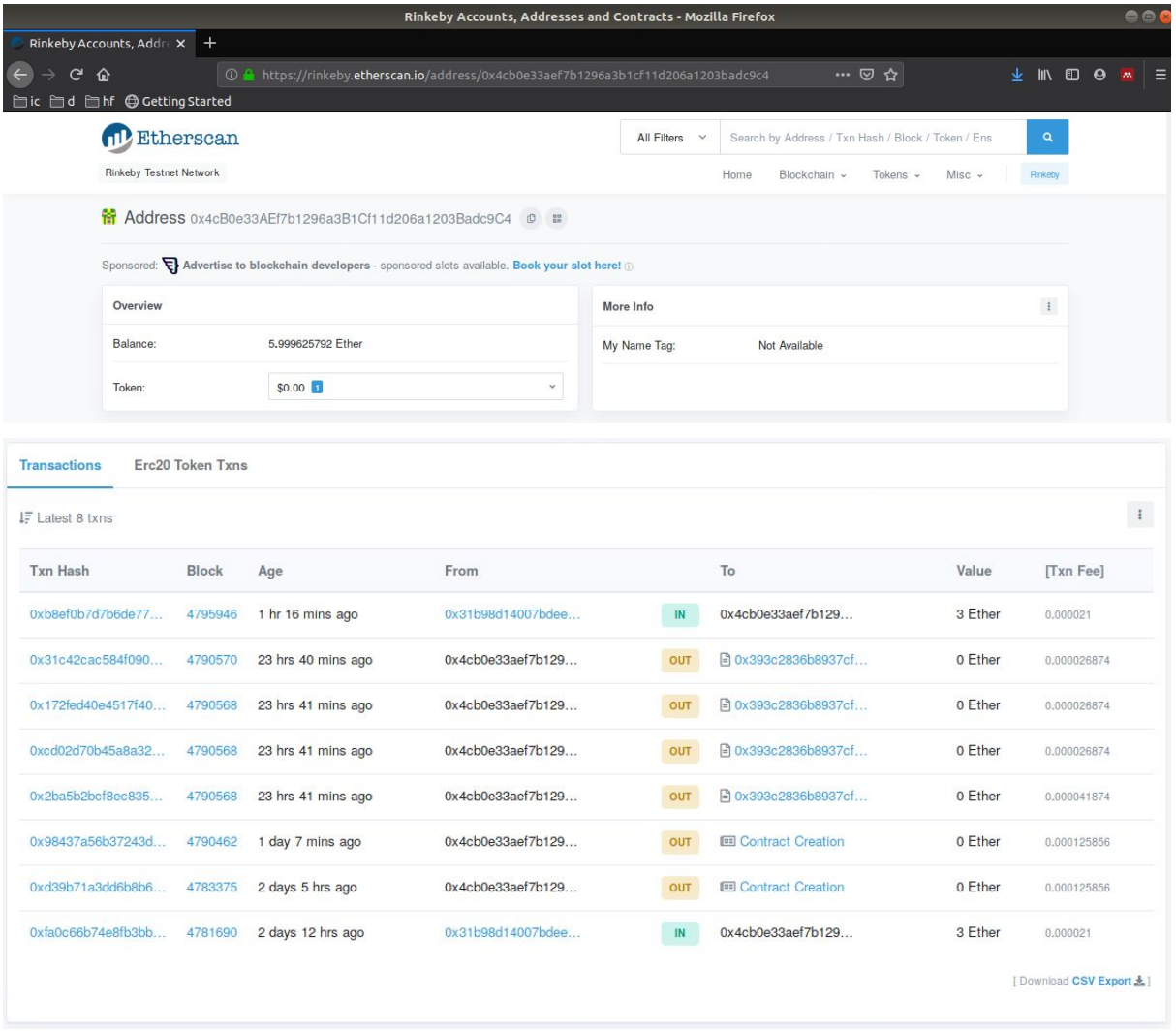
OBS.: Todas as operações dependem de estarmos devidamente sincronizados com a rede.

Blockchain explorer

A rede do Ethereum é pública, portanto todas as operações podem ser vistas pelo público. Neste caso podemos ver as operações de aquisição de fundos e criação de contrato, por exemplo. Para fazer isso basta acessar a página do Etherscan em: https://rinkeby.etherscan.io/



Podemos fazer uma busca por uma conta ou por um contrato específico, basta ter o endereço correspondente.



Referências

http://www.ethdocs.org/en/latest/
https://github.com/ethereum/go-ethereum/wiki
https://solidity.readthedocs.io/en/v0.5.10/
https://geth.ethereum.org/