

Introdução às Tecnologias Blockchain

Visão Geral

Prof. Rogério Aparecido Gonçalves¹
rogerioag@utfpr.edu.br

¹Universidade Tecnológica Federal do Paraná (UTFPR)
Departamento de Computação (DACOM)
Campo Mourão - Paraná - Brasil

Semana de Informática (SEINFO 2023)
SEINFO2023
Introdução às Tecnologia Blockchain



Agenda i

1. Introdução
2. Conceitos e Fundamentos de Blockchain
3. Bitcoin
4. Smart Contracts (Contratos Inteligentes)
5. Ethereum
6. Um pouco mais de Ethereum
7. Prática sobre Ethereum

Agenda ii

8. Leitura do Capítulo 12: *Futher Ethereum*
9. Ambiente e Ferramentas de Desenvolvimento
10. Prática sobre *Ethereum*:
11. Ferramentas de Desenvolvimento e Frameworks
12. Leitura do Capítulo 14: *Development Tools and Frameworks*
13. Instalação das Ferramentas
14. Redes de Testes

Agenda iii

15. Clientes
16. Introdução à Web3
17. Introdução
18. Prática
19. Leitura do Capítulo 15: *Introducing Web3*
20. Instalação das Ferramentas
21. Tokenização

Agenda iv

22. Referências

Introdução

Ministrante



Figura 1: Rogério Gonçalves
(RAG)

Doutor em Ciência da Computação, é professor na UTFPR - Campus Campo Mourão. É revisor do journal Springer Computing e de algumas conferências.

Interesse em: Arquitetura de Computadores, Computação Paralela, Computação Heterogênea, Compiladores e Runtimes e Tecnologias Blockchain.

Ementa

Introdução às Tecnologias Blockchain. Smart Contracts. Ethereum e Solidity. Ferramentas de Desenvolvimento e Frameworks. Redes de Testes e Clientes. Introdução a Web3. Tokenização.

Pré-requisitos

Saber um pouco de Java Script.

Objetivos

- Apresentação de uma Visão Geral sobre o Ecossistema de Tecnologias relacionadas a **Blockchain**. Surgimento e contexto histórico vinculado ao *Bitcoin*. Mas o foco principal está na rede **Ethereum** e componentes do seu Ecossistema. Falaremos um pouco sobre a __Ethereum Virtual Machine (EVM)__ e Contratos Nativos. Além disso, uma perspectiva do usuário é apresentada, mostrando a estrutura dos blocos do *blockchain* da *Ethereum*, *Wallets* e softwares clientes, nós e mineradores, ferramentas e **APIs**, protocolos e Linguagens de Programação Suportados.

Conceitos e Fundamentos de Blockchain

Definição de Blockchain

Definição de Layman

Blockchain is an ever-growing, secure, shared recordkeeping system in which each user of the data holds a copy of the records, which can only be updated if all parties involved in a transaction agree to update.

Definição Técnica

Blockchain is a peer-to-peer distributed ledger that is cryptographically-secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers.

Definição de Blockchain ii

- *Peer-to-peer*
- *Distributed Ledger*
- *Criptograficamente Seguro*
- *Append only (Permitido anexar novos blocos)*
- *Atualizável via consenso dos pares.*

Como a Tecnologia Blockchain foi desenvolvida i

1950s – Hash functions

1970s – Merkle trees - hashes in a tree structure

1970s continued – Research in distributed systems, consensus, state machine replication

1980s – Hash chains for secure logins

1990s – *e-Cash for e-payments*

1991 – Secure timestamping of digital documents.

1992 – Hashcash idea to combat junk emails

1994 – S/KEY application for Unix login.

1997/2002 – *Hashcash*

2008/2009 – Bitcoin (the first blockchain)

Interesse no termo “blockchain”

- Interesse ao longo do tempo (Fonte: Google Trends):

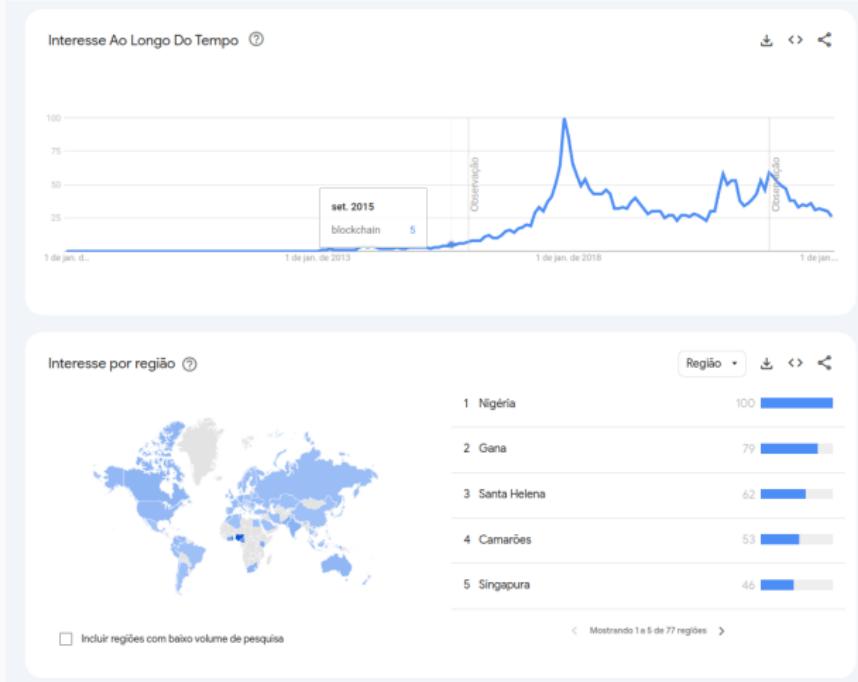
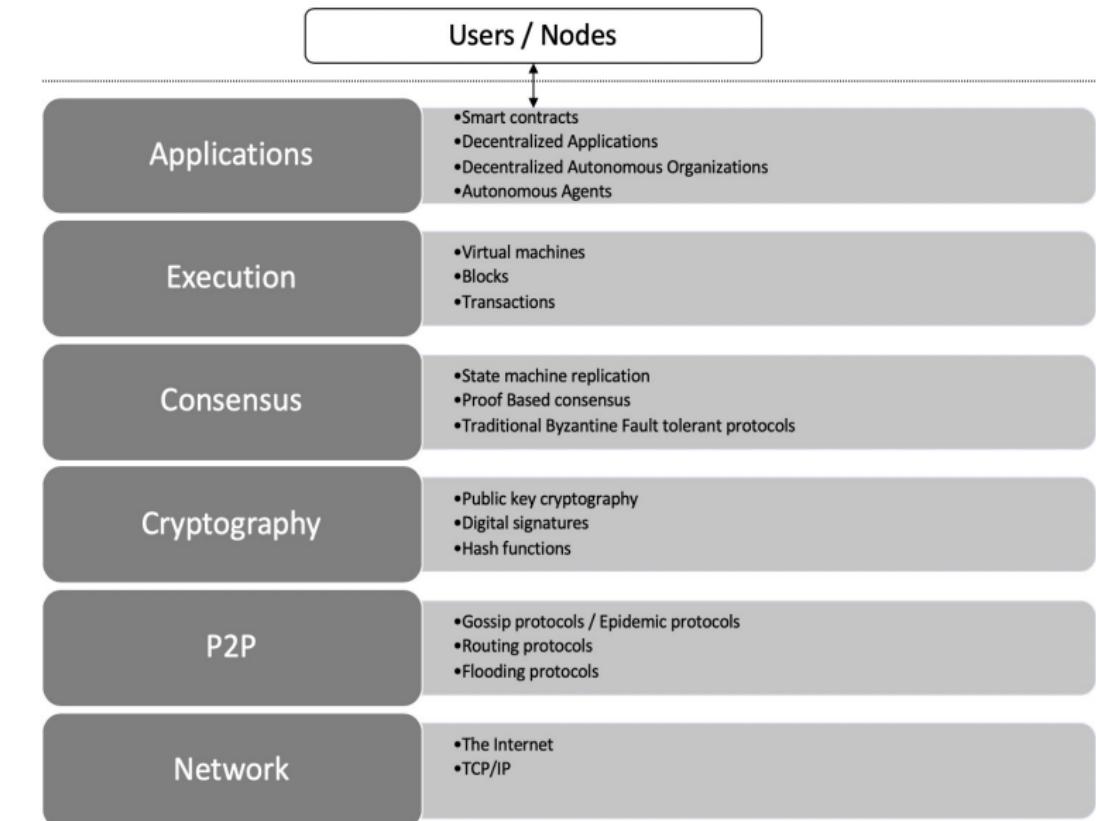


Figura 2: Pesquisas sobre o termo “blockchain”

Visão Arquitetural do Blockchain



Estrutura Genérica de um Blockchain

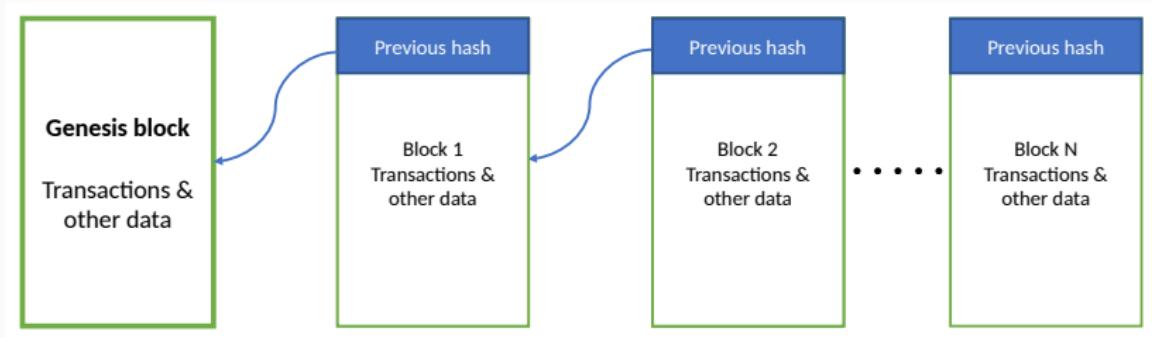


Figura 3: Estrutura Genérica de um Blockchain

Elementos Genéricos de um Blockchain

- Endereços
- Contas
- Transações
- Blocos
- Redes *Peer-to-peer*
- Scripting ou Linguagens de Programação
- Virtual machines
- Máquinas de Estado
- Nós (nodes)
- Contratos Inteligentes (*Smart contracts*)

Como um Blockchain funciona

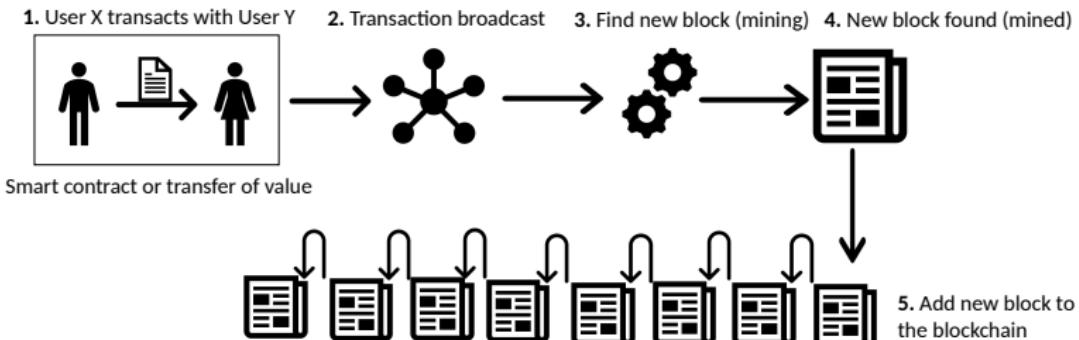


Figura 4: Funcionamento de um Blockchain

Estrutura Genérica de um bloco i

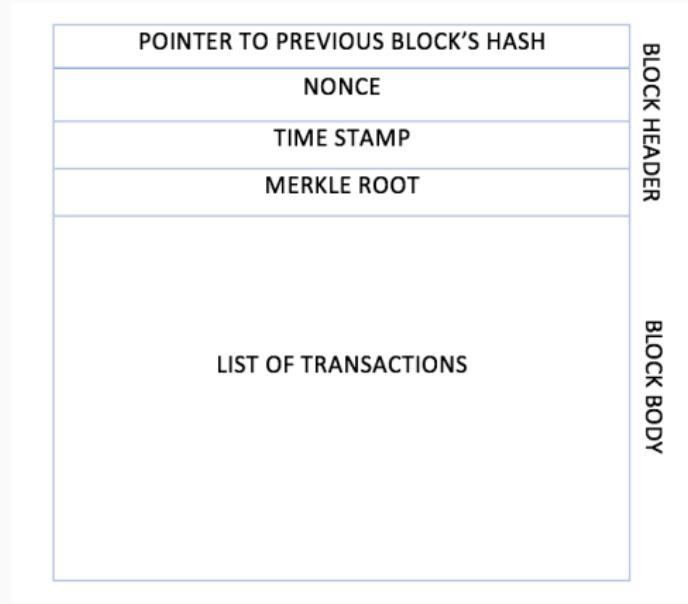


Figura 5: Estrutura de um Bloco

Leitura Recomendada

Capítulo 1: Blockchain 101

Livro: IMRAN BASHIR. Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Bitcoin

- Passos de como enviar e receber pagamentos:
 - A transação começa com um remetente assinando a transação com sua chave privada.
 - A transação é serializada para que possa ser transmitida pela rede.
 - A transação é transmitida para a rede.
 - Mineradores que escutam transações pegam a transação.
 - A transação é verificada quanto à sua legitimidade pelos mineradores.
 - A transação é adicionada ao bloco candidato/proposto para mineração.
 - Uma vez minerado, o resultado é transmitido para todos os nós da rede *Bitcoin*.
 - Normalmente, neste momento, os usuários aguardam até seis confirmações para serem recebidas antes que uma transação seja considerada final; no entanto, uma transação pode ser considerada final na etapa anterior.

- As confirmações servem como um mecanismo adicional para garantir que haja probabilidade muito baixa de uma transação ser revertida, mas, caso contrário, uma vez que um bloco minerado seja finalizado e anunciado, as transações dentro desse bloco serão finais nesse ponto.

Chaves Criptográficas i

- Private keys in Bitcoin
 - Private keys are used to digitally sign the transactions, proving ownership of the bitcoins.
- Public keys in Bitcoin
 - Public keys are used by nodes to verify that the transaction has indeed been signed with the corresponding private key.
- Addresses in Bitcoin
 - A Bitcoin address is created by taking the corresponding public key of a private key and hashing it twice, first with the SHA256 algorithm and then with RIPEMD160.



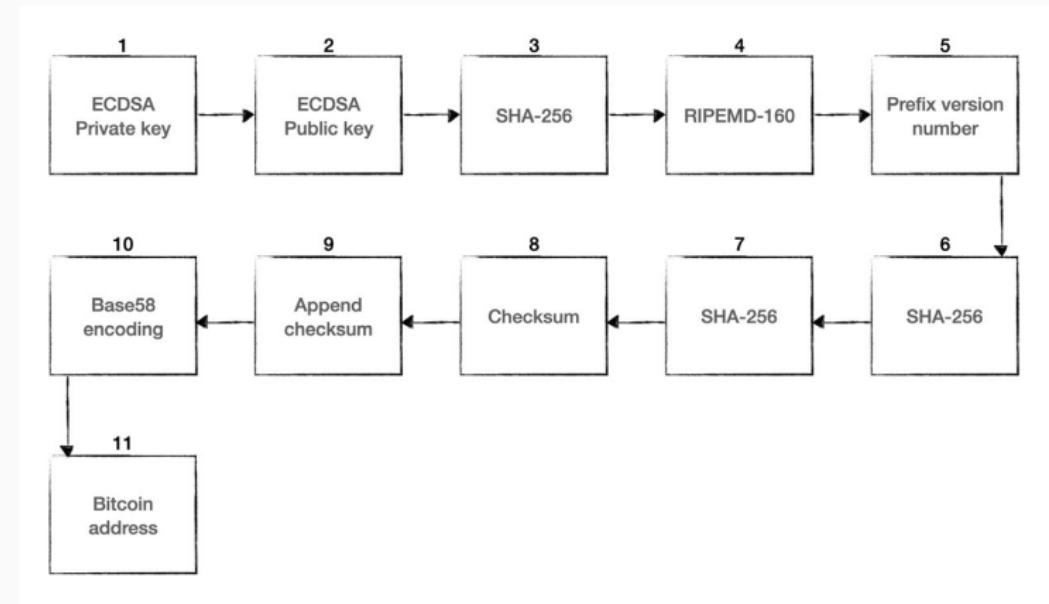
QR code of the Bitcoin address
1ANAgGG8bikEv2fYsTBnRUmxB7QUcK58wt



From bitaddress.org,
a private key and Bitcoin address
in a paper wallet

Geração de Endereços no Bitcoin i

- Para gerar um endereço no Bitcoin, é usado um processo de 11 etapas:



Transações i

- A user/sender sends a transaction using wallet software or some other interface.
- The transaction is signed using the sender's private key.
- The transaction is broadcasted to the Bitcoin network using a flooding algorithm.
- Mining nodes (miners) who are listening for the transactions verify and include this transaction in the next lock to be mined.
- Next, the mining starts.
- Finally, the confirmations start to appear in the receiver's wallet.

Estrutura de dados de uma Transação i

Field	Size	Description
Version number	4 bytes	Used to specify rules to be used by the miners and nodes for transaction processing.
Input counter	1-9 bytes	The number (positive integer) of inputs included in the transaction.
List of inputs	Variable	Each input is composed of several fields, including Previous Tx hash, Previous Txout-index, Txin-script length, Txin-script, and optional sequence number. The first transaction in a block is also called a coinbase transaction. It specifies one or more transaction inputs.
Output counter	1-9 bytes	A positive integer representing the number of outputs.
List of outputs	Variable	Outputs included in the transaction.
Lock time	4 bytes	This field defines the earliest time when a transaction becomes valid. It is either a Unix timestamp or block height.

Estrutura de dados de uma Transação – entradas e saídas

Transaction input data structure

Field	Size	Description
Transaction hash	32 bytes	The hash of the previous transaction with UTXO
Output index	4 bytes	This is the previous transaction's output index, such as UTXO to be spent
Script length	1-9 bytes	Size of the unlocking script
Unlocking script	Variable	Input script (ScriptSig), which satisfies the requirements of the locking script
Sequence number	4 bytes	Usually disabled or contains lock time – disabled is represented by <code>0xFFFFFFFF</code>

Transaction output data structure

Field	Size	Description
Value	8 bytes	The total number (in positive integers) of Satoshis to be transferred
Script size	1 – 9 bytes	Size of the locking script
Locking script	Variable	Output script (ScriptPubKey)

Script i

- Simple stack-based language used to describe how bitcoins can be spent and transferred
- Evaluated from left to right using a Last in, First Out (LIFO) stack
- Composed of two components: elements and operations.
- Scripts use various operations (opcodes) to define their operations.

Opcodes i

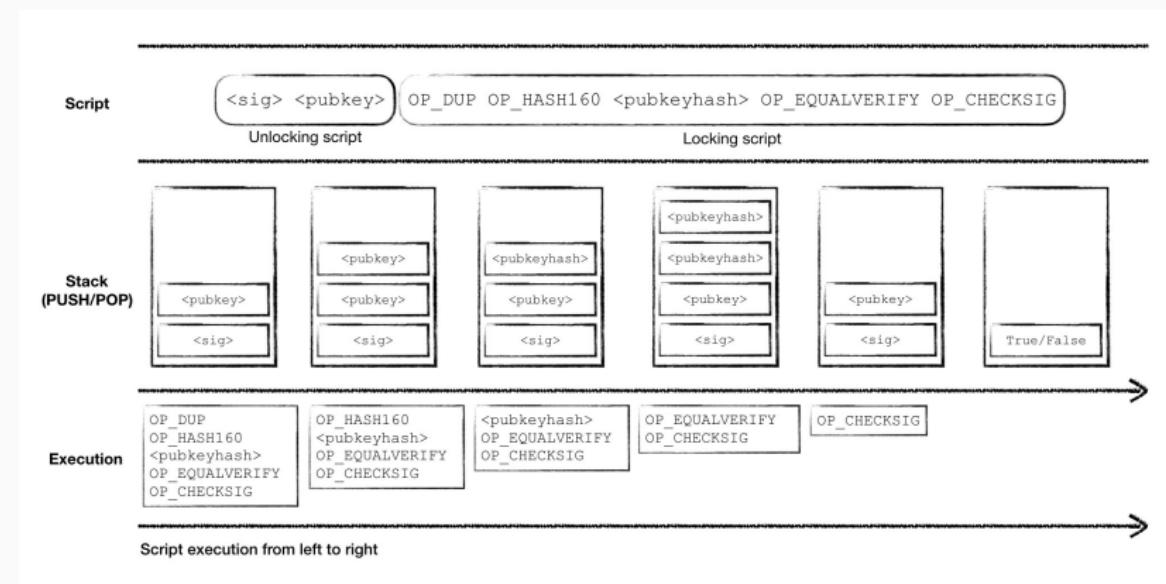
- Here are some examples of a few useful opcodes used in the Script language on the Bitcoin blockchain.

Opcodes ii

Opcode	Description
OP_CHECKSIG	This takes a public key and signature and validates the signature of the hash of the transaction. If it matches, then TRUE is pushed onto the stack; otherwise, FALSE is pushed.
OP_EQUAL	This returns 1 if the inputs are exactly equal; otherwise, 0 is returned.
OP_DUP	This duplicates the top item in the stack.
OP_HASH160	The input is hashed twice, first with SHA-256 and then with RIPEMD-160.
OP_VERIFY	This marks the transaction as invalid if the top stack value is not true.
OP_EQUALVERIFY	This is the same as OP_EQUAL, but it runs OP_VERIFY afterward.
OP_CHECKMULTISIG	This instruction takes the first signature and compares it against each public key until a match is found and repeats this process until all signatures are checked. If all signatures turn out to be valid, then a value of 1 is returned as a result; otherwise, 0 is returned.
OP_HASH256	The input is hashed twice with SHA-256.
OP_MAX	This returns the larger value of two inputs.

P2PKH script execution

- P2PKH is the most commonly used transaction type and is used to send transactions to Bitcoin addresses.



Validação de Transações

During validation, the following are checked:

- That transaction inputs are previously unspent. This validation step prevents double-spending by verifying that the transaction inputs have not already been spent by someone else.
- That the sum of the transaction outputs is not more than the total sum of the transaction inputs. However, both input and output sums can be the same, or the sum of the input (total value) could be more than the total value of the outputs. This check ensures that no new bitcoins are created out of thin air.
- That the digital signatures are valid, which ensures that the script is valid.

Blocos i

- A estrutura de um Bloco Bitcoin é mostrado na tabela:

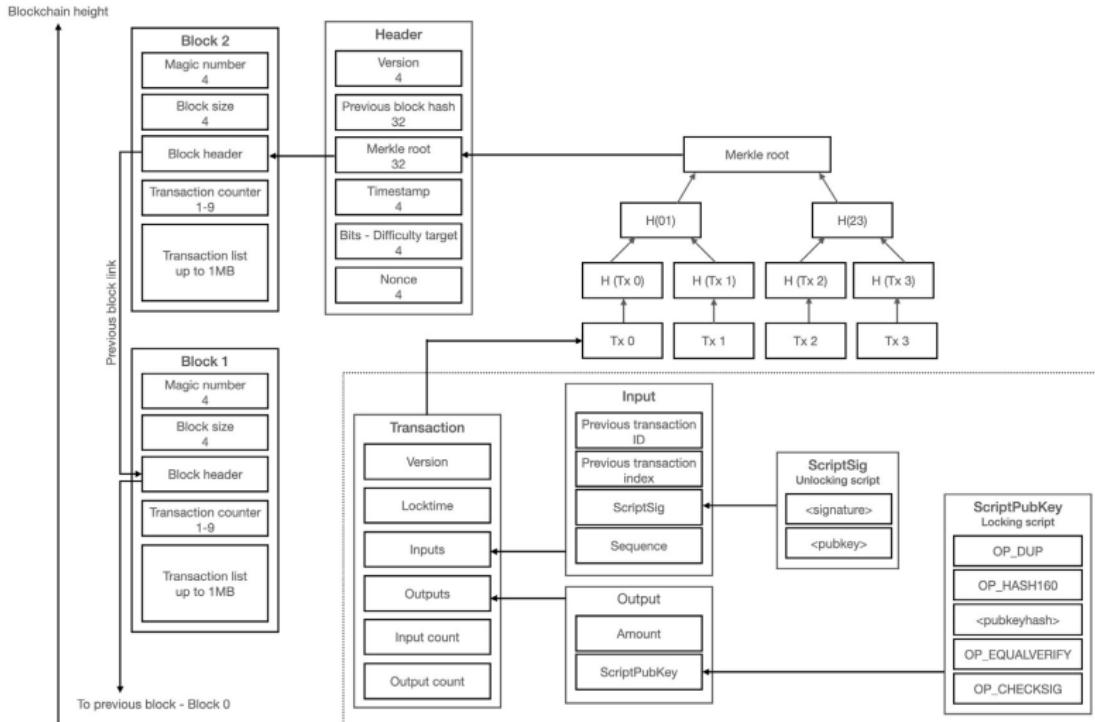
Field	Size	Description
Block size	4 bytes	The size of the block.
Block header	80 bytes	This includes fields from the block header described in the next section.
Transaction counter	Variable	The field contains the total number of transactions in the block, including the coinbase transaction. Size ranges from 1-9 bytes.
Transactions	Variable	All transactions in the block.

Blocos ii

- A estrutura do cabeçalho de um bloco:

Field	Size	Description
Version	4 bytes	The block version number that dictates the block validation rules to follow.
Previous block's header hash	32 bytes	This is a double SHA256 hash of the previous block's header.
Merkle root hash	32 bytes	This is a double SHA256 hash of the Merkle tree of all transactions included in the block.
Timestamp	4 bytes	This field contains the approximate creation time of the block in Unix-epoch time format. More precisely, this is the time when the miner started hashing the header (the time from the miner's location).
Difficulty target	4 bytes	This is the current difficulty target of the network/block.
Nonce	4 bytes	This is an arbitrary number that miners change repeatedly to produce a hash that is lower than the difficulty target.

Uma Visualização da Blockchain do Bitcoin



Bloco Genesis

O Bloco Genesis ou bloco #0 foi *hardcoded* (codificado) por suas características especiais: ele é o único que não aponta para nenhum bloco anterior. No seu *hash* foi encriptado o bloco junto com a mensagem *“The Times 03/Jan/2009 Chancellor on brink of second bailout for banks”*, manchete do jornal naquele dia. Além de servir como prova datada, a manchete escolhida representa justamente uma crítica ao sistema bancário.

Bloco Genesis ii

THE TIMES

Saturday January 3 2009 timesonline.co.uk No E9523 £1.50

Eat Out from £5
More than 900 great restaurants, including four Gordon Ramsay favourites from £15

Start collecting today today. [Purchase tickets](#)

Israel prepares to send tanks and troops into Gaza

Israel allowed foreigners to flee the Gaza Strip as it prepared for a ground offensive. At least 430 Palestinians were killed in a week of strikes. [More](#), page 3

Chancellor on brink of second bailout for banks

Billions may be needed as lending squeeze tightens
[Finance Editor](#) [Deputy Political Editor](#) [Economy](#) [Scandinavia Editor](#)

After a banking crisis last year, the Chancellor will now have to consider whether to bail out the failing banking sector.

The Bank of England has already stepped in, with a decision to provide billions more to the banking system than the £50 billion it had initially planned. Doing so would mean that the central bank would have to issue further bonds to the tune of £25 billion a month, unless there is a further relaxation of the availability of credit.

The idea was to plan to 'keep the banks as healthy as possible without putting too many more people at risk through lending losses.'

[More](#). The Times has learned. [The Bank of England's latest year-](#)

day bank, despite increased pressures, the banks are still lending in the final quarter of the year. In fact, the latest figures show that further expansion of taxpayers' cash injections into the banking system could still be required to support the industry.

The Bank is expected to take yet another step to ease the pressure on the banking system by lowering the base rate of 5 per cent. Doing so would add to the cost of borrowing for small business and individuals.

On Friday, the Chancellor will give his state-of-the-union speech to the Commons. The idea would坐着 the initial funding of the £50 billion rescue package to the Treasury Secretary, as contained in the Budget this week, to the tune of £5 billion.

Details of the Treasury plan are due to be announced on Friday. [More](#), page 3

99p

Pub quiz one the way to nowhere
[More](#), page 4

Giant Killing? Guide to the FA Cup Third Round
[Sports](#)

```
https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp
```

```

39 /**
40 * Build the genesis block. Note that the output of its generation
41 * transaction cannot be spent since it did not originally exist in the
42 * database.
43 */
44 CBlockIndex* CreateGenesisBlock(int nVersion, vector<uint256> vInitTxns,
45     const CScript& vout, const CScriptHash& vchMerkleRoot);
46 CBlockIndex* CreateNewBlock(CBlockIndex* pindexLast, const CScript& vout,
47     const CScriptHash& vchMerkleRoot, CBlockIndex* pindex, CBlockIndex* pindexNext);
48 CBlockIndex* CreateChildBlock(CBlockIndex* pindex, const CScript& vout,
49     const CScriptHash& vchMerkleRoot, CBlockIndex* pindex);
50
51 /**
52 * Main network on which people trade goods and services.
53 */
54 class CChainParams : public CChainParams {
55 public:
56     CChainParams() {
57         int nInitialIndex = CChainParams::MAIN;
58         consensus.SignatureValidation();
59         consensus.CalculateBlockWeights();
60         consensus.MandatorySigScriptSize = 25000;
61         consensus.InsRlpFlagAcceptingAnyHash();
62         setCLOCKTime(consensus.DefaultCLOCKTime());
63     }
64     const CBlockIndex* GetBlockIndexByHash(const uint256 &hash) const;
65     const CBlockIndex* GetBlockIndexByHeight(int nHeight) const;
66     const CBlockIndex* GetBlockIndexByTime(const CTime& nTime) const;
67 }
```

Fonte: <https://github.com/bitcoin/bitcoin/blob/master/src/chainparams.cpp>

Bloco Genesis iii

```
/*
 * Build the genesis block. Note that the output of its generation
 * transaction cannot be spent since it did not originally exist in the
 * database.
 *
 * CBlock(hash=000000000019d6, ver=1, hashPrevBlock=0000000000000000, hashMerkleRoot=4a5e1e, nTime=1231006505, nB
 *     CTransaction(hash=4a5e1e, ver=1, vin.size=1, vout.size=1, nLockTime=0)
 *         CTxIn(COutPoint(000000, -1), coinbase 04ffff001d0104455468652054696d65732030332f4a616e2f3230303920436861
 *             CTxOut(nValue=50.00000000, scriptPubKey=0x5F1DF16B2B704C8A578D0B)
 *             vMerkleTree: 4a5e1e
 */
static CBlock CreateGenesisBlock(uint32_t nTime, uint32_t nNonce, uint32_t nBits, int32_t nVersion,
const CAmount& genesisReward)
{
    const char* pszTimestamp = "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks";
    const CScript genesisOutputScript = CScript() << ParseHex("04678afdb0fe5548271967f1a67130b7105cd6a828e03909a6
        return CreateGenesisBlock(pszTimestamp, genesisOutputScript, nTime, nNonce, nBits, nVersion, genesisReward);
}
```

A carteira de Satoshi

- Carteira: [1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa](#)

The screenshot shows the Blockchain.com Explorer interface for the address [1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa](#). The address has transacted 3,455 times and received a total of 88.5544405 BTC (\$1,286,882.09) and sent 0.0000000 BTC (\$0.00). The current value of the address is 88.5544405 BTC (\$1,286,882.09).

Address

Format	1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa
Transactions	3,455
Total Received	88.5544405 BTC
Total Sent	0.0000000 BTC
Final Balance	88.5544405 BTC

Transactions

Date	Fee	Hash	Inputs	Outputs
2022-09-19 11:43	0.00005788 BTC (9.807 sat/B - 4 160 sat/WU - 500 bytes) (16.626 sat/vByte - 348 virtual bytes)	bb7620043cc1b054adde8a33bf09bb7b689553931e52fb99tb9785e243d94055		+0.00123604 BTC
		3GRmWMBJBxJvN2zG5Tos2bCmsRqj6fGD 394KacQFmCz0s0Cq3yDkA2wYhrW7Q 34dYmHgtsqjCUWKox7u5yNadX2zyfrw		
	0.00000144 BTC (0.640 sat/B - 0.251 sat/WU - 225 bytes) (1.000 sat/vByte - 144 virtual bytes)			+0.000001 BTC

Figura 6: [1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa](#)

A carteira de Satoshi ii

- Essa primeira transação foi incluída no bloco #0, sob o hash
`4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b.`

The screenshot shows the Blockchain.com Explorer interface. The main content area displays the details of the first Bitcoin transaction ever made, which was included in the Genesis Block (block #0). The transaction summary indicates a fee of 0.00000000 BTC and a total value of 50.00000000 BTC sent to the address 1A1dp1eP5QGeL2DMP7fTL55Lmz7DvNa. The transaction was broadcasted on January 03, 2009, at 4:15 PM GMT-2. It has 754,977 confirmations as of the time the screenshot was taken. The transaction details section provides a comprehensive breakdown of the inputs and outputs, including the hash, status (Confirmed), received time (2009-01-03 16:15), size (204 bytes), weight (816), and the fact that it was included in the first block (Included in Block: 0). The total input and output values are both 0.00000000 BTC.

Figura 7: `4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b`

A carteira de Satoshi iii

The screenshot shows the WhatsOnChain website interface for the Bitcoin address `1A1zP1eP5QGefi2DMPTfTL5Lmv7DivfNa`. The top navigation bar includes links for API, About, Classic View, English, Metrics, BSV, New Block, and a search bar. The main content area displays the address, a QR code, and its current balance of `67.97456184 BSV`. It also shows the first seen timestamp as `2009-01-03 18:10:05`. The transaction section lists 1,162 transactions, with the most recent one being a deposit of `0.91 BSV` from `3387418addb4927299e5032f515aa442a6587de54677f98a93b8fe7798e648` on `2011-05-13 21:04:05`.

Figura 8: `1A1zP1eP5QGefi2DMPTfTL5Lmv7DivfNa`

A carteira de Satoshi iv

- Detalhes da Transação:

The screenshot shows the WhatsOnChain interface for a specific Bitcoin transaction. At the top, there are navigation links for 'APB', 'About', 'Classic View', 'English', 'Mandarin', 'CSV', 'New Block', and a search bar. The main content area displays the transaction details for hash `4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b`. The transaction is identified as Block #0. It was timestamped at 2009-01-03 18:15:05 UTC. The fees collected were 0 BSV. There were 757,989 confirmations. The transaction size is 204 B.

Below this, the Hex representation of the transaction is shown, starting with `04ffff985d810445546865205469605732030332f4a618e02f323b3039204380810e63856c4c6f72200f6e20627269606b206f66207365636`. The Decoded section shows the message "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks".

At the bottom, the transaction is summarized with one input and one output. The input is a newly minted coin of 50 BSV. The output goes to the address `1A1zPlePSQGeft1Z0MPYTL5SLm7DifvNs` also containing 50 BSV.

Figura 9: `4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b`

A carteira de Satoshi v

- Scripts

The screenshot shows the WhatsOnChain interface for a specific transaction. At the top, there are navigation links for 'APIs', 'About', 'Classic View', 'English', 'Mandarin', 'CSV', and 'New Block'. A search bar is also present.

The main content area displays the transaction ID: **4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b**. Below it, there are tabs for 'Details' (selected), 'Scripts' (highlighted with a blue border), and 'JSON'.

The 'Input Scripts' section shows one input script:

```
# 0 0 04ffff001d010445440652054696d05732830332f4a616e2f323030392843d8616e83656c6c8f72206f6e2062722896e6b200f66287365636f6e64206261696c8f757420666f722062616e6b73
```

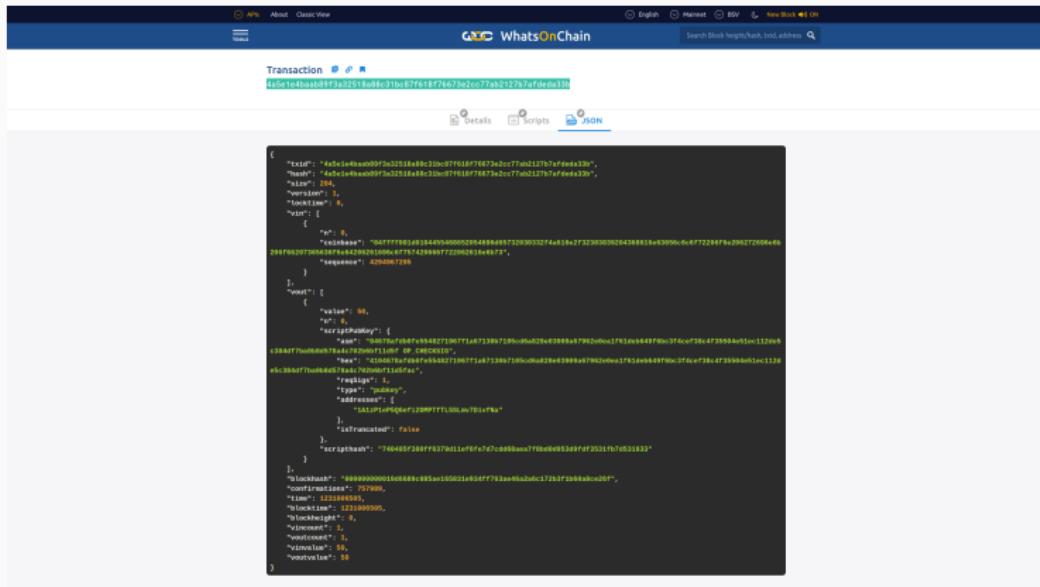
The 'Output Scripts' section shows one output script:

```
# 0 0 04678afdb8fe5548271967f1a87138b7105cd6a829e83909a87962e0ea1f61deb649f6bc3f4cef38c4f35504e51e112de5c984df7ba888d578a4c782b6bf11d5f OP_CHECKSIG
```

At the bottom of the page, there is a footer with links to 'TAAL API', 'WOC API', 'Cookies Policy', 'Terms of Use', 'Privacy', 'About', 'Contact us', 'Join Group', 'Follow Us', and 'WOC Services Status'. A 'PoweredByTAAL' logo and the copyright notice '© 2022 TAAL Distributed Information Technologies Inc.' are also present.

A carteira de Satoshi vi

• JSON



The screenshot shows a web browser displaying the WhatsOnChain website. The URL in the address bar is `https://www.whatsonchain.com/tx/4a5e1e4baa9f73a52518a09c31b08776187767fe2e077a3c12767afdeda33b`. The page has a dark blue header with the WhatsOnChain logo and navigation links for English, Harvest, BSV, New Block, and ON.

The main content area shows a transaction details page. At the top, there are tabs for Details, Scripts, and JSON. The JSON tab is currently selected, displaying the raw JSON representation of the transaction.

```
[{"txid": "4a5e1e4baa9f73a52518a09c31b08776187767fe2e077a3c12767afdeda33b", "hash": "045e1e4baa9f73a52518a09c31b08776187767fe2e077a3c12767afdeda33b", "size": 268, "version": 1, "locktime": 0, "vin": [{"vout": 0, "scriptSig": "3044532531848c3bc0877618776673ec77a3c12767afdeda33b", "sequence": 4394087256}, {"vout": 1, "scriptSig": "3044532531848c3bc0877618776673ec77a3c12767afdeda33b", "sequence": 4394087256}], "vout": [{"script": "OP_RETURN", "value": 0}, {"script": "OP_DUP OP_HASH160 04777f805a23844954000052054884957329303374438a273230302043348818e3d956-0c4f72288fb205272050648294f9620730e055954a3474262160a4775742949772290216e0077", "value": 0}], "scriptSig": "4394087256"}]
```

A carteira de Satoshi vii

The screenshot shows a web browser window for the URL whatsonchain.com/address/1A1zP1ePSQGeF12DMPTfTL5SLav7D1vFNa. The page title is "Address 1A1zP1ePSQGeF12DMPTfTL5SLav7D1vFNa". The main content area displays the following information:

Address: 1A1zP1ePSQGeF12DMPTfTL5SLav7D1vFNa

QR CODE: A QR code corresponding to the address.

Balance: 67.9928777 BSV (Confirmed)

First Seen: 2009-01-03 18:15:05

Script Hash: 9b91df4e368ea28f8dc8423bcf7a4923e3a12d387c875e47a0cfb
F90b5c49161

Script Public Key: 76a9146e907b15cbf27d5425399ebf60fb50ebb8f1888ac

Transaction: 19.574

Details: A button to view transaction details.

JSON: A button to view JSON data.

19.574 Transactions: A section listing 19,574 transactions. The first two transactions are shown in detail:

Index	Transaction ID	Tag
#18413	4922c19661c9942f03196b5c82cafe1b34b13fb498998011122dfcbbf76547	+ 1e-8 BSV
#18412	46306a97eab7f95845669e7488920b719d382fb93c6a3da2bd9ec447c6243b	+ 1e-8 BSV

Bottom Navigation: Includes links for "Index", "115", "116", "117", "118", "119", "1950", and "Next".

Figura 10: Saldo em 14/04/2023

A carteira de Satoshi viii

The screenshot shows the blockchain.com website interface. On the left, there's a sidebar with navigation links like Início, Preços, Gráficos, NFTs, DeFi, Academia, Notícias, Programadores, Wallet, Exchange, Bitcoin, Ethereum, Bitcoin Cash, BTC Testnet, BCH Testnet, and Português. The main content area displays the balance of the Bitcoin address 1A1zPfPqGefZDMPtTLSSlnw7DvNra. The balance is listed as 72.61224698 BTC or \$2.21k USD. Below the balance, there's a banner for BC.GAME with a 760% deposit bonus. The page also shows a summary of the address's history, including its creation date (2009-01-04) and various transaction details.

Figura 11: Saldo em 14/04/2023 Fonte: blockchain.com

Quem é Satoshi Nakamoto?

The screenshot shows a news article from CanalTech. The title is "Elon Musk afirma em tweet que ele não é o criador da criptomoeda Bitcoin". Below the title is a photo of Elon Musk in a dark environment with pipes and equipment. To the left of the main content area, there is a sidebar with a purple background containing the text "Aquele produto que você tá procurando tá aqui!" and a "compartilhar" button. To the right, there is an advertisement for Intel notebooks with a discount offer of 30% off and 7% off on PIX. A vertical sidebar on the far right says "Baixe o app!" and shows icons for iOS and Android. The URL at the bottom of the page is <https://go.gle/dLgJg>.

Figura 12: Elon Musk Fonte: CanalTech

Quem é Satoshi Nakamoto? ii

The screenshot shows a news article from livecoins.com.br. The title is "Há 8 anos, revista expôs vida de Satoshi Nakamoto... e foi processada". The main image is a portrait of Dorian Nakamoto holding up a driver's license. On the left, there are social media sharing buttons for Facebook, Twitter, and LinkedIn. Below the image, a caption reads: "Dorian Nakamoto, apontado falsamente como criador do Bitcoin devido ao seu nome". The text of the article discusses a Newsweek cover from 2009 that claimed Dorian Nakamoto was Satoshi Nakamoto, leading to legal issues. The article is dated 07/03/2022 at 15:18. To the right, there is a sidebar titled "Últimas notícias" with several thumbnail images and titles related to Bitcoin and cryptocurrencies.

Figura 13: Dorian Nakamoto Fonte: Livecoins

Quem é Satoshi Nakamoto? iii

The screenshot shows a news article from livecoins.com.br. The title is "Criador do Linux diz ser Satoshi Nakamoto, o criador do Bitcoin". The article discusses a modification in the Linux kernel source code where Linus Torvalds added the text "eu sou Satoshi". Below the article is a photo of Linus Torvalds. To the right, there are sidebar ads for an Abarth SUV and other news stories.

A modificação também pode ser uma maneira não muito discreta de dizer aos geeks que ele é de fato Satoshi Nakamoto, algo que seria crível porque ele tem as habilidades para isso e alguns fatos se encaixam na história do Bitcoin.

Linus Torvalds | Por: TrustNode | 28/01/2022 18:45

Siga no Google News

Linus Torvalds, o criador do sistema operacional **Linux**, parece ter modificado uma única linha no Kernel do Linux e incluiu uma afirmação de que ele é Satoshi Nakamoto. A modificação diz “**eu sou Satoshi**”, o que pode ser uma brincadeira ou uma afirmação real de que ele criou o Bitcoin.

Linus Torvalds já foi suspeito de ser Satoshi Nakamoto, em parte porque ele criou o Git, que se acredita ter inspirado a blockchain, e em parte

O Primeiro SUV Abarth Do Mundo

Últimas notícias

Na fronteira com Brasil, Paraguai fecha mineradora de criptomoedas suspeita de roubar energia

“Criptomoedas tiram o sono de quem apostou nisso”, diz Banco Central

Citando Luiz, criador do Megaupload fala sobre fim da hegemonia do dólar

Hackers atacam plataforma

Figura 14: Linus Torvalds Fonte: Livecoins

Quem é Satoshi Nakamoto? iv



Figura 15: Linus Torvalds Fonte: cointelegraph

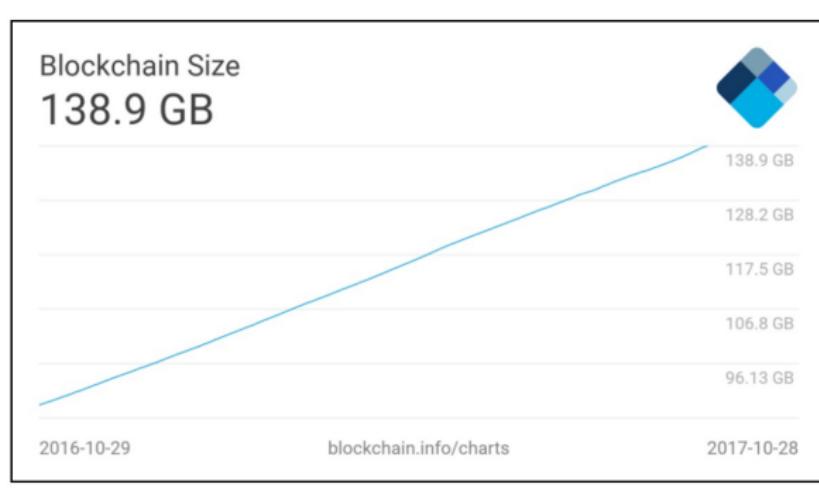
Quem é Satoshi Nakamoto? v



Figura 16: Steve Jobs
Fonte: Exame 10/04/2023

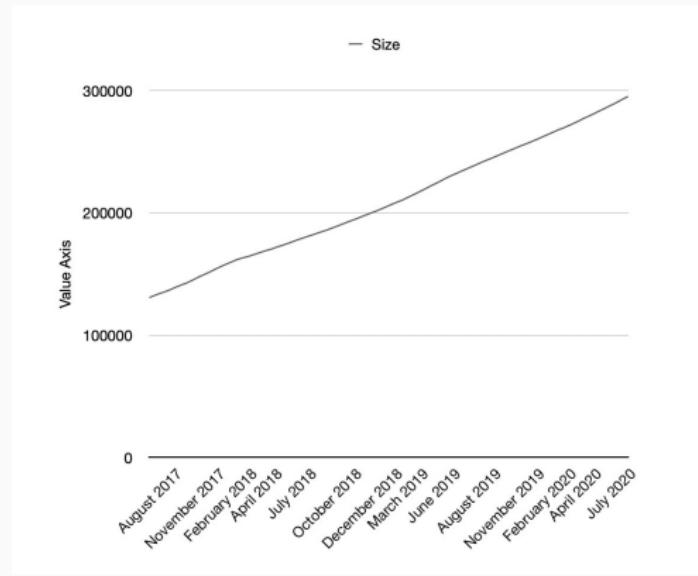
Tamanho do Blockchain do Bitcoin

- O Blockchain do Bitcoin tinha em **October 29, 2017**, aproximadamente: **139GB**



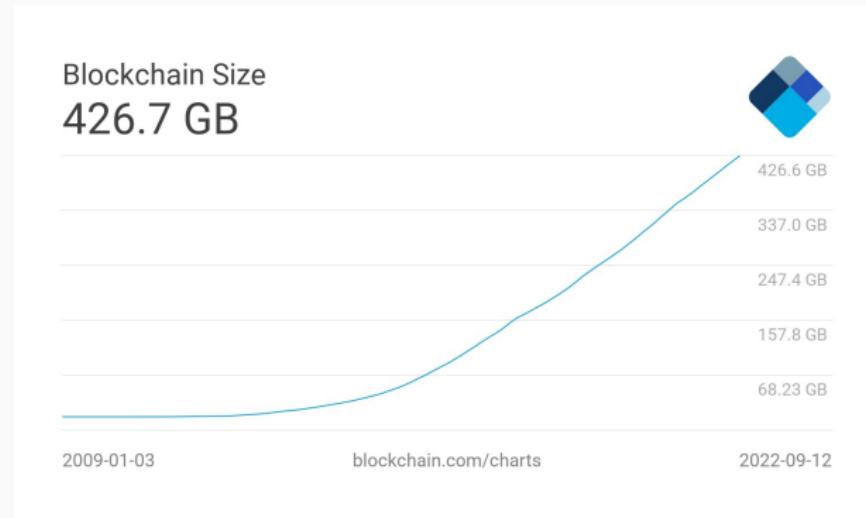
Tamanho do Blockchain do Bitcoin ii

- A figura mostra a evolução de Aug 2017 para Jul 2020.
Aproximadamente, 286GB.



Tamanho do Blockchain do Bitcoin iii

- A figura mostra a evolução de Jan 2009 para Set 2022.
Aproximadamente, **426.7GB**.



- Fonte: <https://www.blockchain.com/charts/blocks-size>

Tamanho do Blockchain do Bitcoin iv

- Tamanho em 14/04/2023: 472.9GB

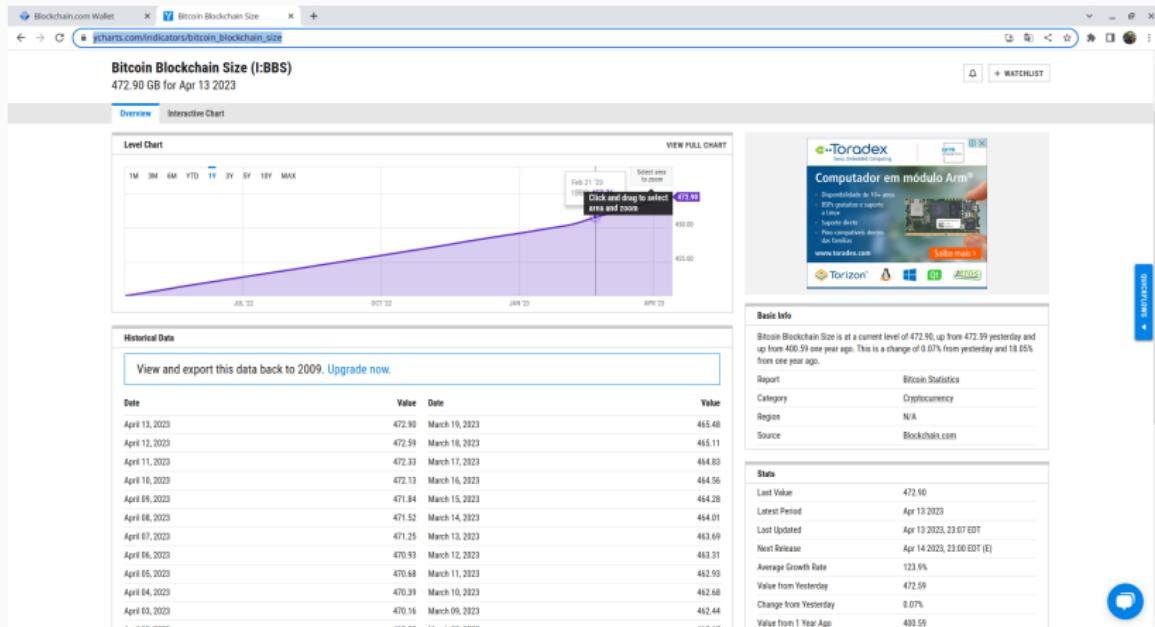
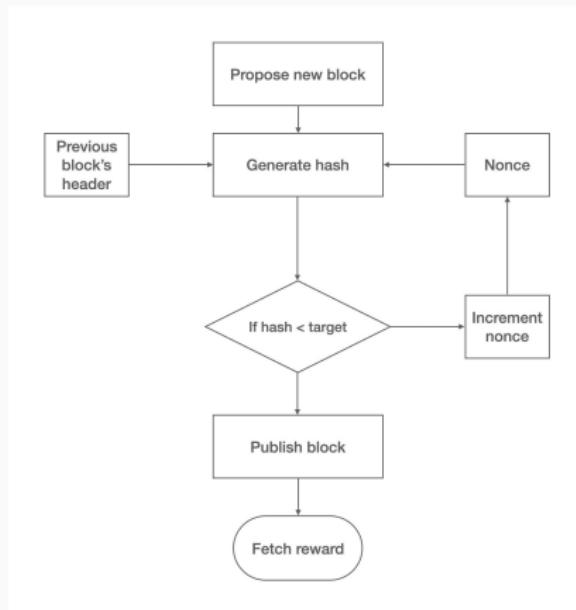


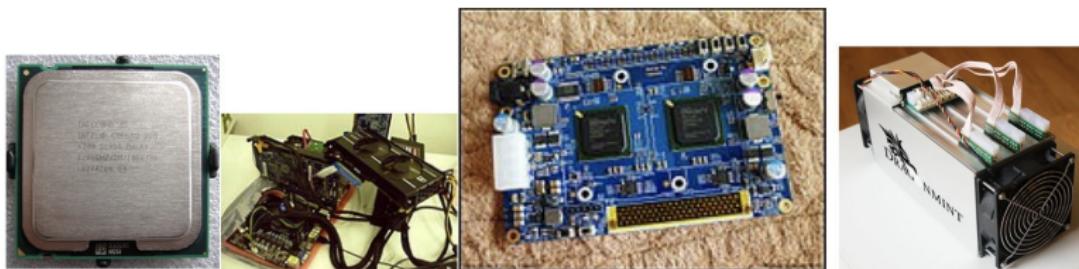
Figura 17: Tamanho em 14/04/2023 Fonte: ycharts

Mineração i

- Sincronização com a rede
- Validação de Transações
- Validação de Bloco
- Criação de novo bloco
- Executar a *Proof of Work (PoW)*
- Buscar Recompensa



Mining systems



Four types of mining hardware. From left to right: a CPU, GPU, FPGA, and an ASIC

Leitura Recomendada

Capítulo 5/6: Introduction Bitcoin: IMRAN BASHIR. Mastering Blockchain: Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Smart Contracts (Contratos Inteligentes)

- Colocar definição
- Falar o que são
- Uso e utilidade
- Citando o artigo original
- Artigo Seminal

Leitura Recomendada

Capítulo 9/10: Smart Contracts

Ethereum

- Vitalik Buterin (<https://vitalik.ca>) conceitualizou Ethereum (<https://ethereum.org>) em Novembro de 2013.
- A ideia central proposta foi o desenvolvimento de uma linguagem **Turing-completa** para permitir o desenvolvimento de programas arbitrários (contratos inteligentes) para *blockchain* e Aplicações Descentralizados (DApps).
- Este conceito difere do *Bitcoin*, onde a linguagem de **script** é limitada e permite apenas as operações necessárias.

Ethereum – Overview ii

ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER PETERSBURG VERSION 4ea7b96 – 2020-06-08

DR. GAVIN WOOD
FOUNDER, ETHEREUM & PARITY
GAVIN@PARITY.IO

ABSTRACT. The blockchain paradigm when coupled with cryptographically-secured transactions has demonstrated its utility through a number of projects, with Bitcoin being one of the most notable ones. Each such project can be seen as a simple application on a decentralised, but singleton, compute resource. We can call this paradigm a transactional singleton machine with shared-state.

Ethereum implements this paradigm in a generalised manner. Furthermore it provides a plurality of such resources, each with a distinct state and operating code but able to interact through a message-passing framework with others. We discuss its design, implementation issues, the opportunities it provides and the future hurdles we envisage.

Figura 18: O *Ethereum Yellow Paper*¹

- O *Ethereum Yellow Paper* foi escrito por Dr. Gavin Wood, o fundador do Ethereum e da Parity (<http://gavwood.com>), e serve como uma especificação formal do protocolo da Ethereum.
- Qualquer pessoa pode implementar um cliente Ethereum seguindo as especificações de protocolo definidas no artigo.

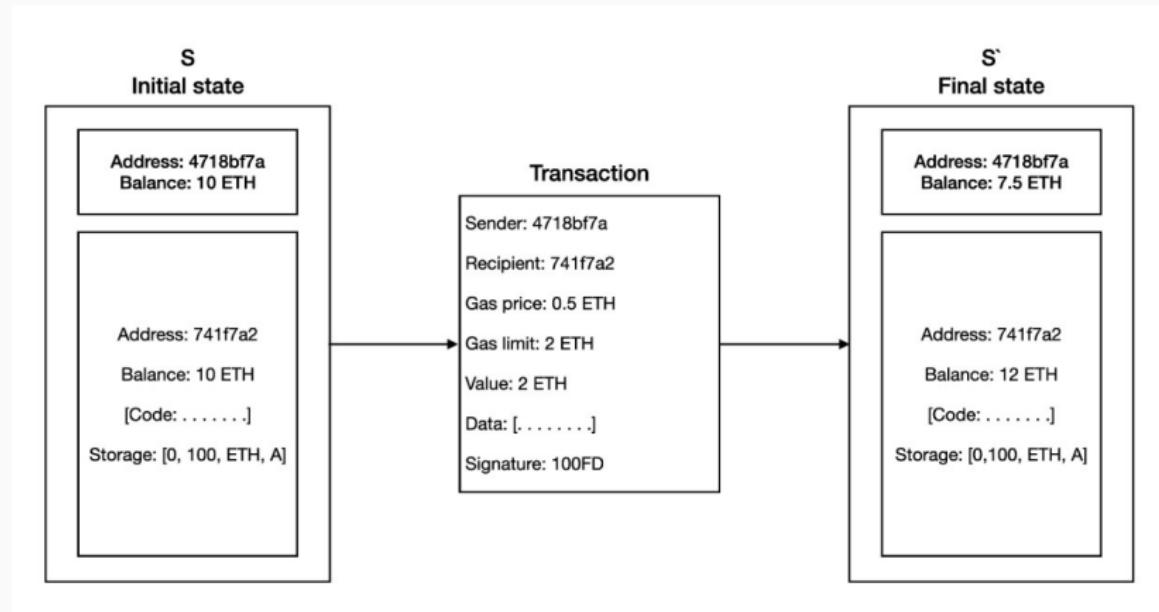
¹<http://ethdocs.org/en/latest/contracts-and-transactions/developer-tools.html#developer-tools>

Ethereum Releases i

- A primeira versão da *Ethereum*, denominada **Olympic**, foi liberada em Maio de 2015.
- Dois meses mais tarde, a versão chamada de **Frontier** foi liberada em Julho.
- Outra versão, a **Homestead** com várias melhorias foi liberada em Março de 2016.
- A release chamada de **Muir Glacier**, que atrasou a **difficulty bomb** (<https://eips.ethereum.org/EIPS/eip-2384>).
- Um grande lançamento antes disso foi **Istanbul**, que incluiu mudanças em torno de privacidade e dimensionamento capacidades.
- Uma lista completa com todas as *releases* anunciadas é mantida em <https://github.com/ethereum/go-ethereum/releases>.

A Blockchain Ethereum i

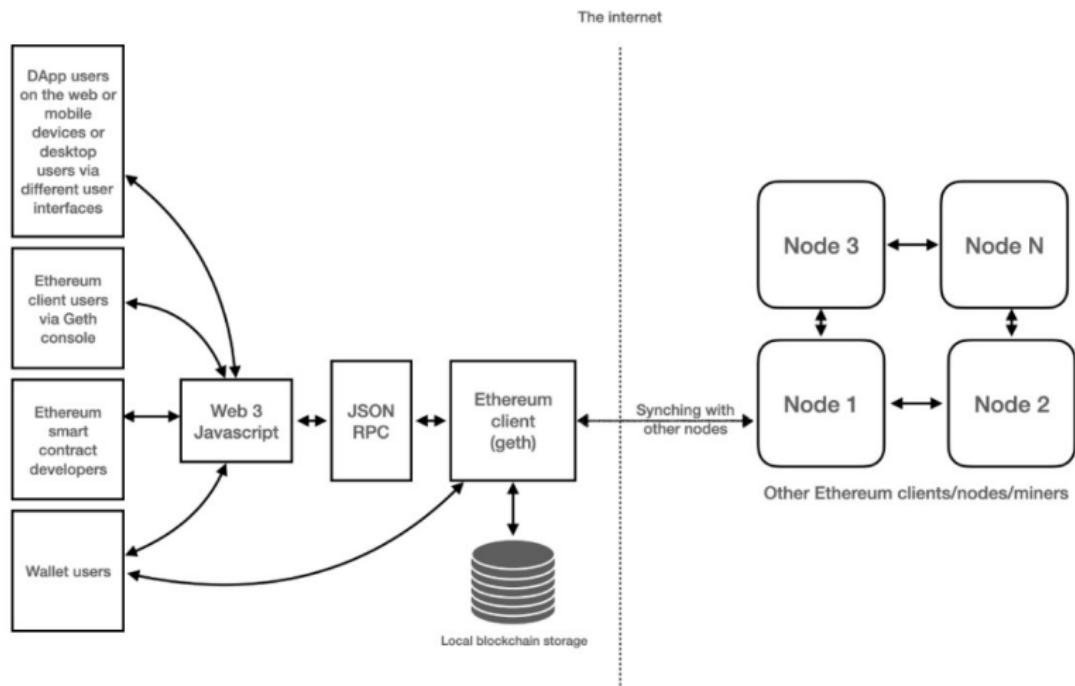
- O Ethereum, assim como qualquer outro *blockchain*, pode ser visualizado como uma máquina de estado baseada em transações.



- A ideia principal da *blockchain* da *Ethereum*, um estado gênese é transformado em um estado final executando transações de forma incremental. A transformação final é então aceita como a versão absoluta e indiscutível do estado. A função de transição de estado *Ethereum* é mostrada, onde a execução de uma transação resultou em uma transição de estado.

- O caso de uso mais comum da rede *Ethereum* é o envio e o recebimento de pagamentos.
- Para isso, o usuário assina a transação e a envia, que se propaga na rede, momento em que os mineradores a pegam, verificam e iniciam a Prova de Trabalho (PoW).
- Se PoW for bem sucedida, o bloco com a transação é finalizado e propagado, e um novo bloco é adicionado à cadeia
- Para enviar e receber transações, um software de carteira é usado: por exemplo, carteiras são usadas em dispositivos móveis.

Arquitetura de Alto Nível da Ethereum



A rede *Ethereum* é uma rede *peer-to-peer* onde os nós participantes mantêm a *blockchain* e contribuem para o mecanismo de consenso. As redes podem ser divididas em três tipos, com base nos requisitos e uso.

A mainnet

A **mainnet** é a atual rede *Ethereum*. Seu ID de rede é 1 e seu ID de cadeia (*chain*) é também 1. Os IDs de rede e de cadeia são usados para identificar a rede. Um explorador de blocos que mostra informações detalhadas sobre blocos e outras métricas relevantes estão disponíveis em <https://etherscan.io>, que pode ser usado para explorar a blockchain Ethereum.

Testnets

Existem um número de redes de testes (testnets) disponíveis para Ethereum. Elas tem como objetivo fornecer um ambiente de testes para contratos inteligentes e DApps antes de serem implantados para produção na rede *blockchain*. Além disso, sendo redes de teste, elas permitem experimentos e pesquisa. A principal testnet é chamada **Ropsten**, que contém todas as características de outras redes de propósito especial menores que foram criados para fins específicos. Por exemplo, outras redes de teste incluem **Kovan** e **Rinkeby**, que foram desenvolvidos para testar as versões do **Byzantium**. As mudanças que foram implementados nessas redes de teste menores também foram implementados em **Ropsten**. Agora a rede de teste **Ropsten** contém todas as propriedades de **Kovan** e **Rinkeby**.

Redes Privadas

As *private nets* são redes privadas que podem ser criadas gerando-se um novo *genesis block*. Este é geralmente o caso em redes *blockchain* privadas, onde um grupo privado de entidades iniciam sua rede *blockchain* e a usam como uma blockchain autorizada ou de consórcio.

Elementos do Ecossistema Ethereum i

- Chaves e Endereços
- Contas
- Transações e mensagens
- Criptomoeda/Tokens Ether
- A Ethereum Virtual Machine (EVM)
- Smart contracts e contratos nativos.

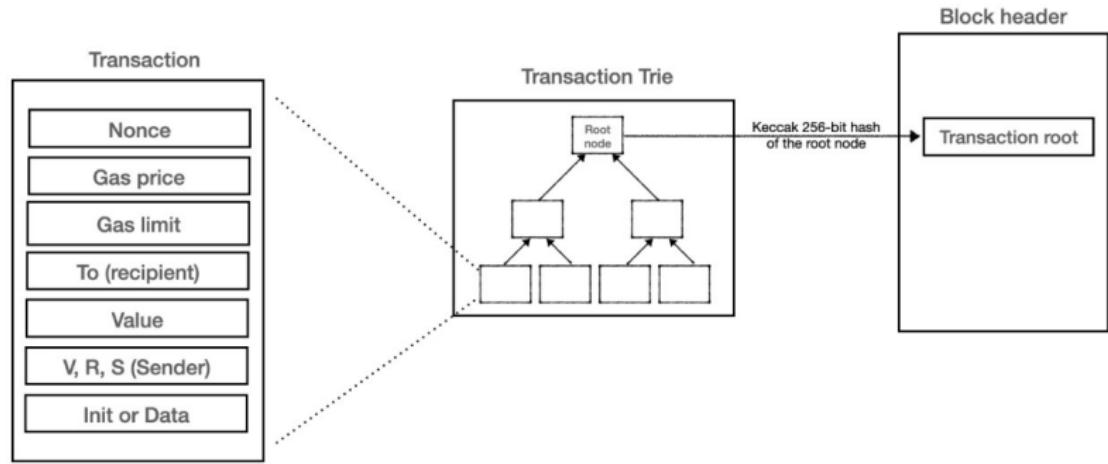
Tipos de contas

- EOAs: *Externally Owned Accounts*. Contas de usuários representadas por um endereço.
- CAs: *Contract accounts*. Criadas como resultado do *deployment* de um contrato inteligente, também representado por um endereço.

Transações e Árvore de Transações (trie) i

Transações

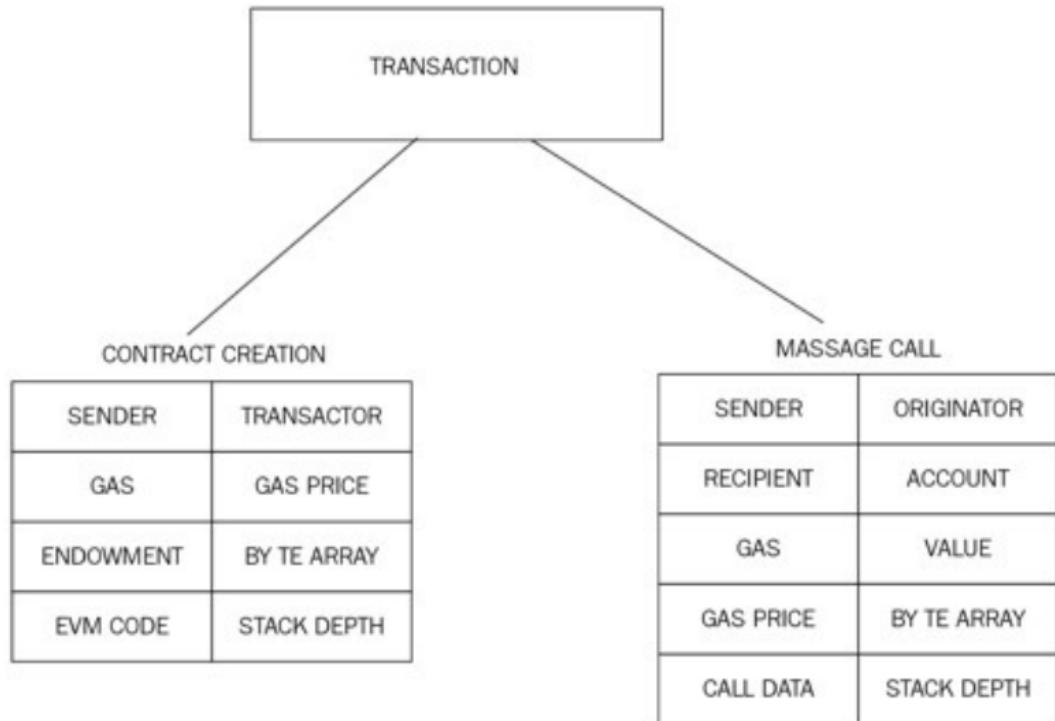
Uma transação no *Ethereum* consiste em vários campos, como mostrado aqui, junto com a *transaction trie*. O diagrama também mostra a relação entre a tentativa de transação e o cabeçalho do bloco.



Tipos de Transações i

- Existem três tipo de transações:
 - Criação de Contrato
 - *Call*
 - Transferência de Valor

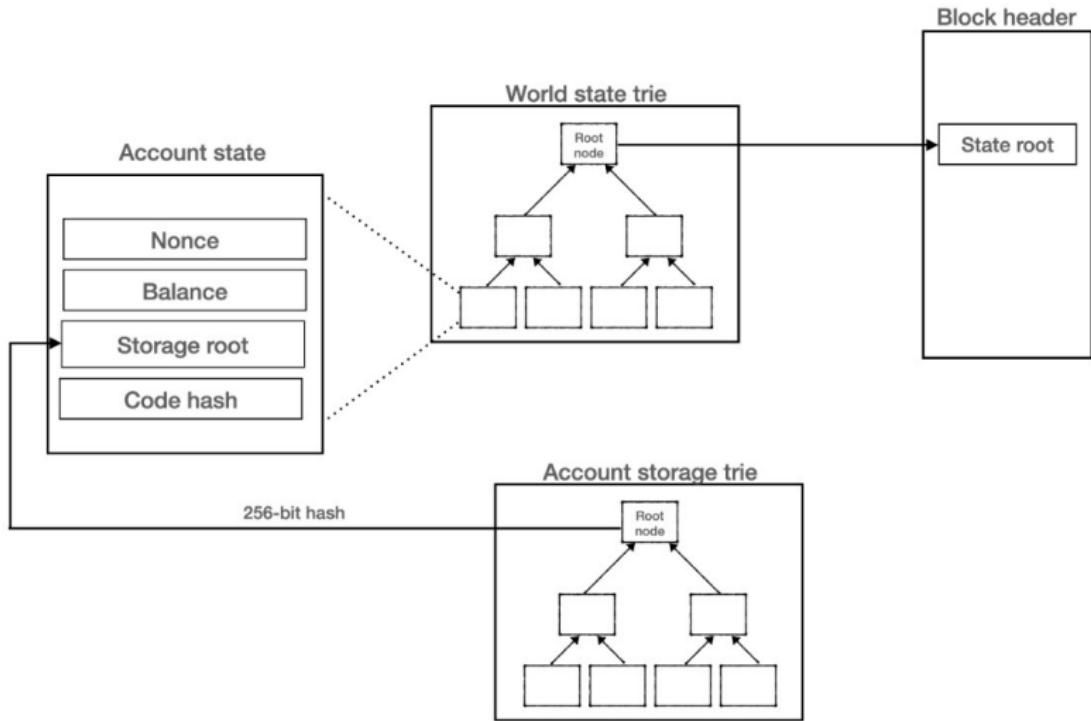
Tipos de Transações ii



Estado da conta e armazenamento na trie i

O diagrama mostra os campos contidos no estado da conta e como os vários elementos estão contidos no *world state* trie:
* World state trie
* State root
* Account state
* Account storage trie

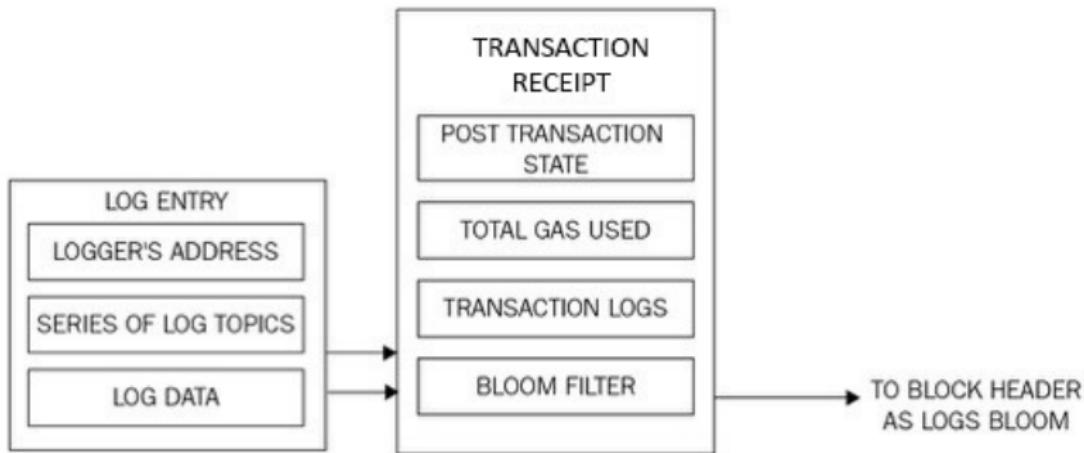
Estado da conta e armazenamento na trie ii



Recibos de Transações i

- Recibos de Transações (transaction receipts) são gerados como resultado da execução de transações.
- Logs também são atualizados em conformidade.
- Ambas as estruturas de dados contêm vários campos, conforme mostrado abaixo:

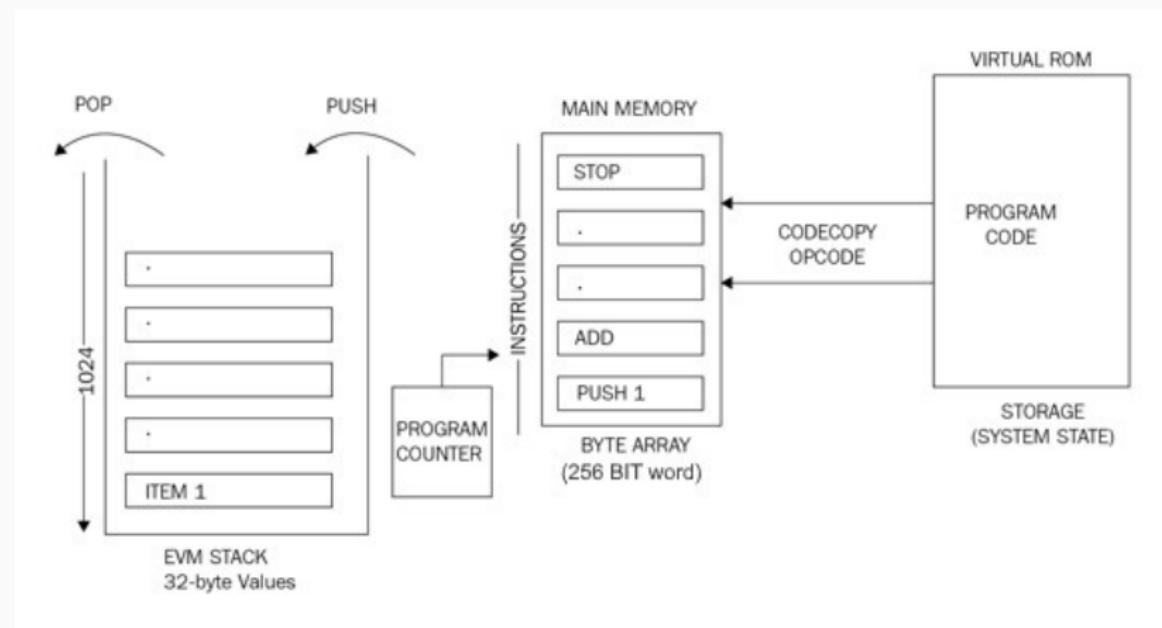
Recibos de Transações ii



The Ethereum Virtual Environment (EVM) i

- Stack size based on LIFO queue: Last In, First Out.
- 1024 stack depth limit
- Turing complete but limited by gas, making it quasi-Turing complete
- Big-endian design
- Storage available to EVM
 - Memory
 - Storage
 - Stack

EVM operation design



Execution environment i

O ambiente de execução do *Ethereum* consiste em vários elementos, como mostrado:

Address of code owner
Sender address
Gas price
Input data
Initiator address
Value
Bytecode
Block header
Message call depth
Permission

Machine State i

Uma Máquina de Estado ou *Machine state* é uma tupla compreendendo vários campos, como mostrado em:

MACHINE STATE

AVAILABLE GAS

PROGRAM COUNTER

MEMORY CONTENTS

NUMBER OF WORDS

STACK CONTENTS

Contratos Nativos i

Existem nove contratos pré-compilados ou contratos nativos na versão Ethereum Istanbul:

- * *The elliptic curve public key recovery function*
- * *The SHA-256-bit hash function*
- * *The RIPEMD-160-bit hash function*
- * *The identity/datacopy function*
- * *Big mod exponentiation function*
- * *Elliptic curve point addition function*
- * *Elliptic curve scalar multiplication*
- * *Elliptic curve pairing*
- * *Blake2 compression function 'F'*

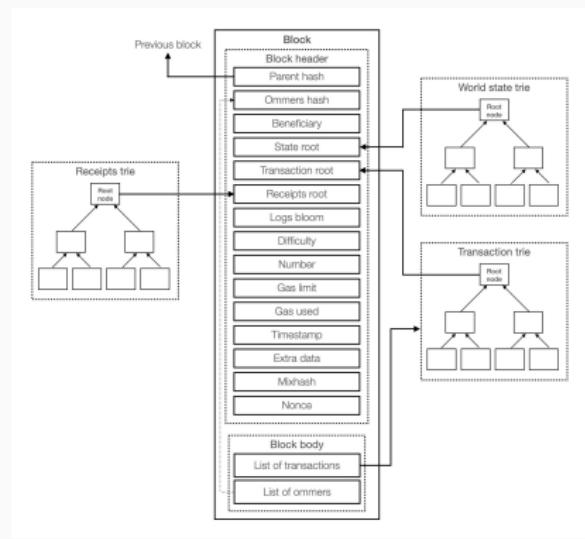
Um pouco mais de Ethereum

Outline i

- Blocos e *Blockchain*
- Wallets e Software Clientes
- Nós e Mineradores
- APIs e ferramentas
- Protocolos suportados
- Linguagens de Programação

Blocks e Blockchain i

Um bloco *Ethereum* consiste em vários campos, conforme diagrama. *State root*, *transaction root* e *receipts root* são *root hashes* de suas respectivas árvores.



Mecanismo de Validação de Blocos

The Ethereum block validation mechanism checks the following conditions:

- * If it is consistent with uncles and transactions. This means that all ommers satisfy the property that they are indeed uncles, and also if the Proof of Work (PoW) for uncles is valid.
- * If the previous block (parent) exists and is valid.
- * If the timestamp of the block is valid. This means that the current block's timestamp must be higher than the parent block's timestamp. Also, it should be less than 15 minutes into the future. All block times are calculated in epoch time (Unix time).

If any of these checks fails, the block will be rejected. A list of errors for which the block can be rejected is presented here:

- * The timestamp is older than the parent
- * There are too many, or duplicate uncles
- * The uncle is an ancestor, or the uncle's parent is not an ancestor
- * There is non-positive difficulty
- * There is an invalid mix digest or PoW

Block finalization is a process that is run by miners to validate the contents of the block and apply rewards. It results in four steps being executed. These steps are described here:

1. **Ommers (uncles) validation.** In the case of mining, determine ommers. The validation process of the headers of stale blocks checks whether the header is valid and whether the relationship between the uncle and the current block satisfies the maximum depth of six blocks. A block can contain a maximum of two uncles.

2. **Transaction validation.** In the case of mining, determine transactions. This process involves checking whether the total gas used in the block is equal to the final gas consumption after the final transaction, in other words, the cumulative gas used by the transactions included in the block.

3. **Reward application.** Apply rewards, which means updating the beneficiary's account with a reward balance. In Ethereum, a reward is also given to miners for stale blocks, which is 1/32 of the block reward. Uncles that are included in the blocks also receive 7/8 of the total block reward. The current block reward is 2 ether. It was reduced first from 5 ether to 3 with the Byzantium release of Ethereum. Later, in the Constantinople release

(<https://blog.ethereum.org/2019/02/22/ethereum-constantinople-st-petersburg-upgrade-announcement/>), it was reduced further to 2 ether. A block can have a maximum of two uncles.

4. **State and nonce validation.** Verify the state and block nonce. In the case of mining, compute a valid state and block nonce.

Mecanismo de Block difficulty i

O mecanismo de dificuldade do bloco é representado pela fórmula abaixo, que garante que os blocos sejam produzidos a uma taxa constante:

$$\text{block_diff} = \text{parent_diff} + \text{parent_diff} // 2048 * \max(1 - (\text{block_timestamp} - \text{parent_timestamp}) // 10, -99) + \text{int}(2 * ((\text{block.number} // 100000) - 2))$$

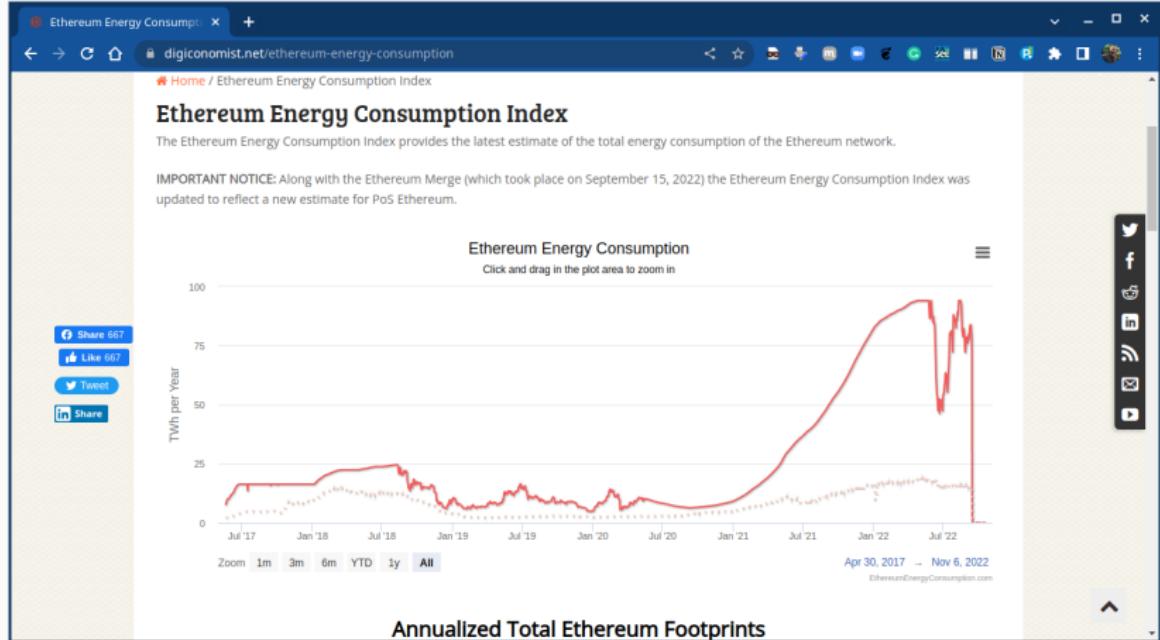
O *gas cost* de uma transação pode ser calculado usando esta fórmula:

$$\text{Totalcost} = \text{gasUsed} * \text{gasPrice}$$

Consumo de Energia i

- Com a última atualização **Merge** que trocaram a *Proof of Work* (PoW) pela *Proof of Stake* (PoS) tendo como uma das motivações a questão ambiental. Houve um grande impacto no consumo de energia.

Consumo de Energia ii



- Wallets
- Light clients
- Existem três tipos de sincronização de clientes:
 - **Full:** Nesse modo de sincronização, o cliente **Geth** faz um *download* completo da *blockchain* para o nó local. Isso significa que ele obtém todos os cabeçalhos e corpos dos blocos e valida todas as transações e blocos desde o bloco *genesis*. No início de 2020, o tamanho do *blockchain* Ethereum era de aproximadamente **210GB**, e baixar e manter isso pode ser um problema.
 - Hoje, 18 de outubro de 2022 o tamanho chega a **966.06GB**, segundo https://ycharts.com/indicators/ethereum_chain_full_sync_data_size.

- **Fast:** Neste modo é feito o *download* completo, mas somente recupera e verifica somente os 64 blocos anteriores ao bloco corrente. Depois disso, ele verifica os novos blocos na íntegra. Não reproduz e verifica todas as transações históricas desde o bloco *genesis*, em vez disso, ele só faz os *downloads* de estado. Isso também reduz significativamente o tamanho do disco do banco de dados *blockchain*. Este é o modo padrão de sincronização do cliente **Geth**.
- **Light:** Este é o modo mais rápido e apenas baixa e armazena o estado atual. Nesse modo, o cliente não baixa nenhum bloco histórico e processa apenas os blocos mais novos.

Ethereum keystore: Key decryption process i

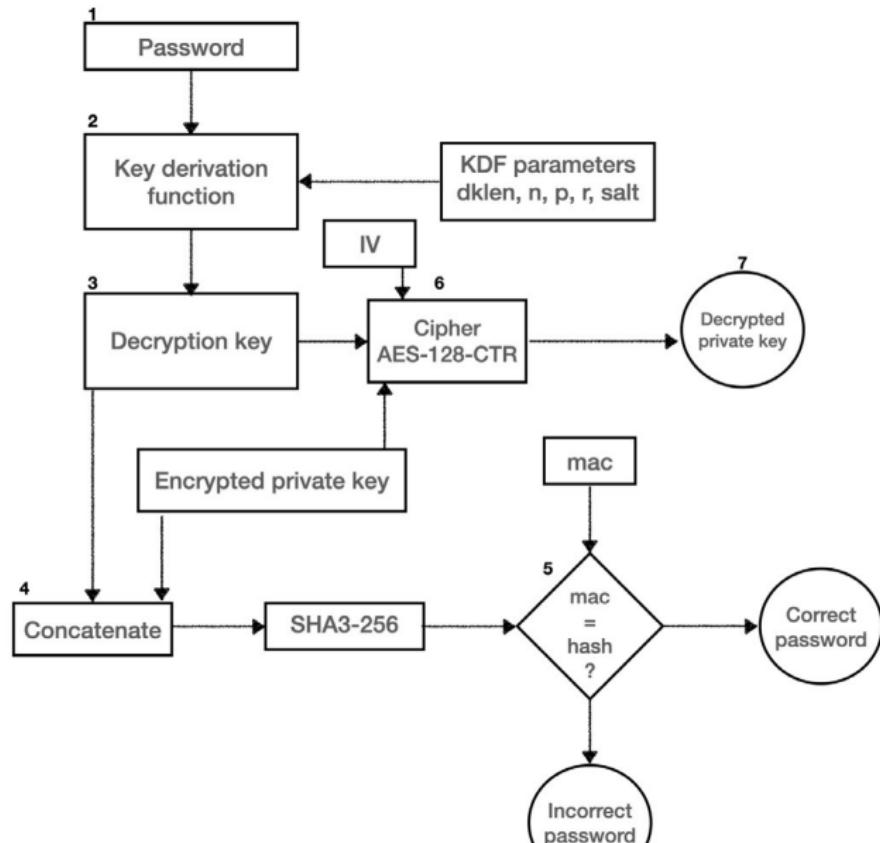
Os arquivos de chave pública e privada são armazenados no diretório keystore. Os arquivos de keystore Ethereum são arquivos JSON, cujo conteúdo se parece com o script abaixo.

Ethereum keystore: Key decryption process ii

```
{  
  "address": "ba94fb1f306e4d53587fcddcd7eab8109a2e183c4",  
  "crypto":  
  {  
    "cipher": "aes-128-ctr",  
    "ciphertext":  
      "b2ab4f94f5f44ce98e61d99641cd28eb00fd794129be25beb8a5fae89ef93241",  
    "cipherparams":  
      {"iv": "a0fdf0a6d314a62ba6a370f438faa57"},  
    "kdf": "scrypt",  
    "kdfparams":  
      {"dklen": 32, "n": 262144, "p": 1, "r": 8},  
    "salt": "be3e99203c24ffcb71a6be2823fc7a211c8cc10d66bc6b448fef420fa0669068"},  
    "mac": "1b0a42d4bf7a8e96d308179e9714718e902727ead7041b97a646ef1c9d6f9ad7"},  
    "id": "7e0772e0-965e-4a05-ad93-fc5d11245ba3",  
    "version": 3  
  }
```

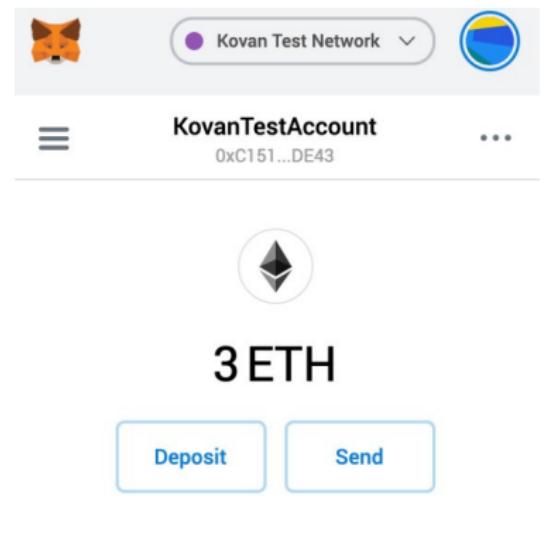
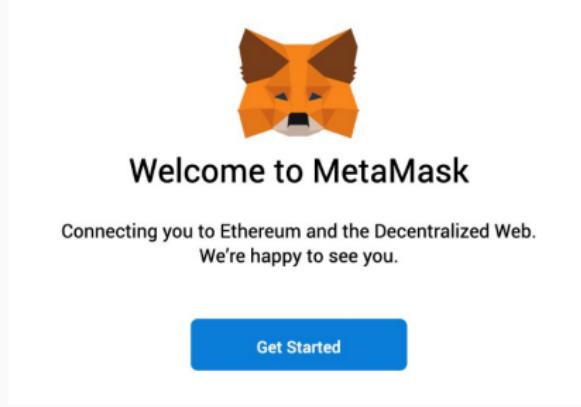
Key decryption process:

Ethereum keystore: Key decryption process iv



MetaMask i

MetaMask é uma carteira de criptomoedas e uma interface para redes *blockchain* sem exigir a instalação de um nó local.



Ver Capítulo 13.

A mineração é o processo pelo qual novos blocos são selecionados por meio de um mecanismo de consenso e adicionados ao *blockchain*. O processo segue os seguintes passos:

- * It listens for the transactions broadcasted on the Ethereum network and determines the transactions to be processed.
- * It determines stale or older blocks and includes them in the blockchain.
- * It updates the account balance with the reward earned from successfully mining the block.
- * Finally, a valid state is computed and the block is finalized, which defines the result of all state transitions.

Ethash é o nome do algoritmo de **Proof of Work** usado no Ethereum.

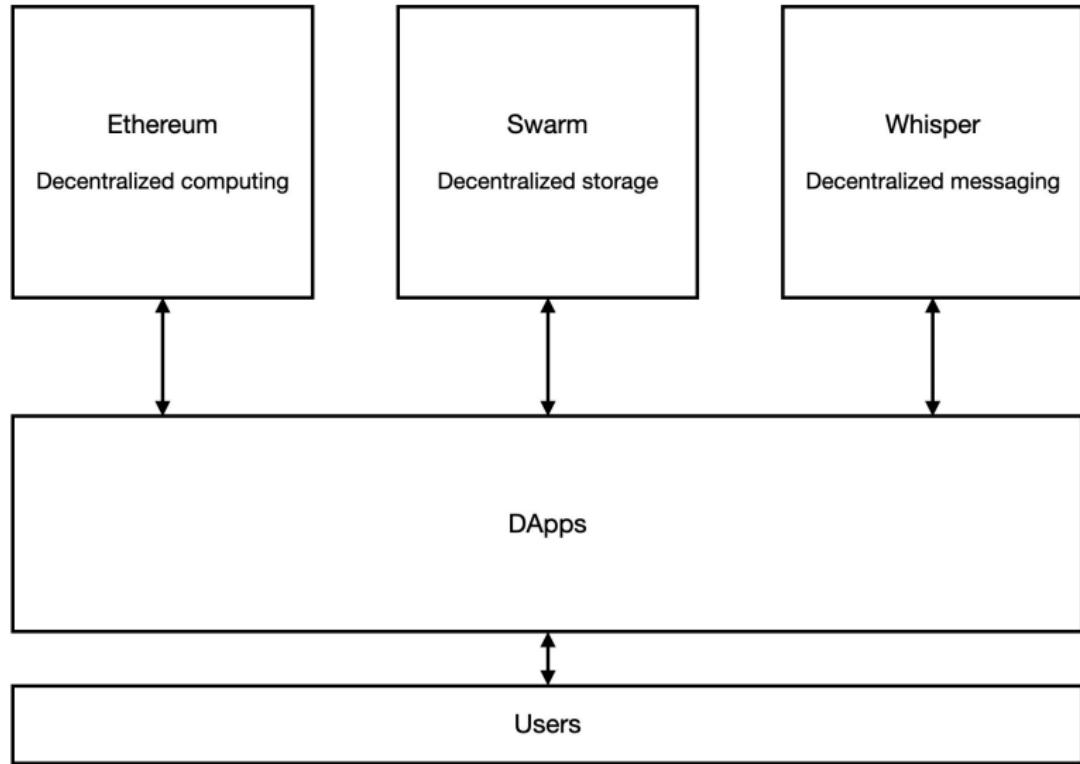
1. First, the header from the previous block and a 32-bit random nonce is combined using Keccak-256. 2. This produces a 128-bit structure called **mix**. 3. **mix** determines which data is to be picked up from the DAG. 4. Once the data is fetched from the DAG, it is “mixed” with the mix to produce the next mix, which is then again used to fetch data from the DAG and subsequently mixed. This process is repeated 64 times. 5. Eventually, the 64th mix is run through a digest function to produce a 32-byte sequence. 6. This sequence is compared with the difficulty target. If it is less than the difficulty target, the nonce is valid, and the PoW is solved. As a result, the block is mined. If not, then the algorithm repeats with a new nonce.

- **CPU:** Mining using a computer's built in CPU.
- **GPU:** Mining using graphical processing units, which provide better performance as compared to CPUs.
- **ASICs:** Specialized hardware—Application-Specific Integrated Circuit designed solely to run mining algorithms. These are currently the most efficient mining tools.
- Mining pools: In mining pools, resources are shared between miners and rewards are split accordingly.

Protocolos Suportados

Dois principais protocolos de suporte, **Swarm** e **Whisper**, são usados para fornecer armazenamento e mensagens descentralizadas, a fim de criar um ecossistema descentralizado completo.

Protocolos Suportados ii



- **Solidity** is one of the high-level languages that has been developed for Ethereum. It uses JavaScript-like syntax to write code for smart contracts. Once the code is written, it is compiled into bytecode that's understandable by the EVM using the Solidity compiler called solc.
- **LLL** is another language that is used to write smart contract code.
- **Serpent** is a Python-like, high-level language that can be used to write smart contracts for Ethereum.
- **Vyper** is a newer language that has been developed from scratch to achieve a secure, simple, and auditable language.

Atividade

- Leitura do Ethereum yellow paper:
<https://ethereum.github.io/yellowpaper/paper.pdf>
- Instalação do Geth e executar ele com a Kovan testnet.

Leitura Recomendada

Capítulo 11: Ethereum 101

Livro: [IMRAN BASHIR](#). Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Capítulo 12: Futher Ethereum

Livro: [IMRAN BASHIR](#). Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Prática sobre Ethereum

Prática sobre Ethereum i

Leitura do Capítulo 12: *Futher Ethereum*

Leitura do Capítulo 12: *Futher Ethereum*

1. Faça a leitura do Capítulo 12: *Futher Ethereum*
2. Instalar as ferramentas e testar os comandos apresentados no capítulo.

O cliente padrão Geth pode ser instalado em sistema derivados do Ubuntu:

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

Em outros Sistemas como o Manjaro:

```
[rag@nitro-ryzen ~]$ sudo pacman -Ss ethereum  
community/go-ethereum 1.10.25-1 [instalado]
```

Ambiente e Ferramentas de Desenvolvimento

Ambiente e Ferramentas de Desenvolvimento

- Apresentação do Ambiente de Desenvolvimento para *Ethereum*.

Ethereum – Redes de Teste i

Parâmetro	Descrição
--testnet	O livro indica que para as redes de teste deve ser usado o parâmetro --testnet para acessar a rede ropsten por padrão ou fornecer o nome da rede, como --testnet rinkeby . Na versão atual os parâmetros são os seguintes.
--ropsten	Ropsten network: pre-configured Proof of Work test network
--rinkeby	Rinkeby network: pre-configured Proof of Authority test network
--goerli	Görli network: pre-configured Proof of Authority test network
--kiln	Kiln network: pre-configured proof-of-work to proof-of-stake test network

--sepolia

Sepolia network: pre-configured proof-of-work
test network

Ethereum – Redes de Teste iii

- Testando a execução:

```
rag@blocker$ geth --ropsten --syncmode snap --http --http.addr 127.0.0.1 --http.port 8545  
INFO [10-25|16:24:31.929] Starting Geth on Ropsten testnet...
```

- Estava dando o seguinte *Warning*:

```
WARN [10-25|17:23:25.126] Post-merge network, but no beacon client seen.
```

Pesquisando na Internet: Post-merge network, but no beacon client seen.

Please launch one to follow the chain!

- Encontrei essa solução na internet:

<https://github.com/ethereum/go-ethereum/issues/25791>

- Indicando a documentação do Ethereum sobre *Consensus Clients*

- Mostra como `geth` deve ser iniciado, com conexão RPC autenticada usando um arquivo `jwtsecret`.
- Por padrão esse arquivo está em `~/.ethereum/geth/jwtsecret`.

```
geth --ropsten --syncmode snap --http --http.addr 127.0.0.1 -  
INFO [10-25|16:24:31.929] Starting Geth on Ropsten testnet...
```

- Note que estou executando na rede de testes **Ropsten**, opção na minha versão do `geth` é diferente do livro. No livro ele diz para usar o parâmetro `--testnet` que por padrão usa a rede de testes **Ropsten**, na minha instalação tem o parâmetro `--ropsten`, como tem o `--mainnet` e outras redes de testes.

Clientes de Consenso

Existem atualmente cinco clientes de consenso que podem ser executado em conjunto com o **Geth**:

[Lighthouse](#): escrito em **Rust**

[Nimbus](#): escrito em **Nim**

[Prysm](#): escrito em **Go**

[Teku](#): escrito em **Java**

[Lodestar](#): escrito em **TypeScript**

- Testei o **Prysm** por ser escrito em **Go**, assim como o **Geth**.
- **Prysm** é uma implementação da especificação do consenso **proof-of-stake** do **Ethereum**.
- Este link apresenta como configurar o **Prism**:

Step 2: Instalando o Prysm#

- Crie no diretório `~/.ethereum`, duas subpastas: `consensus` e `execution`:
- Acesse o diretório `consensus` e execute o comando para baixar o cliente `Prysm` e transformá-lo em executável:

```
$ mkdir prysm && cd prysm
```

```
$ curl https://raw.githubusercontent.com/prysmaticlabs/prysm/
```

Gerando um arquivo **JWT Secret**

- A conexão HTTP entre seu nó beacon e seu nó de execução precisa ser autenticada usando um *token* JWT. Existem diversas formas de gerar este *token*:
 - Usando um gerado *on line* como [este](#). Copie e cole o valor gerado dentro do arquivo `jwt.hex`.
 - Usando OpenSSL para criar o *token* via comando: `openssl rand -hex 32 | tr -d "\n" > "jwt.hex"`.
 - Usar o que foi gerado pelo cliente de execução geth:
`~/.ethereum/geth/jwtsecret`.
 - Usar o próprio Prysm para gerar o `jwt.hex`:

```
## Optional. This command is necessary only if you've previously run
```

```
## Required
```

```
./prysm.sh beacon-chain generate-auth-secret
```

Step 3: Executando um Cliente de Execução#

- Nesta etapa, você instalará um cliente de camada de execução (geth), se ainda não instalou, ao qual o nó beacon do Prysm se conectará.
- Baixe e execute a última versão 64-bit estável do **Geth installer** para seu Sistema Operacional do site [Geth downloads page](#).
- Note que **Geth 1.10.22** contém uma regressão. Atualize para [v1.10.23+](#) se você já não tiver uma mais nova.
- Tenho instalado a versão **1.10.25-stable**:

```
$ geth version
```

Geth

Version: 1.10.25-stable

Git Commit: 69568c554880b3567bace64f8848ff1be27d084d

Git Commit Date: 20220915

Step 4: Executando um nó beacon usando Prysm#

- Use o comando para iniciar um nó beacon que conecta no seu nó de execução local:

```
./prysm.sh beacon-chain --execution-endpoint=http://localhost
```

- Alterei o comando padrão para o conter o *hash* de uma das minhas contas:

```
./prysm.sh beacon-chain --execution-endpoint=http://localhost
```

- As contas podem ser listadas com o comando:

```
$ geth account list
```

- If you're running a validator, specifying a **suggested-fee-recipient** wallet address will allow you to earn what were previously miner transaction fee tips. See [How to configure](#)

Step 5: Executando um validator usando Prysm#

- Next, we'll create your validator keys with the [Ethereum Staking Deposit CLI](#).
- Download the latest stable version of the deposit CLI from the [Staking Deposit CLI Releases](#) page.
- Run the following command to create your mnemonic phrase and keys:

```
./deposit new-mnemonic --num_validators=1 --mnemonic_language=en
```

- Follow the CLI prompts to generate your keys. This will give you the following artifacts:
 1. A **new mnemonic seed phrase**. This is **highly sensitive** and should never be exposed to other people or networked hardware.

Congratulations!

- You're now running a **full Ethereum node** and a **validator**.
- It can take a long time (from days to months) for your validator to become fully activated. To learn more about the validator activation process, see [Deposit Process](#). See [Check node and validator status](#) for detailed status monitoring guidance.
- You can leave your **execution client**, **beacon node**, and **validator client** terminal windows open and running. Once your validator is activated, it will automatically begin proposing and validating blocks.

Atividade

- Leitura do Capítulo 13.

Leitura Recomendada

Capítulo 11: Ethereum 101

Livro: [IMRAN BASHIR](#). Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Capítulo 12: Futher Ethereum

Livro: [IMRAN BASHIR](#). Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Prática sobre **Ethereum**:

- Leitura do Capítulo 12: *Futher Ethereum*
 1. Faça a leitura do Capítulo 12: *Futher Ethereum*
 2. Criar a rede privada local:
 - 2.1 Criar um diretório `mkdir ~/.etherprivate`
 - 2.2 Criar um arquivo `privategenesis.json` em `~/.etherprivate` com o conteúdo:

```
{  
    "nonce": "0x00000000000000042",  
    "timestamp": "0x00",  
    "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",  
    "extraData": "0x00",  
    "gasLimit": 5242880,  
    "difficulty": 1, ...}
```


3. Execute o `geth` indicando o diretório de dados e o *genesis file*:

```
[rag@nitro-ryzen ~]$ geth --datadir ~/.etherprivate init ~/.genesis.json
INFO [10-27|19:59:19.049] Maximum peer count
INFO [10-27|19:59:19.051] Smartcard socket not found, disabling
INFO [10-27|19:59:19.053] Set global gas cap
INFO [10-27|19:59:19.054] Allocated cache and file handles
INFO [10-27|19:59:19.068] Opened ancient database
INFO [10-27|19:59:19.068] Writing custom genesis block
INFO [10-27|19:59:19.068] Persisted trie from memory database
INFO [10-27|19:59:19.069] Successfully wrote genesis state
INFO [10-27|19:59:19.069] Allocated cache and file handles
INFO [10-27|19:59:19.080] Opened ancient database
INFO [10-27|19:59:19.081] Writing custom genesis block
```

```
INFO [10-27|19:59:19.081] Persisted trie from memory database
INFO [10-27|19:59:19.082] Successfully wrote genesis state
[rag@nitro-ryzen ~]$
```

4. Inicie a execução:

```
$ geth --datadir ~/.etherprivate/ --allow-insecure-unlock --r
```

- Iniciar um console para a interação com a execução:

```
$ geth attach ~/.etherprivate/geth.ipc
```

- Criar duas contas, caso não tenha:

Ambiente de Desenvolvimento v

```
> personal.newAccount("admin1234")
"0xedbc36d74d5a1cd64db36e53798bd1781f0c4955"

> eth.accounts
[ "0xedbc36d74d5a1cd64db36e53798bd1781f0c4955" ]

> personal.newAccount("admin1234")
"0x1478d95f8754b3ba7127100dd0bb46578fe7d22a"

> eth.accounts
[ "0xedbc36d74d5a1cd64db36e53798bd1781f0c4955", "0x1478d95f8754b3ba7127100dd0bb46578fe7d22a" ]
```

- Desbloquear as contas:

```
> eth.accounts
[ "0xedbc36d74d5a1cd64db36e53798bd1781f0c4955", "0x1478d95f875
> personal.unlockAccount("0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
Passphrase:
true
> personal.unlockAccount("0x1478d95f8754b3ba7127100dd0bb46578
Unlock account 0x1478d95f8754b3ba7127100dd0bb46578fe7d22a
Passphrase:
true
```

- Verificação dos valores em cada carteira:

Ambiente de Desenvolvimento vii

```
> web3.fromWei(eth.getBalance("0xedbc36d74d5a1cd64db36e53798b  
320  
> web3.fromWei(eth.getBalance("0x1478d95f8754b3ba7127100dd0bb  
0  
> web3.fromWei(eth.getBalance(eth.coinbase), "ether")  
320
```

- Enviar 100 ethers da primeira para a segunda carteira:

```
> eth.sendTransaction({from: "0xedbc36d74d5a1cd64db36e53798b  
> eth.sendTransaction({from: "0xedbc36d74d5a1cd64db36e53798b
```

SyntaxError: SyntaxError: (anonymous): Line 1:73 Unexpected i

Ambiente de Desenvolvimento viii

- Estava ocorrendo esse erro: `SyntaxError: SyntaxError: (anonymous): Line 1:73 Unexpected identifier (and 6 more errors)` quando tentava enviar uma transação.

```
> personal.unlockAccount(personal.listAccounts[0])
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
Passphrase:
true
> personal.unlockAccount(personal.listAccounts[1])
Unlock account 0x1478d95f8754b3ba7127100dd0bb46578fe7d22a
Passphrase:
true
> eth.sendTransaction({from: personal.listAccounts[0], to: pe
"0x797c2a303974e365bf48ac1620da9d3a1e8ad0d53138c3ba06a4cddb1
```

Ambiente de Desenvolvimento ix

```
> eth.sendTransaction({from: "0xedbc36d74d5a1cd64db36e53798bc..."}  
SyntaxError: SyntaxError: (anonymous): Line 1:73 Unexpected identifier  
  
> personal.unlockAccount(personal.listAccounts[0])  
  
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955  
Passphrase:  
true  
> eth.sendTransaction({from: "0xedbc36d74d5a1cd64db36e53798bc..."}  
SyntaxError: SyntaxError: (anonymous): Line 1:73 Unexpected identifier  
  
> personal.unlockAccount(personal.listAccounts[1])  
Unlock account 0x1478d95f8754b3ba7127100dd0bb46578fe7d22a  
Passphrase:
```

```
true
```

```
> eth.sendTransaction({from: "0xedbc36d74d5a1cd64db36e5379bc...")
```

```
SyntaxError: SyntaxError: (anonymous): Line 1:73 Unexpected i
```

```
> eth.sendTransaction({from: personal.listAccounts[0], to: pe...})
```

```
"0x5c599cc300072c38544fa2a8869cf9928b17345b2d75ab43e6a3f23d4b...")
```

```
>
```

- Usando

```
personal.unlockAccount(personal.listAccounts[1]) para  
desbloquear funcionou o envio de transações.
```

- Verificando os valores nas carteiras:

Ambiente de Desenvolvimento xi

```
> web3.fromWei(eth.getBalance("0xedbc36d74d5a1cd64db36e53798b  
9709.999999999999998  
> web3.fromWei(eth.getBalance("0x1478d95f8754b3ba7127100dd0bb  
2e-16
```

- Recuperar o recibo da transação:

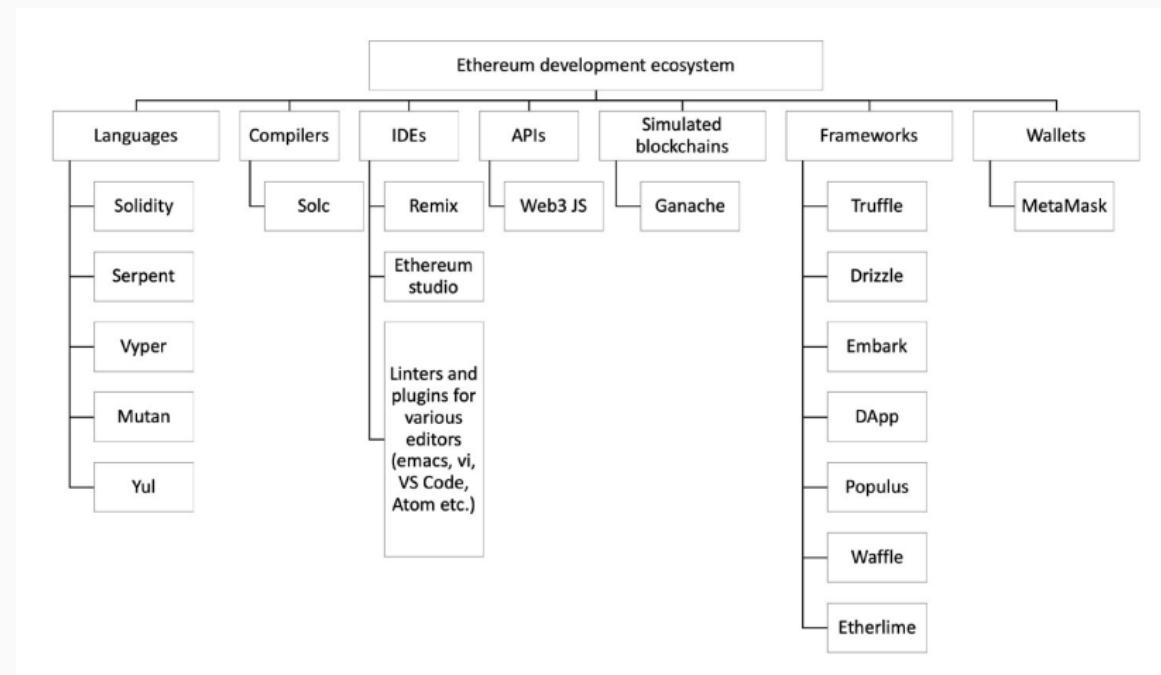
```
eth.getTransactionReceipt("0x797c2a303974e365bf48ac1620da9d3a  
{  
  blockHash: "0x91b7b138aa7ef8aa992e306925e2184cc463fcba65fa9  
  blockNumber: 1834,  
  contractAddress: null,  
  cumulativeGasUsed: 21000,  
  effectiveGasPrice: 1000000000,  
  from: "0xedbc36d74d5a1cd64db36e53798bd1781f0c4955",
```


Ferramentas de Desenvolvimento e Frameworks

Objetivos

- Apresentação de Ferramentas de Desenvolvimento e *Frameworks*.
- Linguagens, Compiladores, Ferramentas e Bibliotecas, *Frameworks*, Desenvolvimento e implantação de contratos e Linguagem Solidity.

Taxonomia do Ecossistema de Componentes de Desenvolvimento Ethereum



- **Solidity:** Tem se tornado a linguagem padrão para escrever contratos para *Ethereum*. O código precisa ser compilado e transformado em *bytecode*, é necessário utilizar o compilador **solc**.
- **Vyper:** Essa linguagem é uma linguagem experimental semelhante ao Python que está sendo desenvolvida para trazer segurança, simplicidade e auditabilidade ao desenvolvimento de contratos inteligentes.
- **Yul:** Esta é uma linguagem intermediária que tem a capacidade de compilar para diferentes back-ends, como EVM e eWasm. Os objetivos de projeto do Yul incluem principalmente legibilidade, fluxo de controle fácil, otimização, verificação formal e simplicidade.
- **Mutan:** Esta é uma linguagem de estilo Go, que foi descontinuada no início de 2015 e não é mais usada.

- **LLL:** Linguagem semelhante ao *Low-Level Lisp-Like*, daí o nome **LLL**, também não é mais usada.
- **Serpent:** Esta é uma linguagem simples e limpa parecida com Python. Ela não é mais usado para desenvolvimento de contratos e não é suportado pela comunidade.
- Leia mais sobre Solidity e Recursos de Desenvolvimento de DApps em **DAPP DEVELOPMENT FRAMEWORKS²**

²<http://ethdocs.org/en/latest/contracts-and-transactions/developer-tools.html#developer-tools>

- O compilador **Solidity** (solc)
- Compilador usado para compilar código de contratos inteligentes e converter eles para *bytecode*.

- **Ganache**
 - Simula um Blockchain Ethereum pessoal com uma interface com usuário (UI), comumente usada no desenvolvimento e testes.
- **Ganache-cli**
 - Versão linha de comando do **Ganache** tem como pre-requisito **NodeJS**.

Ferramentas e Bibliotecas ii

The screenshot shows a blockchain explorer interface with the following details:

ACCOUNTS | **BLOCKS** | **TRANSACTIONS** | **CONTRACTS** | **EVENTS** | **LOGS** | SEARCH FOR BLOCK NUMBERS OR TX HASHES

CURRENT BLOCK: 0 | GAS PRICE: 20000000000 | GAS LIMIT: 6721975 | HARDFORK: PETERSBURG | NETWORK ID: 5777 | RPC SERVER: HTTP://127.0.0.1:7545 | MINING STATUS: AUTOMINING | WORKSPACE: JAZZY-GIRL

MNEMONIC: kick abstract strong shrug forward enlist puppy reunion elephant hip suffer base

HD PATH: m/44'/60'/0'/*/account_index

ADDRESS	BALANCE	TX COUNT	INDEX	EDIT
0x2366e9848803cB00CB82E6E6De3F6D17C4AA9ADA	100.00 ETH	0	0	🔗
0x6805940005154aEdfe6a00A39C588ED668E64D9D	100.00 ETH	0	1	🔗
0x56C21294F4e17dF32486b3d4D12E72D023861edF	100.00 ETH	0	2	🔗
0xAfACDB553412071bB538E36d57ED92d6745C5f94	100.00 ETH	0	3	🔗
0x694AE93a42C43B8E3d10c3F6769Adf2566C1863B	100.00 ETH	0	4	🔗

The preceding screenshot displays a Visual Studio Code window that comprises a file explorer on the left-hand side and a code editor window. Due to syntax highlighting and IntelliSense being enabled by the Solidity plugin, it becomes easy to write smart contract code. The Solidity plugin for Visual Studio is available in the Visual Studio marketplace at <https://marketplace.visualstudio.com/items?itemName=JuanBlanco.solidity>.

- Truffle
 - Framework de desenvolvimento para *Ethereum* com recursos para implantação, teste e depuração.

Frameworks ii

Truffle v5.1.11 – a development framework for Ethereum

Usage: truffle <command> [options]

Commands:

build	Execute build pipeline (if configuration present)
compile	Compile contract source files
config	Set user-level configuration options
console	Run a console with contract abstractions and commands available
create	Helper to create new contracts, migrations and tests
debug	Interactively debug any transaction on the blockchain (experimental)
deploy	(alias for migrate)
develop	Open a console with a local development blockchain
exec	Execute a JS module within this Truffle environment
help	List all commands or provide information about a specific command
init	Initialize new and empty Ethereum project
install	Install a package from the Ethereum Package Registry
migrate	Run migrations to deploy contracts
networks	Show addresses for deployed contracts on each network
obtain	Fetch and cache a specified compiler
opcode	Print the compiled opcodes for a given contract
publish	Publish a package to the Ethereum Package Registry
run	Run a third-party command
test	Run JavaScript and Solidity tests
unbox	Download a Truffle Box, a pre-built Truffle project
version	Show version number and exit
watch	Watch filesystem for changes and rebuild the project automatically

See more at <http://truffleframework.com/docs>

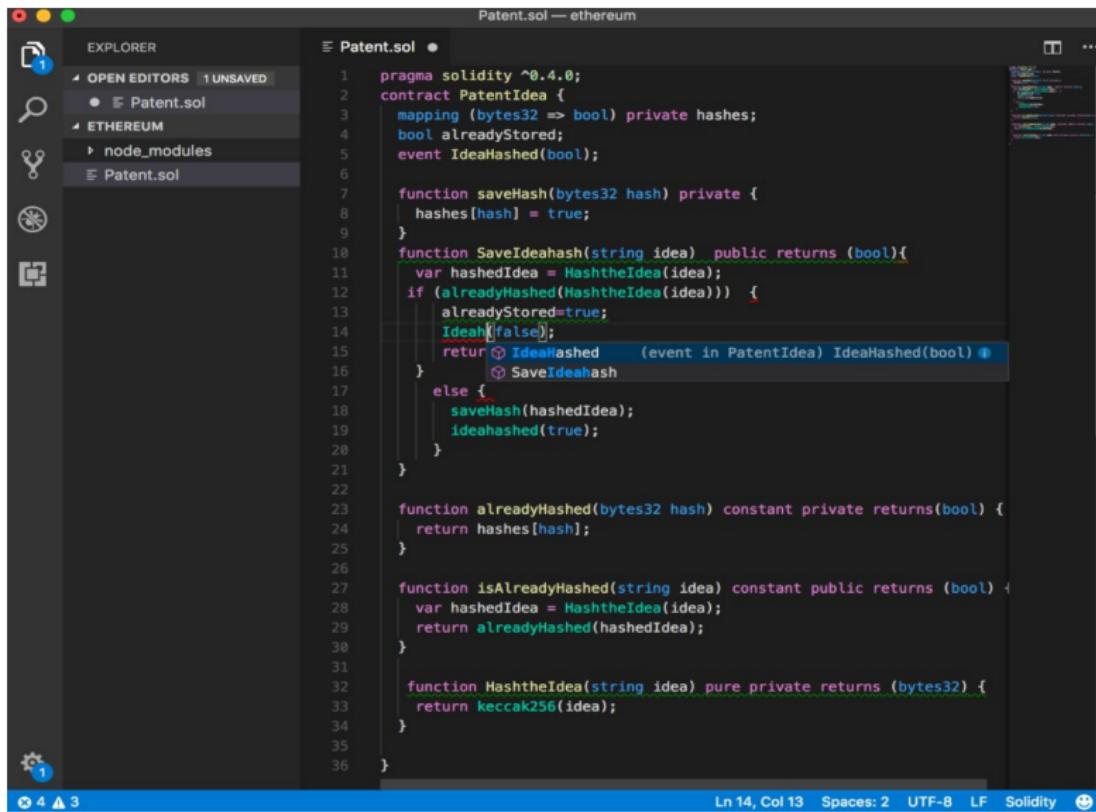
- Drizzle
 - Um conjunto de bibliotecas de *frontend* para o desenvolvimento de interfaces *web*.
 - Torna o desenvolvimento *frontend* para DApps fácil.
 - Tem o NodeJS como pré-requisito.
 - Baseado no *Redux store*.
 - Mantém uma biblioteca de componentes **React**.

- **Embark:** powerful developer platform for building and deploying DApps
- **Brownie:** framework for Ethereum smart contract development and testing
- **Waffle:** another framework for smart contract development and testing
- **Etherlime:** framework that allows DApp development, debugging, testing, and testing in Solidity and Vyper
- **OpenZeppelin:** a toolkit for smart contract development

Desenvolvimento e Implantação

- A escrita de contratos inteligentes é basicamente a escrita de código fonte do contrato em **Solidity** em um editor de texto.
- Existem vários *plugins* e extensões disponíveis para os editores mais comuns, tais como Vim, Atom, VSCode, que fornecem *syntax highlighting* e formatadores para código fonte **Solidity**.

Desenvolvimento e Implantação ii



The screenshot shows a code editor interface for a Solidity smart contract named `Patent.sol`. The code defines a `PatentIdea` contract with methods for saving ideas, checking if an idea is already hashed, and hashing ideas.

```
Patent.sol — ethereum
EXPLORER
OPEN EDITORS 1 UNSAVED
  • Patent.sol
ETHEREUM
  node_modules
    Patent.sol
1 pragma solidity ^0.4.0;
2 contract PatentIdea {
3   mapping (bytes32 => bool) private hashes;
4   bool alreadyStored;
5   event IdeaHashed(bool);
6
7   function saveHash(bytes32 hash) private {
8     hashes[hash] = true;
9   }
10  function SaveIdeaHash(string idea) public returns (bool){
11    var hashedIdea = HashtheIdea(idea);
12    if (alreadyHashed(hashedIdea)) {
13      alreadyStored=true;
14      IdeaHashed(false);
15      return IdeaHashed (event in PatentIdea) IdeaHashed(bool) 1
16    }
17    else {
18      saveHash(hashedIdea);
19      ideahashed(true);
20    }
21  }
22
23  function alreadyHashed(bytes32 hash) constant private returns(bool) {
24    return hashes[hash];
25  }
26
27  function isAlreadyHashed(string idea) constant public returns (bool) {
28    var hashedIdea = HashtheIdea(idea);
29    return alreadyHashed(hashedIdea);
30  }
31
32  function HashtheIdea(string idea) pure private returns (bytes32) {
33    return keccak256(idea);
34  }
35
36 }
```

The code editor has a dark theme and includes a sidebar with icons for file operations and a bottom status bar showing line and column numbers, character count, and file encoding.

The layout of a Solidity source code file i

```
1 pragma solidity ^0.5.0; //specify the solidity compiler version
2 /*
3 this is a simple value checker contract that checks the value provided
4 and returns boolean value (true or false) based on the condition expression
5 evaluation
6 */
7 import "./mapping.sol"; //import a file
8 contract valuechecker {
9     uint price = 10;
10    //price variable declared and initialized with a value of 10
11    event valueEvent(bool returnValue);
12    function Matcher (uint8 x) public returns (bool) {
13        if (x >= price )
14        {
15            emit valueEvent(true);
16            return true;
17        }
18    }
19 }
```

Linguagem Solidity i

- Uma Linguagem de Domínio Específico (DSL)
- *Contract-oriented language*
- JavaScript / C-like
- Amplamente utilizada
- Estaticamente Tipada

Linguagem Solidity i

Leitura Recomendada

Capítulo 14: Development Tools and Frameworks

Livro: IMRAN BASHIR. Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Solidity

Leitura do ***Capítulo 14:***
Development Tools and
Frameworks

1. Faça a leitura do Capítulo 14: Development Tools and Frameworks

Instalação das Ferramentas

Instalação das Ferramentas

2. Instale o Compilador **Solidity** (**solc**). O **solc** converte código de alto nível escrito na linguagem **Solidity** para *bytecode* da *Ethereum Virtual Machine (EVM)*.

Para distribuições **Ubuntu** ou derivados do **Debian**:

```
$ sudo apt-get install solc
```

Outras distribuições como o **Manjaro Linux**, instale o pacote **solidity**:

```
$ sudo pacman -S solidity
```

Feita a instalação, para verificar a versão instalada execute o comando:

```
$ solc --version
solc, the solidity compiler commandline interface
Version: 0.8.17+commit.015f556f+gmp
```

Redes de Testes

Clientes

Introdução à Web3

subtitle: “Introdução ao Web3” date: abstract: “Nesta aula são apresentadas algumas ferramentas de Desenvolvimento e Frameworks para o desenvolvimento e implantação de Contratos Inteligentes. Apresenta uma introdução ao **Web3**, métodos de desenvolvimento, teste e verificação de contratos inteligentes com **Ganache**, console do cliente **Geth** e Remix IDE. Introduz o **Truffle framework**, que também pode ser usado para testar, migrar contratos inteligentes e o Drizzle, para criar *frontends* de **DApps** de maneira mais fácil, com IPFS, para hospedar as páginas web da aplicação.” nocite: | ([Imran 2018](#)) —

Introdução

Objetivos

- Apresentação de Ferramentas de Desenvolvimento e *Frameworks*.
- Explorar a biblioteca **Web3** com o cliente **Geth**, desenvolvimento de contratos, interação com contratos via *frontends*.

- Web3 é uma biblioteca *JavaScript* que pode ser usada na comunicação com um Nó *Ethereum* via comunicação RPC. Web3 expõe métodos que o acesso está disponível sobre RPC.
- A interação com o cliente *Geth* é possível via *Geth JavaScript Console*, que expõe vários métodos de consulta e gerenciamento do blockchain.
- Vimos os comandos para execução do *Geth* e do *Console JavaScript* na **Aula 021 - Prática sobre Ethereum: Ambiente de Desenvolvimento**.

Explorando Web3 com Geth ii

- Iniciar o Nó Geth com suporte ao web3:

```
$ geth --datadir ~/.etherprivate/ --allow-insecure-unlock --networkid 786 --http --http.addr 127.0.0.1 --http.p
```

Explorando Web3 com Geth iii

- Iniciar um console para a interação com a execução:

```
$ geth attach ~/.etherprivate/geth.ipc
Welcome to the Geth JavaScript console!

instance: Geth/v1.10.26-stable-e5eb32ac/linux-amd64/go1.19.3
coinbase: 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
at block: 0 (Wed Dec 31 1969 21:00:00 GMT-0300 (-03))
datadir: /home/rag/.etherprivate
modules: admin:1.0 debug:1.0 engine:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 w

To exit, press ctrl-d or type exit
```

- Verificando se os recursos web3 estão disponíveis:

```
> web3.version
{
  api: "0.20.1",
  ethereum: undefined,
  network: "786",
```

Explorando Web3 com Geth iv

```
node: "Geth/v1.10.26-stable-e5eb32ac/linux-amd64/go1.19.3",
whisper: undefined,
getEthereum: function(callback),
getNetwork: function(callback),
getNode: function(callback),
getWhisper: function(callback)

}

>
```

- Faremos um *deploy* usando o *Geth console*.
- O passo a passo pode ser visto no livro e iremos reproduzir aqui, seguindo a sequência de passos:
 - Executar o *Geth client*.
 - Criar um script de *deployment*, usando a **ABI** e o *bytecode*, e algum código **JavaScript**.
 - Faremos o *deploy* do contrato via linha de comando pelo *Geth console*.
 - Interagir com o contrato via um *frontend web*.

Web3 deployment: Executar o *Geth client* i

- Executar o *Geth client*. [✓]
- Executar o *Geth console*. [✓]

Web3 deployment: Criar um script de **deployment** i

- Compile o contrato com o `solc` ou utilizando o Remix IDE, gerando o binário e a ABI:

```
$ solc --bin --abi -o bin ValueChecker.sol
$ ls
bin deploy.js ValueChecker.sol
$ cd bin
$ ls
valueChecker.abi valueChecker.bin
$ cat valueChecker.bin
6080604052600a60005534801561001557600080fd5b5061018b806100256
1561001057600080fd5b506004361061002b5760003560e01c8063f9d55e2
4a600480360381019061004591906100f2565b610060565b6040516100579
f35b600080548260ff16106100ae577f3eb1a229ff7995457774a4bd31ef7
```

Web3 deployment: Criar um script de **deployment** ii

```
b4b6239c600160405161009d919061013a565b60405180910390a16001905  
80fd5b600060ff82169050919050565b6100cf816100b9565b81146100da5  
506100ec816100c6565b92915050565b60006020828403121561010857610  
8285016100dd565b91505092915050565b60008115159050919050565b610  
600060208201905061014f600083018461012b565b9291505056fea264697  
857c0d0a6d073976f05d5073826c629671c857a375db35d51c64736f6c634
```

```
$ cat valueChecker.abi
```

```
[{"anonymous":false,"inputs":[{"indexed":false,"internalType":"  
"type":"bool"}],"name":"valueEvent","type":"event"}, {"inputs":  
"name":"x","type":"uint8"}],"name":"Matcher","outputs":[{"int  
"name":"","type":"bool"}],"stateMutability":"nonpayable","typ
```

- Preparação do código *JavaScript*:

Web3 deployment: Criar um script de **deployment** iii

```
var valuecheckerContract = web3.eth.contract([{"anonymous":  
  false},  
  {  
    "inputs": [  
      {"name": "value", "type": "uint256"}],  
    "name": "ValueCheckerEvent",  
    "outputs": [  
      {"name": "value", "type": "uint256"}],  
    "signature": "0x6080604052600a60005534801561001557600080fd5b5061  
    60405260043610603f576000357c01000000000000000000000000000000  
    000000900463fffffffff168063f9d55e21146044575b600080f  
    6f600480360381019080803560ff16906020019092919050505  
    5260200191505060405180910390f35b600080548260ff16101  
    74a4bd31ef7b13b6f4491ad1ebb8961af120b8b4b6239c60016  
    505060405180910390a16001905060dc565b5b9190505600a16  
    6f5650d800506c4eb6be2d8d71c0e2c8b0ca50660fde82c7680  
  },  
  {  
    "name": "ValueChecker",  
    "inputs": [  
      {"name": "value", "type": "uint256"}],  
    "name": "checkValue",  
    "outputs": [  
      {"name": "value", "type": "uint256"}],  
    "type": "function"  
  }],  
  {});  
  
var valuechecker = valuecheckerContract.new({  
  from: web3.eth.accounts[0],  
  data: '0x6080604052600a60005534801561001557600080fd5b5061  
    60405260043610603f576000357c01000000000000000000000000000000  
    000000900463fffffffff168063f9d55e21146044575b600080f  
    6f600480360381019080803560ff16906020019092919050505  
    5260200191505060405180910390f35b600080548260ff16101  
    74a4bd31ef7b13b6f4491ad1ebb8961af120b8b4b6239c60016  
    505060405180910390a16001905060dc565b5b9190505600a16  
    6f5650d800506c4eb6be2d8d71c0e2c8b0ca50660fde82c7680  
  },  
  function (e, contract) {  
    console.log(e, contract);  
  }  
});
```

Web3 deployment: Criar um script de **deployment** iv

```
if (typeof contract.address !== 'undefined') {  
    console.log('Contract mined! address: ' + contrac  
}  
})
```

- No Geth console dê um *unlock* na conta:

```
> personal.listAccounts[0]
"0xedbc36d74d5a1cd64db36e53798bd1781f0c4955"
> personal.unlockAccount(personal.listAccounts[0])
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955
Passphrase:
true
>
```

- Cole o código **JavaScript** para fazer o *deploy*:

Web3 deployment: Fazendo o **deploy** pelo **Geth console** ii

```
> var valuecheckerContract = web3.eth.contract([{"anonymous": false, "inputs": [{"name": "value", "type": "uint256"}], "name": "ValueChecker", "outputs": [{"name": "Value", "type": "uint256"}], "events": {}}, {"name": "ValueChecker", "inputs": [{"name": "value", "type": "uint256"}], "name": "Value", "outputs": [{"name": "Value", "type": "uint256"}]}, {"name": "Value", "inputs": [{"name": "value", "type": "uint256"}], "name": "checkValue", "outputs": [{"name": "Value", "type": "uint256"}]}], [{"constant": true, "inputs": [{"name": "value", "type": "uint256"}], "name": "value", "outputs": [{"name": "Value", "type": "uint256"}]}], [{"constant": false, "inputs": [{"name": "value", "type": "uint256"}], "name": "Value", "outputs": [{"name": "Value", "type": "uint256"}]}], [{"constant": false, "inputs": [{"name": "value", "type": "uint256"}], "name": "checkValue", "outputs": [{"name": "Value", "type": "uint256"}]}]).compile();
undefined
> var valuechecker = valuecheckerContract.new({
.....from: web3.eth.accounts[0],
.....data: '0x6080604052600a60005534801561001557600080fd5b50
5260043610603f576000357c0100000000000000000000000000000000000000
ffffffffff168063f9d55e21146044575b600080fd5b348015604f57600080fc
3560ff1690602001909291905050506089565b60405180821515151581526
b600080548260ff1610151560db577f3eb1a229ff7995457774a4bd31ef7b
b6239c6001604051808215151515815260200191505060405180910390a16
165627a7a723058209ff756514f1ef46f5650d800506c4eb6be2d8d71c0e2
gas: '4700000'
.....},
...     function (e, contract) {
... }
```

```
.....      console.log(e, contract);
.....      if (typeof contract.address !== 'undefined') {
.....          console.log('Contract mined! address: ' +
.....          }
.....      })
Error: insufficient funds for gas * price + value undefined
undefined
> miner.start()
null
>
```

- Na execução do *deploy* deu uma mensagem de erro **Error: insufficient funds for gas * price + value undefined**, pois a carteira da conta selecionada não tem saldo suficiente. É necessário minerar para ganhar algum saldo:

```
> miner.start()  
null  
> miner.stop()  
> null
```

- Repetindo o processo de *deploy*:

```
> personal.unlockAccount(personal.listAccounts[0])  
Unlock account 0xedbc36d74d5a1cd64db36e53798bd1781f0c4955  
Passphrase:  
true  
> var valuecheckerContract = web3.eth.contract([{"anonymous":  
undefined  
> var valuechecker = valuecheckerContract.new({  
.....      from: web3.eth.accounts[0],  
.....data: '0x6080604052600a60005534801561001557600080fd5b50  
5260043610603f576000357c0100000000000000000000000000000000000000000000  
fffffff168063f9d55e21146044575b600080fd5b348015604f57600080fc  
3560ff1690602001909291905050506089565b60405180821515151581526  
b600080548260ff1610151560db577f3eb1a229ff7995457774a4bd31ef7b  
b6239c60016040518082151515815260200191505060405180910390a16
```

```
165627a7a723058209ff756514f1ef46f5650d800506c4eb6be2d8d71c0e2
gas: '4700000'
.... },
...     function (e, contract) {
....         console.log(e, contract);
....         if (typeof contract.address !== 'undefined')
....             console.log('Contract mined! address: '
....             })
null [object Object]
undefined
```

- Nos logs do Nó Geth irá aparecer a mensagem de que o contrato foi submetido:

```
INFO [11-24|12:44:17.115] Submitted contract creation
```

- Iniciando a mineração o contrato será minerado:

```
> miner.start()
```

```
null
```

```
> null [object Object]
```

```
Contract mined! address: 0xfb...e486a
```

```
> miner.stop()
```

Web3 deployment: Interagindo com o contrato i

- Interagir com o contrato via *Geth console*
 - Após o *deployment* através da sua **ABI** o contrato estará disponível no *console*:

```
> valuechecker.
```

```
valuechecker.Matcher
```

```
valuechecker.address
```

```
val
```

```
valuechecker._eth
```

```
valuechecker.allEvents
```

```
val
```

```
valuechecker.abi
```

```
valuechecker.constructor
```

```
> valuechecker.address
```

```
"0fbe4899126470af8dd4d37e878f0de486a6cfa71"
```

```
> valuechecker.transactionHash
```

```
"0x975501f4b6c24a46d13ead5840f40bc03460c6be4139cbd6c3d902e737"
```

Web3 deployment: Interagindo com o contrato ii

Percebam o mesmo *address* e *transactionHash* que foram devolvidos no processo de *deploy*.

- A ABI do `valuechecker` está disponível:

```
> valuechecker.abi
[{
    anonymous: false,
    inputs: [{
        indexed: false,
        internalType: "bool",
        name: "returnValue",
        type: "bool"
    }],
    name: "valueEvent",
```

```
type: "event"
}, {
  inputs: [
    internalType: "uint8",
    name: "x",
    type: "uint8"
  ],
  name: "Matcher",
  outputs: [
    internalType: "bool",
    name: "",
    type: "bool"
  ],
  stateMutability: "nonpayable",
```

```
    type: "function"  
}]
```

>

- A função `Matcher` pode ser invocada para a verificação de valores:

```
> eth.getBalance(valuechecker.address)
```

0

```
> valuechecker.Matcher.call(12)
```

true

```
> valuechecker.Matcher.call(10)
```

true

```
> valuechecker.Matcher.call(5)
```

false

>

Web3 deployment: Interagindo com o contrato via frontend web

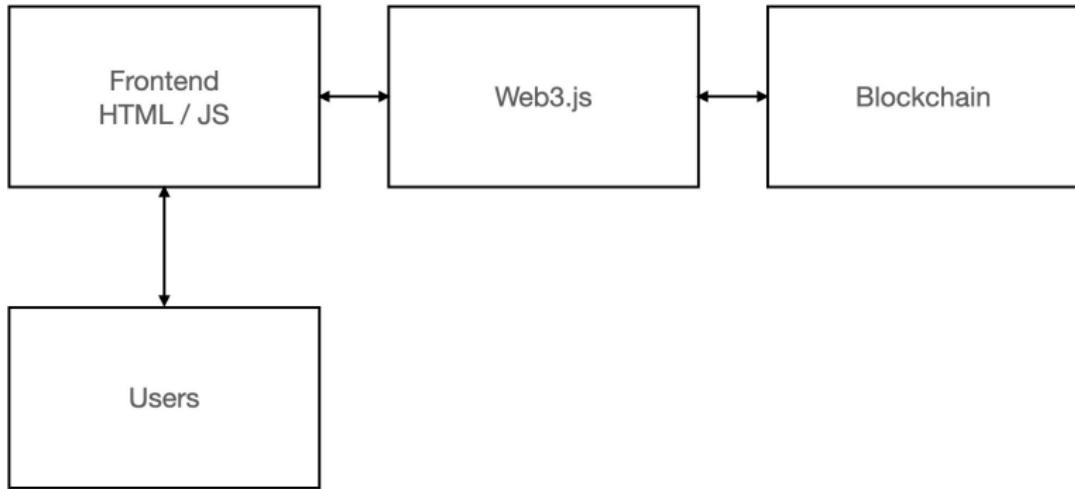
- Interagir com o contrato via um *frontend web*.
- *POST requests* É possível interagir com o Geth via JSON RPC sobre o HTTP. Para esse teste utilizaremos o [curl](#). Lembrando que a porta utilizada foi a 8559.
- **Recuperando a lista de contas:** A lista de contas pode ser obtida utilizando o método `personal_listAccounts`, conforme o comando:

```
$ curl --request POST --data '{"jsonrpc":"2.0","method":"personal_listAccounts","params":{},"id":4}' http://127.0.0.1:8559
```

Um objeto **JSON** é retornado com a lista de contas. No comando **curl**, o parâmetro **--request** é usado para especificar que o comando é uma requisição do tipo **POST** e **--data** é usado para especificar os parâmetros e valores. Finalmente, o **localhost:8559** é usando para indicar o endereço que o **HTTP endpoint** do **Geth** está aberto.

- A interação com *smart contracts* como parte de uma DApps é normalmente feito usando uma interface web desenvolvida utilizando **HTML/JS/CSS**. Algumas bibliotecas e *frameworks* como **React**, **Redux**, e **Drizzle**, podem também ser usadas.

Interagindo com contratos via frontends web ii



Biblioteca Javascript Web3.js i

- Se ainda não instalou a biblioteca `web3.js`, pode instalá-la via `npm` com o comando:

```
$ npm install web3
```

A biblioteca `Web.js` disponibiliza alguns módulos, sendo eles:

- `web3-eth`: Ethereum blockchain e smart contracts.
- `web3-shh`: Protocolo `Whisper` (Comunicação e *broadcast* P2P).
- `web3-bzz`: Protocolo `Swarm`, que fornece armazenamento descentralizado.
- `web3-utils`: Fornece funções úteis para o desenvolvimento de DApps.
- Criando um servidor http para testar a app.

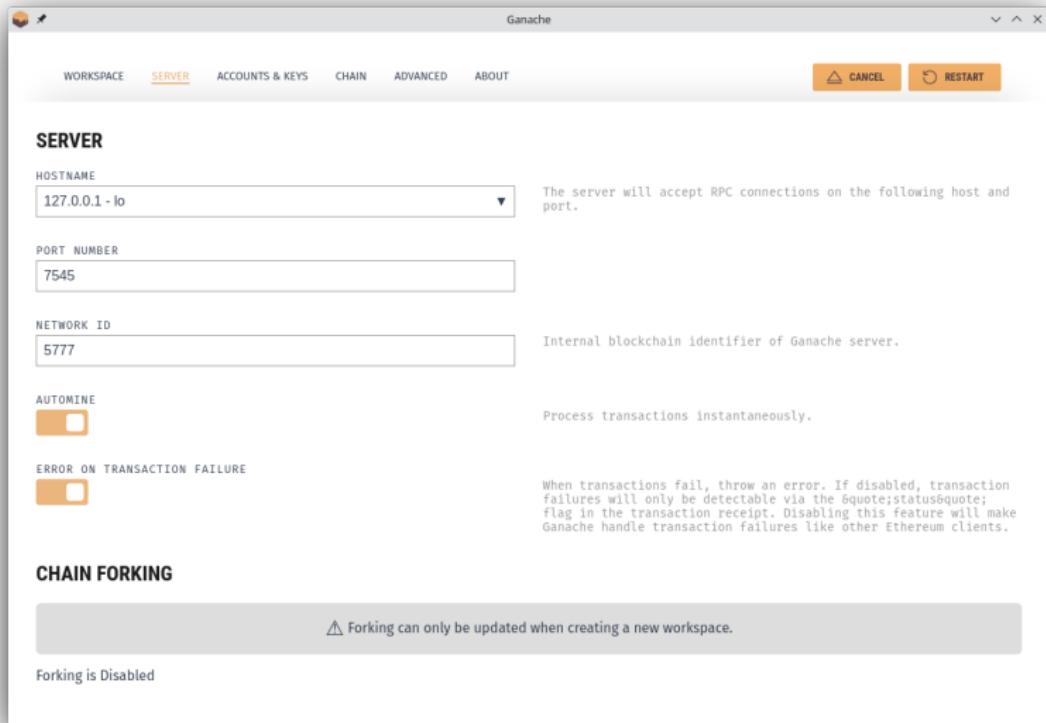
```
# Python 3.x
python3 -m http.server 7777
# If Python version returned above is 2.X
python -m SimpleHTTPServer 7777
```

- Installing and initializing Truffle
 - Truffle initialization is performed using the Truffle init command, which generates a skeleton structure for a project
- Compiling, testing, and migrating using Truffle
 - Several commands available in Truffle can be used to compile, test and deploy smart contracts

Configuração do Ganache i

- We can use Ganache as a local blockchain to provide the RPC interface.

Configuração do Ganache ii



Interagindo com um contrato i

- O console do **Truffle** expõe vários métodos que podem ser usados para interagir com contratos.

```
|truffle(development)> MetaCoin.  
MetaCoin.__defineGetter__  MetaCoin.__defineSetter__  
MetaCoin.__proto__         MetaCoin.hasOwnProperty  
MetaCoin.toLocaleString   MetaCoin.valueOf  
  
MetaCoin.apply             MetaCoin.bind  
MetaCoin.toString          MetaCoin.call  
  
MetaCoin._constructorMethods  MetaCoin._json  
MetaCoin.abi                MetaCoin.addProp  
MetaCoin.ast                 MetaCoin.at  
MetaCoin.bytecode            MetaCoin.caller  
MetaCoin.compiler            MetaCoin.configureNetwork  
MetaCoin.currentProvider     MetaCoin.decodeLogs  
MetaCoin.deployedBinary      MetaCoin.deployedBytecode  
MetaCoin.devdoc              MetaCoin.ens  
MetaCoin.hasNetwork          MetaCoin.interfaceAdapter  
MetaCoin.length              MetaCoin.link  
MetaCoin.name                MetaCoin.network  
MetaCoin.networks             MetaCoin.new  
MetaCoin.resetAddress        MetaCoin.schemaVersion  
MetaCoin.setNetworkType      MetaCoin.setProvider  
MetaCoin.sourceMap            MetaCoin.sourcePath  
MetaCoin.transactionHash     MetaCoin.unlinked_binary  
MetaCoin.userdoc              MetaCoin.web3  
  
MetaCoin.__lookupGetter__    MetaCoin.__lookupSetter__  
MetaCoin.isPrototypeOf       MetaCoin.propertyIsEnumerable  
  
MetaCoin.call               MetaCoin.constructor  
  
MetaCoin._properties          MetaCoin._property_values  
MetaCoin.address              MetaCoin.arguments  
MetaCoin.autoGas              MetaCoin.binary  
MetaCoin.class_defaults       MetaCoin.clone  
MetaCoin.contractName         MetaCoin.contract_name  
MetaCoin.defaults              MetaCoin.deployed  
MetaCoin.deployedSourceMap    MetaCoin.detectNetwork  
MetaCoin.events                MetaCoin.gasMultiplier  
MetaCoin.isDeployed           MetaCoin.legacyAST  
MetaCoin.links                  MetaCoin.metadata  
MetaCoin.networkType           MetaCoin.network_id  
MetaCoin.numberFormat         MetaCoin.prototype  
MetaCoin.schema_version       MetaCoin.setNetwork  
MetaCoin.setWallet              MetaCoin.source  
MetaCoin.timeoutBlocks        MetaCoin.toJSON  
MetaCoin.updatedAt             MetaCoin.updated_at
```

Developing a proof of idea project i

DEPLOY & RUN TRANSACTIONS

Environment: Web3 Provider (Custom (5777) network)

Account: 0x236...A9ADA (98.817 wei)

Gas limit: 3000000

Value: 0 wei

PatentIdea - browser/patent.sol

Deploy or At Address: Load contract from Address

Transactions recorded: 1

Deployed Contracts: PatentIdea at 0x10F...96Ed2 (blockchain)

SaveldeaHash string idea

getTracker

isAlreadyHashed string idea

Low level interactions

patent.sol

```
pragma solidity ^0.5.0;
contract PatentIdea {
    mapping (bytes32 => bool) private hashes;
    bool alreadyStored;
    int tracker=0;
    event ideahashed(bool);
    function saveHash(bytes32 hash) private {
        hashes[hash] = true;
    }
    function SaveIdeaHash(string memory idea) public returns (bool){
        bytes32 hashedIdea = HashtheIdea(idea);
        if (alreadyHashed(HashtheIdea(idea))) {
            alreadyStored = true;
            emit ideahashed(false);
            return alreadyStored;
        }
        saveHash(hashedIdea);
        tracker = tracker+1;
        emit ideahashed(true);
    }
    function alreadyHashed(bytes32 hash) private view returns(bool) {
        return hashes[hash];
    }
    function isAlreadyHashed(string memory idea) public view returns (bool) {
        bytes32 hashedIdea = HashtheIdea(idea);
        return alreadyHashed(hashedIdea);
    }
    function HashtheIdea(string memory idea) private pure returns (bytes32) {
        return bytes32(keccak256(abi.encodePacked(idea)));
    }
    function getTracker() public view returns (int) {
        return tracker;
    }
}
```

listen on network

Search with transaction hash or address

CREATION OF PATENTIDEA PENDING...

[block:113 txIndex:0] from:0x236...A9ADA to:PatentIdea.(constructor) value:0 wei data:0x608.

Creating the ideap project i

The necessary steps to create a proof of idea project, as detailed in the core Mastering Blockchain book, are as follows:

- Write the ideap smart contract
- Compile and test it in the Remix IDE
- Deploy to Ganache using Truffle
- Deploy to your network of choice (this is optional)
- Build a web frontend using Drizzle
- Run the DApp!

This is the resulting interactive frontend of the proof of idea DApp.

Patent DApp ii

The screenshot shows a web-based DApp interface titled "Patent DApp". On the left, there is a "My Account details" section displaying a long Ethereum address: **0xc9Bf76271b9E42E4bF7E1888e0F52351bDb65811**. Below it, the balance is shown as **28680 Ether**. There is also a field labeled "upload patent idea" with the value "hello123" and a "Submit" button.

On the right, there is a "CONTRACT INTERACTION" panel. It shows the account information: **Account 4** (**0xd8B0...E653**). The gas fee is listed as **0.000076** with a note "No Conversion Rate Available". The total amount is also listed as **0.000076** with the same note. At the bottom of this panel are two buttons: "Reject" and "Confirm".

- Traditionally, storage is centralized.
- In order to decentralize the entire blockchain ecosystem, storage services should also be decentralized, and serve as decentralized storage layer of the blockchain.
- DApps can benefit from decentralized storage, where backend data can be stored without fear of censorship or centralized control.

Atividade i

- Instalar as ferramentas do Capítulo e implementar os projetinhos de exemplos.
- Utilizando o **Truffle** baixar o exemplo de projeto **MetaCoin** e fazer o **deploy** no **Ganache**.

Leitura Recomendada

Capítulo 15: Introducing Web3

Livro: IMRAN BASHIR. Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Prática

Objetivos

- Explorar a biblioteca **Web3** com o cliente **Geth**. Fazer o deploy de contratos inteligentes utilizando o console **Geth** e o **Truffle**.

Leitura do ***Capítulo 15:***
Introducing Web3

1. Faça a leitura do [*Capítulo 15: Introducing Web3*](#)

Instalação das Ferramentas

Instalação das Ferramentas

1. Instale o `Node.js` e as bibliotecas necessárias para o Capítulo.
2. Utilizando o `Truffle` baixar o exemplo de projeto `MetaCoin` e fazer o `deploy` no `Ganache`.

Leitura Recomendada

Capítulo 15: Introducing Web3

Livro: IMRAN BASHIR. Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition.

Tokenização

Tokenização

1. Introdução

- 1.1 Falar de Tecnologias Blockchain de forma geral.
- 1.2 Contextualizar e falar que irá trabalhar com Ethereum.
- 1.3 Dar uma introdução sobre Tokenização.

2. Ethereum

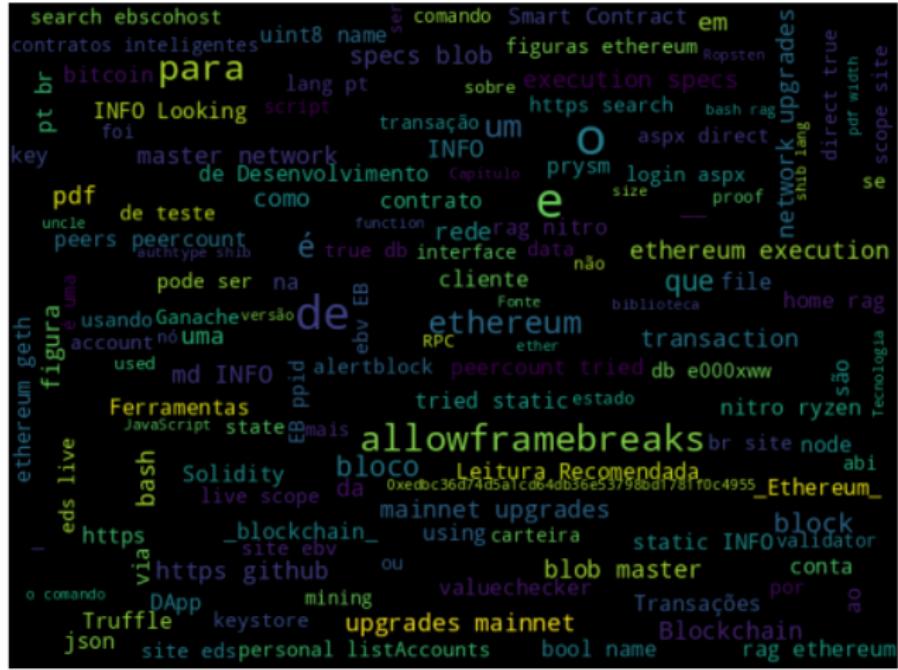
- 2.1 EVM
- 2.2 Redes: principal e testes

3. Solidity e implementação de Contratos Inteligentes

4. Tokenização

- 4.1 O que são tokens
- 4.2 Padrões de Implementação

Word Cloud



Referências

Referências i

Imran, Bashir. 2018. *Mastering Blockchain : Distributed Ledger Technology, Decentralization, and Smart Contracts Explained, 2nd Edition*. Packt Publishing. <https://search.ebscohost.com/login.aspx?direct=true&db=e000xww&AN=1789486&lang=pt-br&site=eds-live&scope=site>.