WELCOME TO ETHERLAND

NODOS RASPBERRY BLOCKCHAIN/ETHEREUM

Pedro Romero Used/Luis Carlos Garcia Gonzalez


ODA LA INFORMACION NECESARIA LA PUEDES ENCONTRAR E
https://github.com/BlockchainEspanaHQ/CommunityLab
Utilizaremos : https://pad.riseup.net/p/lab para compartir info

CLIENTES ETHEREUM

- Interfaz con blockchain capaz de recuperar y verificar información de la cadena de bloques de Etherium, creando varios tipo de nodo

Desde los primeros dias han nacido diversidad de clientes, lo cual demuestra la riqueza del ecosistema y la robustez del estandard definido

Client	Language	Developers	Latest release
go-ethereum	Go	Ethereum Foundation	go-ethereum-v1.4.18
Parity	Rust	Ethcore	Parity-v1.4.0
cpp-ethereum	C++	Ethereum Foundation	cpp-ethereum-v1.3.0
pyethapp	Python	Ethereum Foundation	pyethapp-v1.5.0
ethereumjs-lib	Javascript	Ethereum Foundation	ethereumjs-lib-v3.0.0
Ethereum(J)	Java	<ether.camp>	ethereumJ-v1.3.1
ruby-ethereum	Ruby	Jan Xie	ruby-ethereum-v0.9.6
ethereumH	Haskell	BlockApps	no Homestead release yet

¿ CLIENTES OFICIALES ?

Clientes oficiales

Geth: Cliente en Go Lang
Eth: Cliente C++
Pyethapp: Cliente en Python

Clientes no oficiales:

Parity: Cliente escrito por ethcore en Rust
Ethereumj: Cliente en Java
Ruby-Ethereum: Cliente de Ruby -Etc..

Todos los clientes deberían tener las mismas funcionalidades

TIPOS CLIENTES ETHEREUM

- Los nodos ligeros de eth **estan en desarrollo y el soporte es incompleto**

Full" Sync: Gets the block headers, the block bodies, and validates every element from genesis block.

Fast Sync: Gets the block headers, the block bodies, it processes no transactions until current block - 1024. Then it gets a snapshot state and goes like a full synchronization.

Light Sync: Gets only the current state. To verify elements, it needs to ask to full (archive) nodes for the corresponding tree leaves.

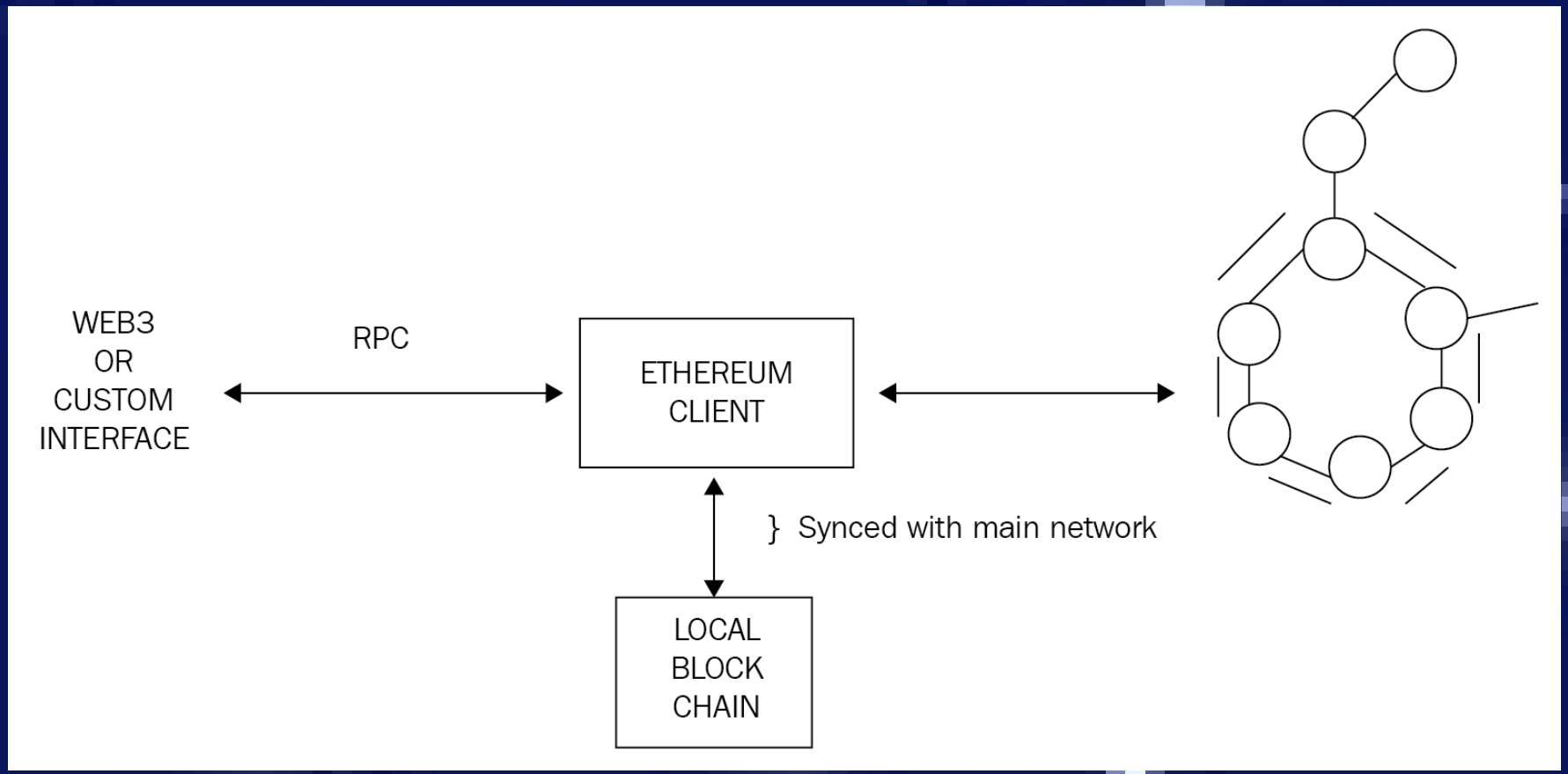
CLIENTE LIGHT ETHEREUM

Nodo Ligero

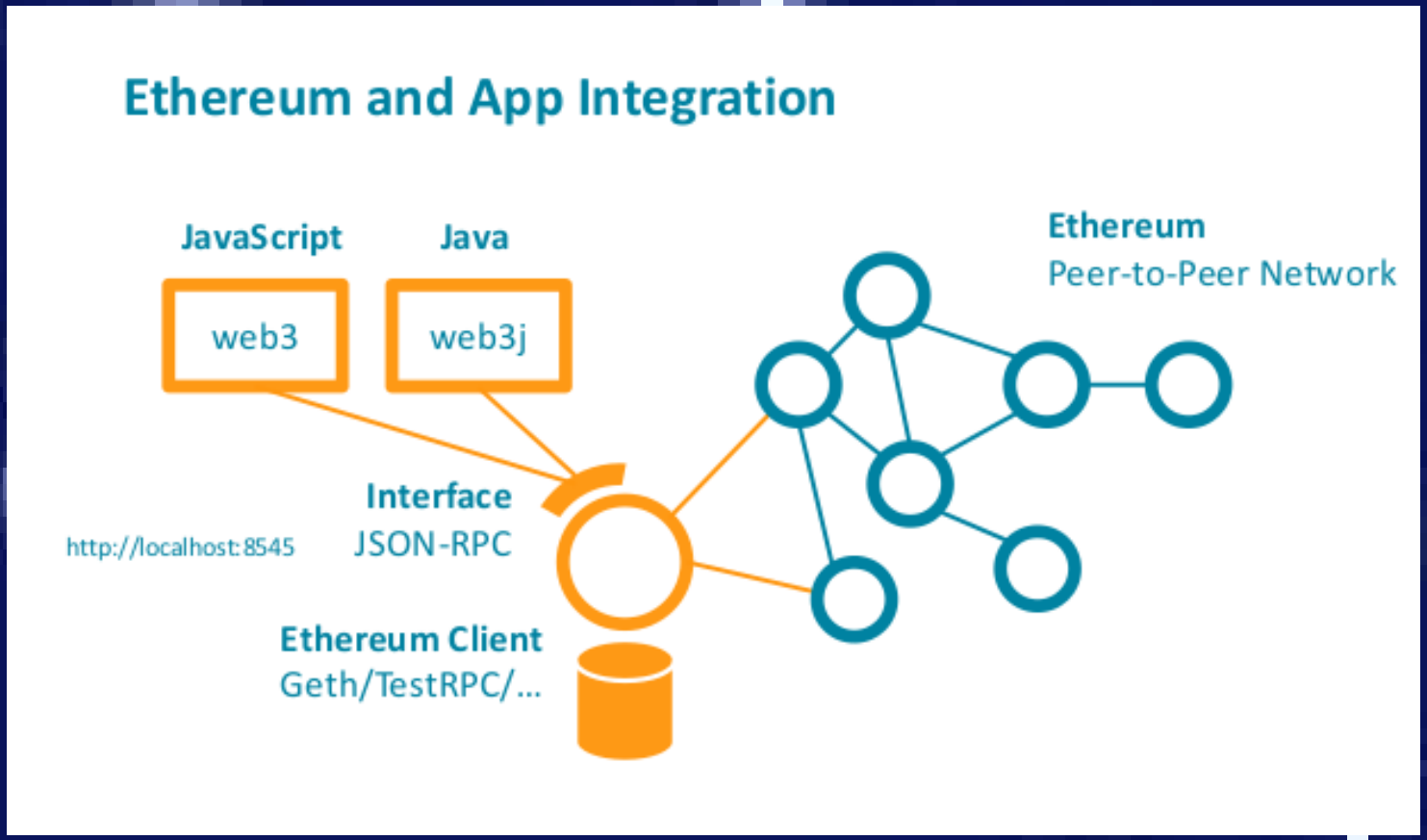
- Diferentes clientes (geth, parity, ...)
- Copia parcial de la blockchain (cabeceras bloque)
- Depende de nodos completos para comprobaciones y acceso al árbol de estado

ARQUITECTURA ETHEREUM

ESTRUCTURA CLIENTE ETHEREUM



INTEGRACION GETH -->APP ETHEREUM



- Interfaces:
  - JSON-RPC
  - Línea de comandos (geth --help)
  - Consola javascript
- Servidor por defecto: http://localhost:8545

GETH: IMPLEMENTANDO UN NODO

Es realmente el interface de linea de comando para ejecutar un full o Light Ethereum node implementado en GO

https://github.com/ethereum/go-ethereum/wiki/geth

Nos permite:

- **Minar real ether**
- **Transferir fondos entre direcciones**
- **Crear contratos y enviar transacciones**
- **Explorar todos la historia de la cadena de bloques**

¿Por qué elegimos Geth?

Cliente más extendido

Compatibilidad multiplaforma

Nos permite generar nodos redes privadas

El cliente de Ethereum es válido tanto para la red principal (Main net), como para diferentes implementaciónes (Ropsten, Private chain...)

PARAMETROS GETH

Flags

La ejecución de Geth admite numerosos "flags", algunos de los más importantes:

Flag	Descripción	Por defecto
--networkid	ID de la blockchain	0 (main net)
--rpc	Permiso a la interfaz RPC	true
--datadir	Path que almacena la blockchain	(Linux) .ethereum/chaindata
--rpcapi	APIs abiertas vía RPC	web3
--rpcport	Puerto de acceso a RPC	8545
--rpccorsdomain	Dominios de acceso a RPC	localhost
--port	Puerto de conexión para otros nodos	30303

TIPOS DE TESTNET DISPONIBLES

En nuestro caso recomendamos usar RINKEBY ya que da soporte a geth.Tambien esta ROPSTEN que soporta Geth pero es casi imposible conseguir monedas de prueba.

- Ejemplo: arrancar nodo

```
$ geth --testnet --syncmode "fast" --rpc --rpcapi
"admin,eth,miner,net,personal,web3"
```

Consola GETH

Hoja de ruta: Probamos.....

```
geth
 . command line

Wallet
 . create account
 . backup
 . check balance
 . sendTransaction

@raspberrypi:~$
```

GETH

Creacion de una Blockchain privada

Dos recursos necesarios para comenzarla:

Network id: Elegid el que deseeis

Archivo genesis: Parámetros que definirán finalmente la blockchain (Genesis block):

gasLimit: Valor que define el total de gas que puede ser gastado en un bloque

difficulty: Valor que define el target de dificultad (Recordar dificultad mining Bitcoin). En una privada, valor bajo (minado rápido)

alloc: Define wallets con prelocalización de Ether Información de los parámetros encontrados en el genesis block pueden ser encontrados en el yellow paper

EJEMPLO DE ARCHIVO GENESIS

- Ejemplo archivo génesis: genesis1.json

```
{

"nonce": "0x0000000000000042",
"timestamp": "0x0",
"parentHash":
"0x0000000000000000000000000000000000000000000000000000000000000000",
"extraData": "0x0",
"gasLimit": "0x8000000",
"difficulty": "0x400",
"mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
"coinbase": "0x3333333333333333333333333333333333333333",
"alloc": {  "3282791d6fd713f1e94f4bfd565eaa78b3a0599d": {
      "balance": "1337000000000000000000"
    },
    "17961d633bcf20a7b029a7d94b7df4da2ec5427f": {
      "balance": "22942700000000000000"
    }
  }
}
```

Inicialización de una blockchain privada

1: Inicialización del bloque génesis:

```
geth --datadir </path/a/bloques/> init genesis.json
Ejemplo:
geth --datadir datos --networkid 123 --nodiscover --maxpeers 0 init
genesis1.json
```

2: Ejecución de la blockchain:

```
$ geth + Flags...
```

Despliegue de una blockchain privada/ Testnet

Crear una blockchain privada con los siguientes parámetros.

Flag	Valor
--networkid	25052017
--rpc	true
--datadir	$HOME/documents/blockchain/private_blockchain
--rpcapi	web3
--rpcport	8546
--rpccorsdomain	localhost
--port	30303

PERO: ¿Y Si elegimos una red Testnet como Rinkeby? :

Podemos arrancar en lugar de utilizar un ID privado, con una testnet: **rinkeby,** con un bloque genesis desde:

```
>wget  -c https://www.rinkeby.io/rinkeby.json

geth --datadir=$HOME/.rinkeby --light init rinkeby.json
```

Despliegue de una blockchain Testnet Rinkeby

Como arrancar ahora nuestro nodo en la red Rinkeby.

```
geth --networkid=4 --datadir=$HOME/.rinkeby --cache=1024 --syncmode=light
--ethstats='yournode:sioux-ethereum@stats.rinkeby.io'
--bootnodes=enode://a24ac7c5484ef4ed0c5eb2d36620ba4e4aa13b8c84684e1b4aa
b0cebea2ae45cb4d375b77eab56516d34bfbd3c1a833fc51296ff084b770b94fb9028c4
d25ccf@52.169.42.101:30303
```

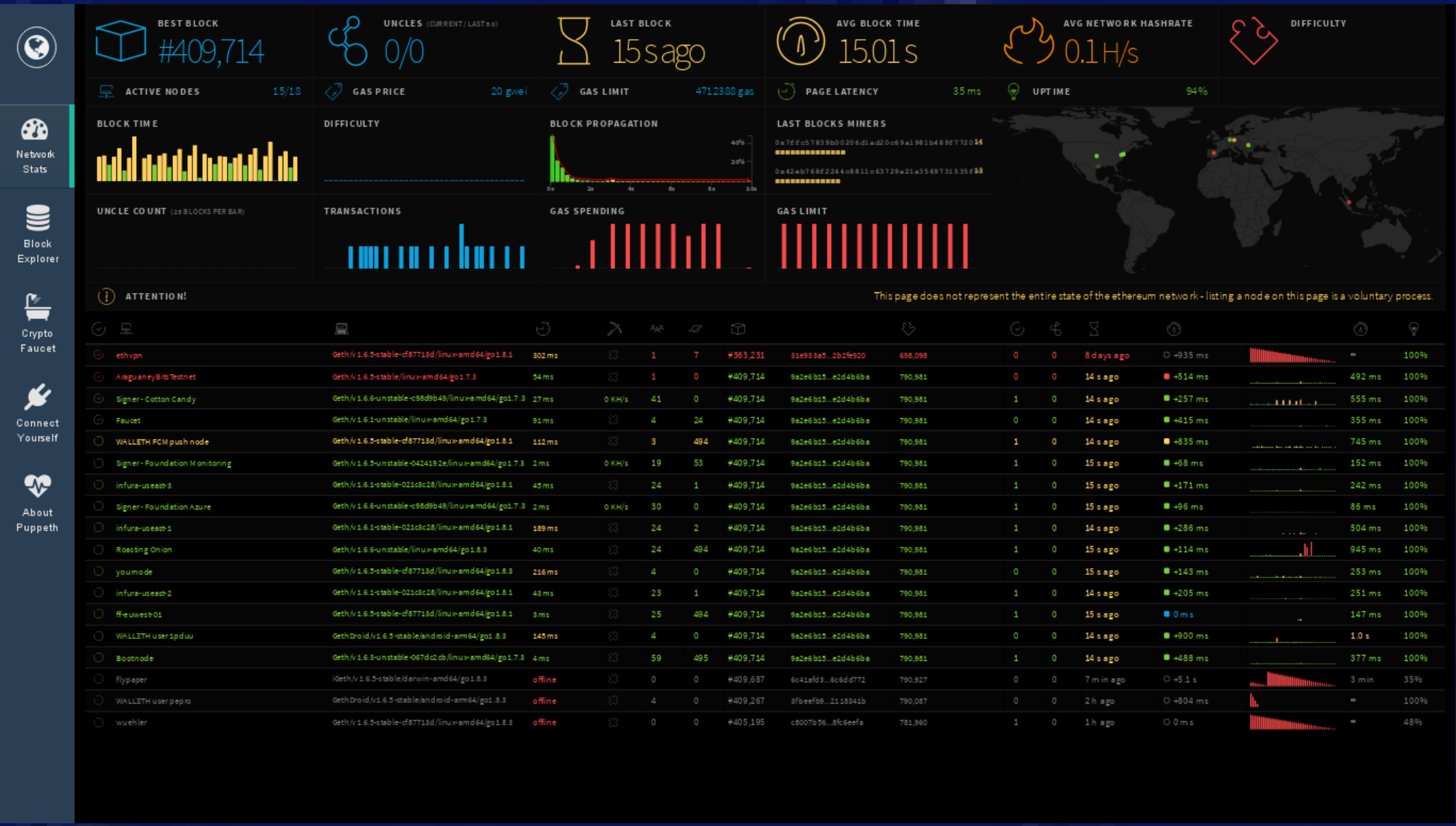
Conectar a la consola geth mediante el endpoint IPC, que aparece al arrancar:

En este caso hemos observado nuestro endpoint IPC en el arranque anterior.
Conectamos: --->

```
geth attach ipc:/home/pi/.rinkeby/geth.ipc
```

NUESTRO NODO EN LA RED RINKEBY

https://www.rinkeby.io/#stats



TestRPC (Ahora se llama Ganache)

Incluso las blockchain privadas son demasiado lentas/ineficientes a la hora de desarrollar, asi que podemos simularla.

testRPC es una Librería en JS que simula un nodo completo de una blockchain privada, o una blockchain real a partir de determinado punto

Para el desarrollo de aplicaciones de prueba, se utilizará esta librería.

```
$ testrpc &
```

Abre servicio WEB3 JSON-RPC: port 8545
--→ Listening on localhost:8545
Nos genera bloque genesis, varios accounts wallet, palabras semilla, etc...

TestRPC (Ahora se llama Ganache)

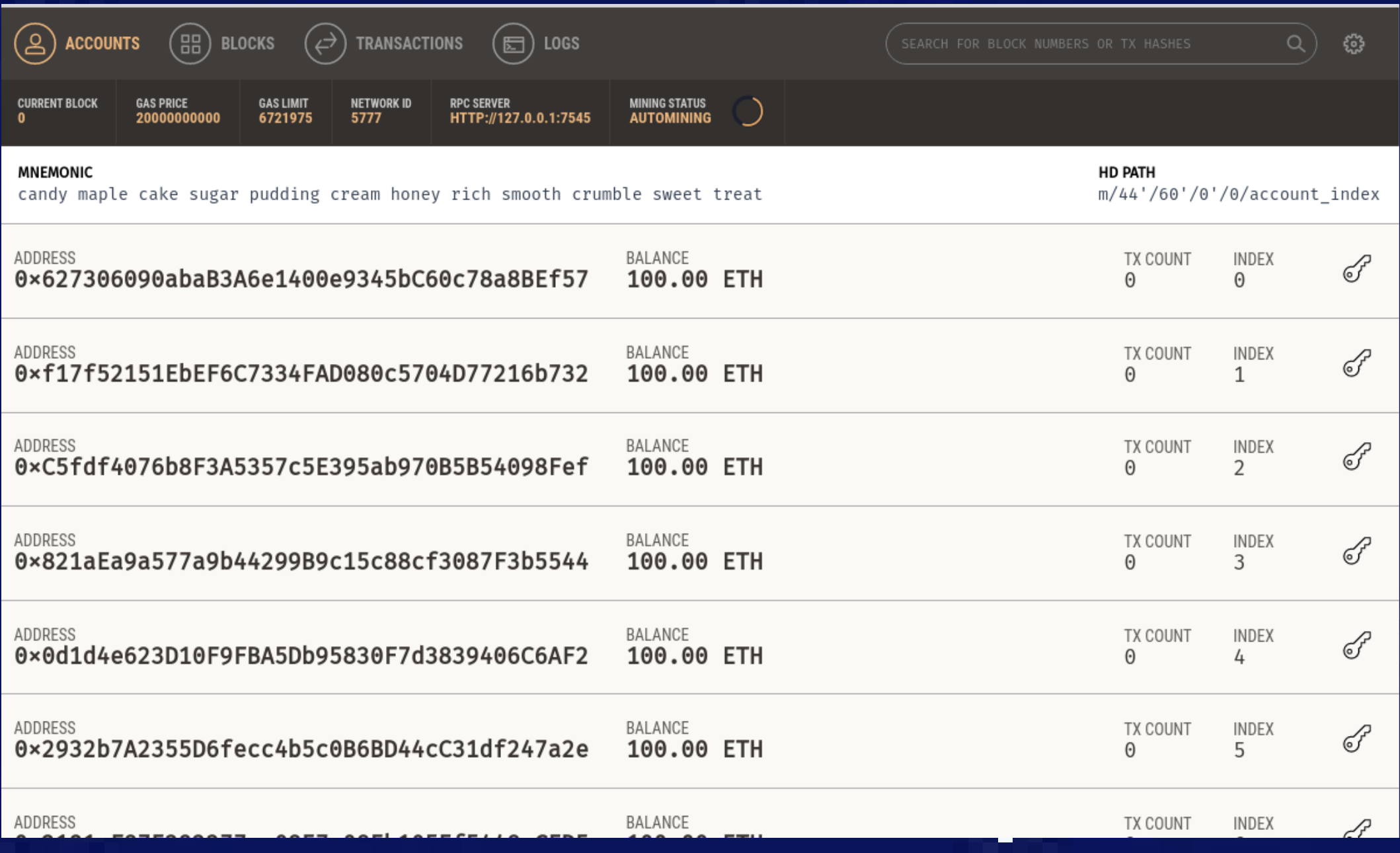
```
(4) 3c2eff069bf69a9108a4168fc3dc496ee7790474358ac5005c465df3f02f9027
(5) 63c57f8441630354ac43f01de5405c625779ad8b93955beab11b644279739597

EthereumJS TestRPC v4.1.3 (ganache-core: 1.1.3)

Available Accounts
==================
(0) 0xcecf0deeb7e66c93e7777e62d1a2bac2bccbaa45
(1) 0xb6b51bfe167d4824c78eaa2c935505edb94fdff8
(2) 0x641d9466b62e373faad347d8dec0ca621de544c9
(3) 0xeb0d71a5f4685f77d58e521e186ee31c193a3d56
(4) 0x8e689117726858l6a0df53094284d63b96b2566f
(5) 0x95613767857c04c1c118a72f8cab6e605f88a87c
(6) 0xa29d408dcb41db3f50e6477f9a960d25c8085847
(7) 0x1cf256b0bf9666d6fddf0c6ca289d2dc63120755
(8) 0x363ca6a071bc02c44e0a18af95d52c5924c131f0
(9) 0x72b3ef060a0c9ba08f1b37203d664e6e0fad7e99

Private Keys
==================
(0) 9c97a646f2aedd08ac0ae22fd6ec1c836b6a3b13a4b7e491bb5a9a07bb8e64c6
(1) 8a1296336098de695c4d1cf976af5794b8f1c203b34da4b534526bfb5b44b51d
(2) f1ff84f0ce7886e63e8ee7c8b3a3994f2567674a400f9f2ad09f331321f08bf2
(3) 2e431377789ba718fb81634129d3e2e38a26f4155b83ce20f655c8cfd0eddf19
(4) 3c2eff069bf69a9108a4168fc3dc496ee7790474358ac5005c465df3f02f9027
(5) 63c57f8441630354ac43f01de5405c625779ad8b93955beab11b644279739597

EthereumJS TestRPC v4.1.3 (ganache-core: 1.1.3)
```

TestRPC (Ahora se llama Ganache)

Options (From the Docs):

-**a or –accounts**: Specify the number of accounts to generate at startup.
-b or --blocktime: Specify blocktime in seconds for automatic mining. Default is 0 and no auto-mining.
-d or --deterministic: Generate deterministic addresses based on a pre-defined mnemonic.
-**m or –mnemonic:** Use a specific HD wallet mnemonic to generate initial addresses.
-**p or --port:** Port number to listen on. Defaults to 8545.
-**h or --hostname:** Hostname to listen on. Defaults to Node's server.listen() default.
-**s or --seed:** Use arbitrary data to generate the HD wallet mnemonic to be used.
-**g or --gasPrice:** Use a custom Gas Price (defaults to 20000000000)
-**l or --gasLimit:** Use a custom Gas Limit (defaults to 0x47E7C4)
-**f or –fork:** Fork from another currently running Ethereum client at a given block. Input should be the HTTP location and port of the other client, e.g. http://localhost:8545. You can optionally specify the block to fork from using an @ sign: http://localhost:8545@1599200.

Ganache Grafico



Conectando a la Blockchain via Consola GETH

Una vez inicializado y corriendo la blockchain, encontraréis un archivo .ipc. Conectar a la blockchain a través de IPC, en el directorio de datos que hayamos lanzado nuestra cadena de bloques:

```
$ geth attach ipc: /your/path/to/your/ipc
```

¡ Ya estamos conectados a nuestro cliente y podemos comunicarnos via Web3!

Nota: Dependiendo de tu sistema operativo, encontrarás tu **chaindata** subdirectorio en:

Linux *- $HOME/.ethereum/geth*
OS/X *- $HOME/Library/Ethereum or HOME/Library/Ethereum/geth*
Windows *- %APPDATA%/Ethereum*

Consola GETH

A través de la consola podemos acceder a diferentes funciones de Web3 y gestión de claves.

Probar interfaces utilizando como ejemplos, veremos algunos en breve:

```
> admin // Gestión de la red
> eth // web3.eth
> personal //Gestión de cuentas y claves personal
> miner //Gestión de minería
```

Realmente son Managament APIs no disponibles via RPC

Ethereum Accounts (Cuentas)

- Agentes autónomos que “viven” en el blockchain
- Tiene una dirección Ethereum de 20 bytes, con:
  - Un nonce usado para que cada transacción pueda ser procesada una sola vez
  - El balance de ether de la cuenta
  - El código de contrato, si existe
  - El almacenamiento de la cuenta

Cuentas de dueño externo

- No tienen código de contrato
- Se usan enviando mensajes firmados por el dueño

Contratos

- Se ejecuta el código cuando llega un mensaje
- Pueden leer/escribir su almacenamiento, enviar mensajes y crear nuevos contratos

Ethereum Accounts y Nonces

Como son usados los Nonces:

-->Each account has a nonce value (account state data)

-->Accounts start with nonce value 0

-->TX: includes sender address and its nonce value

--> TX can only be mined if:

- **Account has sufficient funds**
- **TX nonce == current account nonce**

If TX is mined successfully: Nonce increased by 1

Tipo de cuentas que existen en el World State

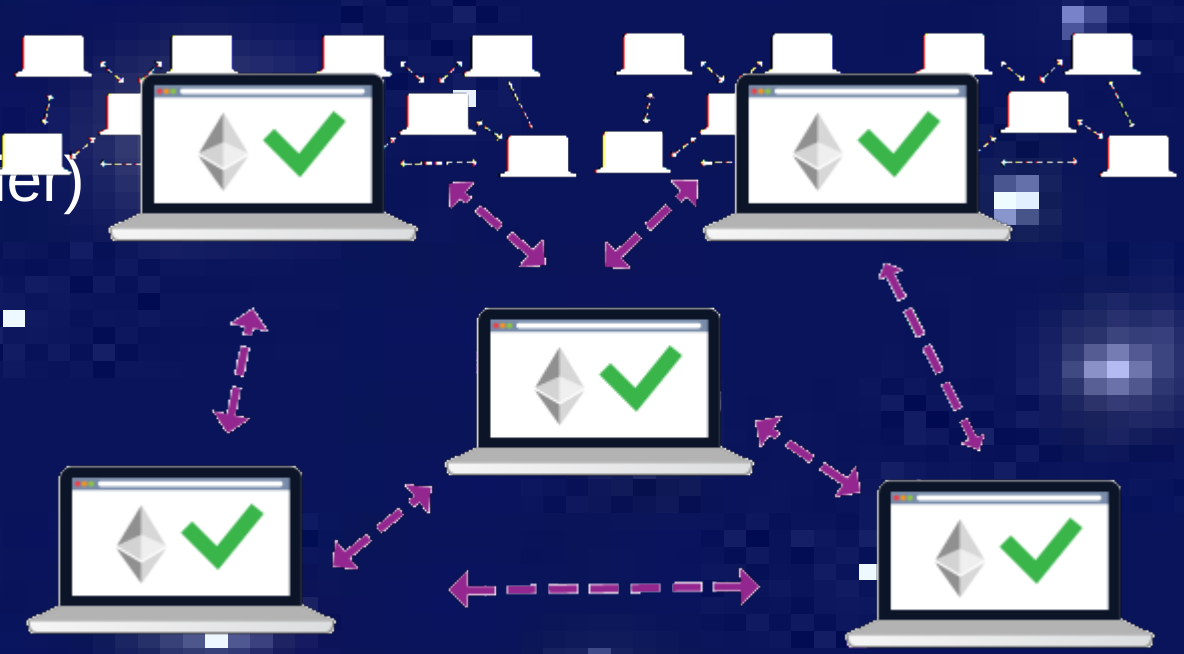
- **Externally Owned Account**

- Address (160-bit identifier)
- Nonce
- Balance

- **Contract Account**

Address (160-bit identifier)
Nonce
Balance
Storage
Code

State History is represented by the blockchain



Consola

Creación de una cuenta

1: A través de la interfaz eth, podemos ver las cuentas disponibles utilizando:

```
> eth.accounts // Array
> eth.accounts[0]  //Primera cuenta
```

2: Podemos crear una cuenta desde la consola, utilizando 'personal'

```
> personal.newAccount()
```

3: Como respuesta, nos pedirán una contraseña, que cifrará la clave privada almacenada

```
Passphrase:
Repeat passphrase:
```

Consola

Importación de una cuenta

1: De la misma manera que hemos creado una cuenta, podemos importarlas de carteras externas. Para ello, únicamente necesitamos realizar:A través de la interfaz eth, podemos ver las cuentas disponibles utilizando:

```
> geth account import <keyfile>    --->claveprivada.txt
Testnet:  geth --testnet account import  claveprivada.txt

Comprobando...: web3.eth.accounts
```

Exportación de una cuenta

Para exportar una cuenta y tratarla desde una wallet diferente,como Etherwallet, únicamente necesitamos utilizar el archivo alojado en keystore.

```
Keystore Path en tu Gnu/Linux  ----> ~/.ethereum/testnet/keystore/
```

TRANSACCIONES

- para enviar un mensaje de una cuenta de dueño externo
- Contiene los siguientes datos:
  - Destino del mensaje
  - Firma identificando el emisor
  - Monto de ether a transferir
  - Campo opcional de datos
  - Valor *STARTGAS*
  - Valor *GASPRICE*
- Protección ante DoS:

STARTGAS

Número máximo de pasos computacionales a ejecutar en la transacción (incluyendo sub-ejecuciones de mensajes)

- cálculo (CPU), ancho de banda y almacenaje

GASPRICE

Fee que el emisor está dispuesto a pagar por cada paso computacional

PROCESO DE TRANSACCIONES

1. Validar que transacción está bien formada (número de valores, firma, nonce es igual al nonce de la cuenta del emisor
2. Calcular transaction fee como STARTGAS * GASPRICE, sustraer ese monto del balance del emisor, incrementar el nonce del emisor. Error si no hay suficiente saldo.
3. Inicializar GAS = STARTGAS, restar costo por byte en la transacción
4. Transferir el valor especificado en tx desde el emisor al receptor
   - Si cuenta destino no existe, crearla
   - Si destino es contrato, ejecutar su código
5. Si ejecución falla (fondos insuficientes; se acabó el GAS), se revierten los cambios salvo transaction fee
6. Devolución de GAS no gastado y pago de fees al minero

TRANSACCIONES

Recordando la estructura de las transacciones:

Recipiente
Firma
Cantidad de Ether a enviar
Campo datos(opcional)

Al realizar una transacción, desde consola, debemos especificar:

Emisor-from (cuenta que vamos a utilizar para la transacción)

Receptor-to (cuenta que va a recibir el ETH)
Valor-value (cantidad que vamos e enviar, en wei)

Consola

Consulta de balance

Desde la consola, consultamos el balance o saldo de la cuenta que hemos creado:

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

Podemos pasarlo  a wei:

web3.fromWei(eth.getBalance(eth.accounts[0]))
```

Consola

Transacciones

Todo esto, utilizando la consola con javascript, se realizaría de la siguiente manera, si queremos utilizar variables:

```
> var sender = eth.accounts[0];

var receiver = eth.accounts[1];

var amount = web3.toWei(0.01, "ether")

eth.sendTransaction({from: sender, to: receiver, value: web3.toWei(1, "ether")})
```

Obteniendo Ethers: Faucet Rinkeby

https://faucet.rinkeby.io/

Solicitando Faucet en la Testnet

Faucet: https://faucet.rinkeby.io/

Para obtener Ether, podemos utilizar un faucet, un servicio web al que le solicitamos envíe ethers a nuestra cuenta creada en la red Rinkeby.

Actualmente necesitamos crear una URL social:
Para poder usar esta operacion creamos un **Twitter**, por ejemplo:

```
Requesting faucet funds into 0x0000000000000000000000000000000000000000
on the #Rinkeby #Ethereum test network.
```

Copiar la URL o enlace de nuestro Twitter en “Crypto Faucet” de Rinkeby.io y pedimos las monedas que necesitemos, 3 ether, podemos usar ***etherescan.io*** para verificar la transacción.

Consultamos el balance recibido:

```
web3.eth.getBalance('0x4d23aa9fc9191c56e45501e6a7955d4a8ab6505ª')
```

Consola GETH

Nuestra primera transaccion

Todo esto, utilizando la consola con javascript, se realizaría de la siguiente manera, sin variables, directamente:

Ejemplo:

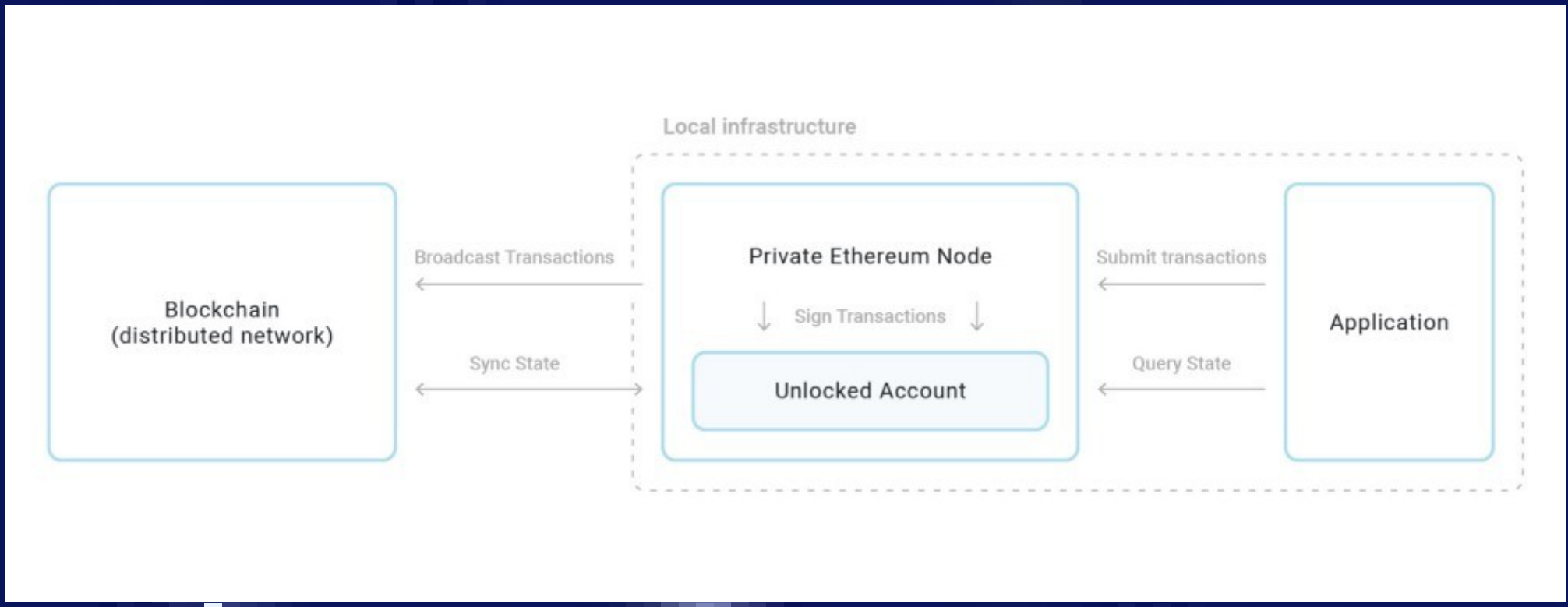
```
 eth.sendTransaction({from: eth.accounts[0], to: "receiver", value: amount})
```

Lanzamos una transaccion Real:

```
> eth.sendTransaction({from: eth.accounts[0], to: "0xb6536c3b131117ac2c1016a8a2e893e842750265", value: 1000})
```

TRANSACCION Y CUENTA EN NODO PRIVADO

Desbloqueo Balance de Cuenta para transaccion



Consola GETH

Bloqueos o problemas

Quizás, ¿ deberiamos de tener problemas de bloqueo y autenticacion a la cuenta?:

Probamos a desbloquear::

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

....Y repetimos la ultima transferencia... ¿Resultado?
¿Por qué no se ha realizado?

```
eth.sendTransaction({from: eth.accounts[0], to: "...", value: "10",
gas:22000,gasPrice:web3.toWei(45,"Shannon")})
```

¿Pudiera ser que el balance total no ha podido ser enviado porque no hay suficiente ETHER residual para ser usado como gas para la transaccion.?

Consola GETH

Problemas con el Gas.

Para un contrato, por ejemplo, podriamos estimar el gas con:

web3.eth.estimateGas

Calcula el gas que ejecuta el contrato localmente, por lo que cuando se extrae la transacción, el resultado puede ser diferente, generalmente se agrega un extra para asegurarse de que funcione.

Para determinar el gas a usar , podemos utilizar:
web3.eth.gasPrice

Para un analisis mas detallado, puedes estudiar:
http://ethgasstation.info

(Este site no parece proveer ninguna API, compruebalo)

Consola GETH

SOLUCION PROPUESTA:

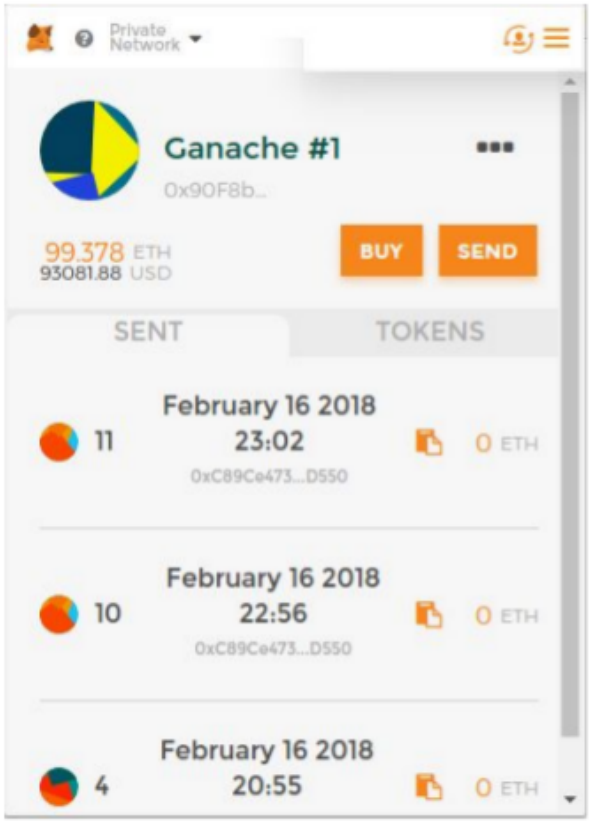
Para un transaccion, por ejemplo, podria quedar algo asi:

```
transactionObject = {
  from: sender,
  to: receiver,
  value: totalBalance
}

var estimatedGas= eth.estimateGas(transactionObject);
web3.eth.getGasPrice(function(error, result){
  gasPrice = result.toNumber();
  var gasValue = gas * gasPrice
  var valueToSend = totalBalance - gasValue;
  });
```

Ejecutamos el objeto:

```
> web3.eth.sendTransaction(transactionObject);
```

Metamask

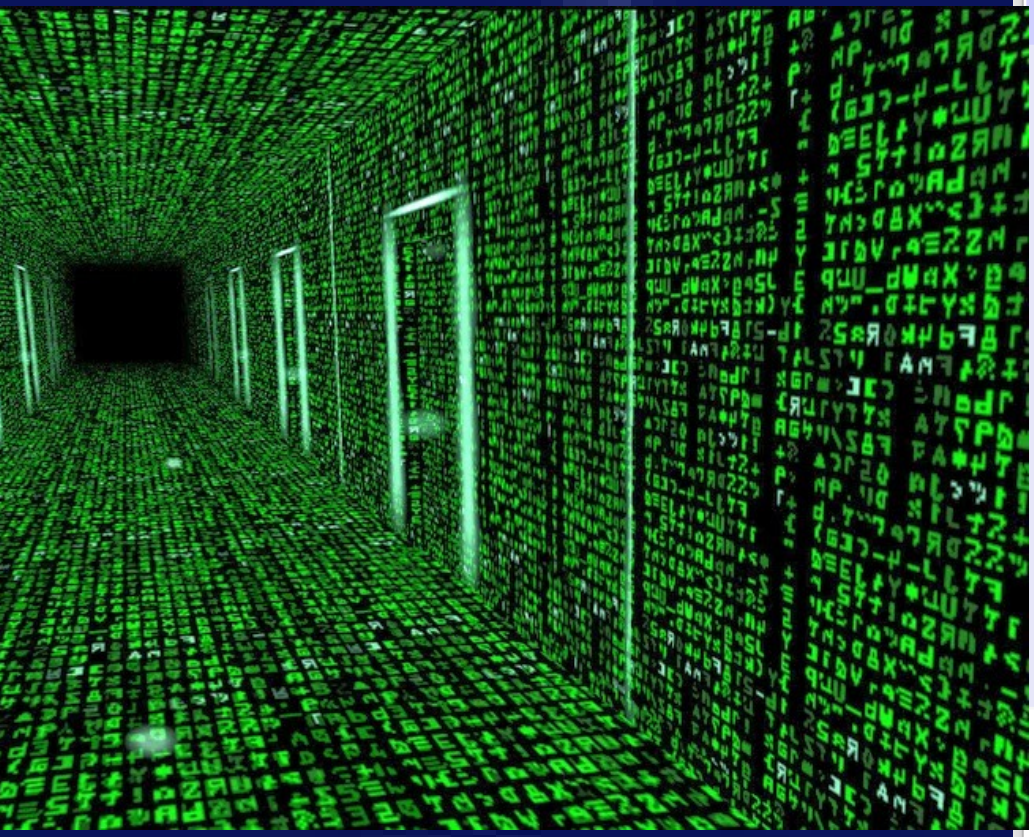


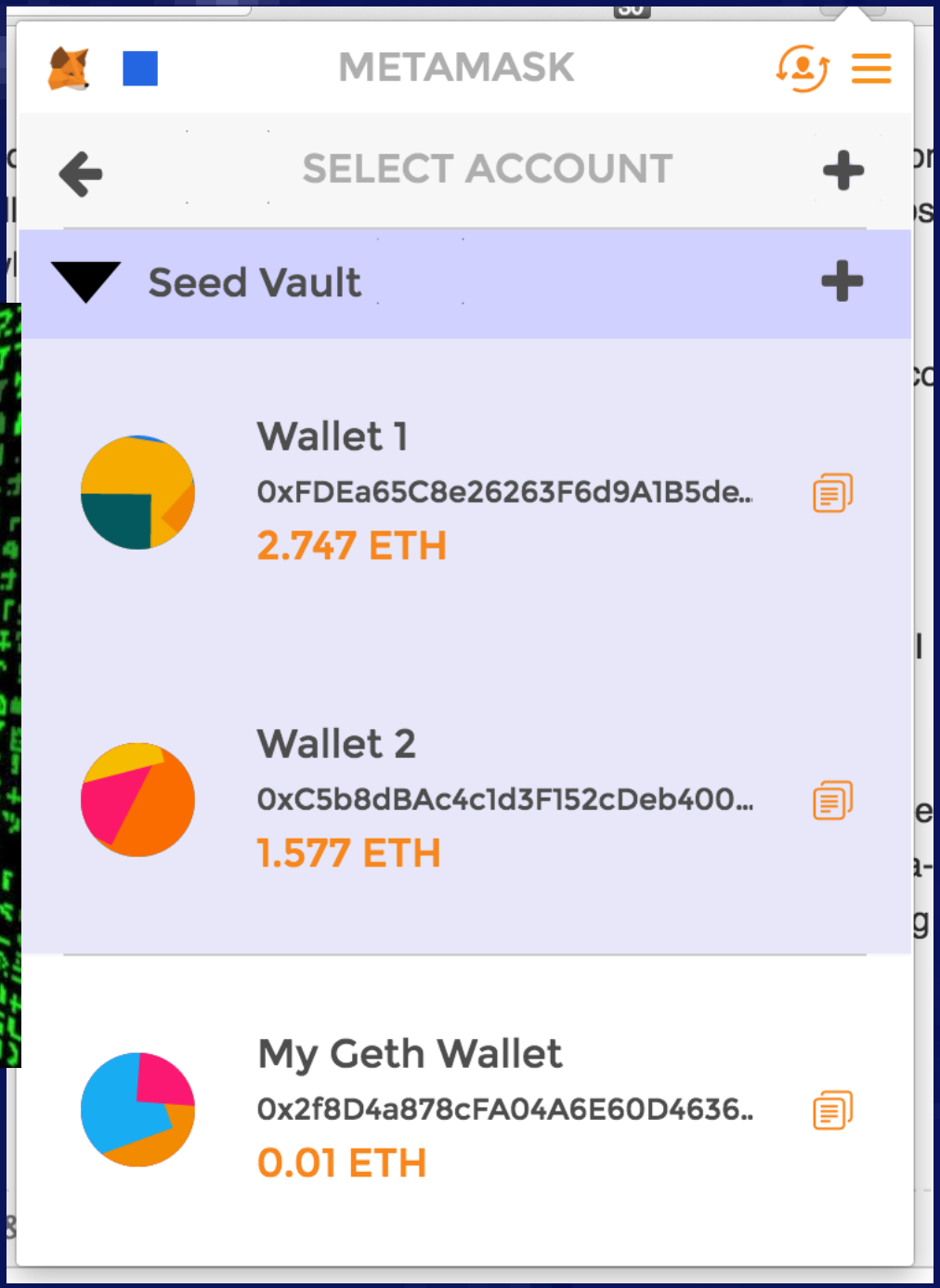
https://metamask.io/

Extensión para conectar una Dapp con un nodo Ethereum

MetaMask injects web3 object and convenience Web3.js library into the javascript context.

JUGUEMOS CON WALLETS DE METAMASK






Creditos:
Pedro Romero Used
hcluster@gmail.com
www.linkedin.com/in/pedro-romero-used
Luis Carlos García
luisc.garcia@gmail.com


THANK YOU
GAME OVER