Ethereum Smart Contracts

Definition, Accounts, EVM

Smart Contracts

What are Smart Contracts

A "smart contract" is simply a program that runs on the Ethereum blockchain. It's a collection of code (its functions) and data (its state) that resides at a specific address on the Ethereum blockchain.

Smart contracts are a type of Ethereum account. This means they have a balance and they can send transactions over the network. However they're not controlled by a user, instead they are deployed to the network and run as programmed.

Smart Contracts

User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract. Smart contracts can define rules, like a regular contract, and automatically enforce them via the code.

To interact with Smart contracts we need to have fully understand: Accounts , Transactions and the Ethereum virtual machine before jumping into the world of smart contracts.

Solidity Programming Language

Smart contracts in Ethereum are written in Solidity. Solidity was initially proposed in August 2014 by Gavin Wood. The language was later developed by the Ethereum project's Solidity team, led by Christian Reitwiessner.

Solidity is a statically-typed programming language designed for developing smart contracts that run on the EVM

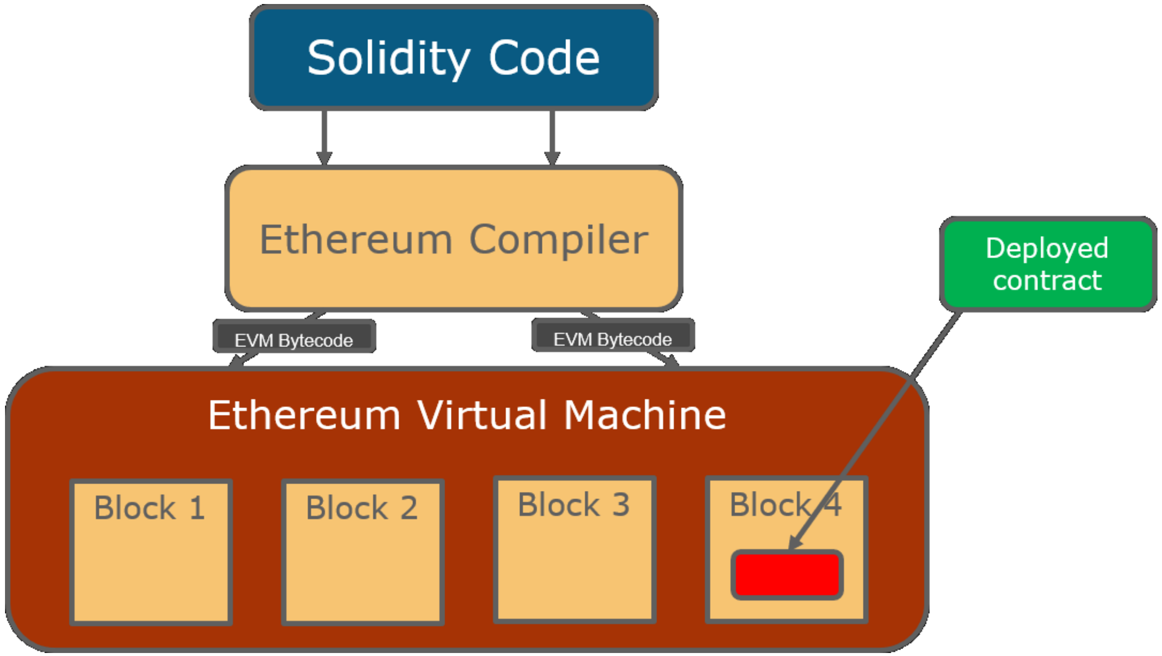
Solidity supports inheritance, libraries and complex user-defined types among other features.

By the time of writing this presentation the latest version of Solidity is v0.7.5

Official Solidity documentation and examples can be found here:

https://docs.soliditylang.org/en/develop/solidity-by-example.html

Ethereum Virtual Machine



Ethereum Virtual Machine

It exists as one single entity maintained by thousands of computers running an Ethereum client.

When people refer to the Ethereum networks as “The world’s computer” they usually mean the Ethereum’s EVN.

To interact with Smart contracts we need to have fully understand: Accounts , Transactions and the Ethereum virtual machine before jumping into the world of smart contracts.

Accounts

What is Ethereum account?

An Ethereum account is an entity with an ether (ETH) balance that can send transactions on Ethereum. Accounts can be user-controlled or deployed as smart contracts

2 types of accounts:

Externally-owned – controlled by anyone with the private keys

Contract – a smart contract deployed to the network, controlled by code.

Difference between different accounts

Externally-owned

- Creating an account costs nothing
- Can initiate transactions
- Transactions between externally-owned accounts can only be ETH transfers

Smart Contract

- Creating an account has a cost because you're using network storage
- Can only send transactions in response to receiving a transaction
- Transactions from an external account to a contract account can trigger code which can execute many different actions, such as transfering tokens or even creating a new contract

Account Fields

Nonce – a counter that indicates the number of transactions sent from the account. This ensures transactions are only processed once. If a contract account, this number represents the number of contracts created by the account

Balance – the number of Wei owned by this address. Wei is a denomination of ETH and there are 1e+18 Wei per ETH.

CodeHash – The Smart Contract code fragments are contained in the state database under their corresponding hashes for later retrieval. For contract accounts, this is the code that gets hashed and stored as the codeHash.

StorageRoot - This tree encodes the hash of the storage contents of this account, and is empty by default

Account key pairs

An account is made up of a cryptographic pair of keys: public and private. They help prove that a transaction was actually signed by the sender and prevent forgeries.Your private key is what you use to sign transactions, so it grants you custody over the funds associated with your account. You never really hold cryptocurrency, you hold private keys – the funds are always on Ethereum's ledger.

This prevents malicious actors from broadcasting fake transactions because you can always verify the sender of a transaction.

Account Creation

When you want to create an account most libraries and Ethereum Clients will generate you a random private key.

A private key is made up of 64 hex characters and can be encrypted with a password.

Example:

fffffffffffffffffffffffffffffffebaaedce6af48a03bbfd25e8cd036415f

The public key is generated from the private key using Elliptic Curve Digital Signature Algorithm. You get a public address for your account by taking the last 20 bytes of the public key and adding 0xto the beginning.

Geth. Example

Contract Accounts

Contract accounts also have a 42 character hexadecimal address:

Example:

0x06012c8cf97bead5deae237070f9587f8e7a266d

The contract address is usually given when a contract is deployed to the Ethereum Blockchain. The address comes from the creator's address and the number of transactions sent from that address (the “nonce”).

Contract Example:

https://etherscan.io/address/0x06012c8cf97bead5deae237070f9587f8e7a266d

Verified Contracts:

https://etherscan.io/contractsVerified

Ethereum Nodes and Clients

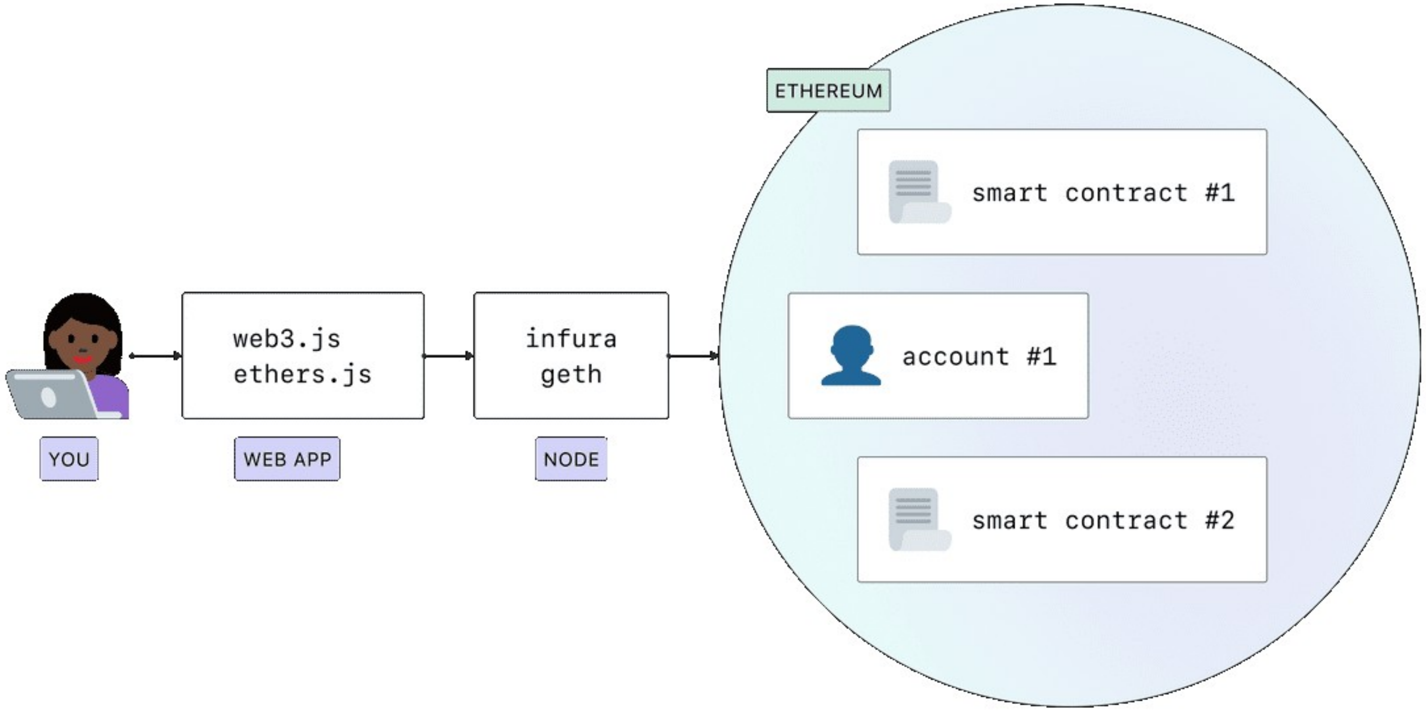
What is ETH Node and ETH Client ?

A node is a machine that is running a client. A client is implementation of Ethereum that verifies all transactions in each block, keeping the network secure and the data accurate.

Many implementations of Ethereum clients exist in a variety of languages. What these client implementations have in common is they all follow a formal specification. This specification dictates how the Ethereum network and blockchain functions.

Why should you run a node? - Running a node allows you to trustlessly and privately use Ethereum, while supporting the ecosystem

Ethereum Nodes and Clients



Ethereum Clients

“Go Ethereum” Geth - https://geth.ethereum.org/

Parity - https://www.parity.io/ethereum/

Infura - 3rd party Open based API

Geth account demo

- different sync modes - https://ethereum.org/en/developers/docs/nodes-and-clients/

Remix IDE

Remix is web based free to use IDE for writing smart contracts

One can reach Remix at: https://remix.ethereum.org/

Remix is always updated with new plugins and features. You can download smart contracts on different all different Ethereum networks:

Mainnet - POW
Ropsten - POW
Rinkeby - POA
Kovan - POA
Görli cross-client POA

https://medium.com/@piyopiyo/list-of-ethereums-major-network-and-chain-ids-2bc58e928508

Using Remix to Create Smart Contract

1) Use Remix IDE and select Solidity Plugin
2) Activate “WALLET CONNECT” from plugin manager and select Metamask or another compatible mode (If using Metamask select Kovan or other Test Network)
3) Use Storage.sol or another Solidity Class (contract) with newer version of solidity
4) Compile the class with the correct version of Solidity
5) Select “Deploy & Run Transactions (select Injected Web3) and Deploy
6) Approve the transaction in Metamask
7) The contract should appear under the “Deployed Contracts” section. Use the transaction id that was used in the creation to check the contract in Kovan (eg. https://kovan.etherscan.io/address/0x889c162eBAD18D37dfCfbc38EA9c69fAc2B08A19)

Working with the Smart Contract

1) In the Deployed Contract section use the Store function to pass parameter to the smart conract (Smart contract should be already created)

2) Approve the transaction and examine it on etherscan (eg https://kovan.etherscan.io/tx/0xd81156cfc069d91587be790e43cefc2c710bc9a55d0a074c6b37c268efc99593)

3)How can we use the retrieve the method ?

4)

Select Deploy & Run Transactions

Setting up Smart contract Dev Environment

Setting up NodeJS and NPM

Install Visual Studio Code or other IDE

Install Truffle (framework for developing full-stack DAPPs)

Install Ganache (local test Blockchain)

Using Infura for 3rd party ETH Client

Infura allows easy API access to GETH functionality without actually installing geth

Install Visual Studio Code or other IDE

Install Truffle (framework for developing full-stack DAPPs)

Install Ganache (local test Blockchain)

Development Process

Create a smart contract and deploy it to local env (Ganache)

Test Smart Contract functionality on local env

Deploy contract on Testnet (Kovan)

Test Smart Contract functionality on Testnet

```
//use this additional package for connecting to Kovan
npm install @truffle/hdwallet-provider
```

Deploy Smart Contract on Mainnet (optional)

Using JSON RCP

Connect to Ganache’s Json RCP service using any tool (Postman)

Get balance of ETH address

```
{
        "jsonrpc":"2.0",
        "method":"eth_getBalance",
        "params":["0xad7421d39326e300f8a54416b04a1D480034b6C4","latest"],
        "id":1
}
```

Get registered accounts

```
"method":"eth_accounts"
```

Get latest block number

```
"method":"eth_blockNumber"
```

Get current chain Id

```
"method":"eth_chainId"
```

Get current gas Price

```
"method":"eth_gasPrice"
```

Sources

Smart Contracts
https://ethereum.org/en/developers/docs/smart-contracts/

EVM
https://ethereum.org/en/developers/docs/evm/

Ecliptic Curve Digital Signature Algorithm
ETH Yellow Paper
Writing Test Smart Contract on Ropsten Testnet with Parity

Smart contracts

Tutorial that we used in class:
Deploy Ethereum Smart Contract in 6 steps

Ethereum Nodes and Clients

https://ethereum.org/en/developers/docs/accounts/

Remix IDE
Turorial

Questions ?

- Thank you for your attention!