author: - Pedro Bonilla Nadal title: Contratos con Ethereum. —

Primeros Pasos

Instalación

Antes de empezar con la implementación, hemos de tener instalada la linea de comandos de ethereum. Puedes encontrar una guía de instalación en inglés y más completa aquí.

Hay tres implementaciones fundamentales hechas para ethereum: Geth, Eth y Pyethapp. La primera de esta se implemento en GO, la segunda en C++ y la tercera en Python, tres de los lenguajes con más fuerza de los últimos tiempos. Existen otras implementaciones de ethereum hechas en lenguajes como RUST, Haskell, javascript entre otras muchas. Nosotros nos centraremos en la primeras. Aun así, explicaremos como instalar las dos primeras, pués se recomienda usar varias implementaciones por razones de fiabilidad de la aplicación instalada.

Geth

Para instalar geth en **ubuntu** escriba en su terminal:

```
sudo apt-get install software-properties-common
sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum
```

Eth

Para instalar eth en **ubuntu** escriba en su terminal:

```
sudo apt-get install cpp-ethereum
```

Ejecutandolo

Tanto Geth como Eth son cli multiproposito que ejecutan un modo completo de ethereum. A pesar de ofrecer múltiples interfaces, como un servidor JSONRPC, nosotros nos centraremos en la consola interactiva. Para abrir la consola de Geth escribimos:

```
geth console
```

Conectandose a un area privada para test

Como podremos imaginar, en nuestros primeros pasos, para hacer programas (o contratos) simples no nos interesa estar en la red general de ethereum.

Como eres el único miembro de la red eres el responsable de encontrar todos los bloques, validar todas las transacciones y ejecutar todos los contratos. También podrías hacer trampas y hacerte a ti mismo un ataque del 51%, sin mucho beneficio de todos modos. Esto hará el desarrollo más fácil (y barato), dado que tienes la flexibilidad de controlar tu propia cadena de bloques. Para generarla (cada geth en la misma linea, están puestos en dos por estética):

```
geth --datadir ~/.ethereum_private init ~/dev/genesis.json

geth --fast --cache 512 --ipcpath ~/Library/Ethereum/geth.ipc
--networkid <id-network> --datadir ~/.ethereum_private  console
```

Para usuarios precavidos y responsables es conveniente cambiar el bloque de inicio, para que no se considere un fork "ilegal" de la cadena principal (que puede ser tomada por defecto). Aquí te explican cómo. Nosotros no lo explicaremos por motivos de tiempo.

Por último, para poner un minero local ejecutamos (todo en la misma linea, está puesto en dos por estética):

```
geth <tu_id> --mine --minerthreads=1
etherbase=0x0000000000000000000000000000000000000000
```

Ahora tomaremos una serie de medidas para evitar que otro usuario que no conozca tu nonce, network-id, y genesis file se conecte a tu red. Para esto primero encontraremos la URL de tu nodo con:

```
admin.nodeInfo.NodeUrl
```

Para que otros usuarios se unan diles que usen el comando:

```
admin.addPeer(<tu_id>)
```

Creando cuentas

Para hacer una cuenta nueva escribimos el comando:

```
personal.newAccount("Una frase de contraseña es mejor que una palabra")
```

Para revisar las cuentas creadas usamos:

```
web3.eth.accounts
```

Por último, para mirar el ahorro de una cuenta. Para ello haremos uso de una variable y miraremos la cuenta de índice i.

```
var primaryAccount = web3.eth.accounts[i]
web3.eth.getBalance(primaryAccount)
```

Para revisar todos los balances podemos utilizar esta función, la cual además es muy entretenida de leer e ir comprendiendo poco a poco.

```
function checkAllBalances() {
  web3.eth.getAccounts(function(err, accounts) {
   accounts.forEach(function(id) {
    web3.eth.getBalance(id, function(err, balance) {
     console.log("" + id + ":\tbalance: " + web3.fromWei(balance, "ether") + " ether");
   });
  });
 });
};
```

Contratos y transacciones

Ahora, antes de nada, tenemos que entender como hacemos un envio simple de ether. Solo consistiría en escribir:

```
eth.sendTransaction({from: <cuenta1>, to:
    <cuenta2>, value: web3.toWei(<cantidad>, <moneda>)})
```

Los envíos se hacen en wei, la mínima divisa aceptada. Por ello escribimos la cantidad de una moneda que queremos enviar y la pasamos a esta con toWei. (10^18 wei = 1 Ether = 0.044 BTC = 445USD$ a 30/11/17 11:09 a.m.)

Compilando un contrato

Los contratos solo con aceptados en los codigos binarios especificos de Ethereum (también llamado bytecode). De todos modos estos suelen estar escritos en algún lenguaje de alto nivel como solidity

Primera opción: Terminal (deprecated?)

```
sudo apt-get install solidity
```

Después los compilamos a bytecode desde la terminal de geth con:

```
eth.compile.solidity(<tu_programa>)
```

Por último, si queremos, podemos especificar la ubicación del compilador con:

```
geth --datadir ~/frontier/00 --solc <directorio>  --natspec
```

Segunda opción: compilación online

Tenemos un compilador online en el repositorio oficial de github .

Tercera opción: node solc

Se nos ofrece en su repositorio una guía de instalación con npm.

Primer ejemplo de contrato.

Tomemos ahora este primer ejemplo de contrato, que solo consiste en una función que pide un número a y devuelve a*7.

```
source = "contract test { function multiply(uint a) returns(uint d) { return a * 7; } }"
contract = eth.compile.solidity(source).test
```

y obtenemos como repuesta el código compilado `605280600...`. Como podemos observar, igual que el lenguaje máquina es poco legible.

Contrato "Hola mundo"

Vamos a escribir ahora el ejemplo clásico de todo lenguaje de programación para que se pueda saber la base de como hacer un contrato inteligente, con la funcionalidad mínima de escribir por pantalla "hola mundo". Como bien sabemos por la memoria, con un contrato se puede hacer basicamente cualquier cosa, por el lenguaje turing completo. Aquí ponemos el código.

```
contract mortal {
    address owner;

    function mortal() { owner = msg.sender; }
    function kill() { if (msg.sender == owner) selfdestruct(owner); }
}

contract greeter is mortal {
    string greeting;


    function greeter(string _greeting) public {
        greeting = _greeting;
    }


    function greet() constant returns (string) {
        return greeting;
```

```
    }
}
```

Compilamos el código (aquí usamos la versión online) y copiamos el web3 development a un nuevo archivo js, y lo ejecutamos con

```
loadScript(<tuArchivo.js>)
```

A veces es necesario permitir la carga con:

```
personal.unlockAccount(web3.eth.accounts[0], "yourPassword")
```