Un poco de Java y +

Otra forma de hablar de nuestro día a día…

BLOCKCHAIN, JAVA, QUÉ ES, TUTORIALES, UN POCO DE

Jugando un poco con Ethereum y Java (web3j+Spring Boot)

Fecha: 27 diciembre 2018 Autor/a: LuisMi Gracia 0 Comentarios

ethereum

+

Blockchain sigue estando de moda….

Podríamos decir que es una estructura de datos descentralizada e **inmutable** dividida en bloques, que se vinculan y aseguran mediante algoritmos criptográficos. Cada bloque individual contiene un hash criptográfico del bloque anterior, una marca de tiempo y datos de transacción. Blockchain es una red peer-2-peer, y durante la comunicación entre nodos, cada nuevo bloque se valida antes de agregar.

Y luego tenemos **Ethereum**. Ethreum es una plataforma descentralizada que proporciona un lenguaje de scripting para el desarrollo de aplicaciones. Se basa en ideas de Bitcoin y usa una nueva criptomoneda llamada Ether.

El core de la tecnología Ethereum es **EVM (Ethereum Virtual Machine**), que es una especie de JVM, pero sobe una red de nodos descentralizados.

Para implementar aplicaciones basadas en Ethereum en el mundo Java se usa la la biblioteca **web3j** , que es

una biblioteca Java muy ligera y reactiva para Java y Android, que nos permite integrarnos de forma sencilla con nodos Ethereum.

En este post construiremos una aplicación Spring Boot que usando web3j comunicará con una red Ethereum.

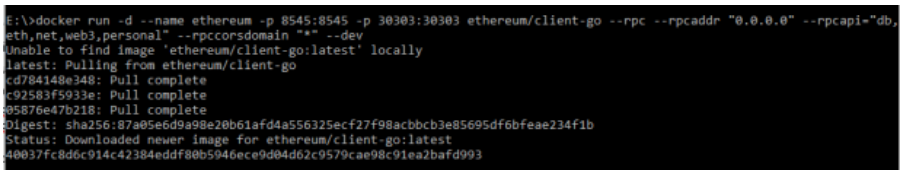
Empecemos:

Lanzando Ethereum en local y jugando un poco con la consola Geth

Aunque hay otras opciones como conectarnos a una red Ethereum en nuestro caso lo que haremos es lanzar un nodo Geth Ethereum en un contenedor Docker y en modo **Development (–dev)**

Para eso basta con ejecutar:

```
docker run -d --name ethereum -p 8545:8545 -p 30303:30303 ethereum/client-go  --rpc --rpcaddr "0.0.0.0"  --rpcapi="db,eth,net,web3,personal"  --rpccorsdomain "*"  –dev
```



Con este comando Docker estoy exponiendo el API RPC Ethereum en el Puerto 8545.

Además al lanzar el contenedor en modo desarrollo tenemos muchos Ethers en la cuenta por defecto (test), de modo que no es necesario minar más Ethers para probar 🙂

Para comunicar con la red Ethereum usaremos la consola interactiva del cliente Geth Javascript, ejecutaremos:

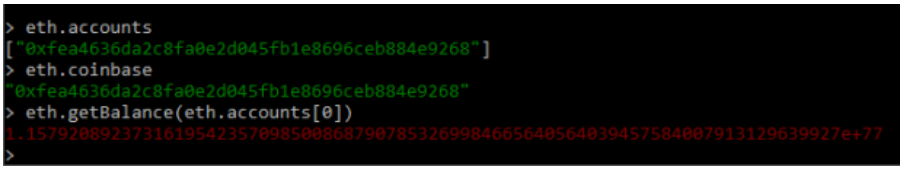
```
docker exec  -it ethereum geth attach ipc:/tmp/geth.ipc
```

Esto nos abre una consola que nos permitirá crear nuevas cuentas, chequear cuántos Ethers tenemos,… por ejemplo:

```
eth.accounts
```

```
eth.coinbase
```

```
eth.getBalance(eth.accounts[0])
```

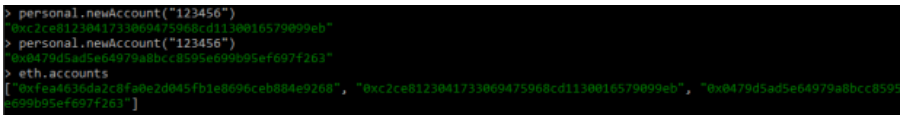


Ahora vamos a crear unas cuentas para luego poder transferir fondos de la cuenta por defecto a las otras, el comando es: `personal.newAccount(password)`

Crearé 2 cuentas (con la misma password):

```
personal.newAccount(“123456”)
```

```
personal.newAccount("123456")
```



Y ahora hare unos traspasos:

```
eth.sendTransaction({from:eth.coinbase,to:eth.accounts[1],value:1000})
```

```
eth.sendTransaction({from:eth.coinbase,to:eth.accounts[2],value:2000})
```

```
eth.getBalance(eth.accounts[0])
```

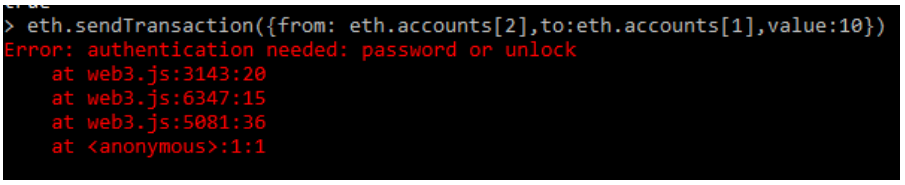
```
eth.getBalance(eth.accounts[1])
```

```
eth.getBalance(eth.accounts[2])
```

Si intento ejecutar:

```
eth.sendTransaction({from: eth.accounts[2],to:eth.accounts[1],value:10})
```

me dirá que tengo las cuentas bloqueadas:

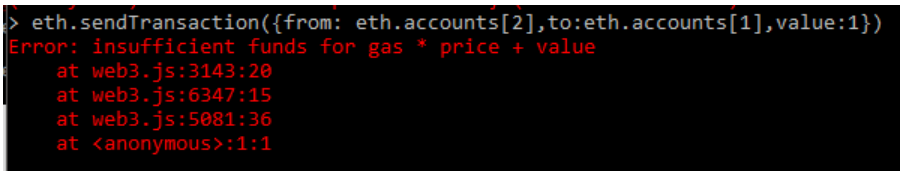


Para desbloquear las cuentas:

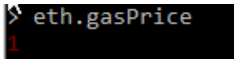
```
personal.unlockAccount(eth.accounts[1],"123456")
```

```
personal.unlockAccount(eth.accounts[2],"123456")
```

Tras esto me saldrá el error:

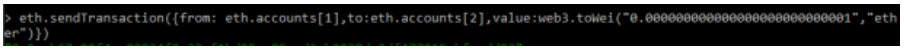


Puedo comprobar el gasPrice y ver si la cuenta origen tiene fondos suficientes:



O bien hacer una transferencia más pequeña usando la función que transforma de Ether a WEI:

```
eth.sendTransaction({from:
eth.accounts[1],to:eth.accounts[2],value:web3.toWei("0.000000000000000000000000001","ether")})
```



Lo que estoy usando desde la consola Geth es el API **web3.js (https://web3js.readthedocs.io/en/1.0/index.html)**

Más info sobre APIs web3

· Accounts v1.0: https://web3js.readthedocs.io/en/1.0/web3-eth-accounts.html#

· Personal v.1.0: https://web3js.readthedocs.io/en/1.0/web3-eth-personal.html

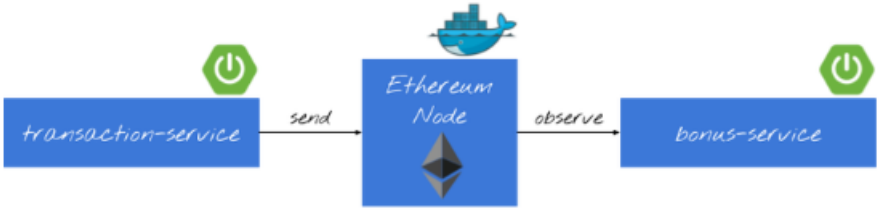
· web3.js v0.20: https://github.com/ethereum/wiki/wiki/JavaScript-API#

Ejemplo Cliente Ethereum con web3j y Spring Boot:

El ejemplo está construido con Spring Boot y tiene 2 funcionalidades:

· Por un lado enviaré transacciones al nodo Geth **(transaction-service)**

· Por otro lado tendré un listener de todas las transacciones que realizará una lógica sencilla, un bonus cada 10 transacciones recibidas (**bonus-service):**



Para seguir el ejemplo lo más sencillo es que os clonéis el repo que contiene el ejemplo en el que me estoy basando (abajo los créditos):

```
git clone https://github.com/piomin/sample-spring-blockchain
```

Una vez clonado el Repo puedo fijarme que en el pom.xml tengo la dependencia de web3j para trabajar con Spring Boot:

```
<dependency>
    <groupId>org.web3j</groupId>
    <artifactId>web3j-spring-boot-starter</artifactId>
    <version>1.6.0</version>
</dependency>
```

Además, si voy al application.yml (en sample-spring-blockchainsrcmainresources) puedo ver la configuración que tengo de mi cliente Geth:

```
spring:
  application:
    name: transaction-service
server:
  port: ${PORT:8090}
web3j:
  client-address: http://192.168.99.100:8545
```

Tengo que cambiar la IP que aparece aquí con la IP que obtengo de la consola Geth al ejecutar `admin.nodeInfo`

Quedando

```
spring:
```

```
application:
  name: transaction-service
server:
  port: ${PORT:8090}
web3j:
  client-address: http://127.0.0.1:8545
```

Comencemos por ver cómo se generan transacciones, para eso vamos a la clase srcmainjavaplpiominserviceblockchainservice**BlockchainService**

Lo primero que vemos es que tenemos inyectado un Bean **Web3j.** Este Bean está incluído en el starter web3j, y se encarga de enviar transacciones al nodo cliente Geth y recibir la respuesta.

En Github veréis:

```
@Service
public class BlockchainService {

    private static final Logger LOGGER = LoggerFactory.getLogger(BlockchainService.class);

    private final Web3j web3j;

    public BlockchainService(Web3j web3j) {
        this.web3j = web3j;
    }
```

Equivalente a

```
@Service
public class BlockchainService {

    private static final Logger LOGGER = LoggerFac

    @Autowired
    Web3j web3j;
```

Este Servicio se invoca desde el BlockchainController,

```
@RestController
public class BlockchainController {

    private final BlockchainService service;

    public BlockchainController(BlockchainService service) {
        this.service = service;
    }

    @PostMapping("/transaction")
    public BlockchainTransaction execute(@RequestBody BlockchainTransaction transaction) throws IOException {
        return service.process(transaction);
    }

}
```

Y su lógica es esta:

```
public BlockchainTransaction process(BlockchainTransaction trx) throws IOException {

    EthAccounts accounts = web3j.ethAccounts().send();
    EthGetTransactionCount transactionCount = web3j.ethGetTransactionCount(
            accounts.getAccounts().get(trx.getFromId()), DefaultBlockParameterName.LATEST).send();

    Transaction transaction = Transaction.createEtherTransaction(
            accounts.getAccounts().get(trx.getFromId()), transactionCount.getTransactionCount(), BigInteger.valueOf(trx.getValue()),
            BigInteger.valueOf(21_000), accounts.getAccounts().get(trx.getToId()), BigInteger.valueOf(trx.getValue()));

    EthSendTransaction response = web3j.ethSendTransaction(transaction).send();

    if (response.getError() != null) {
        trx.setAccepted(false);
        LOGGER.info("Tx rejected: {}", response.getError().getMessage());
        return trx;
    }
    trx.setAccepted(true);
    String txHash = response.getTransactionHash();
    LOGGER.info("Tx hash: {}", txHash);

    trx.setId(txHash);
    EthGetTransactionReceipt receipt = web3j.ethGetTransactionReceipt(txHash).send();

    receipt.getTransactionReceipt().ifPresent(transactionReceipt -> LOGGER.info("Tx receipt: {}", transactionReceipt.getCumulativeGasUsed().intValue()));

    return trx;
}
```

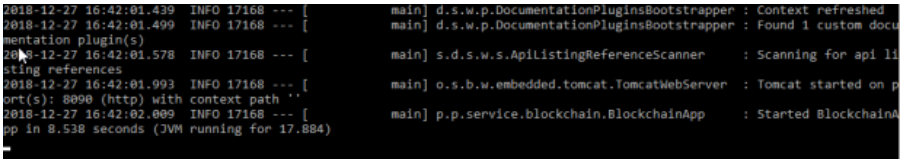
Lanzando transacciones.

veamos cómo invocar al Servicio BlockchainService

1. Ejecutaremos la aplicación Spring Boot con

```
mvn spring-boot:run
```

Esto lanzará la aplicación Spring Boor en el puerto 8090:



(recuerda modificar la IP del nodo Geth en el application.yml)

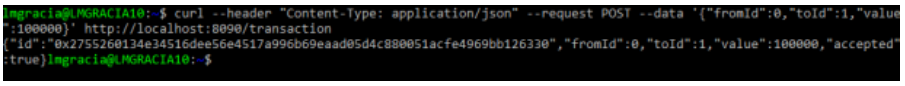
2. Ahora puedo invocar al Controlador en `http://localhost:8090/transaction`

El JSON que espera el controlador es de este tipo:{"fromId":X,"toId":Y,"value":NNN}

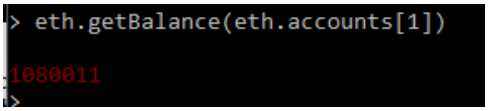
Puedo invocarlo por ejemplo desde curl

```
curl --header "Content-Type: application/json" --request POST --data
'{"fromId":0,"toId":1,"value":100000}' http://localhost:8090/transaction
```

Que me debe devolver que la transacción se ha aceptado



Y puedo volver a revisar cuantos fondos tiene la cuenta 1:



Escuchando Transacciones

El Listener de las transacciones está implementado en la clase pl.piomin.service.blockchain.BlockchainApp

```
@PostConstruct
public void listen() {

    web3j.transactionObservable().subscribe(tx -> {

        LOGGER.info("New tx: id={}, block={}, from={}, to={}, value={}", tx.getHash(), tx.getBlockHash(), tx.getFrom(), tx.getTo(), tx.getValue().intValue());

        try {

            EthCoinbase coinbase = web3j.ethCoinbase().send();
            EthGetTransactionCount transactionCount = web3j.ethGetTransactionCount(tx.getFrom(), DefaultBlockParameterName.LATEST).send();
            LOGGER.info("Tx count: {}", transactionCount.getTransactionCount().intValue());

            if (transactionCount.getTransactionCount().intValue() % 10 == 0) {

                EthGetTransactionCount tc = web3j.ethGetTransactionCount(coinbase.getAddress(), DefaultBlockParameterName.LATEST).send();
                Transaction transaction = Transaction.createEtherTransaction(
                        coinbase.getAddress(), tc.getTransactionCount(), tx.getValue(), BigInteger.valueOf(21_000), tx.getFrom(), tx.getValue());
                web3j.ethSendTransaction(transaction).send();
            }

        } catch (IOException e) {
            LOGGER.error("Error getting transactions", e);
        }

    });

    LOGGER.info("Subscribed");

}
```

Para probarlo lanzaré 10 transacciones vía el comando curl:

```
curl --header "Content-Type: application/json" --request POST --data
'{"fromId":0,"toId":1,"value":1}' http://localhost:8090/transaction
```

Post adaptado desde este artículo de Piotr Minkowski