Lecture 05:
Web3: Interacting with Ethereum

Akash Khosla
Nick Zoghb

BLOCKCHAIN
AT BERKELEY

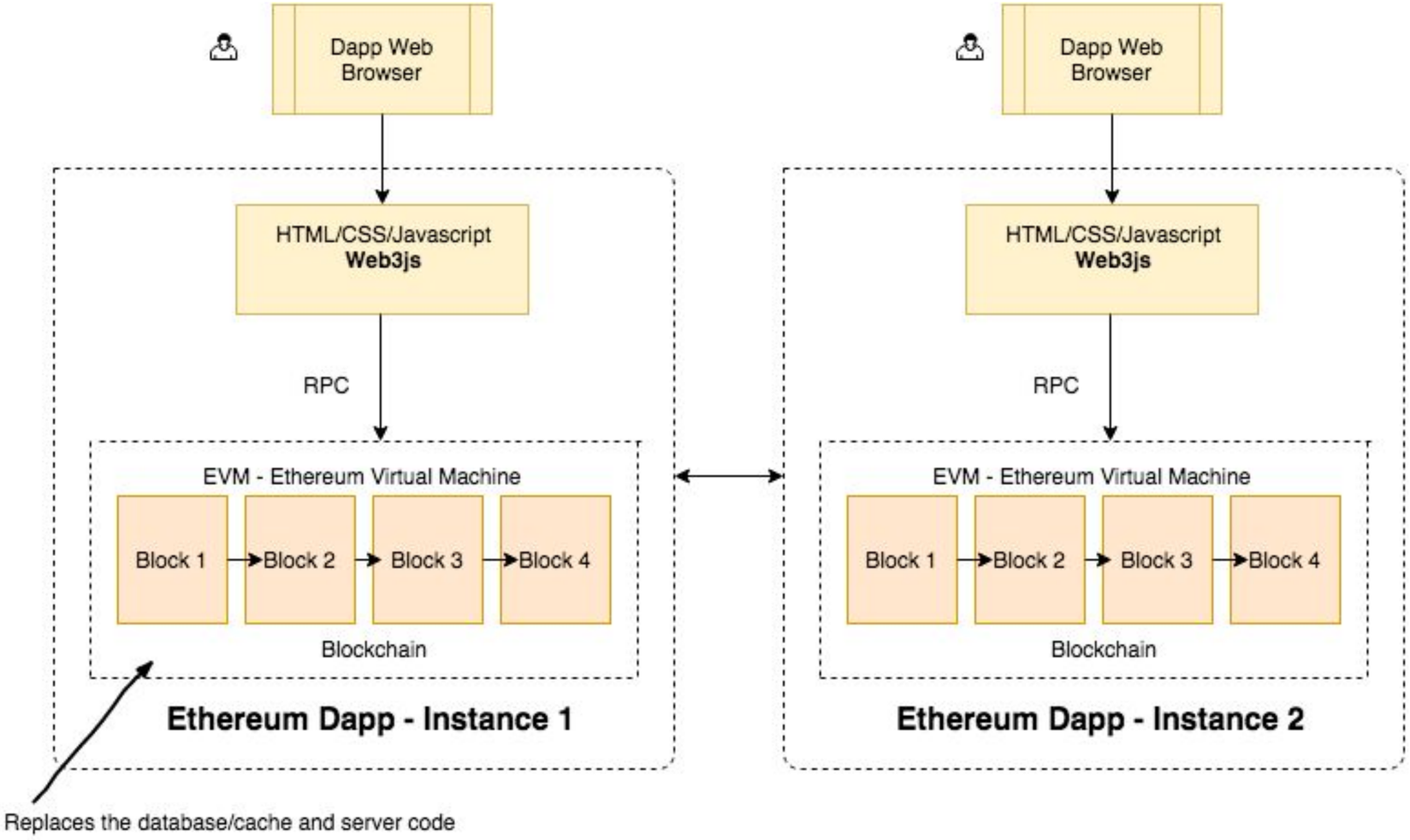
LECTURE OUTLINE

1. WEB3: THE ETHEREUM DAPP API
2. NETWORKS
3. DAPP ARCHITECTURE
4. LET'S BUILD ETHERSCAN

1 WEB3: THE ETHEREUM DAPP API

WHERE ARE WE IN THE STACK?

RIGHT IN THE MIDDLE



AUTHOR: AKASH KHOSLA

HAVE WE BEEN USING THIS?

KIND OF, BUT NOT REALLY

- web3.js is the entrance to the Ethereum blockchain from the client side of a Đapp
- Communicates to nodes on network using RPC (Remote Procedure Calls)
- web3 contains both the eth object and the shh object
  - **`web3.eth`** - for Ethereum blockchain interaction
    - **`web3.eth.abi`**
    - **`web3.eth.accounts`**
    - **`web3.eth.personal`**
  - **`web3.shh`** - for Whisper interaction
  - **`web3.bzz`** - for Swarm interaction

AUTHOR: AKASH KHOSLA

THE ETHEREUM ABI

EXPOSING CONTRACT METHODS

ABI = Application Binary Interface

- An ABI is how you call functions in a contract and get data back
  - It determines how functions are called and in which binary format information should be passed from one program component to the next
- Why is it necessary?
  - You need a way to specify which function in the contract to invoke as well as guarantee to what type of data is returned
- Not part of the core Ethereum protocol, you can define your own ABI - but easier to comply with format provided by web3.js

THE ETHEREUM ABI

ANOTHER ANALOGY

API = Application Programming Interface

ABI = Application Binary Interface

- So therefore an *ABI* is an *API* at a lower level?
- Contract code is stored as *bytecode* in binary form on the blockchain under a specific address
  - You can access the binary data in the contract through the ABI
  - The ABI defines the structures and methods that you will use to interact with binary contract (just like an API)

Read more here.

THE ETHEREUM ABI

THE LOVELY ABI FORMAT

```
contract Test {
function Test(){ b = 0x1234567890123456789012345678901234567890; }
event Event(uint indexed a, bytes32 b);
event Event2(uint indexed a, bytes32 b);
function foo(uint a) { Event(a, b); }
bytes32 b;
}
```

```
[{
"type":"event",
"inputs": [{"name":"a","type":"uint256","indexed":true},{"name":"b","type":"bytes32","indexed":false}],
"name":"Event"
}, {
"type":"event",
"inputs": [{"name":"a","type":"uint256","indexed":true},{"name":"b","type":"bytes32","indexed":false}],
"name":"Event2"
}, {
"type":"function",
"inputs": [{"name":"a","type":"uint256"}],
"name":"foo",
"outputs": []
}]
```

AUTHOR: AKASH KHOSLA

WHAT CAN WEB3.JS DO?

EVERYTHING

```
52      function onAddressKeyUp() {
53          var address = document.getElementById('address').value;
54          document.getElementById('nameOf').innerText = web3.eth.namereg.name(address);
55      };
56
57      function onNameKeyUp() {
58          var name = document.getElementById('name').value;
59          document.getElementById('addressOf').innerText = web3.eth.namereg.addr(name);
60      };
61
62  </script>
63  </head>
64  <body>
65      <i>This example shows only part of namereg functionalities. Namereg contract is available <a href="https://github.com/ethereum/dapp-bin
66      </i>
67      <h1>Namereg</h1>
68      <h3>Search for name</h3>
69      <div>
70          <text>Address: </text>
71          <input type="text" id="address" onkeyup="onAddressKeyUp()"></input>
72          <text>Name: </text>
73          <text id="nameOf"></text>
74      </div>
75      <h3>Search for address</h3>
76      <div>
77          <text>Name: </text>
78          <input type="text" id="name" onkeyup="onNameKeyUp()"></input>
```

Read more [here]

AUTHOR: NICK ZOGHB

WHAT CAN WEB3.JS DO?

EVERYTHING: REACT.JS

JavaScript

```
52      function onAddressKeyUp() {
53          var address = document.getElementById('address').value;
54          document.getElementById('nameOf').innerText = web3.eth.namereg.name(address);
55      };
56
57      function onNameKeyUp() {
58          var name = document.getElementById('name').value;
59          document.getElementById('addressOf').innerText = web3.eth.namereg.addr(name);
60      };
61
```

JavaScript

HTML

```
62  </script>
63  </head>
64  <body>
65      <i>This example shows only part of namereg functionalities. Namereg contract is available <a href="https://github.com/ethereum/dapp-bin
66      </i>
67      <h1>Namereg</h1>
68      <h3>Search for name</h3>
69      <div>
70          <text>Address: </text>
71          <input type="text" id="address" onkeyup='onAddressKeyUp()'></input>
72          <text>Name: </text>
73          <text id="nameOf"></text>
74      </div>
75      <h3>Search for address</h3>
76      <div>
77          <text>Name: </text>
78          <input type="text" id="name" onkeyup='onNameKeyUp()'></input>
```

HTML

AUTHOR: NICK ZOGHB

WHAT CAN WEB3.JS DO?

EVERYTHING

- It is the official DApp API that is run on all of the Ethereum nodes
  - Interfaces with Ethereum nodes from the network using **JSON-RPC** calls
- **Main Capabilities**:
  - Interact with contract functions
  - Deploy contracts
  - Send **raw transactions** to contracts with extra data
  - Query the blockchain for data
    - Includes logged **events** by any contract along with block data
- Can have our back-end on chain, and our interface off chain
- Use GitHub pages to host my apps with server level functionality? We'll see.

- getProtocolVersion
- isSyncing
- getCoinbase
- isMining
- getHashrate
- getGasPrice
- getAccounts
- getBlockNumber
- getBalance
- getStorageAt
- getCode
- getBlock
- getBlockTransactionCount
- getUncle
- getTransaction
- getTransactionFromBlock
- getTransactionReceipt
- getTransactionCount
- sendTransaction
- sendSignedTransaction
- sign
- signTransaction
- call
- estimateGas
- getPastLogs
- getCompilers

WHAT CAN WEB3.JS DO?

EVERYTHING: EVENTS

Example of Web3 instantiation and event listening:

```
var Web3 = require('web3');
var web3 = new Web3();
web3.setProvider(new web3.providers.HttpProvider("http://localhost:8545"));

window.onload = function () {
  var filter = web3.eth.namereg.Changed();
  filter.watch(function (err, event) {
  // live update all fields
    onAddressKeyUp();
    onNameKeyUp();
    onRegisterOwnerKeyUp();
   });
};
```

WHAT CAN WEB3.JS DO?

EVERYTHING: EVENTS

Example of Web3 instantiation and event listening in web3 v1.0:

```
myContract.once('MyEvent', {
    filter: {myIndexedParam: [20,23], myOtherIndexedParam:
'0x123456789...'}, // Using an array means OR: e.g. 20 or 23
    fromBlock: 0
}, function(error, event){ console.log(event); });
```

WHAT ELSE CAN WEB3.JS DO?

EVERYTHING: SMART CONTRACTS

Example of writing and compiling a Solidity smart contract:

```
// solidity code code
var source = "" +
"contract test {\n" +
"   function multiply(uint a) constant returns(uint d) {\n" +
"       return a * 7;\n" +
"   }\n" +
"}\n";
var compiled = web3.eth.compile.solidity(source);
var code = compiled.code;
// contract json abi, this is autogenerated using solc CLI
var abi = compiled.info.abiDefinition;
var myContract;

...
```

AUTHOR: NICK ZOGHB

(Source)

BLOCKCHAIN AT BERKELEY

WHAT ELSE CAN WEB3.JS DO?

EVERYTHING: SMART CONTRACTS

Example continued:

```
...

web3.eth.contract(abi).new({data: code}, function (err, contract) {
if(err) {
  console.error(err);
  return;
} else if(contract.address) {
  myContract = contract;
  console.log('address: ' + myContract.address);
  document.getElementById('status').innerText = 'Mined!';
  document.getElementById('call').style.visibility = 'visible';
});
```


(Source)

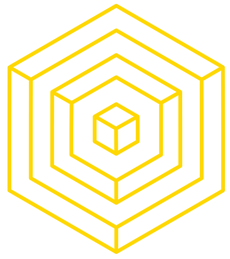
JSON-RPC API

CONTEXT BEHIND ETHEREUM'S APIs

- JSON is a lightweight data-interchange format
- RPC is used to make *remote function calls*
- **JSON-RPC** is a stateless, light-weight remote procedure call (RPC) protocol.
  - Defines several data structures and the rules around their processing.
  - It is transport agnostic in that the concepts can be used within the same process, over sockets, over HTTP, or in many various message passing environments

Side note:

Serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer) or transmitted (for example, across a network connection link) and reconstructed later (possibly in a different computer environment).



...AND MORE

READ THE DOCS

getBlockNumber

```
web3.eth.getBlockNumber([callback])
```

Returns the current block number.

Returns

`Promise` returns `Number` - The number of the most recent block.

Example

```
web3.eth.getBlockNumber()
.then(console.log);
> 2744
```

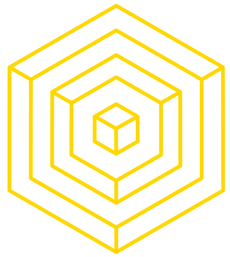
getBalance

```
web3.eth.getBalance(address [, defaultBlock] [, callback])
```

Get the balance of an address at a given block.

Parameters

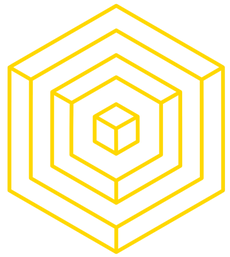
AUTHOR: NICK ZOGHB



Other implementations

- Python Web3.py
- Haskell hs-web3
- Java web3j

Easy to make, most used languages now have some kind of implementation in progress.



2 NETWORKS

TESTNETS

NETWORKS YOU CAN TEST ON

- Identical to the mainnet functionality
  - Copy of the protocol with controlled risk
  - Sometimes uses different consensus
- Minor difference in network/client parameters
  - Different genesis block
  - Different network ID
  - Lower block difficulty

Read more here.

AUTHOR: NICK ZOGHB

PRIVATE VS. PUBLIC

WHEN WOULD YOU USE EITHER

- Public
  - More realistic latency
  - Point to endpoint, setup is done
  - Must obtain ether from faucet (or running a node)
- Private (testrpc)
  - More control over parameters
  - Faster, more ideal
  - Like operating on a bare database
  - Certain operations require a real instance (revert and throw)

GETH

THE ETHEREUM CLI CLIENT

- Entry point into the Ethereum network
  - Main net
  - Test net
  - Private net

Read more here.

GETH

RISKS

Disclaimer

Safety caveats

Security warnings

- **You are responsible for your own computer security.** If your machine is compromised you **will** lose your ether, access to any contracts and maybe more.
- **You are responsible for your own actions.** If you mess something up or break any laws while using this software, it's your fault, and your fault only.
- **You are responsible for your own karma.** Don't be a jerk and respect others.
- This software is open source under a GNU Lesser General Public License license.

Legal warning: Disclaimer of Liabilites and Warranties

Short version

- **The user expressly knows and agrees that the user is using the ethereum platform at the user's sole risk.**

Read more here

AUTHOR: NICK ZOGHB

Build on parity

The Parity Ethereum client is built from the ground up to the highest standards of software development.

High Performance

Tuned, hand-optimised use of low-level Rust-language
JITEVM turbocharges execution of complex contracts
Multi-level in-memory caching

Ultra Reliable

Memory and concurrency safety guaranteed by Rust language.
Actor-based modularity ensures maximal resilience.
Unit-tested and peer-reviewed from day one.

Future Proof

Standard, JSON-based, chain-specification format
EVM plugins allow native speed contracts
Actor-based modular architecture with IPC

Small Footprint

Rust's ownership tracking facilitates minimal memory footprint
Cache management gives fine control to user
State-trie pruning minimises storage footprint

Easy to use

1-line install on Mac and Linux
Docker images available
Library APIs are fully documented

Compatibile

100% consensus test conformant implementation
Complies with standard devp2p network protocol
Fully compatible with JSON-RPC API

GETH VS PARITY

YOUR VERY OWN

Parity (written in Rust)

- With the pruning algorithm, hard drive usage won't grow exponentially
- Cool browser-based GUI
- Passive mode to reduce CPU and network load on leaf nodes
- Warp sync allows you to sync from scratch in hours as opposed to days

Geth (go-ethereum, written in Golang)

- Has the clique consensus implemented to access Rinkeby testnet (or build your own private Ethereum-based enterprise network)
- Is purported to be more thoroughly audited / Is generally seen as the reference implementation of Ethereum.

Read more here.

GETH

THE ETHEREUM CLI CLIENT

- Example mainnet call

```
$ geth --fast --cache=512 console
```

  - Start geth in fast sync mode (**`--fast`**), causing it to download more data in exchange for avoiding processing the entire history of the Ethereum network, which is very CPU intensive.
  - Bump the memory allowance of the database to 512MB (**`--cache=512`**), which can help significantly in sync times
  - Start up Geth's built-in interactive JavaScript console, (via the trailing console subcommand) through which you can invoke all official web3 methods as well as Geth's own management APIs. This is optional and if you leave it out you can always attach to an already running Geth instance with geth attach.

(Source)

GETH

THE ETHEREUM CLI CLIENT

- Example testnet call

```
$ geth --testnet --fast --cache=512 console
```

  - Instead of using the default data directory (~/.ethereum on Linux for example), Geth will nest itself one level deeper into a testnet subfolder (~/.ethereum/testnet on Linux). Note, on OSX and Linux this also means that attaching to a running testnet node requires the use of a custom endpoint since geth attach will try to attach to a production node endpoint by default. E.g. geth attach <datadir>/testnet/geth.ipc. Windows users are not affected by this.
  - Instead of connecting the main Ethereum network, the client will connect to the test network, which uses different P2P bootnodes, different network IDs and genesis states.

(Source)

GETH

THE ETHEREUM CLI CLIENT

- Example sync call to *Ropsten* chain

```
geth --testnet removedb
geth --testnet --fast --bootnodes
"enode://20c9ad97c081d63397d7b685a412227a40e23c8bdc6688c6f37e97cfbc22d2b4d1db1510d8f61e6a
8866ad7f0e17c02b14182d37ea7c3c8b9c2683aeb6b733a1@52.169.14.227:30303,enode://6ce05930c72a
bc632c58e2e4324f7c7ea478cec0ed4fa2528982cf34483094e9cbc9216e7aa349691242576d552a2a56aaeae
426c5303ded677ce455ba1acd9d@13.84.180.240:30303"
```

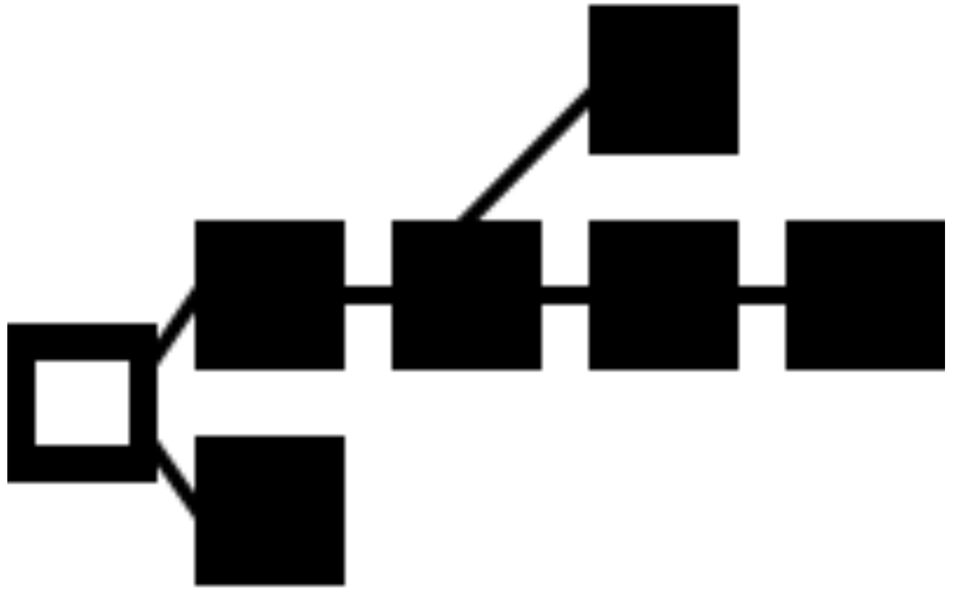
- What's an "enode" ?

(Source)

GENESIS BLOCK

THE BEGINNING

- The very first block on the network
  - No parent
  - Block 0
- Usually a .json file



(Source)

PRIVATE NETWORKS: GENESIS BLOCK

YOUR VERY OWN

CustomGenesis.json

```
{
    "nonce": "0x0000000000000042",
    "timestamp": "0x0",
    "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "extraData": "0x0",
    "gasLimit": "0x8000000",
    "difficulty": "0x400",
    "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
    "coinbase": "0x3333333333333333333333333333333333333333",
    "alloc": {
    }
}
```

(Source)

PRIVATE NETWORKS: GENESIS BLOCK

YOUR VERY OWN

- Prefund accounts for easier testing

```
"alloc": {
  "0x0000000000000000000000000000000000000001": {"balance": "111111111"},
  "0x0000000000000000000000000000000000000002": {"balance": "222222222"}
}
```

- Initialize your Geth node with the genesis state.

```
$ geth init path/to/genesis.json
```

(Source)

PRIVATE NETWORKS: SETUP

YOUR VERY OWN

- To initialize all nodes to desired genesis state, start a bootstrap node that others can use to find each other in your network; This creates an enode URL

```
$ bootnode --genkey=boot.key
$ bootnode --nodekey=boot.key
```

- Start every subsequent Geth node pointed to the bootnode for peer discovery

```
$ geth --datadir=path/to/custom/data/folder --bootnodes=<bootnode-enode-url-from-above>
```

- Geth instance to mine transactions on the network

```
$ geth <usual-flags> --mine --minerthreads=1
--etherbase=0x0000000000000000000000000000000000000000
```

(Source)

PUBLIC NETWORKS: OPTIONS

NETWORK ID

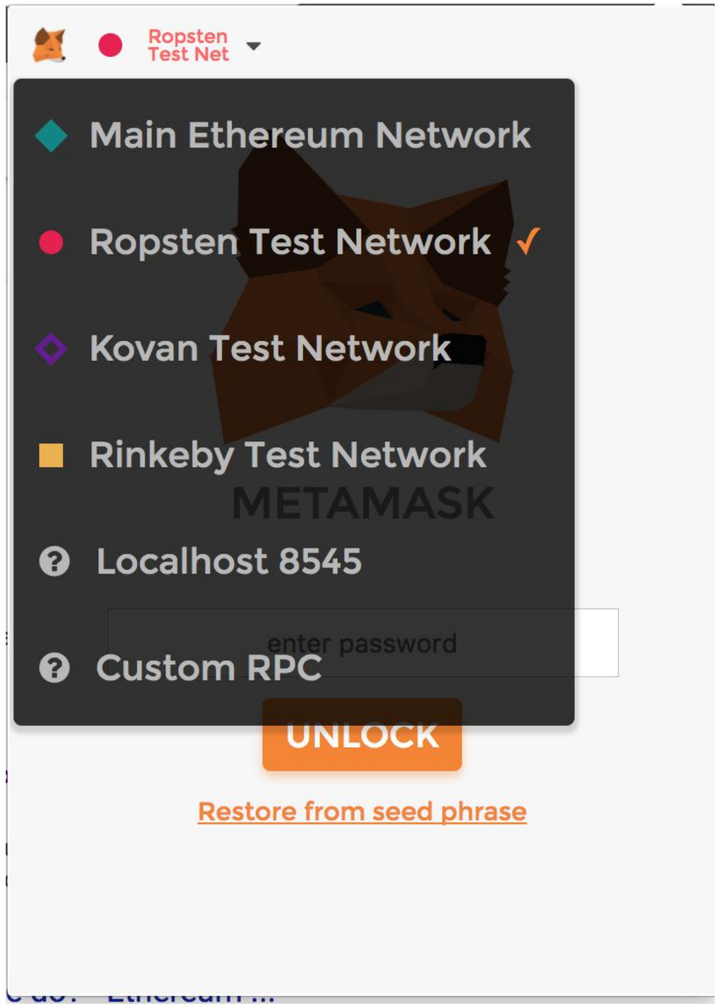
- 0: Olympic, Ethereum public pre-release testnet
- 1: Frontier, Homestead, Metropolis, the Ethereum public main network
- 1: Classic, the (un)forked public Ethereum Classic main network, chain ID 61
- 1: Expanse, an alternative Ethereum implementation, chain ID 2
- 2: Morden, the public Ethereum testnet, now Ethereum Classic testnet
- 3: Ropsten, the public cross-client Ethereum testnet
- 4: Rinkeby, the public Geth Ethereum testnet
- 42: Kovan, the public Parity Ethereum testnet
- **7762959: Musicoin, the music blockchain**

Read more here.

ROPSTEN

NOT RECOMMENDED

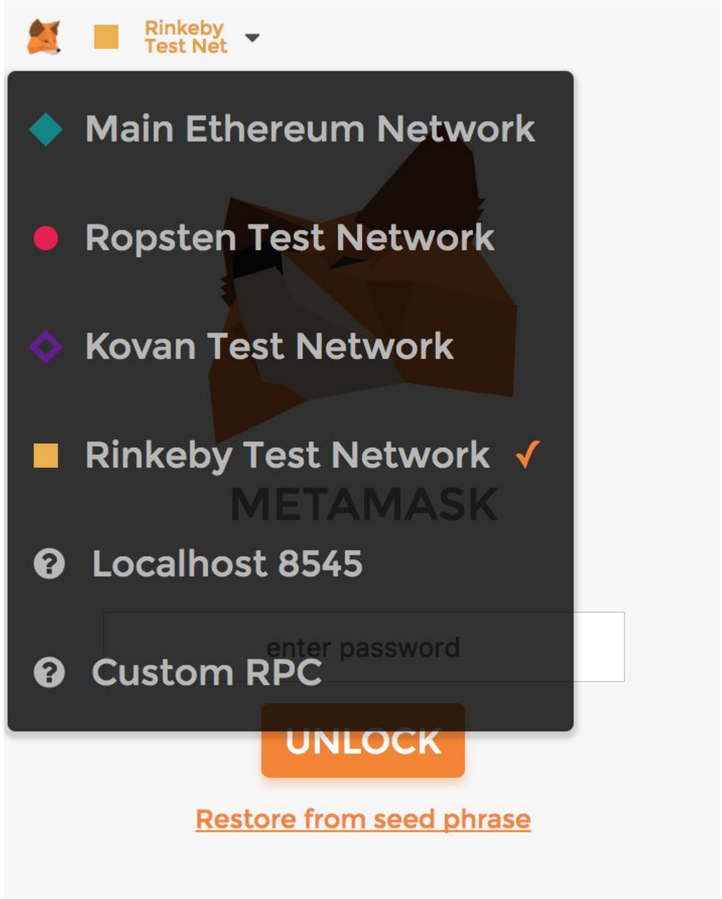
- Currently, it's an older, kind of "meh" testnet
  - Uses Proof of Work
    - Slow block times, pointless incentives and wasted resources
  - Often subjected to spam attacks
- Proof of Work cannot work securely in a network with no monetary value!



RINKEBY

RECOMMENDED

- Newer, official Ethereum testnet
  - Uses Proof of Authority
    - Rely on trusted validators to ensure that valid transactions are added to blocks, processed and executed by the EVM faithfully
    - 4 second block time
  - Requires geth (or any other client that supports clique consensus)



PROOF OF AUTHORITY?

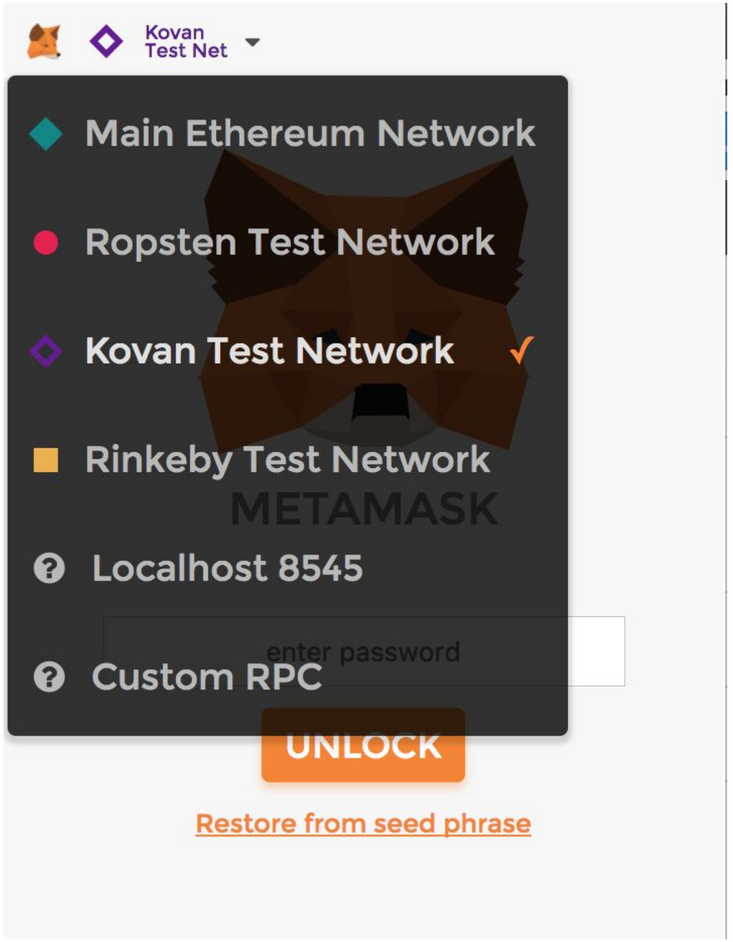
CONSENSUS FOR TESTNETS

- Immune to spam attacks , Lower block times, faster deployment and testing, no mining
- You have a number of pre-approved authority nodes (called sealers, think of these as mining nodes)
- Any new node that you want to add has to be voted on by the currently approved set of authority nodes, this gives you full control over which nodes can seal blocks (mine) on your network
- To make sure a malicious signer cannot do too much harm to the network any signer can sign at most one of a number of consecutive blocks **(floor(SIGNER_COUNT / 2) + 1)**
- The same consensus is applied when an authority node is removed from the network

KOVAN

ANOTHER PROOF OF AUTHORITY TESTNET

- Uses Proof of Authority consensus engine from Parity client (not compatible with geth), so Parity exclusive network
- Run by consortium of companies



Grid Singularity

Attores
Smart Contracts as a Service

parity

TenX

NIVAURA

MELONPORT

DIGIX GLOBAL

ETHERSCAN
The Ethereum Block Explorer

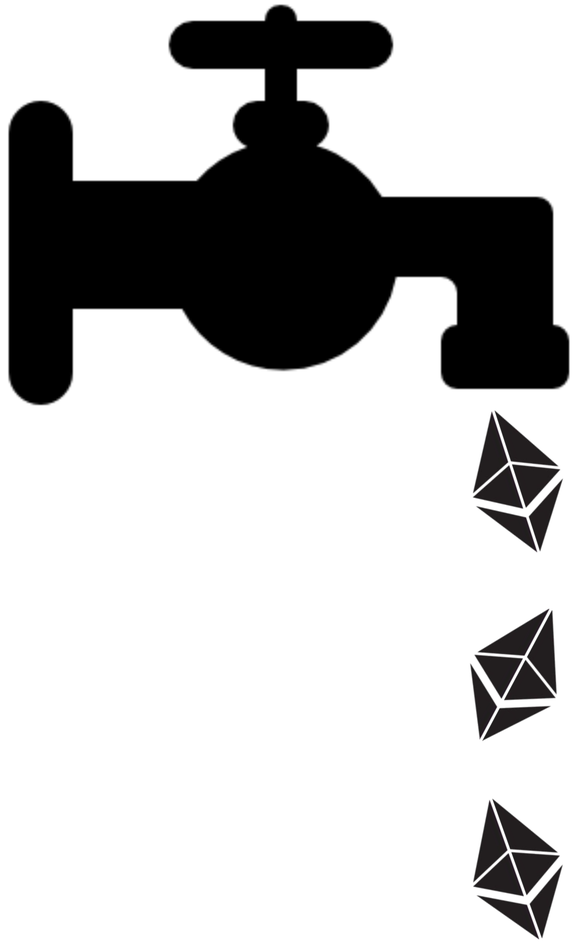
MAKER

AUTHOR: AKASH KHOSLA

PUBLIC NETWORKS: FAUCETS

INFINITE MONEY

- Allow users to request funds
  - Used to test smart contracts
- No value on the regular network
- Convenient
- Safeguarded against spam (controlled by central authority)



AUTHOR: NICK ZOGHB

3 DAPP ARCHITECTURE

MANAGEMENT API

A WAY TO MANAGE AND QUERY ETHEREUM NODES

- Besides the official DApp APIs (eth, shh, web3), Ethereum node clients will have support for management (JSON RPC)
- Geth provides the following extra management API namespaces:
  - **admin**: Geth node management
  - **debug**: Geth node debugging
  - **miner**: Miner and DAG management
  - **personal**: Account management
  - **txpool**: Transaction pool inspection
- Important for building production DApps!

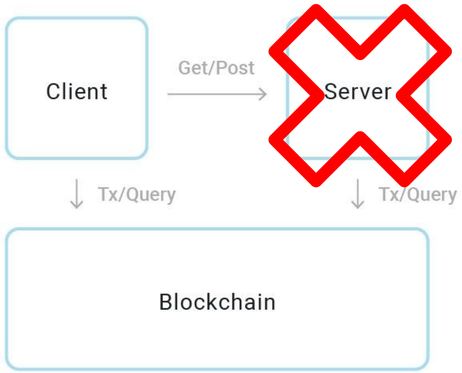
admin	debug	miner	personal	txpool
addPeer	backtraceAt	setExtra	ecRecover	content
datadir	blockProfile	setGasPrice	importRawKey	inspect
nodeInfo	cpuProfile	start	listAccounts	status
peers	dumpBlock	stop	lockAccount	
setSolc	gcStats	getHashrate	newAccount	
startRPC	getBlockRlp	setEtherbase	unlockAccount	
startWS	goTrace		sendTransaction	
stopRPC	memStats		sign	
stopWS	seedHashsign			

CLIENT-BLOCKCHAIN IN SERVERLESS APPS

THE MAIN MODEL FOR DOING DAPPS

- This is the canonical flavour of an Ethereum app
- Entire flow happens between the client and the blockchain
- Set up such that you distribute client code to your users
- Usually static page with **web3** calls, host anywhere like GitHub pages
- If clients have **bzz** (**Swarm**) or **IPFS** protocol support then you can do complete decentralized by hosting the pages/content on there

Source



QUERYING THE BLOCKCHAIN

CLIENT-BLOCKCHAIN IN SERVERLESS APPS

- The app needs to be able to read information from the blockchain
- Requires connection to an Ethereum node
- Web3 Provider handles the actual connection to a node
- `web3.setProvider(new web3.providers.HttpProvider('http://localhost:8545'));`
- Can use the Mist client, or a browser plugin known as Metamask, which acts as a lightweight client to the blockchain

Source

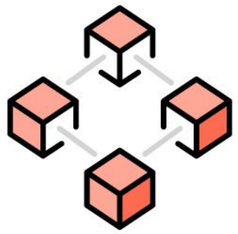
AUTHOR: AKASH KHOSLA

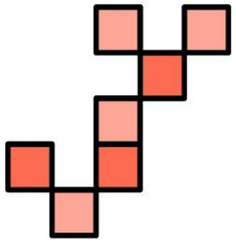
WHAT IF I DON'T HAVE MIST/METAMASK?

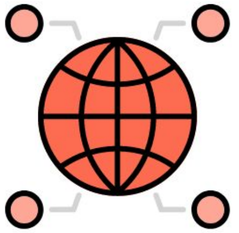
CLIENT-BLOCKCHAIN IN SERVERLESS APPS

- If you only need to query the blockchain without sending transactions, then you can establish a connection to a public Ethereum node
- Public in the sense that we have a geth/parity node ready without exposing the **personal** API (part of the JSON-RPC Ethereum API) for managing accounts
- If you don't want to host it yourself, you can leverage Infura, a service that offers public nodes at no cost
- Careful as you are trusting a third-party controlled node, for the sake of ease-of-use.
  - Real third party nodes will disable anything beyond querying

Source



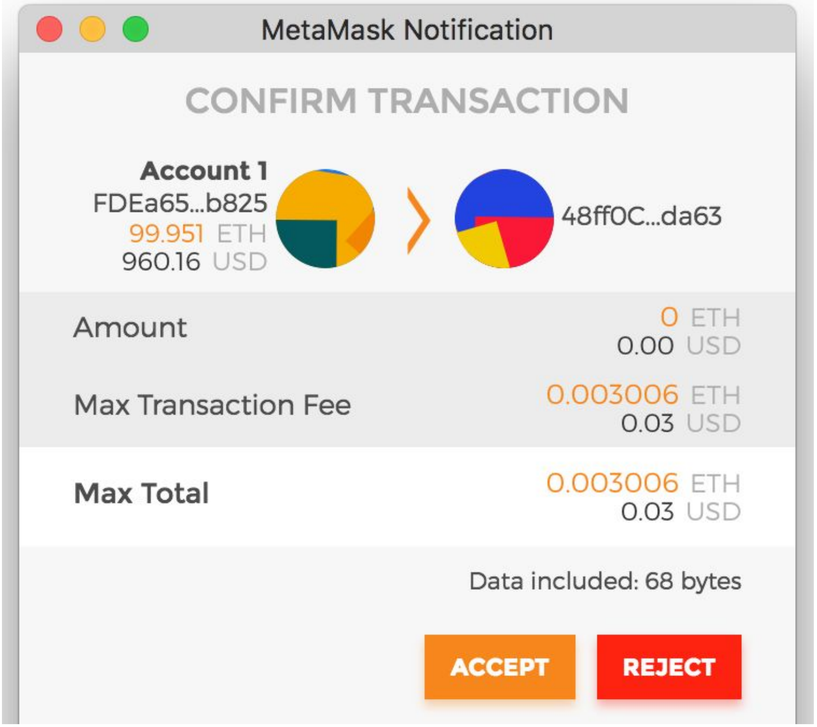




CONTRACT INTERACTION THROUGH TXNS

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

- Querying is easy, but what if you want to have users **submit transactions and perform state changes** on smart contracts?
- With Mist and Metamask, this is easy, both provide a way to manage the user accounts and request the user to approve a transaction **when the application is asking for ETH**
- **Mist** will provide a gateway to the user's local node (geth, parity) and its accounts (via the personal API) for signing transactions, while **Metamask** will sign transactions client-side and relay them to the Metamask public nodes

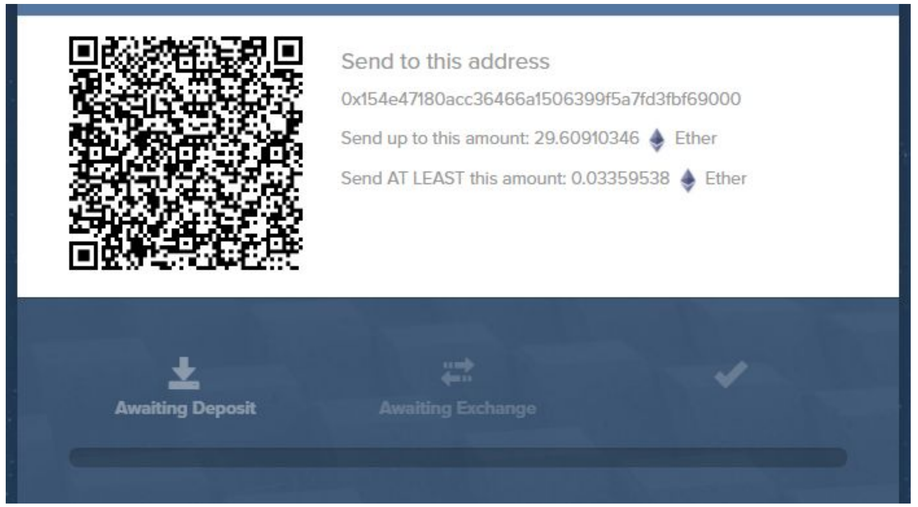


Source

CONTRACT INTERACTION THROUGH TXNS

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

- You can also use light mode in Geth to send transactions
- If you don’t require your user to install Metamask or Mist to use your app, you’ll need to direct them to manually send the transactions from whichever wallet they work with
- Most applications will implement this by asking the user to send a certain amount of ETH to an address, optionally including a scannable QR code or copy-to-clipboard button



Source

CONTRACT INTERACTION THROUGH TXNS

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

- Your application may monitor the contract **events** to update the user interface as the transaction sent out-of-band is executed.
  - Since watching events is simply a way of querying the blockchain, it can easily be done through a public node
- For executing contract functions, you may need to request the user to send ETH along with additional data to execute a specific function
- You can use the request method of a contract function to easily obtain the data needed for running a method, and present that to the user along with the target address

Source

CONTRACT INTERACTION THROUGH TXNS

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

- Now, for executing contract functions, you may need to request the user to send ETH **along with additional data** to execute a specific function.
- You can use the **request** method of a contract function (web3) to easily obtain the data needed for running a method, and present that to the user along with the target address.

```
SimpleToken.at(tokenAddress).transfer.request("0xbcfb5e3482a3edee72e101be9387626d2a7
e994a", 1e18).params[0].data
// returns data for invoking transfer with args =>
'0xa9059cbb000000000000000000000000000000000000000000000000000000010000000000
0000000000000000000000000000000000de0b6b3a7640000'
```

Source

DOING TXNS MANUALLY IS POOR

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

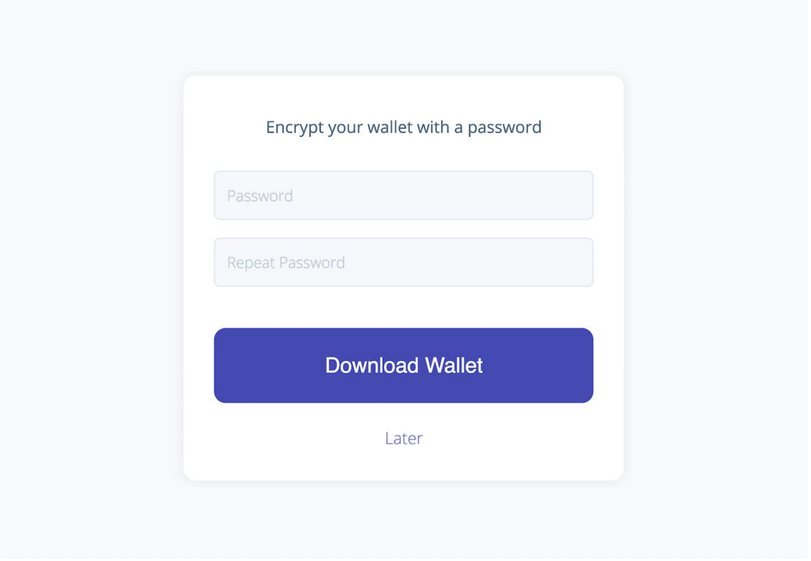
- It requires a fairly complex interaction from the user - not all users know how additional data works in Ethereum or how to send it along in a transaction
- Use only if you cannot think about how to create a UI to include that additional data as a dropdown or field with limited inputs
- However, if you do plan on using this manual method as a way to interact with your DApp, then make sure you include a **fallback function** with reasonable behavior
  - If the user forgets to include additional data when manually sending the transaction, he should not lose the funds sent
  - Or design the fallback such that it cleverly executes a desirable outcome

Source

WALLET MANAGEMENT IN THE APP

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

- Another option for having your users perform complex interactions
- Have people create a new Ethereum account address through your DApp, which will be used to send transactions directly from your code
- The user will need to seed the account with Ether, either from other account or from some exchange service
- Once the account has the balance to pay for transaction fees, the client side code may execute any needed operations on behalf of the user (See EtherDelta)



Source

WALLET MANAGEMENT IN THE APP

CLIENT-BLOCKCHAIN IN SERVERLESS APPS - INTERACTIONS

- Your application will use the private key of the generated account to sign any transaction client-side and then relay them to a public node for execution
- This pattern requires a lot more involvement - you need to add features of generating, encrypting and importing an Ethereum account
  - Also all transactions need to be crafted and signed manually and then sent as a raw transaction to a node
  - Users still also need to keep their accounts safe
  - But once set up, your user can perform Ethereum transactions directly from your application with zero friction at all, without any other third party software

How do you decide what form of interaction to use? Depends on usage patterns (continuous, one-time or infrequent)

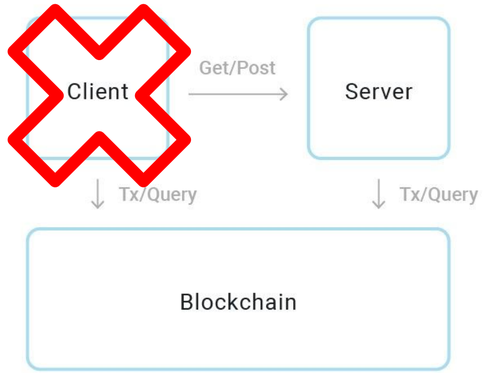
Source

SERVER TO BLOCKCHAIN MODEL

ALTERNATIVE TO CLIENT BLOCKCHAIN?

- Suppose we add a server to the mix and leave the client aside
  - The server in this case can be an application server, scripts or batch processes
- Option 1: Local Node
- Option 2: Offline signing and public nodes

Source



AUTHOR: AKASH KHOSLA

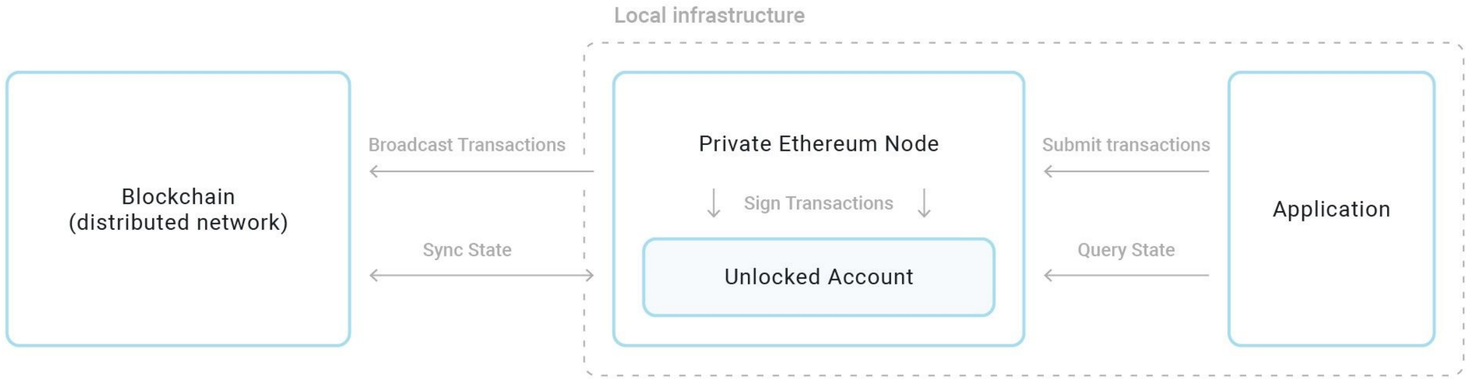
OPTION 1: LOCAL NODE

SERVER TO BLOCKCHAIN MODEL

- First solution is to setup a local Ethereum node and use its JSON RPC interface to perform all your blockchain operations
- For convenience, keep an unlocked account for running transactions from your application available on your node (unlock flag for Geth and Parity)
- Make sure the JSON RPC interface is not accessible anywhere but from your application because if someone gains access to your node they can steal your funds

Precaution: Keep the funds on the unlocked account to a minimum and seed it from a different source as needed

Source



AUTHOR: AKASH KHOSLA

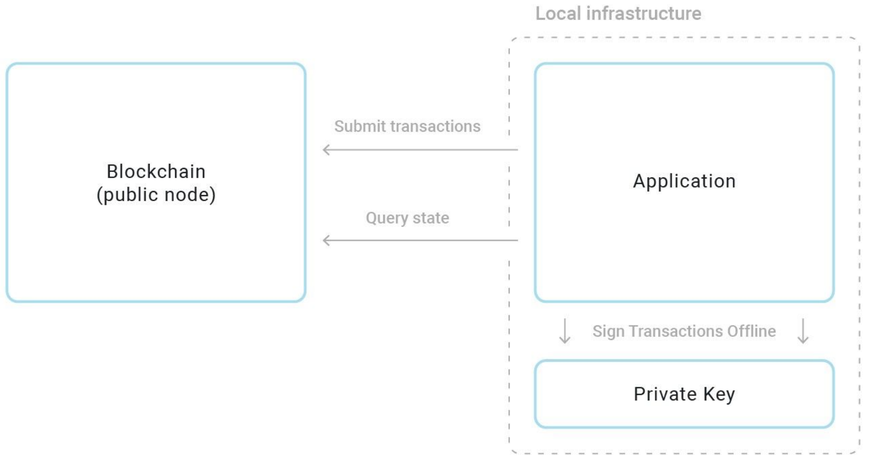
OPTION 2: OFFLINE SIGNING W/ PUB NODES

SERVER TO BLOCKCHAIN MODEL

- Sign transactions offline as explored previously, and relay them to a public node
- If you go down this route you are blindingly trusting the public node you connect to as it can choose to not relay them to the network or provide a fake response to your queries
- Can be mitigated by connecting and sending transactions/queries to multiple nodes simultaneously but this would make your codebase much more complex

Truffle can be configured such that you transparently sign transactions offline and send them to an Infura node

Source



CLIENT SERVER AND BLOCKCHAIN MODEL

WORKING WITH EVENTS

- Having both client and server interacting simultaneously with the blockchain implies that you **may need to coordinate both of them**
  - Examples: Server reacts to an **action performed on-chain** by the client or we want to visualize certain state-changes on a contract for a client
- To observe changes on a contract, we listen for contract **events**
  - Ensure that all relevant actions have an associated event and use indexed arguments so that observers can filter events relevant to them
  - A client will listen to events that affect them directly, while servers may monitor all contracts related to the app

Source

CAREFUL WITH ON-CHAIN DEPENDENCE

CLIENT SERVER AND BLOCKCHAIN MODEL

- If you rely on transactions or events, ensure to only react upon them after a certain number of confirmations - as it could be subject to a **chain reorganization**
  - When a node on the Ethereum network realises that what it thought was the canonical chain turned out not to be
    - When this happens, the transactions in the latter part of its chain (i.e. the most recent transactions) are reverted and rather the transactions in the newer replacement are executed.
    - Happens often due to 12 sec block time, essentially a fork resolution
- Therefore: You should wait a dozen or more blocks before acting upon an on-chain event and let the user know that the transaction was successful but not yet confirmed

Source

OK, BUT WHY DO WE EVEN NEED A SERVER

CLIENT SERVER AND BLOCKCHAIN MODEL

- In a traditional client-server context, server is permanent storage, enforces business logic and coordinates clients - all of which can now be performed on-chain
- But there are still many uses for a server backing your app
  - On chain code cannot directly work with off-chain services
  - If you want to integrated third party services, use external data or perform basic actions like sending an email you need a server
  - The server can also act as a cache or indexing engine for your smart contracts
  - Large data is expensive on Ethereum due to gas cost, you could use a server to store large amounts and then keep a hash on-chain for verification

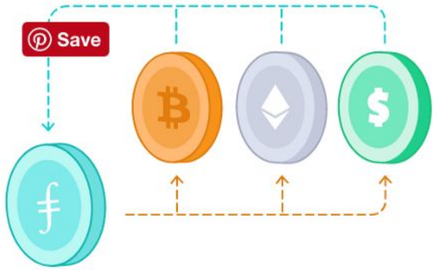
Source

WILL SERVERS BE A THING IN THE FUTURE?

CLIENT SERVER AND BLOCKCHAIN MODEL

- More projects are providing seamless integrations with the EVM, like Filecoin, Storj, Truebit and Oraclize.it, so maybe things will be different with the traditional server

EARN FILECOIN FOR HOSTING FILES



EXCHANGE FILECOIN FOR USD, BTC, ETH AND MORE

RELIABLY STORE FILES AT HYPERCOMPETITIVE PRICES

Source

SEE YOU NEXT TIME

Web3 and Smart Contract Architecture

BLOCKCHAIN
AT BERKELEY