Blockchain sem neblina

Da primeira transação
ao próprio serviço na RpcNode



Sem complicação · com desenhos e exemplos

Sem complicação

Blockchain sem neblina

Da primeira transação ao seu próprio serviço na RpcNode



Este livro é para você se

- já ouviu a palavra “blockchain”, mas não entende como funciona;
- quer aceitar criptomoeda no seu site;
- programa e ainda nunca conversou com um nó.

O que você vai precisar

- curiosidade — não é preciso saber nada antes;
- um computador com terminal, para repetir os exemplos;
- as primeiras requisições funcionam sem cadastro e sem chave.

Como ler os capítulos



Autor: rpcnode.dev · 2026

Sumário

Parte 1. Por que um provedor

1. Quero aceitar cripto: do que eu preciso? 4
2. Quanto custa fazer tudo sozinho? 9
3. O que o provedor oferece 15

Parte 2. Blockchain do zero

4. Dinheiro e registros sem intermediário 21
5. Blocos e cadeia 26
6. Chaves, endereços e assinatura 31
7. O caminho da transação 36
8. Como a rede chega a um acordo 41
9. Moedas, tokens e contratos inteligentes 47

Parte 3. Nós e RPC

10. Os tipos de nó 53
11. Como conversar com um nó 59
12. O nó próprio por dentro 66
13. Como escolher um provedor 72

Parte 4. Como as redes funcionam

14. O Bitcoin e seus parentes 77
15. Ethereum: contas, gas e eventos 82
16. Parentes da Ethereum e “segundos andares” 88
17. TRON: energia no lugar do gas 93
18. Solana: tudo guardado em contas 99
19. TON: cartas em vez de telefonemas 105
20. Outras abordagens: de Sui a Cosmos 111
21. Qual rede escolher 118

Parte 5. RpcNode

22. Os primeiros cinco minutos na RpcNode 125
23. O que acontece com a requisição 131
24. Histórico e dados de arquivo 137
25. Planos, preço da chamada e limites 143
26. Eventos por endereço: webhooks e sockets 149
27. Pagamentos e monitoramento de transações 155

Parte 6. Mão na massa

28. Montando a loja 161
29. Quando tudo dá errado 168
30. Segurança e lista antes do lançamento 175

Parte 7. E depois

31. DeFi: finanças sem intermediário 182

No fim do livro

Conclusão 189
Apêndice A — Glossário 191
Apêndice B — As redes em uma página 193
Apêndice C — Cola de requisições 195

Capítulo 1. Quero aceitar cripto: do que eu preciso?

Tudo começou com um cliente

Artiom tem uma pequena loja virtual: ele torra café e manda pelo correio. Cartão, transferência, pagamento na entrega — tudo isso já funciona. Até que chega uma mensagem:

> "Olá! Quero encomendar dois quilos. Posso pagar com criptomoeda?"

Artiom topa. Instala uma carteira no celular, pega um endereço e manda para o cliente. Poucos minutos depois, o dinheiro aparece na tela.

Lá pelo décimo pedido, conferir a carteira na mão cansa. Ele quer que o próprio site perceba que o pedido foi pago.



Entendendo: o que é um nó

Para saber qualquer coisa sobre uma blockchain, é preciso perguntar a quem a guarda. Esse alguém se chama **nó** (em inglês, node).

Imagine um caderno coletivo onde está anotado quem transferiu quanto para quem. O caderno não fica trancado num único cofre. Há cópias dele com muitos participantes no mundo todo, e eles vivem comparando umas com as outras. Isso é a blockchain, e vamos destrinchá-la no capítulo 5.

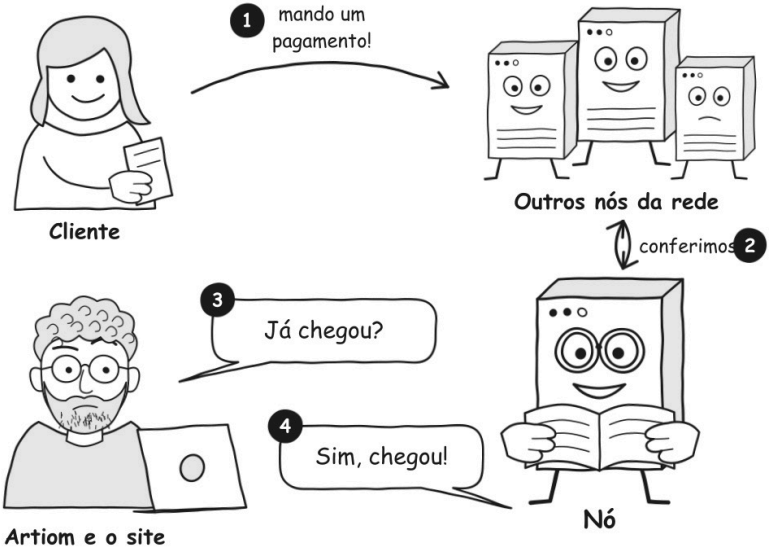
Um nó é um computador rodando o programa da rede. Ele faz quatro coisas:

1. Guarda a sua própria cópia dos registros.
2. Recebe registros novos de outros nós e os confere pelas regras da rede.

3. Passa adiante as transferências que lhe enviam.
4. Responde às perguntas dos programas: “qual é o saldo?”, “a transferência passou?”.

Há vários tipos de nó: alguns não guardam o histórico inteiro. Falamos disso no capítulo 10.

O site de Artiom não olha dentro da blockchain por conta própria. Ele faz uma pergunta ao nó, e o nó responde.



Na prática: a primeira pergunta ao nó

Os programas conversam com o nó seguindo regras simples: mandam uma requisição e recebem uma resposta. Um desses conjuntos de regras se chama **JSON-RPC**. Veja uma pergunta a um nó da rede Ethereum: “quantas moedas há neste endereço?”

```
curl https://ENDERECO_DO_NO \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["ENDERECO_DA_CARTEIR
A","latest"]}'
```

Aqui, `ENDERECO_DO_NO` é a porta em que batemos, e `ENDERECO_DA_CARTEIRA` é de quem queremos saber o saldo. A palavra `latest` quer dizer “no momento mais recente”.

O nó responde:

```
{"jsonrpc":"2.0","id":1,"result":"0xde0b6b3a7640000"}
```

Parece uma sopa de letrinhas, mas é um número escrito em hexadecimal: 1.000.000.000.000.000.000. O nó conta nas menores frações da moeda, chamadas **wei**. Um ETH tem exatamente essa quantidade. Ou seja, há 1 ETH no endereço.

Então o truque todo é uma pergunta e uma resposta. Falta decidir de quem será esse `ENDERECO_DO_NO`.

Dois caminhos: nó próprio ou provedor

Você pode rodar um nó por conta própria. Ou pode se conectar a um nó de outra pessoa, que cuida dele para você. Esse tipo de serviço se chama **provedor**. Ele entrega o `ENDERECO_DO_NO` pronto. A RpcNode, de que trata este livro, é um desses serviços.

Pergunta	Nó próprio	Provedor
Quando dá para começar	Depois de instalar e sincronizar: de horas a semanas, conforme a rede	Depois do cadastro, em poucos minutos
Pelo que se paga	Servidor, discos, tráfego e o tempo de um especialista	O plano do serviço
Quem atualiza e conserta	Você	O provedor
Várias redes	Um nó para cada uma	Em geral, tudo num painel só
O que se arrisca	O nó ficar para trás ou quebrar enquanto você dorme	Depender do serviço de outros

Os dois caminhos têm seu lugar. O nó próprio se justifica se você precisa de privacidade especial, se a carga é muito alta ou se participa do funcionamento da própria rede.

> **Artiom:** "Eu torro café, não faço plantão em servidor. Mas, afinal, quanto custa fazer tudo sozinho?"

Resposta curta: para várias redes, com servidores reserva e histórico completo, a conta chega a dezenas de milhares de dólares. Um único nó pequeno sai muito mais barato. A conta detalhada está no capítulo 2.

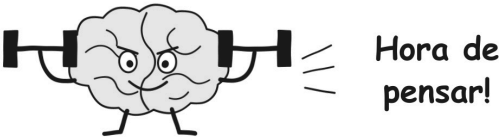
Cilada: "vou dar ao provedor a chave da carteira"

Nunca. Para saber o saldo ou ver um pagamento, o nó só precisa do endereço. A chave privada e a frase secreta mandam no dinheiro: quem as conhece pode levar tudo. Um provedor honesto não precisa delas. Veremos as chaves no capítulo 6.



E uma segunda ressalva. A requisição acima mostra apenas a moeda nativa da rede, no caso o ETH. Se o cliente pagar com um token, como o USDT, essa pergunta não vai enxergar. Para tokens se pergunta de outro jeito: é o assunto do capítulo 9.

Pense bem



1. Artiom confere um pagamento, mas o nó dele está uma hora atrasado em relação à rede. O que ele vai ver e o que vai escrever ao cliente?
2. A loja tem três pedidos por dia. E se fossem três mil? O que muda na escolha entre nó próprio e provedor?

Dica para a primeira: um nó atrasado mostra, com toda a honestidade, o que existia uma hora atrás.

O essencial

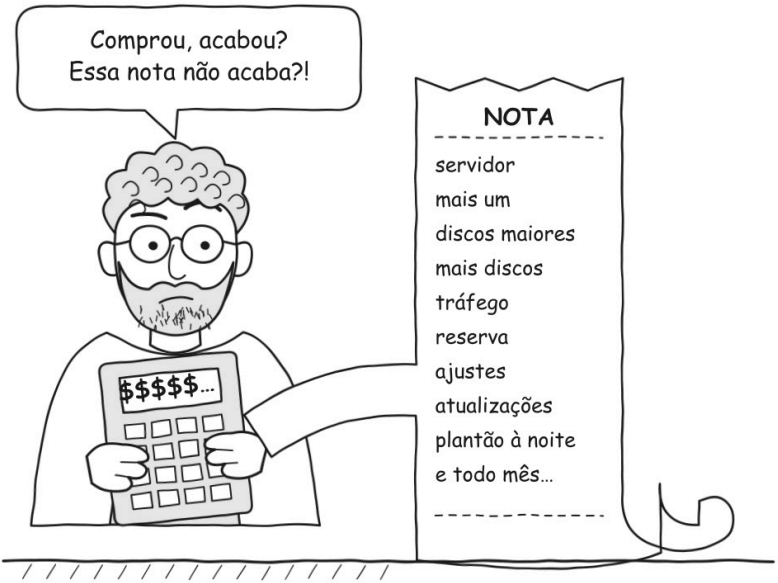
- Um programa fica sabendo o que há na blockchain fazendo perguntas a um nó.
- Você pode manter o próprio nó ou contratar um provedor.
- O nó precisa do endereço, mas não da sua chave privada.

A seguir: capítulo 2. Artiom pega a calculadora e calcula quanto custariam servidores próprios.

Capítulo 2. Quanto custa fazer tudo sozinho?

Artiom pega a calculadora

O raciocínio de Artiom é simples: “Compro um servidor, instalo um nó e os pagamentos andam sozinhos. Pago uma vez só”. Ele abre a calculadora e começa a anotar.



Entendendo: do que é feita a conta

O servidor é só a pontinha. O resto não aparece no começo.



1. **Servidor.** Processador, memória e discos NVMe rápidos.
2. **Servidor reserva.** Se um quebra, o outro responde enquanto o primeiro é consertado.
3. **Os discos crescem.** A blockchain só acrescenta. A cadeia do Bitcoin, no ano até novembro de 2026, passou de uns 700 para 786 GB.
4. **Tráfego.** O nó troca dados o tempo todo. Um liteserver da TON precisa, no pico, de uns 16 TB por mês.
5. **Atualizações.** A rede muda as regras, e o programa antigo deixa de acompanhar.
6. **Gente.** Alguém tem de configurar tudo isso, vigiar e acordar no meio da noite.

Quanto de máquina cada rede pede

Estes são os requisitos, segundo a documentação oficial, para um nó comum, não para um que participa das votações da rede.

Rede	Processador	Memória	Disco
Bitcoin	a partir de 1 GHz	2 GB	recomendam 750 GB, mas a cadeia em si já tem cerca de 786 GB
Ethereum	4+ núcleos	32 GB	NVMe de 4 TB
BNB Smart Chain	16 núcleos	64 GB	SSD a partir de 3 TB; o snapshot completo da base já passa de uns 7,4 TB, o reduzido, cerca de 1,9 TB
TRON	16 núcleos	32 GB	SSD a partir de 3,5–4 TB
Solana (nó RPC)	16 núcleos ou mais	a partir de 512 GB	vários NVMe: contas 1 TB+, ledger 1 TB+, snapshots 500 GB+

Notou as esquisitices? Para o Bitcoin recomendam um disco menor que a própria cadeia. Para a BNB o mínimo é 3 TB, mas o snapshot completo tem mais que o dobro disso. A documentação nem sempre acompanha a rede, então se compra disco com folga.

E se for preciso o histórico inteiro?

Um nó comum conhece bem o presente. Para perguntar "qual era o saldo há um ano?", é preciso um nó **de arquivo**, assunto do capítulo 10. Ele é bem mais pesado. O arquivo da TON precisa de 16–20 TB. No Ethereum o tamanho depende muito do programa: de 2 a 14 TB ou mais. Os arquivos não entram na nossa conta: com eles sai mais caro.

Fazendo as contas

Dá para comprar um servidor próprio, mas vamos calcular o aluguel. É uma estimativa a partir das tabelas de preços públicas de empresas de hospedagem em 27 de novembro de 2026, sem impostos nem taxa de instalação. Os preços mudam.

Classe do servidor	O que tem dentro	Mais ou menos por mês
Médio	8 núcleos, 64 GB, 2×2 TB NVMe	US$ 180
Grande	16 núcleos, 128 GB, 4–8 TB NVMe	US$ 460–560
Muito grande	512 GB de memória ou mais	US$ 1.160–2.350

Agora, cinco redes. Pegamos preços mais perto do piso e colocamos dois servidores por rede: o principal e o reserva.

Rede	Classe	Um servidor, por mês	Com reserva, por mês
Bitcoin	médio	US$ 180	US$ 360
Ethereum	médio	US$ 180	US$ 360
BNB Smart Chain	grande	US$ 500	US$ 1.000
TRON	grande	US$ 500	US$ 1.000
Solana	muito grande	US$ 1.200	US$ 2.400
Total		**US$ 2.560**	**US$ 5.120**

Artiom: "5.120 dólares por mês são 61.440 por ano. E isso é só o aluguel?!"

É. Usando os preços mais altos da primeira tabela, dá uns US$ 92.000 por ano. É daí que vêm as "dezenas de milhares de dólares".

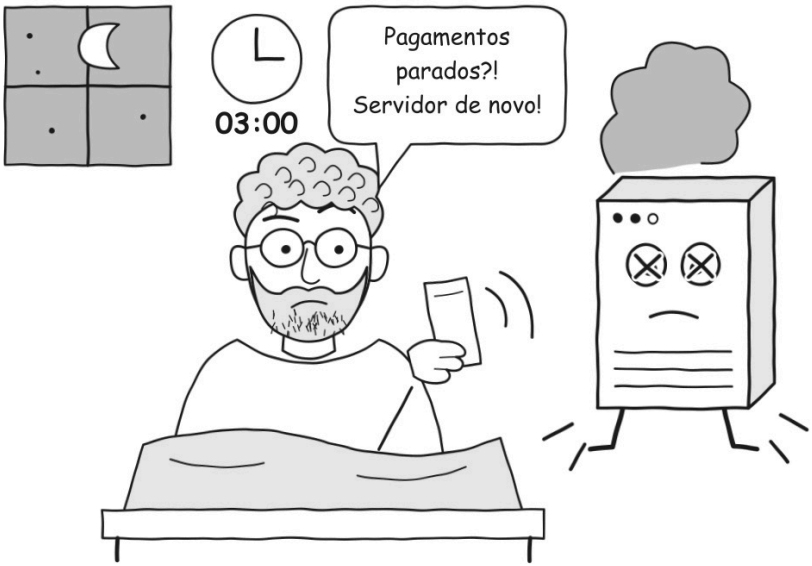
Manter uma única rede é bem mais barato. O Bitcoin é pouco exigente: um servidor mais modesto que o médio dá conta.

E quem vai cuidar disso?

Servidor não se mantém sozinho. Segundo o portal Habr Career, um engenheiro DevOps de nível pleno na Rússia ganha cerca de 160–290 mil rublos por mês. Uma pessoa só não consegue ficar de plantão 24 horas, sem folga e sem férias. Mesmo duas já custam cerca de 3,8–7 milhões de rublos por ano.

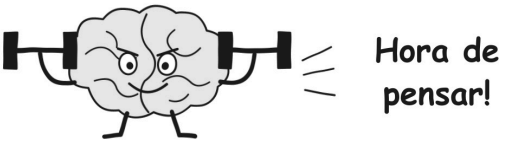
Cilada: "compro um servidor e pronto"

Um servidor só é um ponto único de falha. O disco encheu, o programa travou, a rede se atualizou — e o site deixa de ver o pagamento. O cliente mandou o dinheiro, mas o pedido consta como "não pago".



E mais uma coisa que muita gente esquece. Leia as regras da hospedagem antes de pagar: algumas proíbem programas ligados a criptomoedas e podem desligar o servidor.

Pense bem



1. Artiom só precisa de Bitcoin, com um servidor reserva de classe média. Quanto dá por ano?
2. Às cinco redes se somou uma sexta, em servidores grandes de US$ 500, também com reserva. Quanto a conta anual aumenta?
3. Os clientes pagam três vezes por dia. Eles compensam esse gasto?

Respostas: 1) 360 × 12 = US$ 4.320. 2) 1.000 × 12 = US$ 12.000. 3) A decisão é sua, e o capítulo 3 ajuda.

O essencial

- Não se paga uma vez só, e sim todo mês.
- Cinco redes com reserva são mais de US$ 60.000 por ano só de aluguel, sem arquivos e sem gente.
- Os discos crescem, as redes se atualizam, alguém precisa ficar de plantão.

A seguir: capítulo 3. O que o provedor assume e quando o nó próprio ainda é necessário.

De onde vêm os números

Todas as páginas foram consultadas em 27 de novembro de 2026.

- Bitcoin, requisitos: bitcoin.org/en/bitcoin-core/features/requirements
- Bitcoin, tamanho da cadeia: api.blockchain.info/charts/blocks-size
- Ethereum: ethereum.org/developers/docs/nodes-and-clients/run-a-node
- BNB Smart Chain: docs.bnbchain.org/bnb-smart-chain/developers/node_operators/node_best_practices; github.com/bnb-chain/bsc-snapshots
- TRON: tronprotocol.github.io/documentation-en/using_javatron/installing_javatron; developers.tron.network/docs/deploy-the-fullnode-or-supernode
- Solana: docs.anza.xyz/operations/requirements
- TON: docs.ton.org/nodes/cpp/run-liteserver; docs.ton.org/nodes/cpp/run-archive-liteserver
- Salários: career.habr.com/professions/devops (dados de 2025–2026)

Os preços de servidores são uma estimativa arredondada do autor a partir das tabelas públicas de várias empresas de hospedagem.

Capítulo 3. O que o provedor oferece e quando o nó próprio ainda é necessário

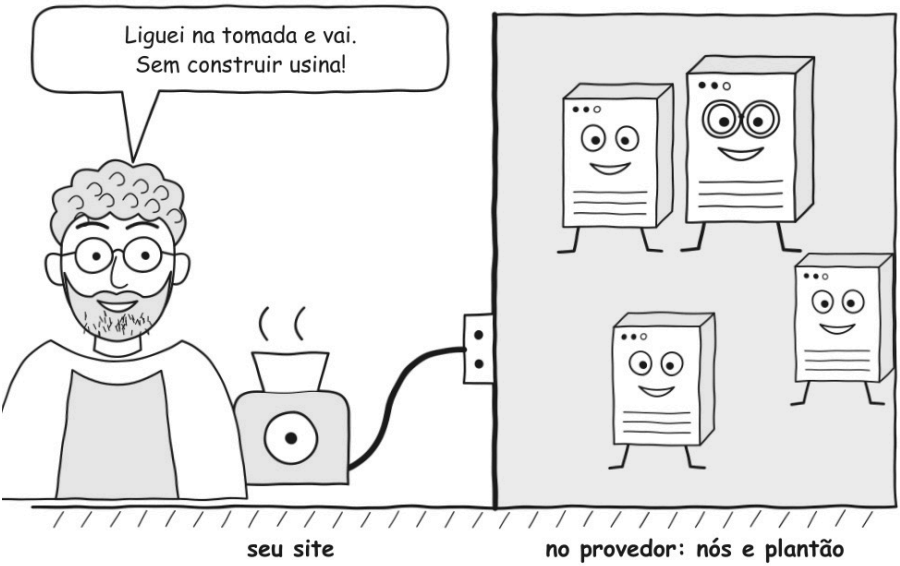
Artiom na encruzilhada

Depois das contas do capítulo 2, a calculadora está de cara para baixo. Cinco redes com reserva dão mais de 60.000 dólares por ano só de aluguel. Café não rende tanto.



Entendendo: o que o provedor assume

Você tira eletricidade da tomada; não constrói uma usina para ligar a chaleira. Com os nós é igual: o **provedor** mantém os nós com ele e dá a você um endereço para fazer perguntas.



Em geral, ficam do lado dele cinco preocupações:

1. **Máquinas e discos.** Comprar, alugar, ampliar quando o espaço acaba.
2. **Sincronização.** O nó não pode ficar atrás da rede.
3. **Atualizações.** A rede muda as regras — o programa é trocado a tempo.
4. **Troca do que quebrou.** As requisições vão para um nó saudável.
5. **Plantões.** Quem acorda de madrugada não é Artiom.

Há provedores de todo tipo, e eles prometem coisas diferentes. O que dessa lista você realmente recebe se verifica antes de pagar. Como escolher, veremos no capítulo 13.

O que continua com você

- **As chaves da carteira.** Só você as tem.
- **O endereço do endpoint.** É como se chama o endereço que o provedor entrega. Muitas vezes sua chave de acesso pessoal vai embutida nele. Guarde-o como uma senha: requisições de estranhos gastam o seu limite.
- **O seu código.** Ele precisa sobreviver a um erro, repetir a requisição e não contar o mesmo pagamento duas vezes.

Três termos de qualquer plano

Termo	O que significa	Pergunte-se
Requisições por segundo (RPS)	Com que frequência se pode perguntar	Quantos clientes vão chegar ao mesmo tempo?
Limite, ou créditos	Quanto se pode perguntar ao todo num período. Perguntas diferentes custam diferente	Dá até o fim do mês?
Acesso de arquivo	Se dá para perguntar sobre o passado distante	Preciso do histórico antigo?

Na prática: uma pergunta a um nó de verdade

No capítulo 1, no lugar do endereço estava `ENDERECO_DO_NO`. Agora vamos pôr um de verdade. A RpcNode tem nós públicos gratuitos, que não pedem cadastro nem chave. Vamos perguntar o número do último bloco da rede Ethereum:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

Foi isto que o nó respondeu quando este capítulo estava sendo escrito:

```
{"jsonrpc":"2.0","id":1,"result":"0x1946393"}
```

`0x1946393` é 26.502.035. O seu número será maior: a rede não fica parada. Repita a requisição daqui a um minuto e veja.

Os nós públicos são compartilhados por todos, por isso têm restrições: em 27 de novembro de 2026, são 5 requisições por segundo por endereço, e os métodos mais "caros" ficam fechados. Para conhecer, é o bastante. Uma loja precisa do próprio endpoint.

Quanto custa

Como exemplo, os dois primeiros planos da RpcNode em 27 de novembro de 2026. Os valores atuais estão no site.

	Free	Starter
Preço por mês	US$ 0	US$ 19
Créditos	15 milhões	50 milhões
Requisições por segundo	15	40
Endpoints	3	5
Redes	Ethereum e Solana	todas
Acesso de arquivo	não	sim

Os nós da RpcNode ficam em três regiões: Frankfurt, EUA e Japão.

> **Artiom:** "19 dólares contra 5.120 por mês? Qual é a pegadinha?"

São coisas diferentes. Um servidor próprio é inteiro seu. No plano, você recebe uma fatia da capacidade total, com limites. Para uma loja pequena a fatia basta, e não faz sentido pagar por uma "usina" inteira.

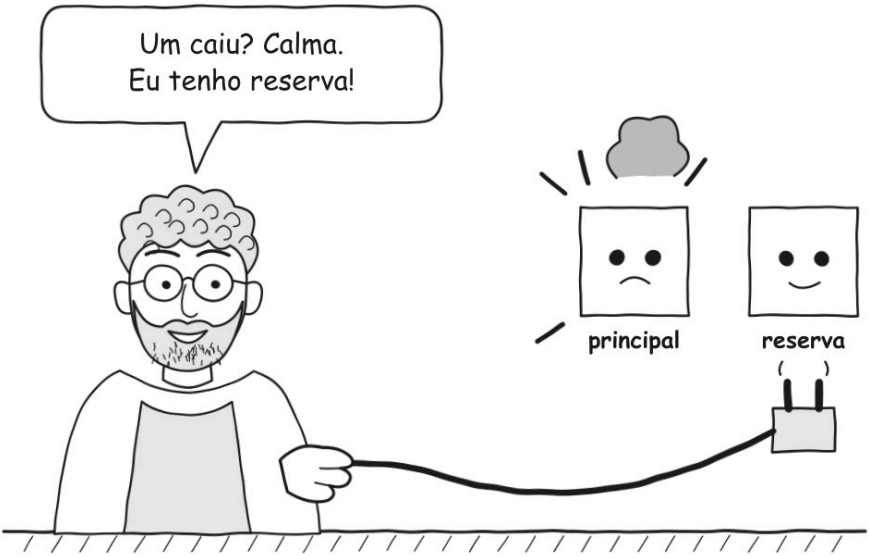
Quando o nó próprio ainda é necessário

- **Privacidade.** O provedor vê as suas requisições: quais endereços você consulta e de onde pergunta.
- **Carga enorme.** São tantas requisições que o plano sai mais caro que ter servidores.
- **Participação na rede.** Você é validador ou desenvolve a própria rede.
- **Configurações especiais.** Você precisa de um método ou modo que o provedor não oferece.
- **Exigências da lei ou do contratante.** Os dados só podem ser processados por você.

Existe também o caminho misto: nó próprio para a rede principal, provedor para as outras e como garantia.

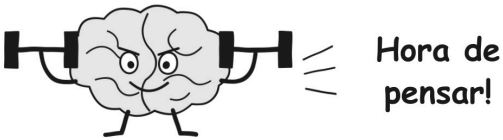
Cilada: "o provedor nunca vai cair"

Tudo cai: servidores, links de comunicação e os serviços dos provedores também. Se o pagamento é importante para você, decida antes o que o site faz quando não vem resposta.



- Não diga ao cliente "não pago" quando você simplesmente não conseguiu perguntar.
- Repita a requisição mais tarde, e não cem vezes seguidas.
- Tenha um endereço reserva: um segundo provedor ou o seu próprio nó.

Pense bem



1. O site de Artiom confere o pagamento uma vez por segundo para cada um de 20 pedidos em espera, com uma requisição por conferência. Isso cabe em 15 requisições por segundo?
2. O que o cliente deve ver se o nó passar dez minutos sem responder?
3. Quem deles precisa de nó próprio: a loja de café ou a equipe que desenvolve a própria rede?

Dica para a primeira: 20 é mais que 15. Confira com menos frequência ou pergunte com mais esperteza: como, no capítulo 26.

O essencial

- O provedor assume as máquinas, a sincronização, as atualizações e os plantões.
- As chaves, o endereço do endpoint e o comportamento do site nas falhas continuam com você.
- O nó próprio é necessário por privacidade, carga alta, participação na rede ou exigências especiais.

Fim da primeira parte. Artiom escolheu um provedor e recebeu a resposta de um nó de verdade. Mas que "bloco" número 26.502.035 é esse que o nó informou? **A seguir:** parte 2, capítulo 4. Dinheiro e registros sem intermediário.

Capítulo 4. Dinheiro e registros sem intermediário

Onde fica o seu dinheiro?

Artiom recebeu uma transferência no cartão. Nenhuma cédula saiu do lugar. Então o que aconteceu? Alguém diminuiu o número ao lado do nome do cliente e aumentou o número ao lado do nome de Artiom.

Dinheiro eletrônico é um registro. E quem cuida dos registros se chama intermediário: um banco ou um serviço de pagamento.



Entendendo: para que serve o intermediário e onde ele atrapalha

O intermediário faz muita coisa útil. Confere se há dinheiro suficiente. Não deixa gastar o mesmo duas vezes. Pode resolver uma disputa e devolver um pagamento.

Mas todos os registros estão nas mãos dele. Ele pode parar uma transferência, cobrar taxa, bloquear a conta ou estar fechado no fim de semana. Isso pesa ainda mais quando o cliente mora em outro país.

> **Artiom:** "Então que o cliente me mande o dinheiro como um arquivo. Igual a uma foto. Sem intermediário nenhum!"

Por que não dá para simplesmente mandar um arquivo

Uma foto pode ser enviada cem vezes, e ela continua com você. Com dinheiro não pode ser assim: a mesma moeda pagaria duas pessoas. Isso se chama **gasto duplo**.



Durante muito tempo só houve uma solução: alguém que manda mantém a lista e diz de quem é a moeda agora. Ou seja, de novo um intermediário.

A ideia: todos guardam os registros

Em 2008, um autor sob o pseudônimo Satoshi Nakamoto descreveu outro caminho, e em janeiro de 2009 a rede Bitcoin começou a funcionar. A ideia é esta:

1. A lista de transferências não fica escondida com um só. **Muitos têm uma cópia.**
2. Cada nova transferência é **conferida pelos participantes segundo regras comuns**: há dinheiro e ele ainda não foi gasto?
3. Os participantes **entram em acordo** sobre quais registros valem, sem chefe.



Essa lista comum, mantida em conjunto, é o que se chama **blockchain**. Como os registros formam uma cadeia, veremos no capítulo 5; como os participantes entram em acordo, no capítulo 8.

Na prática: o jogo do caderno coletivo

Não precisa de código. Chame dois amigos e dê a cada um uma folha. No começo, todos anotam: Ana tem 10 moedas, Boris 5, Vera 0. A regra é uma só: **ninguém pode dar mais do que tem**.

Nº	Quem quer transferir	Aceitar?	Ana	Boris	Vera
início			10	5	0
1	Ana → Vera 4				
2	Vera → Boris 1				
3	Boris → Ana 7				
4	Ana → Boris 6				

Preencha os quadrinhos vazios e depois compare as folhas. Se todas estiverem iguais, vocês acabaram de chegar a um acordo sem chefe. A resposta está no fim do capítulo.

O que a blockchain dá e o que não dá

Dá	Não dá
Transferência direta, a qualquer dia e hora	Botão de "cancelar"
A mesma conta acessível de vários países	Preço estável da moeda: a cotação pode variar muito
Registros que qualquer um pode conferir	Sigilo: as transferências de qualquer endereço são visíveis a todos
Ninguém precisa confiar a outro a guarda da lista	Um suporte que recupere a chave perdida

Nem tudo oscila: há moedas que tentam se manter em torno de um dólar, chamadas **stablecoins**. Mas por trás das mais difundidas há uma empresa emissora, que pode congelar o dinheiro de um endereço. Os detalhes estão no capítulo 9.

Os intermediários não sumiram: há corretoras, carteiras, provedores. Mas eles são diferentes entre si. Uma corretora pode ficar com as suas moedas. O provedor da primeira parte não guarda dinheiro: só leva as suas perguntas até o nó.

Cilada: "se eu não gostar, cancelo"

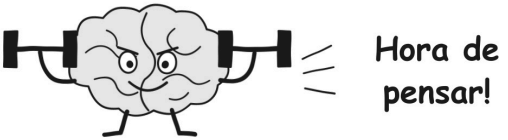
Uma transferência confirmada não se cancela. Errou o endereço — o dinheiro foi embora. Só o destinatário pode devolvê-lo, se quiser e se aquele endereço for mesmo de alguém.



Por isso o endereço é conferido duas vezes, e antes de uma quantia alta se manda primeiro um pedacinho como teste.

Uma historinha. Em maio de 2010, pagaram 10.000 bitcoins por duas pizzas. Aquela transferência continua anotada no caderno coletivo até hoje.

Pense bem



1. Depois do jogo, Vera tem 3 moedas. Ao mesmo tempo, ela escreve a Ana "te dou 3" e a Boris "te dou 3". O que os participantes devem fazer?
2. E se um participante aceitou a primeira transferência e outro, a segunda? Quem está certo?
3. O que é melhor para Artiom: poder cancelar um pagamento ou não poder? E para o cliente dele?

Dicas: 1) só uma pode ser aceita. 2) É a pergunta mais difícil, e o capítulo 8 é dedicado a ela.

O essencial

- Dinheiro eletrônico são registros, e alguém precisa cuidar deles.
- A blockchain é uma lista comum, mantida por muitos segundo regras comuns.
- Livrar-se do intermediário também é assumir a responsabilidade: não há quem cancele a transferência nem recupere a chave.

Resposta do jogo: a transferência 1 é aceita (6, 5, 4); a 2 é aceita (6, 6, 3); a 3 é recusada, Boris só tem 6 (6, 6, 3); a 4 é aceita (0, 12, 3). O total de moedas continua 15.

A seguir: capítulo 5. Que "bloco" é esse que o nó informou e por que não dá para reescrever um registro antigo às escondidas.

Capítulo 5. Blocos e cadeia: por que não dá para reescrever o passado às escondidas

Aquele tal número

No capítulo 3, o nó respondeu a Artiom: o último bloco é o de número 26.502.035. Que bloco é esse? Por que ele tem número? E o que impede alguém de corrigir um registro do caderno na surdina?

Entendendo: um bloco é uma página do caderno

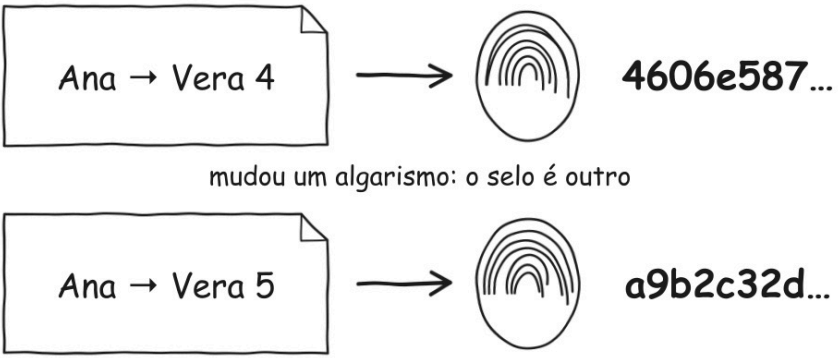
As transferências não são anotadas uma a uma. Elas são juntadas em pacotes, como linhas numa página. Essa página se chama **bloco**. Nela há:

1. **Transferências** e outras operações, chamadas de transações.
2. O **número** de ordem da página.
3. A **hora** em que foi criada.
4. O **selo da página anterior.** Aí está todo o segredo.

O selo, ou hash

Para qualquer texto dá para calcular uma sequência curta de números e letras. É o **hash**, que vamos chamar de selo. Ele lembra uma impressão digital:

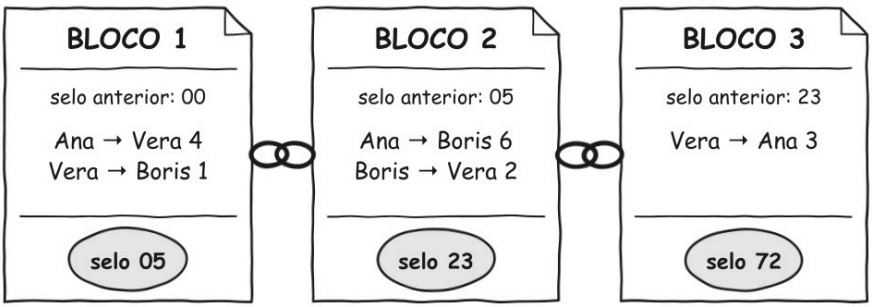
- o mesmo texto sempre tem o mesmo selo;
- mudou uma letra que seja — o selo é completamente outro;
- pelo selo não se reconstrói o texto;
- achar outro texto com o mesmo selo é, na prática, impossível.



São os começos de hashes SHA-256 verdadeiros das duas linhas do desenho. O SHA-256 é usado pelo Bitcoin; outras redes calculam os selos de outras maneiras.

A cadeia

Em cada página está anotado o selo da anterior. E a página recebe o próprio selo a partir de tudo o que há nela, **inclusive esse selo anterior**.

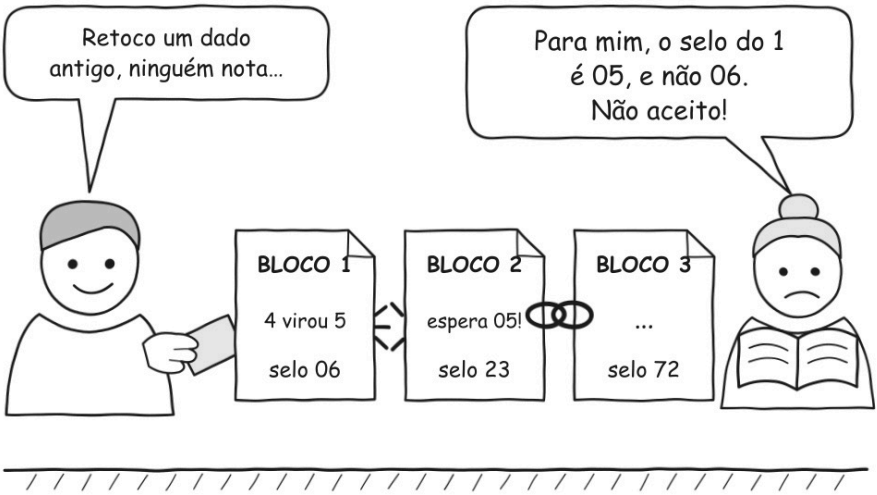


Assim as páginas se engatam numa **cadeia de blocos**. Em inglês, blockchain. O nome é só isso.

Os selos do desenho são de brinquedo, com dois algarismos: assim dá para calcular de cabeça.

O que acontece se alguém mexer no passado

Um golpista troca, no primeiro bloco, o 4 por 5. O selo do bloco muda. Mas no segundo bloco está anotado o antigo. A cadeia se rompe, e isso se vê na hora.



Ele pode reescrever todos os selos seguintes também. Mas os outros participantes têm suas próprias cópias, e a versão dele não vai bater com elas. Para que a versão dele fosse aceita, teria de vencer o resto da rede na força. Isso é muito caro. Como exatamente a rede decide qual cadeia é a verdadeira, veremos no capítulo 8.

Veja uma cadeia de verdade

Estes são três blocos da rede Ethereum de 26 de novembro de 2026. Os selos são compridos, por isso aparecem só o começo e o fim.

Número do bloco	Seu selo	Selo do anterior	Transações
26.502.033	`0xbe58…d764`	`0x2844…c876`	229
26.502.034	`0x7bb7…b7a4`	`0xbe58…d764`	229
26.502.035	`0xeab2…c904`	`0x7bb7…b7a4`	515

Passe o dedo: o selo de uma linha se repete na seguinte. Entre esses blocos passaram exatamente 12 segundos de cada vez. No Bitcoin, um bloco novo surge mais raramente, em média a cada 10 minutos.

O primeiríssimo bloco não tem anterior. O primeiro bloco do Bitcoin foi criado em 3 de janeiro de 2009.

Na prática: o seu próprio selo

Vamos inventar uma regra de brinquedo:

Selo = (selo anterior × 3 + todas as quantias da página); do resultado, ficam os dois últimos algarismos.

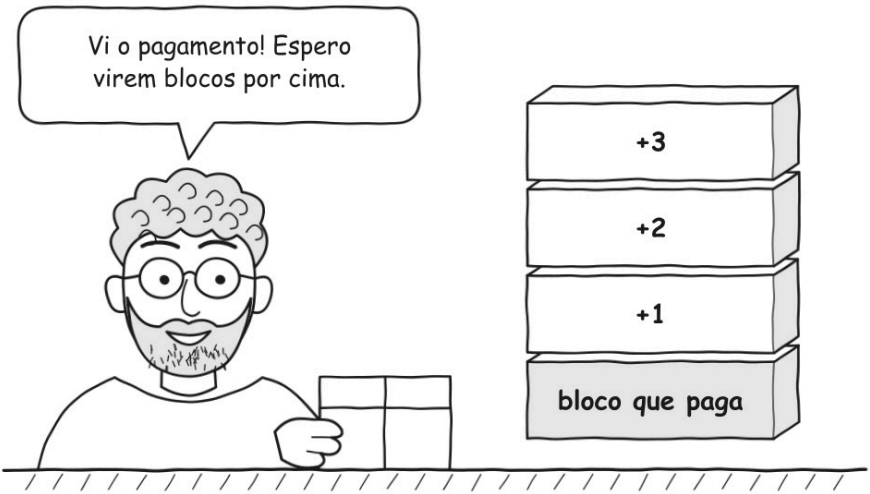
Bloco	Quantias transferidas	Selo	Se no bloco 1 o 4 virar 5
1	4 e 1		
2	6 e 2		
3	3		

Para o primeiro bloco, o selo anterior é 0. Preencha as duas colunas. Quantos selos mudaram?

Cilada: "vi no bloco, então é para sempre"

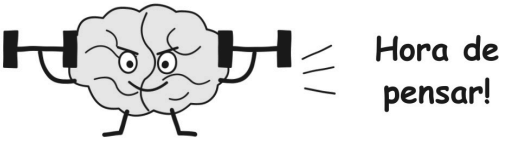
Os blocos mais recentes às vezes são substituídos por outros: partes da rede ainda não tiveram tempo de se acertar. Quanto mais blocos se empilham por cima, mais difícil é remover o seu registro.

Isso é contado em **confirmações**: o próprio bloco com o pagamento é a primeira, e cada bloco seguinte acrescenta mais uma.



Quanto esperar depende da rede e da quantia. Por uma xícara de café, menos; por uma encomenda grande, mais. Detalhes no capítulo 8.

Pense bem



1. Engane o nosso selo: troque no bloco 1 as quantias 4 e 1 por outras, de modo que o selo não mude. O que isso diz sobre a nossa regra?
2. Por que é mais difícil mudar um bloco de três anos atrás do que o de ontem?
3. Se surge um bloco a cada 12 segundos, quantos blocos saem em uma hora?

Respostas: 1) por exemplo, 3 e 2: a soma é a mesma. Um hash de verdade perceberia. 2) Seria preciso refazer tudo o que se empilhou por cima. 3) 300.

O essencial

- Um bloco é uma página com transações, número, hora e o selo da página anterior.
- Alterar um registro antigo muda o selo e rompe a cadeia.
- Um bloco fresco merece menos confiança: espera-se pelas confirmações.

Resposta do jogo: selos 05, 23, 72. Depois da troca, 06, 26, 81: os três mudaram.

A seguir: capítulo 6. Como a rede sabe que a transferência foi enviada mesmo pelo dono do dinheiro? Chaves, endereços e assinatura.

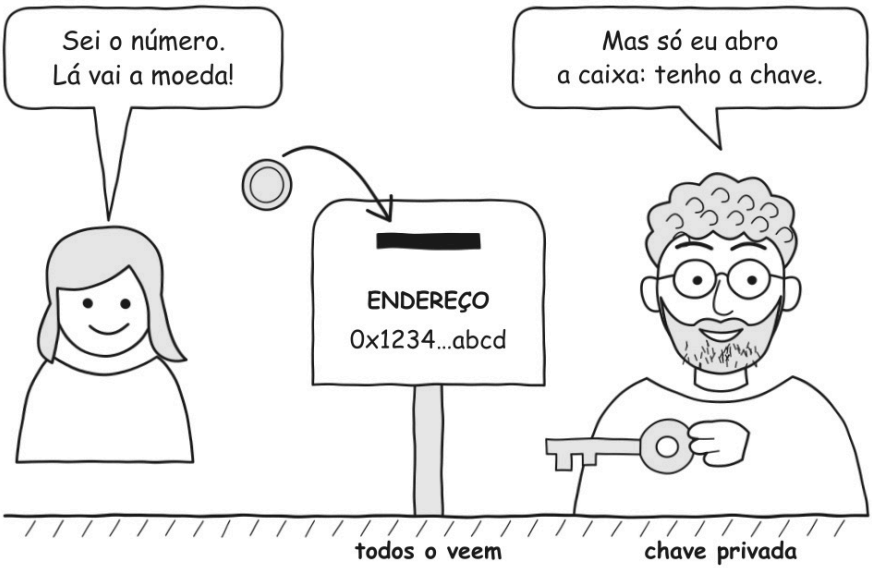
Capítulo 6. Chaves, endereços e assinatura

Como a rede sabe que o dinheiro é seu?

No caderno coletivo não há documentos de identidade. Ninguém pergunta o nome nem confere a foto. Então o que impede qualquer um de escrever: "Artiom me dá tudo"?

Entendendo: a caixa de correio e a chave

Imagine uma caixa de correio na rua. O número dela todo mundo sabe, e qualquer um pode jogar uma moeda pela fenda. Mas abrir a caixa e pegar a moeda, só quem tem a chave.



O quê	Com o que se parece	A quem mostrar
Endereço	O número da caixa	A quem quiser, para receber dinheiro
Chave pública	O modelo pelo qual se confere a assinatura	Pode a todos
Chave privada	A chave da caixa	**A ninguém**

A ligação entre eles tem mão única. Da chave privada se obtém a pública; da pública, o endereço. Fazer o caminho de volta é, na prática, impossível. Em algumas redes o endereço é a própria chave pública, só que escrita de um jeito especial.

A **carteira** é um programa ou aparelho onde as chaves ficam guardadas. Não há moedas dentro dela: elas continuam sendo registros no caderno coletivo.

A assinatura

Para dar moedas a alguém, o dono redige a transferência e a **assina** com a chave privada. A chave não é enviada a lugar nenhum. Para a rede vão a transferência e a assinatura.



O nó confere duas coisas: se a assinatura foi feita com a chave justamente daquele endereço e se o texto não foi mudado depois de assinado. A assinatura de uma transferência não serve para outra. E a mesma transferência a rede não executa uma segunda vez.

Na prática: uma assinatura de brinquedo

Tudo se calcula numa calculadora. A chave privada é o número **7**, a pública é o número **3**. Só o primeiro é segredo.

> Regra: multiplique pelo número escolhido e, depois de cada passo, fique com o resto da divisão por 33.

Assinando a mensagem "4". Tomamos o 4 tantas vezes quantas a chave privada mandar, ou seja, 7:

Quantos quatros	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Resto	4	16	31	25	1	4	**16**

Cada passo é o resto anterior multiplicado por 4. Por exemplo, o terceiro: 16 × 4 = 64, e o resto da divisão por 33 é 31. Depois do sétimo passo temos a assinatura: **16**.

Conferindo. Tomamos a assinatura 16 tantas vezes quantas a chave pública mandar, ou seja, 3: 16; depois 16 × 16 = 256, resto 25; depois 25 × 16 = 400, resto **4**. Saiu a nossa mensagem. A assinatura é verdadeira!

Agora alguém corrige a mensagem para "5" e deixa a assinatura 16. A conferência continua dando 4, e não 5. A falsificação aparece.

É só a ideia. Números pequenos assim, um estudante quebra. Nas redes de verdade, as chaves são números gigantescos e a matemática é outra.

A frase secreta

Copiar a chave privada à mão é incômodo: é uma sequência comprida. Por isso muitas carteiras dão uma **frase secreta** de 12 ou 24 palavras. A partir dela se recuperam as chaves e todos os endereços.



- Quem conhece a frase é o dono do dinheiro.
- Perdeu a frase e o aparelho — não há quem recupere.
- Não fotografe, não guarde em anotações nem em conversas.
- Não digite em sites nem dite ao "suporte". O suporte de verdade não pede essa frase.

Uma senha pode ser trocada; uma chave privada, não. Se ela vazou, só resta transferir o dinheiro com urgência para um endereço novo.

O que isso significa para a loja

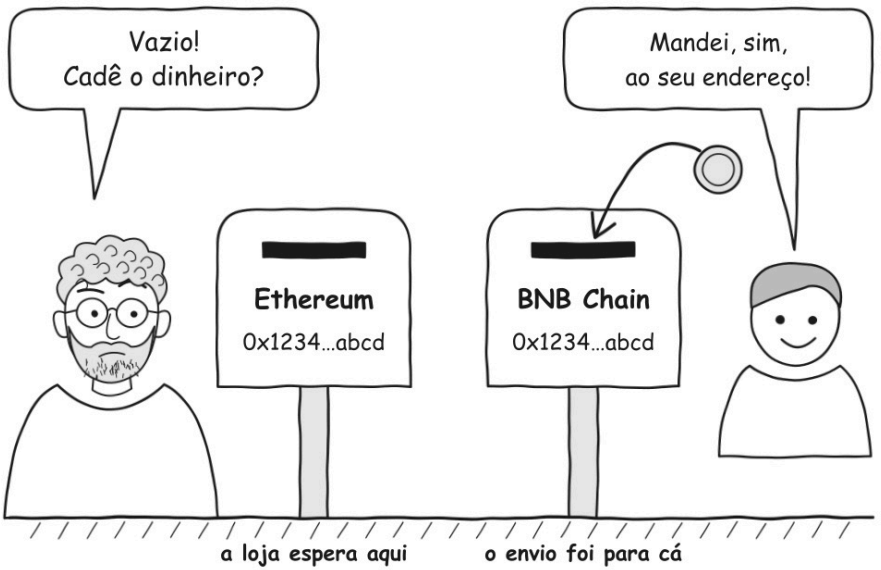
Para **receber** dinheiro e ver o pagamento, o site de Artiom só precisa do endereço. A chave privada não é necessária no servidor da loja, e não vale a pena guardá-la lá: se invadirem o site, levam o dinheiro junto. É prático dar a cada pedido um endereço próprio, assunto do capítulo 28.

Cilada: "o endereço é o mesmo, então chega"

Os endereços de redes diferentes têm caras diferentes:

Rede	O endereço começa com
Bitcoin	`1`, `3` ou `bc1`
Ethereum e redes parecidas	`0x`
TRON	`T`
TON	em geral `EQ` ou `UQ`
Solana	não há começo especial

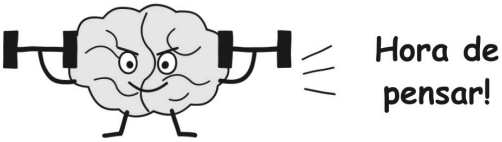
Mas na Ethereum, na BNB Smart Chain, na Polygon e em muitas outras redes o endereço tem a **mesma** cara. Só que os cadernos são diferentes. O cliente mandou pela rede errada — a loja não vê o pagamento.



Por isso na cobrança não se escrevem só o endereço e a quantia, mas também a **rede**.

E há mais uma artimanha de golpistas: eles mandam uma transferência minúscula de um endereço parecido com um que você conhece. A aposta é que você vá copiá-lo do histórico. Confira o endereço inteiro, e não só os primeiros e últimos caracteres.

Pense bem



1. Assine a mensagem "2" com a chave 7 e confira a assinatura com a chave 3.
2. Artiom imprimiu o endereço da loja no cartão de visita. É perigoso?
3. O "suporte da carteira" pede a frase secreta para devolver uma transferência travada. O que fazer?

Respostas: 1) a assinatura é 29, e a conferência dá 2 de novo. 2) Assim não roubam o dinheiro, mas todos verão o que entra nesse endereço. 3) Não informar nada: são golpistas.

O essencial

- O endereço se mostra a todos; a chave privada e a frase secreta, a ninguém.
- A assinatura prova que o dono da chave autorizou a transferência e que ela não foi alterada.
- A rede é tão importante quanto o endereço.

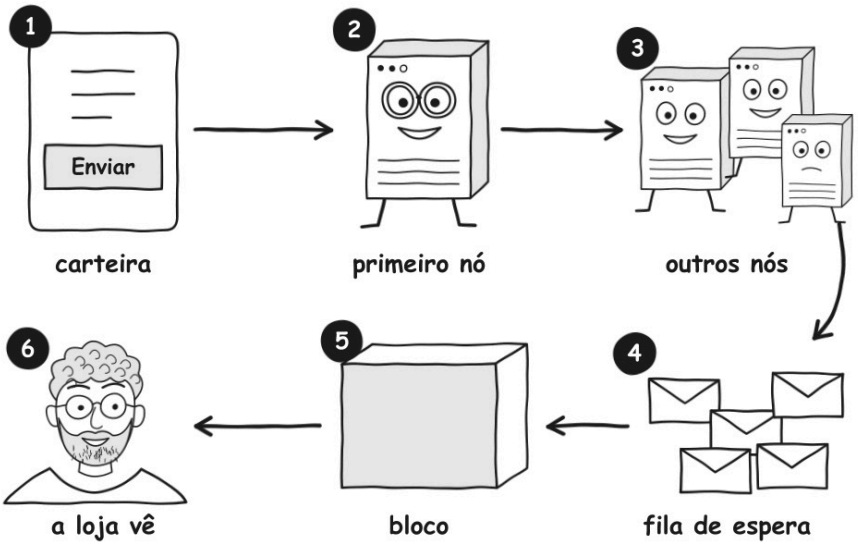
A seguir: capítulo 7. A transferência está assinada. O que acontece com ela entre o botão "Enviar" e o registro no bloco?

Capítulo 7. O caminho da transação: do botão “Enviar” até o bloco

“Já mandei!” — “Aqui não chegou nada”

O cliente apertou o botão e escreve: “Mandei”. Artiom olha — nada de dinheiro. Alguém está enganando? Muito provavelmente, ninguém. A transferência ainda está a caminho.

Entendendo: seis paradas



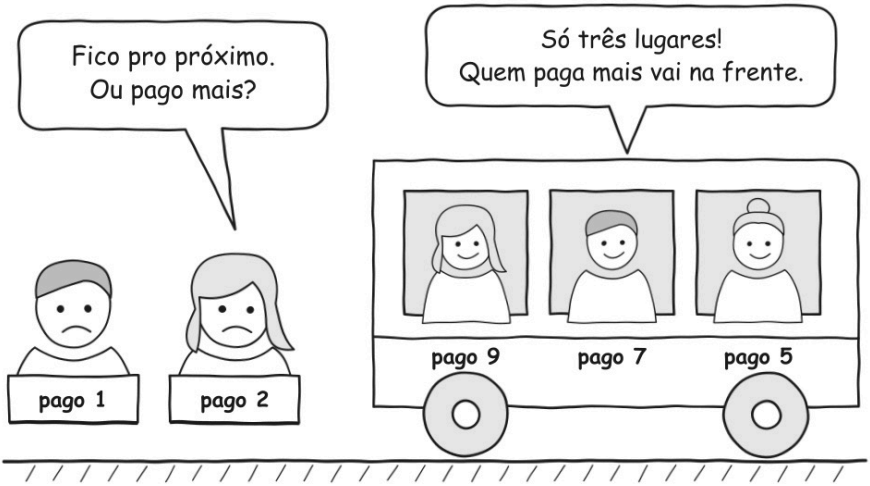
1. A **carteira** redige a transferência e a assina com a chave.
2. O **primeiro nó** recebe a transferência e confere: a assinatura vale? há dinheiro suficiente?
3. Ele a repassa a **outros nós**, que a passam adiante.
4. A transferência espera numa **fila**. Em inglês ela se chama mempool.
5. Quem está na vez de montar a página põe a transferência num **bloco**.
6. A **loja** a enxerga e espera pelas confirmações.

Os nomes e detalhes mudam de rede para rede: em algumas, por exemplo, não há fila geral, e a transferência vai direto a quem está montando o bloco.

Cada transferência tem seu próprio "selo": o **hash da transação**. É como o código de rastreio de uma encomenda: por ele dá para saber onde a transferência está.

A taxa: lugares no ônibus

Num bloco não cabe tudo. Imagine um ônibus de três lugares e cinco pessoas no ponto.



Para entrar no bloco, o remetente paga a **taxa** da rede. Quanto mais gente querendo, mais caro o lugar. Quem ofereceu pouco espera o próximo ônibus e às vezes nem chega a embarcar.

- Quem paga a taxa é o remetente, e ela não vai para a loja.
- Ela não depende da quantia transferida, e sim do "peso" da transferência e da lotação da rede.
- Cada rede calcula esse "peso" do seu jeito: é o assunto da quarta parte do livro.

Veja uma transferência de verdade

É uma transferência comum na rede Ethereum, bloco 26.502.085. Perguntamos por ela ao nó usando o hash da transação:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
       "method":"eth_getTransactionReceipt",
       "params":["HASH_DA_TRANSFERENCIA"]}'
```

A resposta é longa. Eis o principal dela, convertido em números comuns:

Campo	Valor	O que significa
`status`	`0x1`	Executada com sucesso
`blockNumber`	26.502.085	Em que bloco está
`gasUsed`	21.000	O "peso" de uma transferência simples
taxa	0,000043 ETH	"Peso" × preço por unidade
quantia transferida	0,000041 ETH	Vem da própria transferência

Reparou? Desta vez se pagou pelo envio **mais do que se enviou**. Para a taxa, tanto faz quanto dinheiro vai dentro.

E esse número muda o tempo todo. Pergunte você mesmo:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_gasPrice","params":[]}'
```

O nó informará o preço atual de uma unidade de "peso" em wei. Já no nó público de Bitcoin da RpcNode, no mesmo dia, havia quase 83.000 transferências esperando na fila.

O que pode acontecer com a transferência

Estado	O que significa	O que a loja faz
Esperando	Ainda na fila	Esperar, não enviar o produto
No bloco	Registrada, mas com poucas confirmações	Checar o sucesso e esperar
Confirmada	Já há blocos suficientes por cima	Enviar o pedido
Não executada	Está no bloco, mas a transferência não aconteceu	Não contar como pagamento
Caiu da fila ou foi substituída	Não entrou em bloco	Pedir ao cliente que olhe a carteira

Cilada: "se está no bloco, está pago"

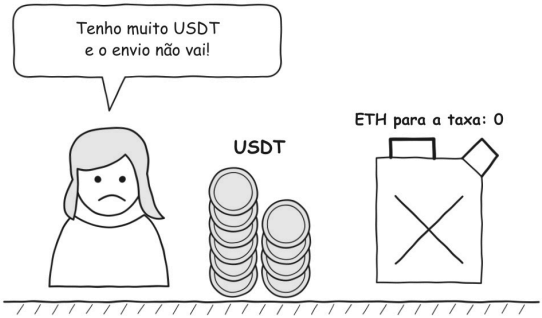
Na Ethereum e em redes parecidas, uma transferência pode entrar num bloco e mesmo assim não ser executada. Por exemplo, o cliente não tinha tokens suficientes. A taxa é cobrada do mesmo jeito, e o dinheiro não chega a você.



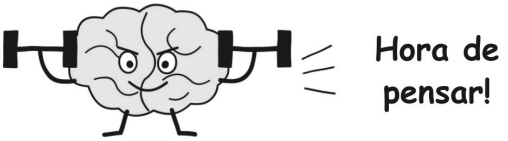
Por isso não se olha só o bloco, mas também o resultado: no exemplo acima é o campo `status`. E ainda se conferem o destinatário, a quantia, a moeda e a rede.

Segunda cilada: o tanque vazio

A taxa é cobrada na moeda nativa da rede. Na Ethereum é o ETH, mesmo que você esteja enviando USDT. Na TRON o pagamento sai em TRX e em recursos especiais da rede.



Pense bem



1. Num bloco cabem três transferências. Na fila, ofereceram pelo lugar 4, 9, 1, 7 e 2. Quem vai no primeiro bloco?
2. O cliente mandou o hash da transação e uma captura de tela com "sucesso". Isso basta para despachar o café?
3. O cliente tem 100 USDT na rede Ethereum e nem um pingo de ETH. O que ele deve fazer?

Respostas: 1) 9, 7 e 4. 2) Não: uma captura pode ser forjada; confira o hash do seu lado. 3) Primeiro conseguir um pouco de ETH para a taxa.

O essencial

- Entre o "mandei" e o "recebi" há um caminho: nó, fila, bloco, confirmações.
- A taxa compra um lugar no bloco e não depende da quantia.
- Pagamento é uma transferência bem-sucedida, na moeda certa, na rede certa, com confirmações suficientes.

A seguir: capítulo 8. Quem monta os blocos e como todos concordam que a cadeia verdadeira é justamente aquela?

Capítulo 8. Como a rede chega a um acordo

Quem escreve a próxima página?

No caderno coletivo não há chefe. Milhares de participantes têm cópias, e cada um pode propor a sua próxima página. Qual delas é a verdadeira?



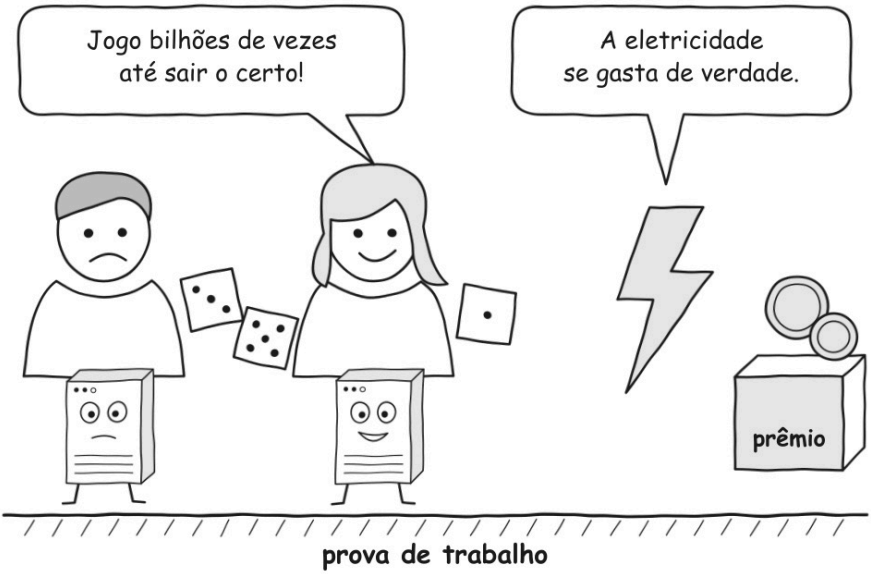
Entendendo: o consenso

As regras pelas quais os participantes chegam a uma resposta comum se chamam **consenso**. A missão delas é fazer com que trapacear saia mais caro do que jogar limpo.

Simplesmente contar votos não dá: um golpista abriria um milhão de programas e ganharia um milhão de votos. Por isso o voto precisa custar alguma coisa. Há duas maneiras principais.

Maneira 1: prova de trabalho

Em inglês, Proof of Work; o processo em si é chamado de **mineração**. Os participantes apostam corrida para resolver um problema que não se resolve com esperteza, só na tentativa. É como jogar dados até sair uma combinação rara.

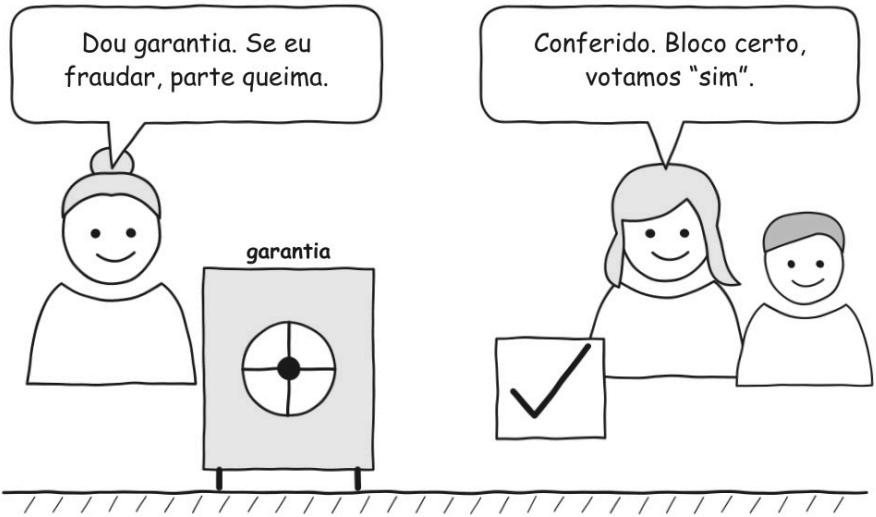


O vencedor acrescenta o bloco e ganha a recompensa: moedas novas e as taxas. Conferir a resposta dele é fácil; achá-la é caro. É assim que o Bitcoin funciona:

- um bloco surge em média a cada 10 minutos;
- a cada 2016 blocos, a dificuldade do problema é ajustada para manter esse ritmo;
- hoje cada bloco paga 3,125 BTC em moedas novas, e mais ou menos a cada quatro anos a recompensa cai pela metade.

Maneira 2: prova de participação

Em inglês, Proof of Stake, ou **staking**. Em vez de eletricidade, o participante deposita moedas como garantia. Ele se chama **validador**. Os validadores se revezam propondo blocos, e os demais conferem e votam.

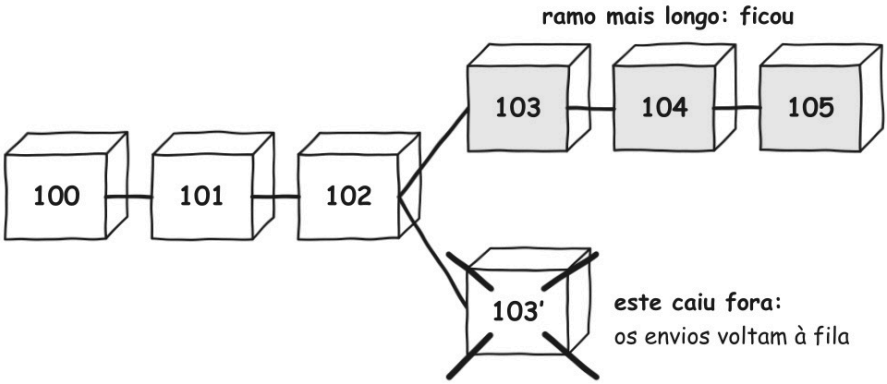


Em caso de infração grave, parte da garantia é queimada; isso se chama slashing. A Ethereum passou para esse método em 15 de setembro de 2022, e o consumo de eletricidade da rede caiu cerca de 99,95%. Para ser validador por conta própria são necessários 32 ETH.

Há variações. Na TRON os blocos são escritos por 27 representantes eleitos, escolhidos pelos donos de moedas. Na BNB Smart Chain também trabalha um círculo limitado de validadores.

Quando surgem duas páginas de uma vez

A rede é grande, e as notícias não viajam instantaneamente. Às vezes aparecem dois blocos com o mesmo número, e a cadeia se bifurca por um tempo. Depois um ramo vence e o outro é descartado. Isso se chama **reorganização**.



A transferência de um bloco descartado não some sem deixar rastro: em geral ela volta para a fila e entra em outro bloco. Mas pode não entrar. É para isso que servem as confirmações do capítulo 5.

No desenho vence o ramo mais longo. Mais exatamente, no Bitcoin se escolhe o ramo em que foi investido mais trabalho; nas redes com staking, o que recebeu mais votos de garantia.

Finalidade: quando já não se desfaz

Finalidade é o momento a partir do qual o bloco não será mais substituído. Em algumas redes ele é anunciado explicitamente; em outras, a certeza apenas cresce a cada bloco.

Rede	Mais ou menos quando o registro é tido como definitivo
Bitcoin	Não há marca explícita. Para quantias altas recomendam 6 confirmações, cerca de uma hora
Ethereum	Normalmente 13–15 minutos, às vezes mais
TRON	Normalmente cerca de um minuto
BNB Smart Chain	Cerca de um segundo ou menos
Solana	Depois da atualização de 2026, projetada para frações de segundo
TON	Cerca de um segundo

Dados de 27 de novembro de 2026, segundo a documentação das redes. Esses números mudam com as atualizações.

Na prática: peça três graus de "frescor"

Aos nós da Ethereum não se pede só o bloco mais novo. A palavra `latest` do capítulo 1 tem duas parentes: `safe` (já seguro) e `finalized` (definitivo).

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
       "method":"eth_getBlockByNumber",
       "params":["finalized",false]}'
```

Troque `finalized` por `safe` e `latest` e compare os números. Quando este capítulo estava sendo escrito, o nó respondeu assim:

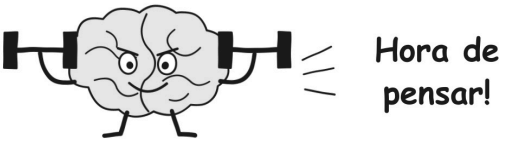
O que se pediu	Número do bloco	Atraso em relação ao mais novo
`latest`	26.502.104	—
`safe`	26.502.070	34 blocos, cerca de 7 minutos
`finalized`	26.502.038	66 blocos, cerca de 13 minutos

Cilada: "em todas as redes é igual"

"Espero uma confirmação e pronto" é uma regra perigosa. Uma confirmação significa graus de segurança bem diferentes em redes diferentes. Decida pela rede e pela quantia:

- quanto mais caro o pedido, mais se espera;
- para quantias altas, aguarde a finalidade, onde ela existe;
- não entregue o que não dá para reaver com base numa transferência de um bloco recém-saído.

Pense bem



1. Por que no caderno não se pode votar pelo princípio “um programa, um voto”?
2. Os blocos do Bitcoin saem a cada 10 minutos. Quanto se espera, em média, por 6 confirmações? E por 3?
3. Um cliente leva um café para viagem e outro leva 200 kg de grãos. Com qual deles Artiom deve esperar a finalidade?

Respostas: 1) programas não custam nada, dá para rodar quantos se quiser. 2) Cerca de uma hora e cerca de meia hora. 3) Com o que leva 200 kg.

O essencial

- Consenso são as regras pelas quais os participantes, sem chefe, escolhem uma única cadeia.
- Um voto custa trabalho (mineração) ou garantia (staking).
- Blocos recentes podem ser substituídos; o definitivo vem depois, e cada rede tem o seu jeito.

A seguir: capítulo 9. Moedas, tokens e contratos inteligentes: por que USDT não é a mesma coisa que ETH.

Capítulo 9. Moedas, tokens e contratos inteligentes

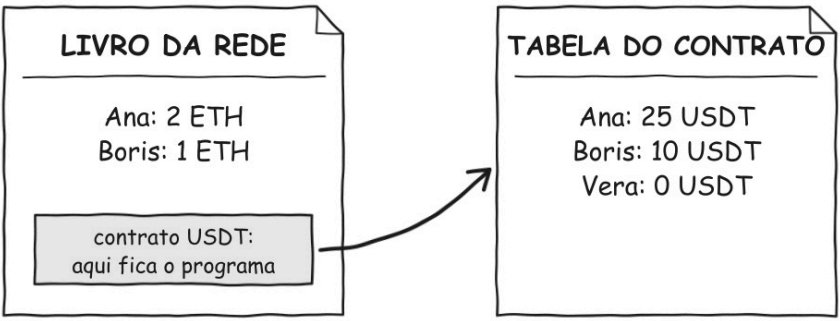
"Eu paguei, e aí não aparece nada"

O cliente mandou 25 USDT na rede Ethereum. Artiom pergunta ao nó, como no capítulo 1, e o saldo não mudou. O dinheiro sumiu? Não. Só era preciso perguntar de outro jeito.

Entendendo: moeda e token

A **moeda nativa** é aquela que a própria rede controla: ETH na Ethereum, TRX na TRON, BNB na BNB Smart Chain. É com ela que se paga a taxa.

Um **token** são registros mantidos por um programa que vive dentro da rede. O USDT é um token.



	Moeda nativa	Token
Quem mantém os registros	a própria rede	um contrato inteligente
Exemplo na Ethereum	ETH	USDT, USDC
Com que se paga a taxa	com ela mesma	com a moeda nativa da rede
Como perguntar o saldo	`eth_getBalance`	`eth_call` ao contrato

Cada rede organiza os tokens à sua maneira. Aqui olhamos para a Ethereum e as redes parecidas com ela.

Contrato inteligente: a máquina sem vendedor

Um **contrato inteligente** é um programa gravado na blockchain. Ele lembra uma máquina de venda automática: você coloca, aperta e recebe o resultado conforme regras definidas de antemão. Todos os nós o executam do mesmo jeito.



- Não é um contrato com assinaturas, é um programa mesmo.
- Em geral não dá para alterá-lo, mas alguns autores guardam para si essa possibilidade.
- O programa pode ter um "dono" com poderes especiais.
- Um erro no programa pode custar dinheiro, e não há quem desfaça.

Na prática: o saldo de um token

O método `eth_call` significa "pergunte ao contrato sem mudar nada". A rede não cobra taxa por essa pergunta: nada é escrito no caderno. Vamos perguntar ao contrato do USDT quanto USDT há no endereço dele mesmo:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0x70a08231000000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"},
  "latest"]}'
```

`to` é o contrato a quem perguntamos. Já o `data` é a colagem de três pedaços:

Pedaço	O que significa
`0x70a08231`	qual é a pergunta: "qual o saldo?" (`balanceOf`)
24 zeros	enchimento até o comprimento certo
40 caracteres	de quem é o saldo: o endereço sem o `0x`

Quando este capítulo estava sendo escrito, o nó respondeu assim:

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000000
014d0a9aa6c5"}
```

Depois dos zeros vem `14d0a9aa6c5`, que é 1.430.402.016.965. O USDT na Ethereum tem seis casas decimais, então no endereço do contrato há **1.430.402,016965 USDT**.

De onde vieram? Pelo visto, são principalmente transferências para o lugar errado: alguém colou o endereço do contrato no lugar do endereço do destinatário. O remetente não tem um jeito comum de recuperá-las.

Onde fica a vírgula

O contrato só guarda números inteiros. Quantas casas separar quem diz é uma propriedade à parte: `decimals`.

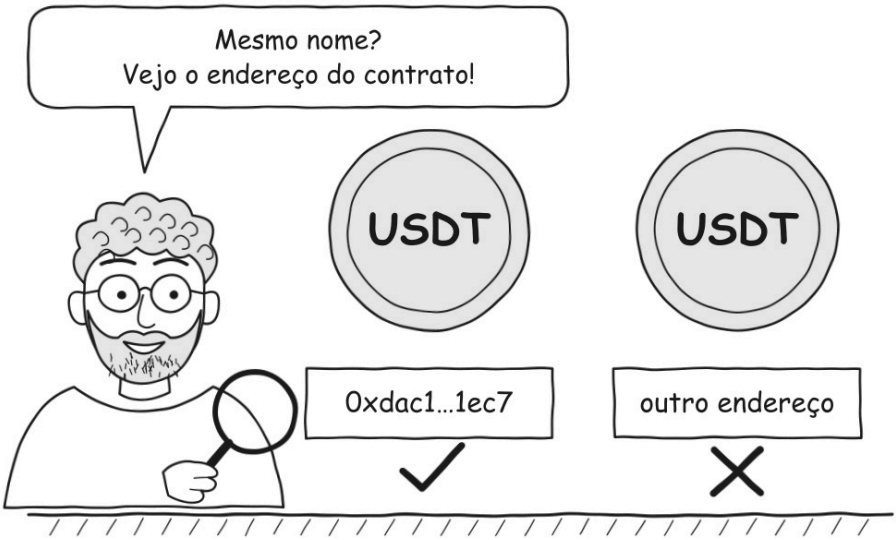
O quê	Casas decimais
ETH, a moeda nativa	18
USDT na Ethereum	6
token "Tether USD" (USDT) na BNB Smart Chain	18

O nome é o mesmo, mas as casas são o triplo. Confundiu — errou um trilhão de vezes. Além disso, o endereço desse token da BNB Smart Chain não está entre os contratos de USDT no site da empresa Tether: nome igual não significa emissão igual.

Verificado em 27 de novembro de 2026 com requisições a nós públicos e pela página tether.to/en/supported-protocols.

Cilada: o token falso

Qualquer um pode chamar o próprio token de "USDT". O nome não prova nada. O USDT verdadeiro se reconhece por um par: **a rede e o endereço do contrato**.



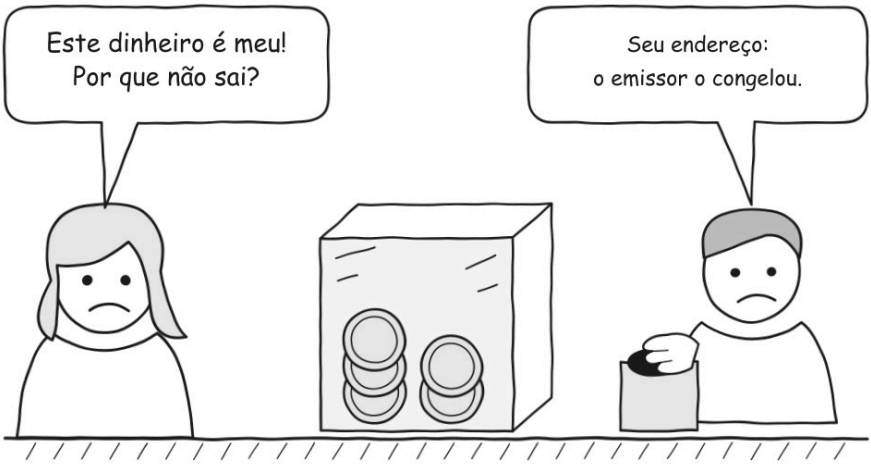
Por isso a loja confere tudo de uma vez:

- a rede;
- o endereço do contrato do token;
- o destinatário;
- a quantia, com o número certo de casas;
- o sucesso e as confirmações.

Stablecoins: práticas, mas têm dono

Uma **stablecoin** procura se manter em torno de um dólar. No caso do USDT, isso é uma promessa da empresa emissora, não uma lei da natureza.

O emissor também tem poder: os termos de uso da Tether preveem expressamente o direito de congelar tokens (versão de 26 de fevereiro de 2026, tether.to/en/legal).

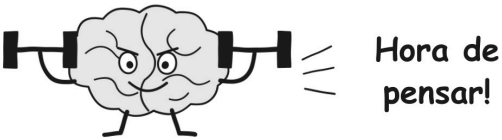


Segunda cilada: "autorize o gasto"

Um site pode pedir que você assine uma **autorização** (`approve`): "este contrato pode gastar os meus tokens". Se você autorizar um golpista sem limite de quantia, ele os levará depois, sem nova assinatura. Ele nem precisa saber a frase secreta.

> **Regra:** se não entende o que está assinando, não assine.

Pense bem



1. Para o USDT na Ethereum, o nó devolveu o número 25.000.000. Quantos USDT são?
2. O cliente mandou um token chamado USDT, mas com outro endereço de contrato. O pedido está pago?
3. Artiom tem 100 USDT na Ethereum e 0 ETH. Ele consegue mandá-los ao fornecedor?

Respostas: 1) 25. 2) Não. 3) Não: não há com que pagar a taxa, ver capítulo 7.

O essencial

- A moeda é controlada pela rede; o token, por um contrato inteligente.
- O token se reconhece pela rede e pelo endereço do contrato, e não pelo nome.
- A quantia se lê levando em conta as casas decimais.

Fim da segunda parte. Artiom já sabe o que são bloco, chave, taxa e token. Mas o nó continua sendo uma caixa-preta para ele. **A seguir:** parte 3, capítulo 10. Os tipos de nó.

Capítulo 10. Os tipos de nó

“E quanto havia lá naquela época?”

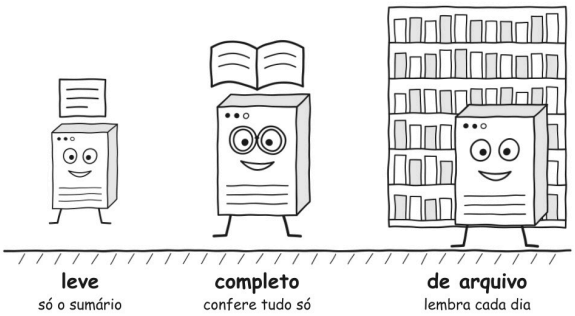
Artiom está fechando a contabilidade e quer saber quanto havia no endereço da loja alguns meses atrás. O nó responde muito bem sobre hoje, mas sobre o passado se recusa. A blockchain não guarda tudo?

Entendendo: registros e saldos

Aqui se escondem duas coisas diferentes:

- **Histórico**: os próprios registros, ou seja, blocos e transferências.
- **Estado**: quem possui o quê num momento específico, isto é, saldos e dados dos contratos.

É como no banco: o extrato das operações existe, mas o saldo de cada minuto do ano passado não está guardado à parte. Seria preciso recalculá-lo. Os nós se diferenciam por quanto disso tudo eles lembram.

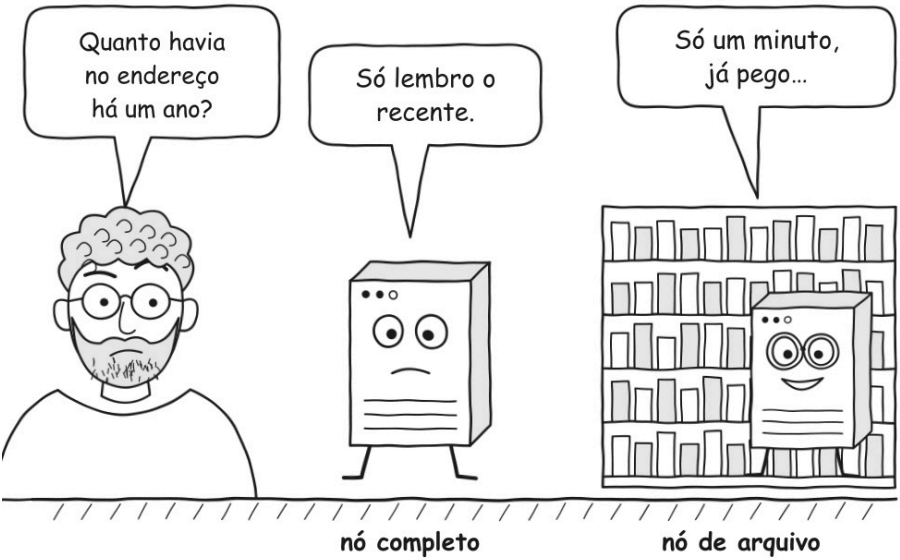


Os três tipos principais

O **nó completo** confere os blocos pelas regras da rede e guarda o estado recente. Na Ethereum isso costuma ser cerca dos últimos 128 blocos, ou seja, uns 25 minutos. A profundidade exata depende do programa e das configurações.

O **nó de arquivo** confere tudo desde o começo e não apaga o que baixou. Ele lembra o estado em qualquer bloco, mas exige muito mais disco.

O **nó leve** baixa só os cabeçalhos dos blocos e pede os detalhes a outros. A ideia é boa, mas na Ethereum, em 27 de novembro de 2026, não há um nó leve pronto para o uso diário.



O nó podado: conferiu e jogou fora

O Bitcoin tem um modo de economizar disco. O **nó podado** (pruned) confere tudo e depois apaga os blocos antigos. Ele continua sendo um nó completo de pleno direito, só não consegue entregar o que apagou. Segundo os desenvolvedores do Bitcoin Core, até 10 GB lhe bastam.

A Ethereum também ganhou essa possibilidade: desde julho de 2025, os programas de nó conseguem não guardar o conteúdo dos blocos mais antigos, os que surgiram antes da passagem para o staking, em setembro de 2022. É uma configuração, e não está ligada em toda parte. Por isso até um nó "completo" pode não lembrar registros muito antigos.

Na prática: pergunte sobre o passado

Vamos pedir o saldo de um mesmo endereço duas vezes. Primeiro, o de agora:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance",
  "params":["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":"0x2a"}
```

`0x2a` são 42 wei, uma migalha. Agora troque `"latest"` por `"0xe4e1c0"`, ou seja, pelo bloco número 15.000.000. O nó público responde assim:

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
 "message":"archive data is not available on the public endpoint,
 use a personal endpoint"}}
```

Em português: “dados de arquivo não estão disponíveis no endpoint público; use um endpoint pessoal”. Não é defeito, é o limite do serviço gratuito. Na RpcNode, o acesso de arquivo está nos planos pagos; no Free ele não existe (capítulo 3).

Quanto isso pesa

Nó	Disco, aproximadamente
Bitcoin, podado	a partir de 10 GB
Bitcoin, completo	cerca de 900 GB
Ethereum, completo	cerca de 2 TB ou mais
Ethereum, de arquivo	de 2 a 14 TB ou mais, conforme o programa

Dados da Ethereum tirados da página do ethereum.org sobre como rodar nós, em 27 de novembro de 2026.

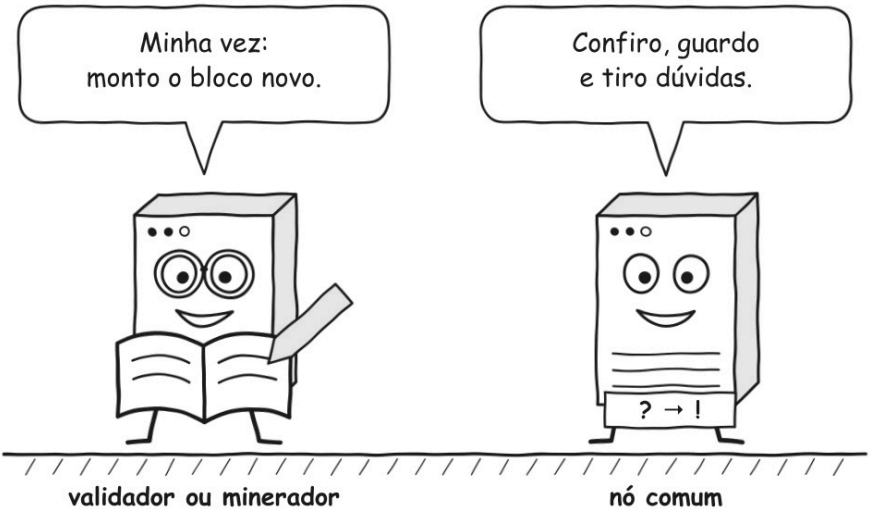
A segunda linha você pode conferir por conta própria: o nó do Bitcoin informa honestamente o próprio tamanho.

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getblockchaininfo","params":[]}'
```

Na resposta comprida, ache duas linhas. Quando este capítulo estava sendo escrito, havia ali: `"size_on_disk":896027659764` (cerca de 896 GB, contando os arquivos auxiliares) e `"pruned":false` — o nó não é podado.

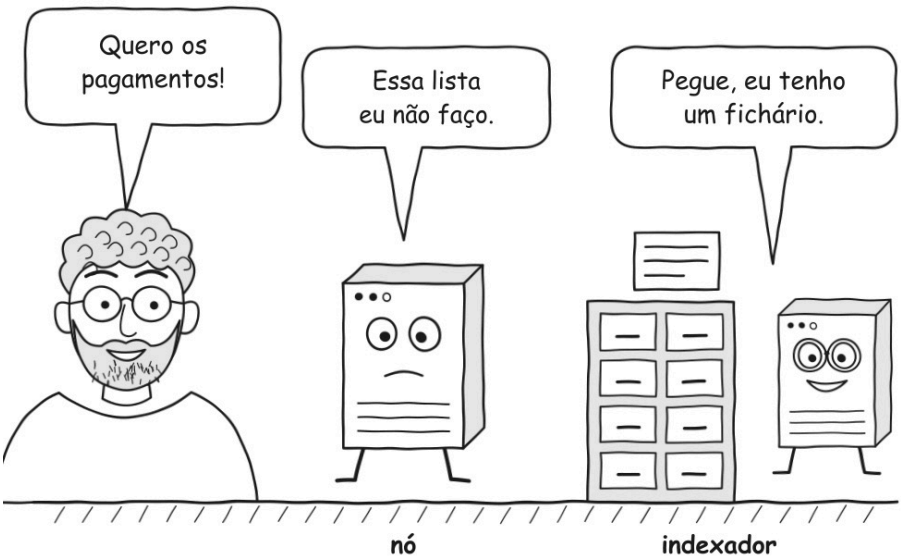
E quem escreve os blocos?

Esse é um papel à parte. O **validador** ou **minerador** participa da criação dos blocos (capítulo 8). Para conferir pagamentos, não é preciso ser um: basta um nó que lê, confere e responde.



Cilada: "vou pedir ao nó todos os meus pagamentos"

No conjunto padrão de perguntas a um nó da Ethereum não existe "mostre todas as transferências deste endereço". Nem o nó de arquivo sabe fazer isso. O método `eth_getTransactionCount` informa apenas o número de transações **enviadas** pelo endereço.

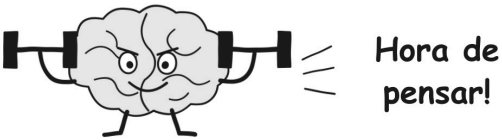


Para listas assim é preciso um **indexador**: um programa que lê os blocos e organiza os registros num fichário. É assim que funcionam também os sites exploradores de blockchain. Para a loja, é mais simples anotar os próprios pagamentos desde o começo e acompanhar os novos, assunto do capítulo 26.

O que serve para a sua tarefa

Tarefa	O que serve
Conferir um pagamento recente	um nó completo comum
Saber o saldo numa data passada	um nó de arquivo
Obter a lista de todas as transferências de um endereço	um indexador
Participar da criação de blocos	um validador ou minerador

Pense bem



1. Artiom quer fazer um gráfico do saldo da loja ao longo de um ano, com um ponto por dia. Que nó é necessário?
2. Um nó podado de Bitcoin apagou os blocos antigos. Dá para confiar a ele a conferência dos novos?
3. Para mostrar a um cliente todos os pedidos anteriores dele com os pagamentos, o que é preciso: um nó de arquivo ou um banco de dados da própria loja?

Respostas: 1) de arquivo. 2) Sim, ele conferiu tudo antes de apagar. 3) Um banco de dados próprio: o nó não mantém listas assim.

O essencial

- Histórico são os registros; estado são os saldos num momento específico.
- O nó completo lembra o estado recente; o de arquivo, qualquer um.
- O nó não entrega a lista de transferências de um endereço: para isso é preciso um indexador.

A seguir: capítulo 11. Já fizemos uma dezena de perguntas ao nó. Hora de entender em que língua: JSON-RPC, REST, gRPC e WebSocket.

Capítulo 11. Como conversar com um nó

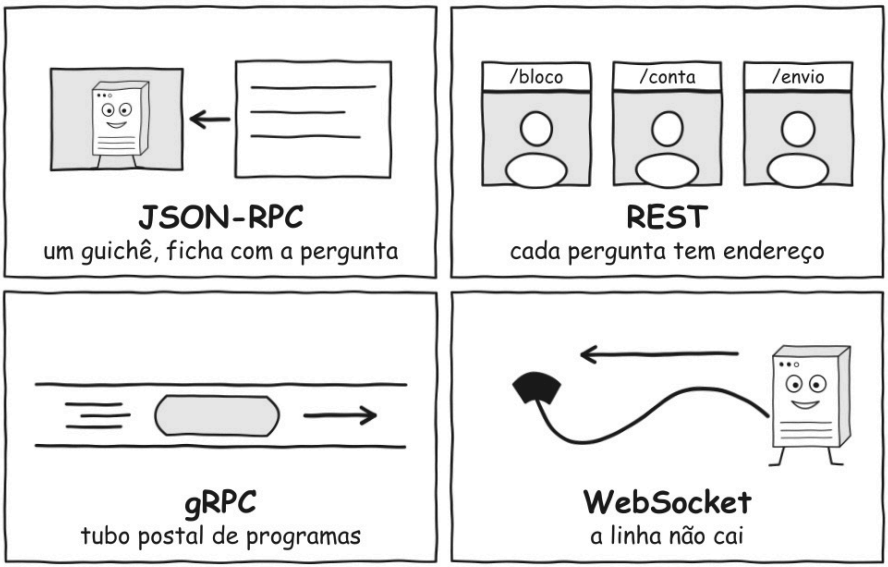
Que letras são essas nas requisições?

Artiom já copiou comandos deste livro uma dezena de vezes, e eles funcionaram. Hora de entender o que está escrito neles: o que é RPC, para que tantas chaves e parênteses e por que na documentação de redes diferentes as requisições têm caras diferentes.

Entendendo: o que são API e RPC

Uma **API** é um guichê pelo qual um programa se dirige a outro seguindo regras conhecidas de antemão. **RPC** (chamada remota de procedimento) é um jeito de pedir a um programa em outro computador que faça algo, como se ele estivesse ao lado. O nó aceita pedidos assim.

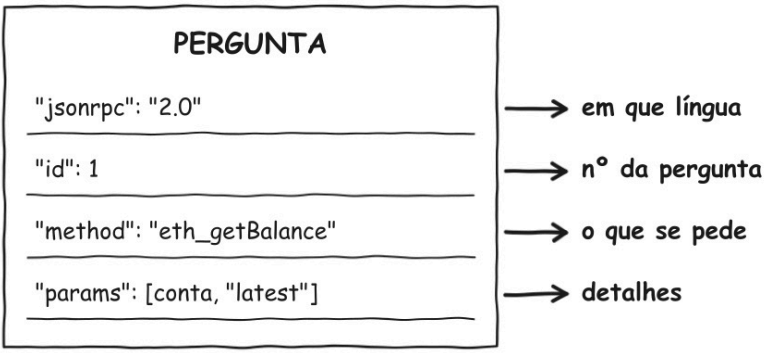
Há várias formas de conversar:



Forma	Como é	Onde aparece
JSON-RPC	Um só endereço; a requisição diz o método e os parâmetros	Ethereum e parecidas, Solana, Bitcoin
REST	Cada pergunta tem seu caminho no endereço	TRON, TON, Aptos
gRPC	Mensagens binárias compactas num esquema rígido	Troca entre programas, por exemplo na Sui
WebSocket	Uma conexão que não se fecha	Quando o próprio nó deve avisar as novidades

O WebSocket não é uma língua, e sim uma **linha de comunicação**: na Ethereum, passam por ela as mesmas mensagens JSON-RPC. Algumas redes têm várias formas ao mesmo tempo.

JSON-RPC peça por peça



JSON é um jeito simples de escrever dados: chaves, nomes entre aspas e valores. Eis o comando do capítulo 3, destrinchado linha a linha:

Pedaço do comando	O que significa
`curl`	o programa que envia a requisição pelo terminal
`https://…`	o endereço do endpoint
`-H '…'`	a etiqueta "aqui dentro vai JSON"
`-d '{…}'`	a pergunta em si

Na resposta se olham o `result` (o que deu certo) e o `error` (o que não deu). Alguns nós mandam só um desses campos; outros, os dois, e aí o vazio vem como `null`. O `id` da resposta é o mesmo da pergunta: é assim que se faz o par.

Várias perguntas de uma vez

Dá para juntar perguntas numa lista entre colchetes. Isso se chama **lote** (batch):

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '[{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]},
       {"jsonrpc":"2.0","id":2,"method":"eth_chainId","params":[]}]'
```

```
[{"jsonrpc":"2.0","id":1,"result":"0x1946420"},
 {"jsonrpc":"2.0","id":2,"result":"0x1"}]
```

A primeira resposta é o número de bloco 26.502.176; a segunda, o número da rede: o da Ethereum é 1. Pelas regras do JSON-RPC, as respostas de um lote podem vir em qualquer ordem, por isso olhe o `id`. E lembre: o lote economiza viagens, mas não limite. Na RpcNode, um lote de dez perguntas conta como dez requisições, e é aceito ou recusado por inteiro.

E agora o REST

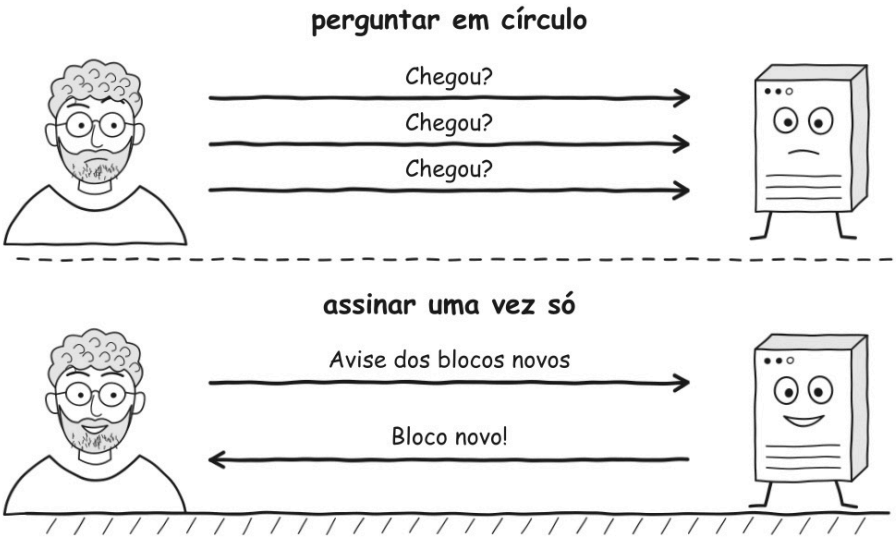
Aqui a pergunta vai embutida no próprio endereço. Vamos pedir à rede TRON o bloco mais recente:

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/getnowblock
```

Chega uma resposta comprida sobre um bloco com transferências. Quando este capítulo estava sendo escrito, era o bloco número 88.351.889, com 194 transações. Outra pergunta, outro caminho depois do endereço.

WebSocket: em vez de perguntar, escutar

Perguntar "chegou?" a cada segundo é desperdício. Numa linha aberta dá para assinar uma vez só, e o próprio nó vai mandando as novidades.



Os nós públicos da RpcNode não respondem por WebSocket: é preciso um endpoint pessoal. O endereço dele começa com `wss://`. Veja um esboço em JavaScript:

```
const ws = new WebSocket('wss://SEU_ENDPOINT')

ws.onopen = () => ws.send(JSON.stringify({
  jsonrpc: '2.0', id: 1,
  method: 'eth_subscribe', params: ['newHeads'],
}))

ws.onmessage = (event) => console.log(event.data)
ws.onclose = () => console.log('a linha caiu')
```

Linhas caem. O programa precisa saber reconectar, assinar de novo e conferir o que perdeu. As assinaturas também consomem créditos.

gRPC

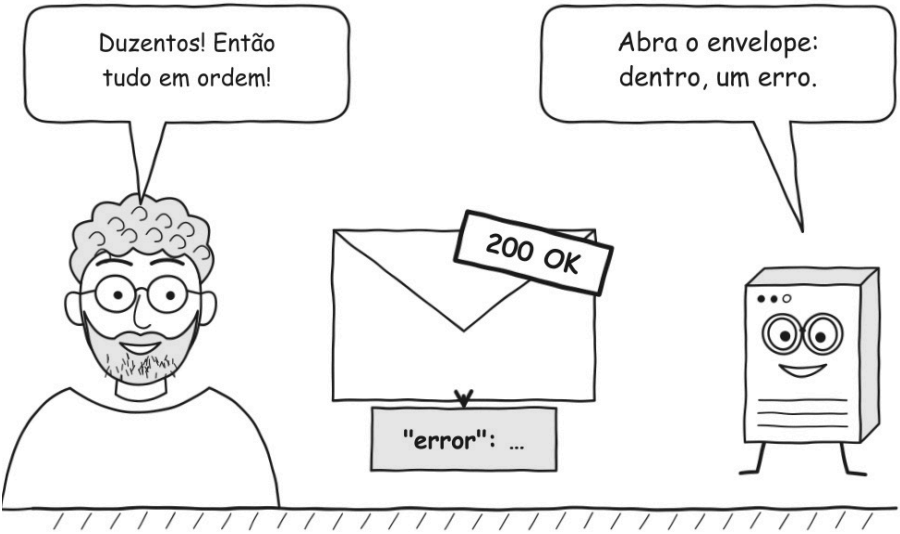
É uma forma para programas que fazem questão de velocidade e formato rígido. As mensagens não se leem a olho, e seus campos são descritos de antemão num arquivo à parte. Uma loja iniciante em geral não precisa dele. Na RpcNode, nem todas as redes têm gRPC: veja a lista de transportes do endpoint.

Cilada: "200 quer dizer que está tudo bem"

200 é o código de entrega: o envelope chegou. Mas dentro pode haver um erro. Vamos pedir o saldo do endereço "hello":

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance","params":["hello","latest"]}'
```

O código da resposta é 200, mas dentro vem um `error` de número `-32602`: parâmetros inválidos.



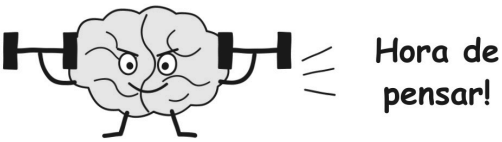
Por isso o programa sempre confere tanto o código quanto o campo `error`.

Erros frequentes

Código	O que houve	O que fazer
`-32700`	Não dá para interpretar o texto da requisição	Conferir chaves e aspas
`-32600`	A requisição está fora das regras	Conferir os campos
`-32601`	Método não encontrado ou indisponível	Conferir o nome e o plano
`-32602`	Parâmetros inválidos	Conferir os valores
`-32005` (HTTP 429)	RpcNode: frequência alta demais	Esperar e perguntar com menos frequência
`-32006` (HTTP 402)	RpcNode: créditos insuficientes	Conferir o consumo e o plano

Os quatro primeiros códigos são do padrão JSON-RPC 2.0; os dois últimos são códigos da RpcNode.

Pense bem



1. Na primeira requisição do lote, escreva `eth_blockNumbr` no lugar de `eth_blockNumber`. Que código vem? A segunda pergunta, correta, é respondida?
2. O que é melhor para um painel de "último bloco" que deve se atualizar sozinho: perguntar a cada segundo ou assinar?
3. O programa recebeu HTTP 200 e na mesma hora marcou o pedido como pago. O que está errado?

Respostas: 1) `-32601`, e um só para todos: a RpcNode recusa o lote inteiro, e a segunda pergunta também fica sem resposta. 2) Assinar. 3) Não foram conferidos nem o campo `error` nem a transferência em si (capítulo 7).

O essencial

- O JSON-RPC diz o método dentro da requisição; o REST, no endereço.
- O WebSocket mantém a linha aberta para que o próprio nó avise as novidades.
- Sucesso não é o código 200, e sim uma resposta sem `error` e com o que você esperava.

A seguir: capítulo 12. Vamos espiar o outro lado do guichê: como vive um nó próprio, o que é sincronização e por que ela quebra.

Capítulo 12. O nó próprio por dentro: instalação, sincronização e panes

“Isso se resolve numa noite!”

Um programador conhecido tenta convencer Artiom: “Baixou, rodou, e você tem seu próprio nó”. Rodar realmente não demora. O que demora é o que começa depois.

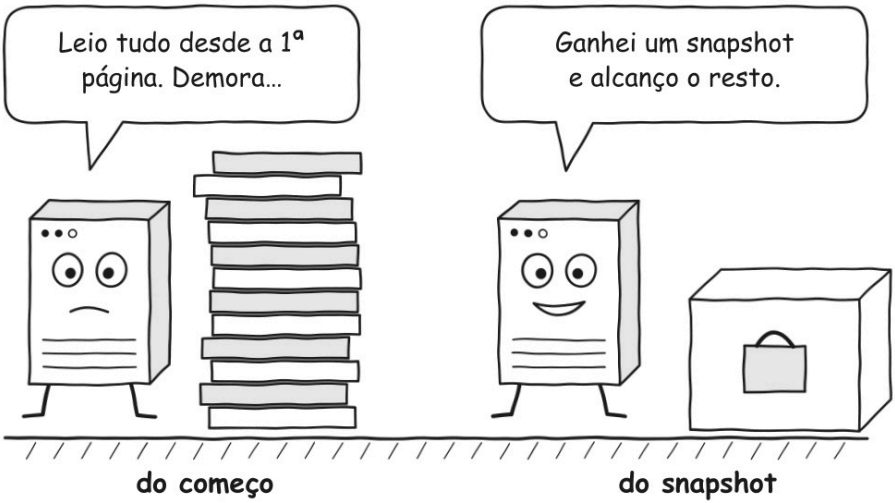
Entendendo: a vida de um nó



1. **Servidor.** Os requisitos vimos no capítulo 2.
2. **Programa.** Cada rede tem o seu, e muitas vezes há vários à escolha. Na Ethereum são necessários dois ao mesmo tempo: o **cliente de execução** (executa transferências e contratos) e o **cliente de consenso** (acompanha qual cadeia é a verdadeira).
3. **Sincronização.** O nó alcança a rede.
4. **Vigilância.** Ficou para trás? Há espaço suficiente?
5. **Atualizações.** A rede muda as regras — o programa é trocado com antecedência.

Sincronização: alcançar a rede

Um nó novo não sabe nada. Enquanto não alcançar os outros, não adianta perguntar a ele sobre um saldo: ele vai responder com o passado.



- **Desde o começo.** O nó confere cada bloco por conta própria. Seguro e demorado. A primeira carga do Bitcoin tem cerca de 740 GB.
- **A partir de um snapshot.** Pega-se uma cópia pronta da base e se alcança o restante. É mais rápido, mas você confia em quem fez o snapshot.
- **Caminhos intermediários.** Os programas da Ethereum costumam baixar o estado recente e conferi-lo com os cabeçalhos dos blocos.

Em tempo, vai de algumas horas a muitos dias: depende da rede, do método, do disco e da conexão.

Na prática: uma configuração em duas linhas

O começo do arquivo de configuração `bitcoin.conf` pode ser assim:

```
server=1
prune=5000
```

Linha	O que significa
`server=1`	aceitar perguntas por RPC; enquanto não se liberar outra coisa, só deste mesmo computador
`prune=5000`	nó podado (capítulo 10): apagar blocos antigos, mantendo uns 5 GB deles; os dados auxiliares ocupam espaço além disso

Isto não é um manual completo. Antes de instalar, leia a documentação do seu programa.

Como saber se o nó já alcançou a rede

Dá para perguntar diretamente ao nó. Na Ethereum:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_syncing","params":[]}'
```

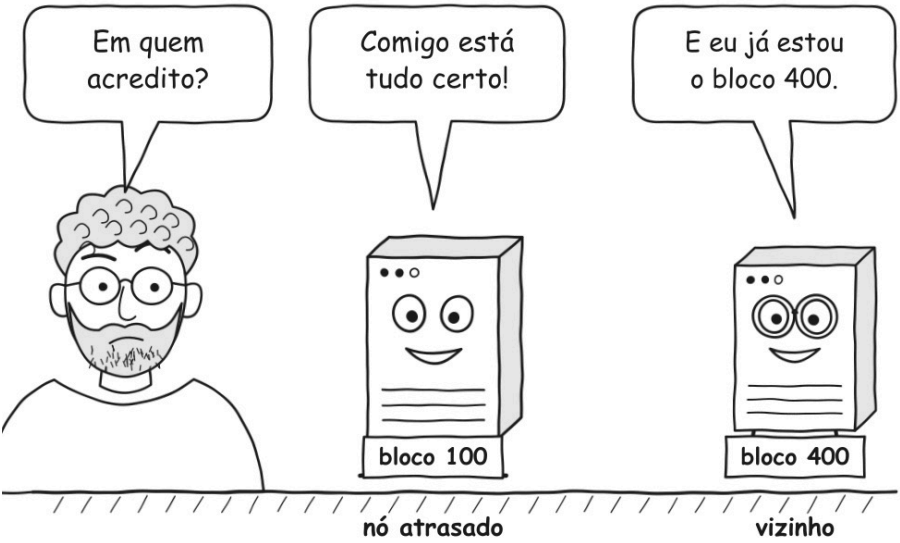
```
{"jsonrpc":"2.0","id":1,"result":false}
```

`false` quer dizer "neste momento não estou correndo atrás". Enquanto o nó ainda está alcançando a rede, vêm no lugar uns números: onde ele está e aonde vai. No Bitcoin se olha a resposta do `getblockchaininfo`, do capítulo 10:

Campo	Valor na verificação	O que significa
`blocks`	977.715	quantos blocos foram conferidos
`headers`	977.715	de quantos blocos se tem notícia
`initialblockdownload`	`false`	a primeira carga já ficou para trás

Cilada: "ele mesmo disse que está tudo bem"

O nó julga pelo que sabe. Se perdeu o contato com os outros, pode nem desconfiar de que ficou para trás.



Por isso se conferem duas coisas: o **frescor do último bloco** e a **comparação com outra fonte**. O frescor é fácil de saber. Peça a hora do último bloco:

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber",
  "params":["latest",false]}' | grep -o '"timestamp":"0x[0-9a-f]*"'
```

```
"timestamp":"0x6b08aacb"
```

São os segundos passados desde 1970. Subtraia-os da hora atual (coloque o seu número):

```
echo $(( $(date +%s) - 0x6b08aacb ))
```

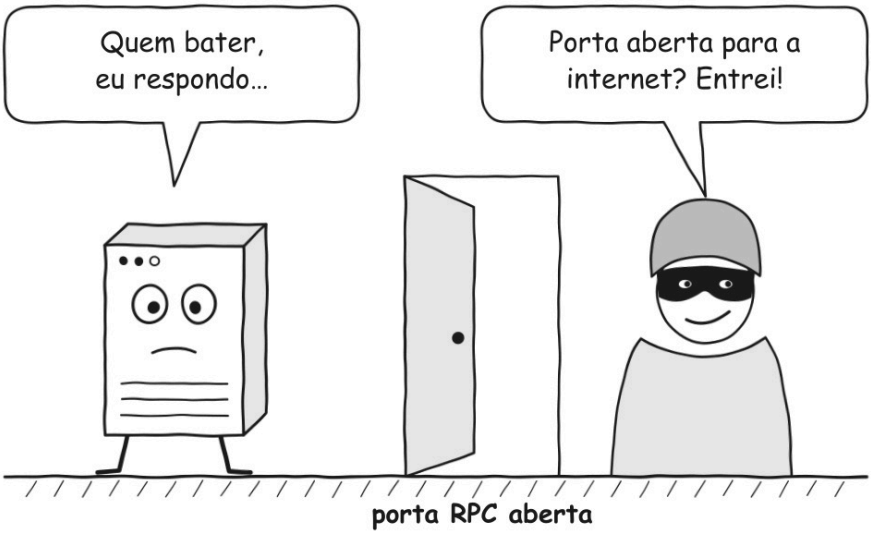
O resultado é a idade do bloco em segundos. Quando este capítulo estava sendo escrito, eram 7 segundos: para a Ethereum, com um bloco a cada 12 segundos, é o normal. Mas, se no seu nó derem milhares, ele ficou para trás. Lembra a pergunta do capítulo 1 sobre o nó uma hora atrasado?

O que quebra

O que se vê	Causa frequente	O que fazer
O nó parou	Acabou o espaço em disco	Vigiar o espaço e ampliar com antecedência
Vive atrasado	Disco ou conexão fracos, poucos vizinhos	Verificar a máquina e a rede
“Empacou” depois de uma atualização da rede	Versão antiga do programa	Atualizar antes da data anunciada
Não sobe depois de uma queda de energia	A base se corrompeu	Restaurar de uma cópia ou sincronizar de novo
Fica lento sem motivo aparente	Está sobrecarregado com requisições de estranhos	Fechar o acesso a quem é de fora

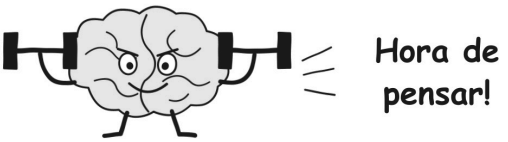
Segunda cilada: a porta aberta

O Bitcoin Core e o Geth, de fábrica, só aceitam perguntas RPC do próprio computador. Não é à toa. Ao abrir a porta para a internet inteira, você deixa entrar qualquer um e, se houver uma carteira no nó, arrisca o dinheiro.



- Libere o acesso só aos seus endereços (firewall).
- Coloque na frente do nó uma verificação de chave.
- Não guarde uma carteira com dinheiro no nó que responde ao site.

Pense bem



1. O `eth_syncing` responde `false`, mas o último bloco tem 40 minutos. O nó está saudável?
2. O disco tem 2000 GB, 1900 GB estão ocupados e entram 5 GB por dia. Em quantos dias tudo para?
3. O site fica em outro servidor. Pode abrir a porta RPC para todos, "para o site conseguir perguntar"?

Respostas: 1) Não. 2) Em no máximo 20 dias, e o conserto começa hoje. 3) Não: libere só o endereço do site.

O essencial

- Ligar o nó é o começo do trabalho, não o fim.
- A saúde se verifica pelo frescor do bloco e pela comparação com outra fonte.
- A porta RPC não se abre para a internet inteira.

A seguir: capítulo 13. Se nada disso é o seu passatempo favorito: como escolher um provedor e o que perguntar a ele.

Capítulo 13. Como escolher um provedor

Todos prometem o melhor

Artiom decidiu: nó próprio não é para ele. Abre os sites dos provedores e lá é um festival de pontos de exclamação.



Entendendo: primeiro, olhe para si

Comparar provedores sem conhecer as próprias necessidades não adianta. Primeiro responda:

- De que redes você precisa? Precisa de redes de teste?
- Quantas requisições por segundo e por mês?
- Precisa de histórico antigo (arquivo)?
- Precisa saber dos eventos na hora (WebSocket, notificações)?

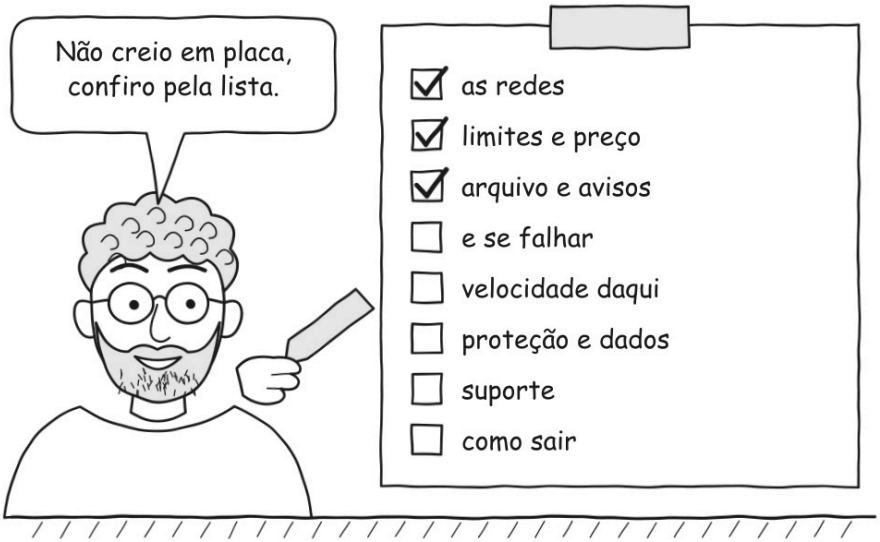
Calcule as suas requisições

Artiom tem 20 pedidos esperando pagamento, e o site confere cada um a cada 10 segundos:

> 20 pedidos × 6 vezes por minuto = 120 requisições por minuto, ou 2 por segundo.
> Em 30 dias: 5.184.000 requisições.

Depois multiplique pelo preço do método em créditos: ele varia de método para método. E agora pense se é preciso perguntar tantas vezes. A cada 30 segundos, são três vezes menos requisições.

A lista de perguntas



Pergunte	Sinal de alerta
Minhas redes estão lá? E os métodos de que preciso nelas?	Lista enorme de redes sem lista de métodos
Limites: requisições por segundo e no total; o que acontece ao ultrapassar?	"Ilimitado" sem números; cobranças extras nebulosas
Arquivo, WebSocket, gRPC: em que plano e para quais redes?	"Temos tudo", mas sem detalhes
Falha de um nó: vão trocar para outro? Onde ficam os servidores?	Sem resposta
Disponibilidade: o que prometem, como medem, o que acontece se descumprirem?	Uma porcentagem sem período e sem condições
Proteção: dá para restringir endereços e sites? O que registram sobre as requisições?	A chave funciona na mão de qualquer um
Suporte: para onde escrever e em que língua respondem?	Só um formulário anônimo
Saída: os métodos são padronizados? Dá para pagar por mês?	Prazo obrigatório longo

O que a porcentagem esconde

“99% de disponibilidade” soa quase como “sempre”. Vamos calcular quanto tempo parado isso permite em 30 dias:

Prometido	Pode ficar fora do ar
99%	7 horas e 12 minutos
99,9%	cerca de 43 minutos
99,99%	cerca de 4 minutos

E pergunte **o que exatamente** conta como funcionando: o site do provedor ou o seu endpoint na sua rede.

Na prática: meça você mesmo a velocidade



As medições dos outros são feitas de outro lugar. Faça cinco requisições do servidor onde o seu site vai morar:

```
for i in 1 2 3 4 5; do
  curl -s -o /dev/null -w '%{time_total} s\n' \
    https://public-ethereum-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
  sleep 1
done
```

Você verá cinco números: o tempo de resposta em segundos. Não olhe só o melhor: o cliente também vai pegar o pior. Aqui a conexão é aberta do zero a cada vez; um programa com conexão permanente costuma ser mais rápido. E compare não só a velocidade, mas também o frescor do bloco (capítulo 12).

Faça essas perguntas a nós também

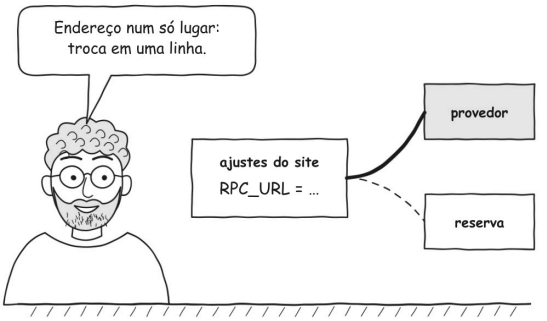
O livro é sobre a RpcNode, então vamos examiná-la também. Dados de 27 de novembro de 2026.

Pergunta	RpcNode
Redes	25 na lista do site
Experimentar de graça	plano Free e nós públicos sem chave
Limites	Free: 15 requisições por segundo e 15 milhões de créditos; Starter: 40 e 50 milhões
Ao ultrapassar	não há cobrança por excedente: as requisições são recusadas (429 ou 402, capítulo 11)
Arquivo	nos planos pagos
WebSocket e gRPC	pelo endpoint pessoal, não para todas as redes
Onde ficam os servidores	Frankfurt, EUA, Japão
Proteção	o endpoint pode ser restrito por endereços IP e por sites
Privacidade	o provedor vê as suas requisições (capítulo 3)
Disponibilidade e suporte	veja as condições em vigor no site rpcnode.dev

Cilada: escolher pelo número maior

Mil redes não ajudam se as suas duas funcionam mal. O preço mais baixo não ajuda se o método de que você precisa é caro ou indisponível. Pegue o plano gratuito ou o mais barato e passe alguns dias rodando as suas requisições de verdade.

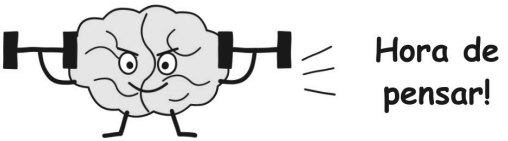
Não crie raízes



- Mantenha o endereço do endpoint numa única configuração, e não no código.
- Sempre que possível, use os métodos padrão da rede.

- Tenha um plano reserva e um dia mude para ele, só para testar.

Pense bem



1. 20 pedidos, conferência a cada 30 segundos. Quantas requisições em 30 dias?
2. Prometeram 99,9%, mas o endpoint ficou duas horas fora do ar no mês. A promessa foi cumprida?
3. O endereço do endpoint está escrito em 14 pontos do código. Por que isso é ruim?

Respostas: 1) 1.728.000. 2) Não: eram permitidos cerca de 43 minutos. 3) Na hora de trocar é fácil esquecer um ponto, e uma chave no código vaza mais facilmente.

O essencial

- Primeiro calcule as suas necessidades, depois compare.
- Teste as promessas com as suas medições e as suas requisições.
- Deixe aberta a possibilidade de ir embora.

Fim da terceira parte. Artiom já sabe o que é um nó, como falar com ele e de quem obtê-lo. Mas as redes são diferentes! **A seguir:** parte 4, capítulo 14. O Bitcoin e seus parentes.

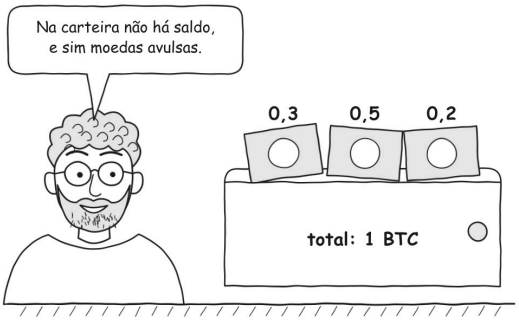
Capítulo 14. O Bitcoin e seus parentes

“E o saldo dele, onde está?”

Um cliente quer pagar em bitcoins. Artiom já tem experiência: agora é só pedir ao nó o saldo do endereço, como na Ethereum. Mas o nó do Bitcoin não tem uma pergunta tão simples. Por quê?

Entendendo: não é conta, são moedas

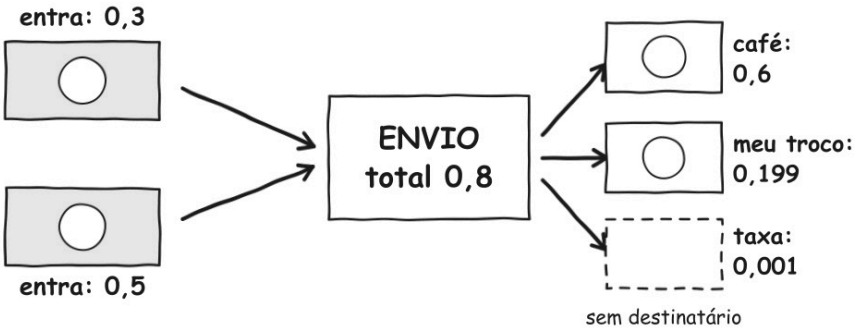
Na Ethereum, o endereço tem um saldo, como uma conta bancária. No Bitcoin, não. Lá existem “pedaços de dinheiro” separados, como cédulas de valores diferentes.



Cada pedaço desses se chama **UTXO** (saída de transação não gasta). O saldo é a soma de todos os seus pedaços não gastos. Quem soma é a carteira ou um indexador (capítulo 10).

A transferência: entradas, saídas e troco

Não dá para rasgar um pedaço da cédula. É preciso entregá-la inteira e receber o troco. Aqui é igual: a transferência gasta os pedaços antigos por completo (**entradas**) e cria pedaços novos (**saídas**).



- Uma saída vai para o destinatário.
- A segunda é o **troco**, para você mesmo.
- Tudo o que não for anotado em nenhuma saída fica com o minerador. É a **taxa**: entradas menos saídas.

> 0,8 − 0,6 − 0,199 = 0,001 BTC de taxa

O preço da taxa não é calculado sobre a quantia, e sim sobre o tamanho da transferência: em **satoshis por byte virtual**. Um satoshi é a centésima milionésima parte de um bitcoin. Quanto mais pedaços na entrada, maior a transferência e mais cara ela fica.

A primeiríssima transferência entre pessoas

Em 12 de janeiro de 2009, no bloco 170, um pedaço de 50 BTC virou dois: 10 BTC e 40 BTC. Quem recebeu os dez, o programador Hal Finney, contou ele mesmo que Satoshi os tinha enviado como teste. O troco, 40 BTC, voltou ao remetente. Não houve taxa.

Na blockchain não há nomes: sobre Finney se sabe pelo relato dele mesmo (bitcointalk.org, 2013).

Essa transferência ainda pode ser consultada:

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getrawtransaction","params":
  ["f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16",true]}'
```

Na resposta, procure `vin` (as entradas, aqui uma) e `vout` (as saídas, aqui duas: `10.0` e `40.0`).

Na prática: a moeda já foi gasta?

O método `gettxout` pergunta: "este pedaço ainda está inteiro?". O pedaço é identificado pelo número da transferência e pelo número da saída, contando a partir de zero. Vamos perguntar

por aqueles 10 BTC:

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"gettxout","params":
  ["f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16",0]}'
```

```
{"jsonrpc":"2.0","result":null,"id":1}
```

`null` : não existe tal pedaço não gasto, ele foi gasto há muito tempo. Agora, a recompensa do primeiríssimo bloco depois do inicial:

```
curl https://public-bitcoin-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"gettxout","params":
  ["0e3e2357e806b6cdb1f70b54c3a3a17b6714ee1f0e68bebb44a74b1efd512098",0]}'
```

Aqui vêm `"value":50.00000000` e quase um milhão de confirmações. Quando este capítulo estava sendo escrito, ninguém ainda tinha tocado nesses 50 BTC.

Mais três fatos sobre o Bitcoin

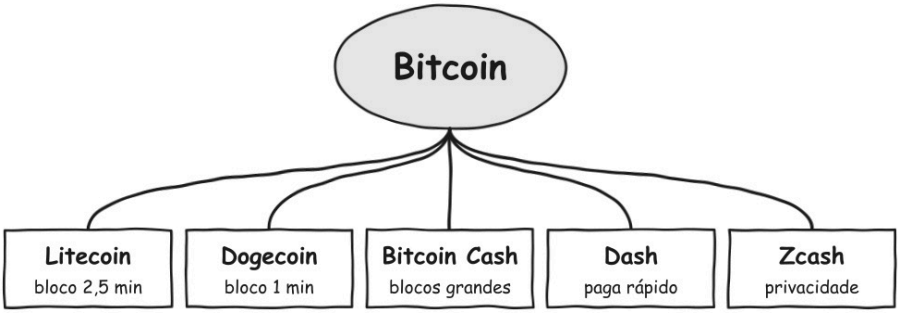
- Nunca haverá mais que **21 milhões** de moedas.
- A recompensa do minerador não pode ser gasta antes que se acumulem 100 blocos por cima.
- Para quantias altas, esperam-se 6 confirmações, cerca de uma hora (capítulo 8).

Cilada: o troco esquecido

Uma carteira comum acrescenta o troco sozinha. Mas, se um programa montar a transferência na mão e esquecer a saída do troco, todo o resto vira taxa.



Os parentes



parentes no feitio, mas cada um com sua rede e suas moedas

Rede	Bloco, aprox.	O que tem de diferente	O endereço começa com
Bitcoin	10 min	o mais velho da família	`1`, `3`, `bc1`
Litecoin	2,5 min	outro algoritmo de mineração, o Scrypt	`L`, `M`, `ltc1`
Dogecoin	1 min	não há teto de emissão; pode ser minerada junto com a Litecoin	`D`
Bitcoin Cash	10 min	blocos bem maiores	`bitcoincash:q…`
Dash	2,5 min	pagamentos rápidos (InstantSend) e proteção dos blocos contra substituição (ChainLocks)	`X`
Zcash	75 s	as transferências podem ser ocultadas	abertos `t1`, ocultos `zs`, `u1`

Eles funcionam de modo parecido, por isso as perguntas são quase iguais. Vamos pedir a altura da cadeia a cinco redes de uma vez:

```
for net in bitcoin litecoin doge bch dash; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"getblockcount","params":[]}'
  echo; sleep 1
done
```

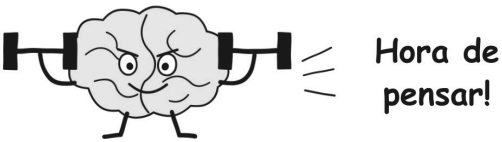
```
bitcoin: {"jsonrpc":"2.0","result":977717,"id":1}
litecoin: {"result":3219941,"error":null,"id":1}
doge: {"result":6474162,"error":null,"id":1}
bch: {"result":979125,"error":null,"id":1}
dash: {"result":2578936,"error":null,"id":1}
```

Segunda cilada: "tem error, então é erro"

Olhe as respostas: em quatro redes o campo `error` está sempre presente. O valor `null` significa "tudo certo". Verifique não se o campo existe, mas o que há dentro dele, como aconselhou o capítulo 11.

E não confunda as redes. Os parentes não aceitam as moedas uns dos outros. No Bitcoin Cash, o formato antigo de endereço tinha a mesma cara que o do Bitcoin, por isso lá se introduziu um novo, com o prefixo `bitcoincash:`.

Pense bem



1. Na carteira há pedaços de 0,3, 0,5 e 0,2 BTC. É preciso pagar 0,4, com taxa de 0,001. Um pedaço basta? Qual será o troco?
2. Por que enviar 1 BTC formado por cem pedacinhos é mais caro do que enviar um pedaço inteiro?
3. Uma entrada de 1,0 BTC, uma única saída de 0,4 BTC. Onde foram parar os 0,6?

Respostas: 1) sim, o pedaço de 0,5; o troco é 0,099. 2) Mais entradas, transferência maior. 3) É a taxa. O mais provável é que tenham esquecido o troco.

O essencial

- No Bitcoin não há saldo em conta; há pedaços não gastos (UTXO).
- A taxa são as entradas menos as saídas; não esqueça o troco.
- Os parentes se parecem no funcionamento, mas são redes diferentes.

A seguir: capítulo 15. Ethereum: contas, gas e eventos.

Capítulo 15. Ethereum: contas, gas e eventos

“Por que mandar dólares é mais caro?”

Artiom mandou ao fornecedor um pouco de ETH e depois USDT. As quantias eram parecidas, mas no USDT cobraram mais que o dobro de “gas”. Que gas é esse e por que é assim?

Entendendo: dois tipos de conta

Na Ethereum, o endereço tem um saldo, como no banco. Nada a ver com os pedaços do Bitcoin (capítulo 14). E há dois tipos de conta.



	Conta comum	Contrato
Quem manda	o dono da chave	o programa dele
Pode iniciar sozinho uma transferência	sim	não, só quando é chamado
Tem código	normalmente não	sim

Quem executa os programas é a **EVM**, a máquina virtual da Ethereum. É como um videogame dentro de cada nó: o mesmo programa dá o mesmo resultado em toda parte.

Na prática: onde fica o programa

O método `eth_getCode` mostra o código que há num endereço. Vamos perguntar pelo contrato do USDT e ficar só com o começo da resposta:

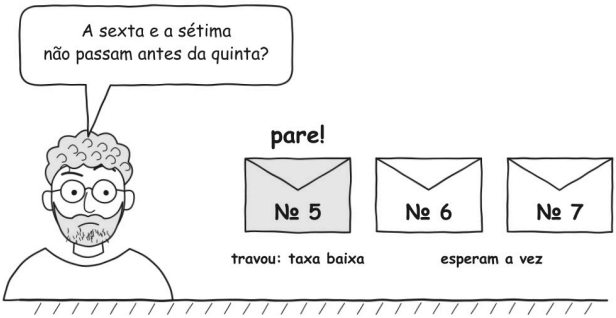
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getCode","params":
  ["0xdac17f958d2ee523a2206206994597c13d831ec7","latest"]}' | head -c 80; echo
```

Você verá o começo de uma longa sequência de caracteres: é o programa, com mais de 11 mil bytes. Troque o endereço por `0x0000000000000000000000000000000000000dEaD` e virá `"0x"`, ou seja, não há código.

Desde 2025, uma conta comum também pode delegar suas ações a um programa (atualização Pectra). Nesse caso a resposta começa com `0xef0100`. Ou seja, "tem código" já nem sempre significa "é um contrato".

O número de ordem: nonce

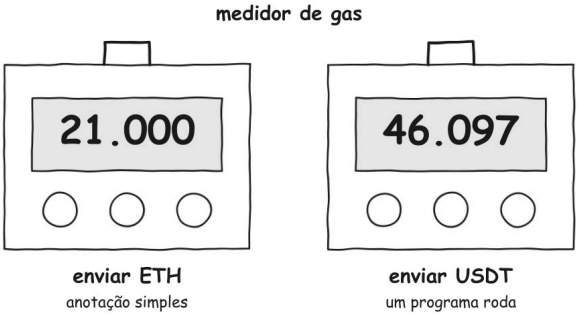
Cada transação de uma conta recebe um número: 0, 1, 2 e assim por diante. Ele se chama **nonce**. A rede executa as transações rigorosamente em ordem e não aceita o mesmo número uma segunda vez.



Se uma transferência com taxa pequena travou, as seguintes esperam atrás dela. As carteiras oferecem "acelerar": enviam uma nova transferência com o mesmo número e uma taxa maior.

Gas: o medidor de trabalho

O **gas** mede quanto trabalho uma transação exige. O simples registro "transferir ETH" custa hoje 21.000. Transferir um token é rodar um programa, e precisa de mais gas.



Taxa = gas consumido × (preço base + gorjeta)

- O **preço base** é definido pela própria rede. Esse dinheiro é **queimado**.
- A **gorjeta** vai para o validador: quanto maior, com mais boa vontade ele pega a sua transação.
- O preço do gas é contado em **gwei**, a bilionésima parte de um ETH.

No bloco cabe uma quantidade limitada de gas. Quando os blocos estão mais da metade cheios, o preço base sobe; quando menos, ele cai. Por exemplo, o bloco 26.502.250 comportava 60 milhões de gas e estava 93% cheio.

A carteira define um **limite de gas**. Só se cobra o que foi consumido. Mas, se o limite não bastar, a transferência não é executada, e o gas gasto não volta.

Uma transferência de USDT de verdade

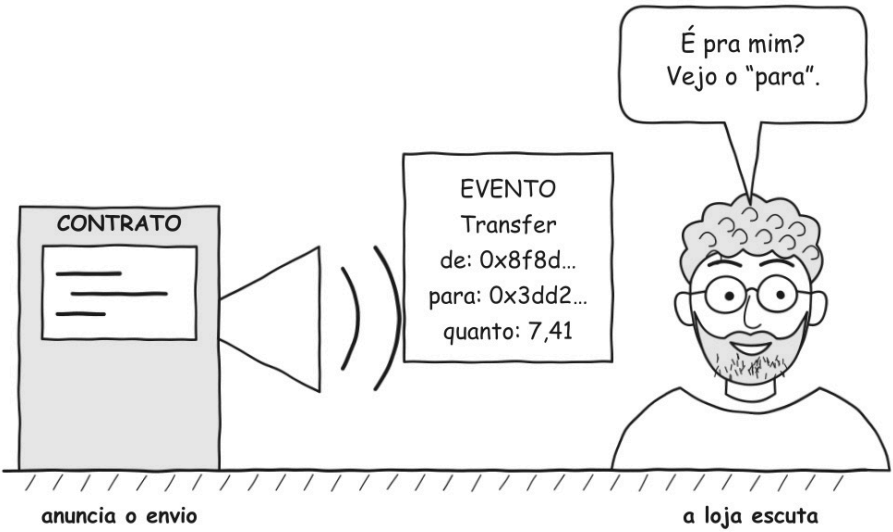
É uma transferência de 7,41 USDT daquele mesmo bloco. Até então o remetente já tinha feito 423.464 transações.

O quê	Valor
gas consumido	46.097
preço base	0,058 gwei
gorjeta	2 gwei
taxa	cerca de 0,000095 ETH
dos quais queimados	cerca de 0,000003 ETH

Eventos: como o contrato avisa sobre uma transferência

Os saldos dos tokens ficam dentro do contrato. Para o mundo ficar sabendo de uma transferência, o contrato anuncia um **evento** (log). Nos tokens ele se chama `Transfer`: de

quem, para quem, quanto.



Os eventos ficam no recibo da transação, do capítulo 7. Vamos pedir o recibo desta transferência:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":
  ["0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016"]}'
```

Na resposta, encontre `logs`:

Campo	Começo e fim	O que significa
`address`	`0xdac1…1ec7`	quem anunciou: o contrato do USDT
`topics[0]`	`0xddf2…b3ef`	o tipo de evento: Transfer
`topics[1]`	`…8f8d…884e`	de quem
`topics[2]`	`…3dd2…a9c0`	para quem
`data`	`…711150`	7.410.000, ou seja, 7,41 USDT

E o método `eth_getLogs` sabe procurar eventos. Vamos contar as transferências de USDT no bloco mais recente:

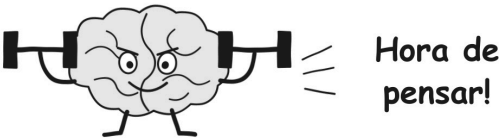
```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef"],
  "fromBlock":"latest","toBlock":"latest"}]}' | grep -o '"logIndex"' | wc -l
```

Sai o número de transferências em um bloco. Sobre blocos passados o nó público não responde: isso já é arquivo (capítulo 10).

Cilada: "eu só escuto os eventos"

- **A transferência do próprio ETH não gera evento Transfer.** Uma loja que só espera eventos deixa passar um pagamento desses. Futuras atualizações da rede podem mudar isso.
- **O ETH pode vir de um contrato**, por exemplo, de certas carteiras. Nesse caso o seu endereço não aparece na própria transação.
- **Qualquer contrato pode anunciar um evento.** Confira o `address`: o token é o verdadeiro? (capítulo 9)

Pense bem



1. Transferência de ETH, 21.000 de gas, preço com gorjeta de 2 gwei. Qual é a taxa em gwei e em ETH?
2. Vários blocos seguidos estão 30% cheios. O que acontece com o preço base?
3. A loja só acompanha os eventos Transfer do contrato do USDT. O cliente pagou em ETH. A loja vai ver?

Respostas: 1) 42.000 gwei, ou 0,000042 ETH. 2) Ele vai baixar. 3) Não.

O essencial

- Uma conta é comandada por uma chave ou por um programa.
- A taxa é o gas multiplicado pelo preço: o base mais a gorjeta.
- As transferências de tokens são conhecidas pelos eventos; o ETH precisa ser pego à parte.

A seguir: capítulo 16. A Ethereum tem muitos parentes: outras redes EVM e os “segundos andares”.

Capítulo 16. Os parentes da Ethereum e os “segundos andares”

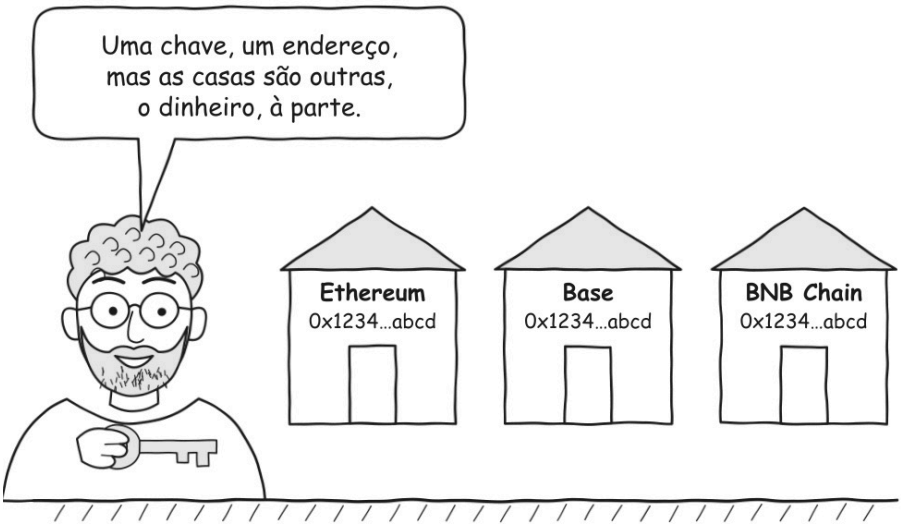
“Mandei pela Base”

O cliente escreve: “Mandei USDC pela Base, o endereço é o mesmo”. Artiom fica confuso. É outro token? Outra rede? E por que o endereço é o mesmo?

Entendendo: o mesmo molde, casas diferentes

Muitas redes adotaram a EVM do capítulo 15. Por isso elas têm endereços iguais, os mesmos métodos `eth_…`, e o mesmo programa de contrato pode rodar em cada uma. Essas redes são chamadas de **compatíveis com EVM**.

Mas os cadernos delas são **diferentes**. Blocos próprios, saldos próprios, taxas próprias.



Para não confundir as redes, cada uma tem um número: o **chain ID**. Ele entra na assinatura, por isso uma transferência assinada para uma rede não é aceita por outra.

Na prática: pergunte o número a seis redes

```
for net in ethereum base arb bsc mantle monad; do
  echo -n "$net: "
  curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 1
done
```

Vêm números em hexadecimal. Aqui estão eles em números comuns, junto com a frequência com que saíam blocos na verificação:

Rede	Chain ID	Um bloco a cada, aprox.	Com que se paga a taxa
Ethereum	1	12 s	ETH
Base	8453	0,2 s	ETH
Arbitrum One	42161	0,3 s	ETH
BNB Smart Chain	56	0,45 s	BNB
Mantle	5000	2 s	MNT
Monad	143	0,3 s	MON

Parentes do primeiro tipo: os independentes

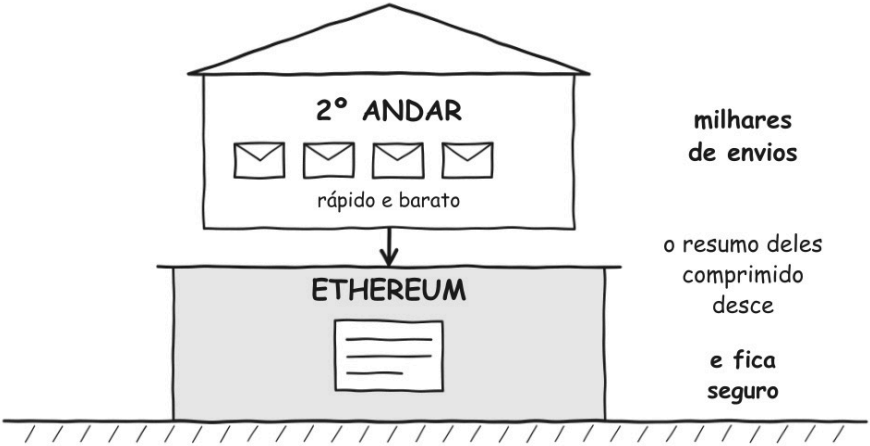
Eles têm validadores próprios e entram em acordo sobre os blocos por conta própria.



A Polygon PoS, além disso, deixa periodicamente na Ethereum marcas dos seus blocos. Entre os independentes está também a Monad, lançada em novembro de 2025.

Parentes do segundo tipo: os segundos andares (L2)

Uma **L2** executa as transferências no próprio espaço, rápido e barato, e envia para a Ethereum registros comprimidos delas. O tipo mais comum se chama **rollup**, algo como "enrolar tudo num pacote".



- A ordem das transferências costuma ser definida por um único operador, o **sequenciador**. Por isso é tudo tão rápido. E por isso, nesse ponto, você confia nele.
- Os rollups **otimistas** (Arbitrum One, OP Mainnet) consideram os registros corretos, mas dão um tempo para contestá-los.
- Os rollups **ZK** (por exemplo, a Mantle) anexam uma prova matemática.
- A Base passou em 2026 a um código próprio e combina vários tipos de prova.

Por causa do tempo de verificação, não dá para trazer rapidamente o dinheiro de volta à Ethereum pela ponte oficial:

Saindo da rede	Quanto esperar para sacar na Ethereum
Arbitrum One	no mínimo 7 dias
OP Mainnet	7 dias
Base	5 dias; em alguns casos, 1 dia

Segundo a documentação das redes em 27 de novembro de 2026. Esses prazos mudam: na OP Mainnet há uma reformulação prevista para janeiro de 2027.

“Vi rápido” não quer dizer “definitivo”

Lembra `latest`, `safe` e `finalized`, do capítulo 8? Veja o que respondeu um nó da Base:

O que se pediu	Atraso em relação ao bloco mais novo
`latest`	—
`safe`	cerca de 17 segundos
`finalized`	cerca de 19 minutos

O bloco surge em frações de segundo, mas só se torna definitivo quando o registro dele já é definitivo na própria Ethereum.

A taxa no segundo andar

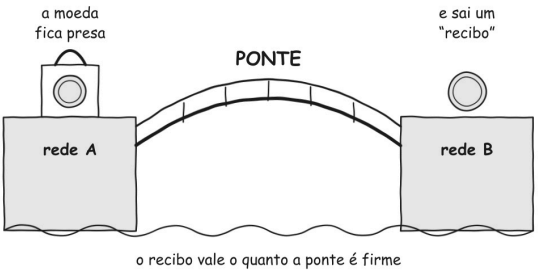
Ela se compõe principalmente de duas partes: o trabalho na própria L2 e o pagamento pelo registro na Ethereum. Às vezes há pequenos acréscimos. Num recibo da Base, a segunda parte está no campo `l1Fee`:

```
curl https://public-base-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt","params":
  ["0xa1d4398ce62b570a2810ad2b2ed347f388f9ad85aee4b62ba3215917cf3ac7e5"]}'
```

Aqui o trabalho na Base custou cerca de 0,0000035 ETH, e o registro na Ethereum, cerca de 0,0000000006 ETH. Nem sempre é tão barato.

Pontes

Para se “mudar” de uma rede para outra, usa-se uma **ponte**. Muitas vezes ela tranca as moedas numa rede e emite na outra um substituto delas.



Por isso **USDC** e **USDC.e** são tokens diferentes, com endereços de contrato diferentes. O primeiro é emitido pela própria empresa Circle; o segundo aparece por meio de uma ponte.

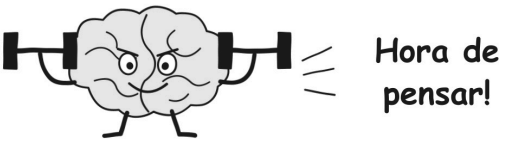
Cilada: mandou pela rede errada

No capítulo 6 dissemos: a loja não vai ver um pagamento desses. É verdade. Mas o dinheiro está perdido?

- **A chave do endereço está com você**: em geral dá para recuperar o dinheiro mudando a carteira para aquela rede. Para movê-lo, você vai precisar da moeda dela para a taxa.
- **É um endereço numa corretora**: a chave não está com você. Se a corretora não suporta aquela rede, o mais provável é que não dê para recuperar o dinheiro.
- **É uma carteira-contrato**: na outra rede ela pode nem existir. Vai ser preciso a ajuda de um entendido.

> **Regra:** não conte a frase secreta a ninguém, nem a quem promete "resgatar" a transferência (capítulo 6).

Pense bem



1. A loja esperava USDT na Ethereum, mas recebeu na BNB Smart Chain, no próprio endereço dela. A chave está com Artiom. O que fazer?
2. Um bloco na Base surgiu em 0,2 segundo. Um pagamento alto já pode ser tido como definitivo?
3. O cliente mandou USDC.e no lugar de USDC. É a mesma coisa?

Respostas: 1) mudar a carteira para a BNB Smart Chain; para mover é preciso BNB. 2) Não, espere o finalized. 3) Não, é outro token: quem decide se aceita é a loja.

O essencial

- As redes EVM se parecem, mas são cadernos diferentes: confira o chain ID.
- Há redes independentes e segundos andares sobre a Ethereum.
- Um bloco rápido ainda não é definitivo.

A seguir: capítulo 17. TRON: energia no lugar do gas e a estrada favorita das stablecoins.

Capítulo 17. TRON: energia no lugar do gas e a estrada favorita das stablecoins

"Pode ser por TRC-20?"

Um cliente pergunta: "Vocês aceitam USDT por TRC-20?" É a terceira vez na semana que Artiom ouve isso. O que é TRC-20 e por que todo mundo pede?

Entendendo: uma rede independente

A **TRON** é uma blockchain separada, com moeda própria, o **TRX**. Sua menor fração se chama **sun**: um TRX tem um milhão de sun.

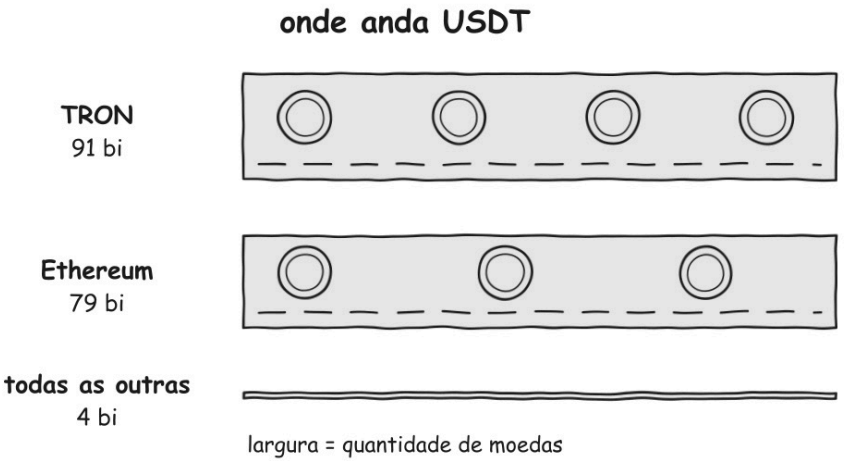
A TRON funciona de um jeito muito parecido com a Ethereum: saldos em contas, contratos inteligentes, eventos. Os programas são executados por uma máquina virtual própria, parente próxima da EVM do capítulo 15.

TRC-20 são as regras pelas quais se fazem tokens na TRON, o irmão do padrão ERC-20 da Ethereum. "USDT por TRC-20" quer dizer: o token USDT na rede TRON.

Os blocos saem a cada 3 segundos. Quem os escreve são 27 representantes eleitos (capítulo 8).

A estrada favorita das stablecoins

O USDT vive em muitas redes, mas de forma bem desigual. Veja os números da própria Tether, empresa que emite a moeda, consultados em 27 de novembro de 2026:



Mais da metade de todo o USDT está na TRON. Por isso os clientes escolhem tanto essa rede.

Fonte: app.tether.to/transparency.json. Contaram-se as moedas emitidas menos a reserva ainda não posta em circulação.

O endereço começa com T

Por dentro, um endereço TRON são 20 bytes, como na Ethereum, com o prefixo `41`. Mas para as pessoas ele é mostrado de outro jeito, começando com a letra `T`:

Como as pessoas veem	Como é por dentro (endereço do contrato do USDT)
`TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t`	`41a614f803b6fd780986a42c78ec9c7f77e6ded13c`

Na forma com `T` há um dígito de controle escondido: dá para perceber um erro de digitação. Vamos perguntar ao nó sobre um endereço verdadeiro e depois trocar a última letra dele:

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t","visible":true}'

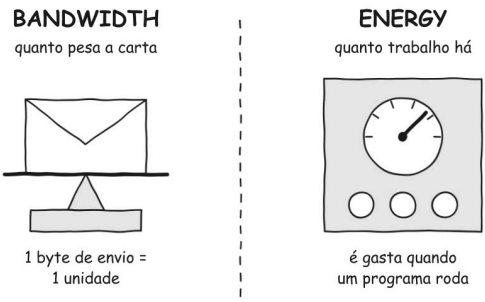
curl https://public-tron-mainnet.rpcnode.dev/wallet/validateaddress \
  -d '{"address":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6u","visible":true}'
```

```
{"result":true,"message":"Base58check format"}
{"result":false,"message":"Invalid address"}
```

`"visible":true` quer dizer "estou escrevendo os endereços na forma com T".

Duas cobranças no lugar do gas

No capítulo 7 prometemos contar como a TRON calcula o "peso" de uma transferência. Ela usa duas medidas.



- **Bandwidth**: pelo espaço. O consumo é mais ou menos igual ao tamanho da transferência em bytes. Toda transferência paga.

- **Energy** (energia): pelo trabalho do programa. É necessária quando se chama um contrato, portanto também numa transferência de USDT.

Um endereço que já recebeu TRX pelo menos uma vez ganha um presente: **600 unidades de bandwidth por dia**. O que se gasta se recupera aos poucos, ao longo de 24 horas. Energia grátis não existe.

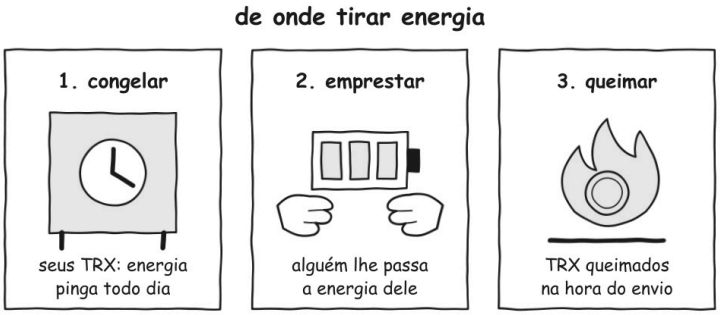
Se algo falta, a rede pega TRX e os queima. A que preço? Vamos perguntar ao nó:

```
curl -s https://public-tron-mainnet.rpcnode.dev/wallet/getchainparameters |
  python3 -c "import json,sys
for p in json.load(sys.stdin)['chainParameter']:
    if p['key'] in ('getFreeNetLimit','getTransactionFee','getEnergyFee'):
        print(p['key'], p['value'])"
```

```
getTransactionFee 1000
getEnergyFee 100
getFreeNetLimit 600
```

Ou seja, uma unidade de bandwidth custa 1000 sun, e uma unidade de energia, 100 sun. Esses preços são alterados por votação dos representantes.

De onde tirar energia

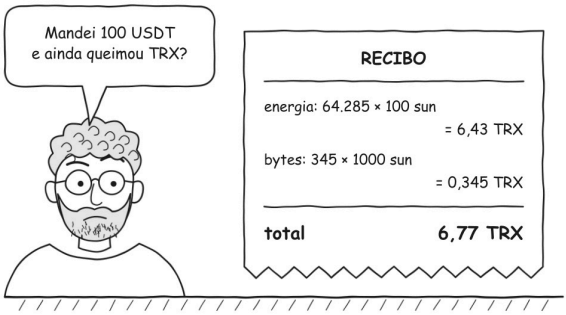


- **Congelar TRX** (em inglês, staking). As moedas continuam suas, e o endereço recebe todo dia uma parte da energia total. Para retirá-las, depois de descongelar se esperam 14 dias.
- **Pegar emprestado**: o dono de TRX congelados transfere a energia dele para outro endereço.
- **Não fazer nada**: o que faltar é pago com a queima de TRX.

Quanto congelar? Depende de quantas moedas todos os outros congelaram. No dia da verificação, para uma transferência de USDT por dia bastariam cerca de 6600 TRX congelados.

O recibo de uma transferência de verdade

Eis uma transferência de 100 USDT do bloco 88.352.467. O remetente não tinha nem energia nem bandwidth livre:



O recibo se obtém com o método `gettransactioninfobyid`. No nó público ele está fechado, é preciso um endpoint pessoal, por isso isto é um esboço:

```
curl https://SEU_ENDPOINT/walletsolidity/gettransactioninfobyid \
  -d '{"value":"0eb7b372f5a268ce56430392de8db2fd970455bb0970c67b47a7b2673abb513c"}'
```

Campo da resposta	Valor	O que significa
`energy_usage_total`	64.285	total de energia gasta
`energy_penalty_total`	49.635	dos quais, acréscimo por popularidade
`energy_fee`	6.428.500	queimado pela energia, em sun
`net_fee`	345.000	queimado pelos bytes, em sun
`fee`	6.773.500	total, ou 6,7735 TRX
`result`	`SUCCESS`	o programa rodou com sucesso

Que acréscimo é esse? Quando um mesmo contrato é usado demais, a rede encarece as chamadas a ele. No USDT, naquele dia, gastava-se quase 4,4 vezes mais energia do que sem o acréscimo.

E mais: se o destinatário ainda não tinha USDT, é preciso o dobro de energia, 130.285. O contrato tem de abrir uma linha nova para ele.

Bloco recente e bloco definitivo

Os nós da TRON têm dois conjuntos de endereços. `/wallet/…` responde com os dados mais recentes. `/walletsolidity/…`, só com os definitivos. Um bloco se torna definitivo quando, na

altura dele ou acima, 19 dos 27 representantes já publicaram blocos seus. Isso costuma levar cerca de um minuto.

No dia da verificação, o bloco mais recente era o 88.352.763, e o definitivo, o 88.352.746.

Para experiências existe também a rede de testes Nile, com nó público próprio: `public-tron-nile.rpcnode.dev`.

Cilada: "está no bloco, então chegou"

Vamos olhar as transferências diretas de USDT no último bloco definitivo:

```
curl -s https://public-tron-mainnet.rpcnode.dev/walletsolidity/getnowblock > block.json
python3 <<'EOF'
import json
usdt = '41a614f803b6fd780986a42c78ec9c7f77e6ded13c'
rows = []
for t in json.load(open('block.json'))['transactions']:
    v = t['raw_data']['contract'][0]['parameter']['value']
    data = v.get('data', '')
    if v.get('contract_address') == usdt and data.startswith('a9059cbb'):
        amount = int(data[72:136], 16) / 1_000_000
        rows.append((t['txID'][:8], t['ret'][0]['contractRet'], amount))
print('transferências de USDT:', len(rows))
print('não concluídas:', sum(r[1] != 'SUCCESS' for r in rows))
for r in rows[:3]:
    print(*r)
EOF
```

Os seus números serão outros. Já no bloco 88.352.467, de 41 transferências assim, duas não se concluíram. Uma delas era de 100.000 USDT:

```
fbfec9fa OUT_OF_ENERGY 100000.0
```

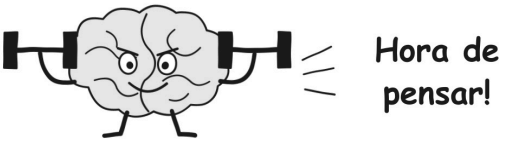
`OUT_OF_ENERGY`: faltou energia, o programa parou e o USDT não foi a lugar nenhum. Mas o registro no bloco existe, e numa captura de tela a quantia tem ar respeitável.

Quanta energia se pode gastar quem define é o remetente: o campo `fee_limit`, um teto em sun. Se o teto não bastar, a transferência não se conclui, e o que foi queimado não volta.

O que a loja precisa lembrar

- Espere o bloco definitivo e confira o `result` no recibo.
- Procure no recibo o evento `Transfer` do contrato `TR7NHq…Lj6t`. Assim você enxerga também as transferências feitas por outros programas. O USDT tem seis casas decimais.
- Para mover USDT do seu endereço, ele precisa ter TRX ou energia. Um endereço por pedido é prático, mas recolher o dinheiro de cem endereços são cem transferências pagas.
- A primeiríssima transferência de TRX para um endereço novo custa ao remetente pelo menos 1 TRX a mais.

Pense bem



1. Um endereço tem 600 de bandwidth grátis e muita energia. Ele manda seguidas duas transferências de 345 cada. O que acontece com a segunda?
2. Não há energia, e o destinatário ainda não tinha USDT: são precisas 130.285 unidades a 100 sun. Quantos TRX serão queimados pela energia?
3. O cliente mandou o hash. A transação está no bloco e traz `OUT_OF_ENERGY`. Despachar o café?

Respostas: 1) sobram 255 grátis, que não bastam para 345; pelos 345 inteiros se queimam 0,345 TRX, não se soma em partes. 2) 13,0285 TRX. 3) Não, não houve transferência.

O essencial

- Na TRON há duas medidas: bandwidth pelos bytes e energy pelo trabalho do programa.
- Sem energia, queimam-se TRX, e bem mais no USDT do que num simples envio.
- Pagamento é uma transferência bem-sucedida de USDT verdadeiro num bloco definitivo.

A seguir: capítulo 18. Solana: tudo guardado em contas.

Capítulo 18. Solana: tudo guardado em contas

"Cadê os meus USDC?"

Um cliente pagou em USDC. Artiom pergunta ao nó o saldo da própria carteira e vê só umas migalhas de SOL. Nenhum USDC. Foi enganado?

Entendendo: uma rede feita de contas

A **Solana** é uma rede independente, com a moeda **SOL**. A menor fração se chama **lamport**: um SOL tem um bilhão deles.

Um endereço Solana são 32 bytes escritos com letras e números. Ele não tem um começo especial (capítulo 6) nem dígito de controle por dentro.

Tudo o que a rede guarda fica em **contas** (accounts). Aqui, uma conta não é só dinheiro, mas qualquer gaveta com endereço. Na Ethereum, o contrato guarda os próprios dados dentro de si. Na Solana, **o programa fica de um lado, e os dados, de outro**.



Toda conta tem um programa dono (campo `owner`). Só ele pode mudar os dados da conta.

Na prática: a "identidade" de um token

Cada token tem uma conta que funciona como documento de identidade, chamada **mint**. Vamos pedir a do USDC:

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getAccountInfo","params":
  ["EPjFWdd5AufqSSqeM2qN1xzybapC8G4wEGGkZwyTDt1v",{"encoding":"jsonParsed"}]}'
```

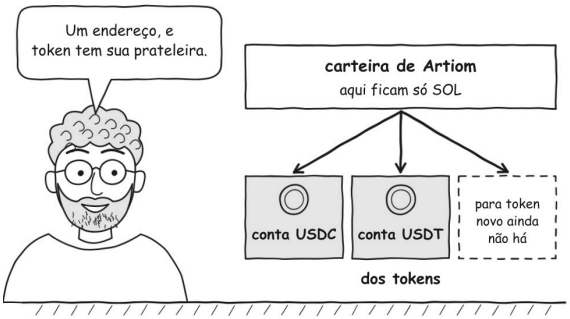
Eis o principal da resposta:

Campo	Valor	O que significa
`owner`	`Tokenkeg…Q5DA`	a conta é comandada pelo programa de tokens
`supply`	`8513455604386747`	quanto foi emitido, nas menores frações
`decimals`	6	ou seja, cerca de 8,5 bilhões de USDC
`mintAuthority`	`BJE5…5ruG`	quem pode emitir mais
`freezeAuthority`	`7dGb…Crar`	quem pode congelar (capítulo 9)

O USDC não tem programa próprio. Um único programa de tokens atende milhares de tokens, que se distinguem por suas identidades. Há dois programas gerais desses: o comum, **Token** (onde estão o USDC e o USDT), e o mais novo, **Token-2022**, com recursos extras.

Conta de token: uma prateleira separada

Quanto USDC Artiom tem não está anotado nem na carteira nem na identidade do token, e sim numa **conta de token**. Cada token tem a sua.



A conta de token principal se chama **ATA** (associated token account). O endereço dela é calculado a partir de três coisas: o endereço da carteira, a identidade do token e o programa de tokens. Por isso se passa ao cliente o **endereço da carteira**, e o programa dele acha a prateleira sozinho.

Vamos olhar uma conta de token de verdade. Primeiro perguntamos quantos tokens há nela e depois quanto SOL:

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTokenAccountBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'

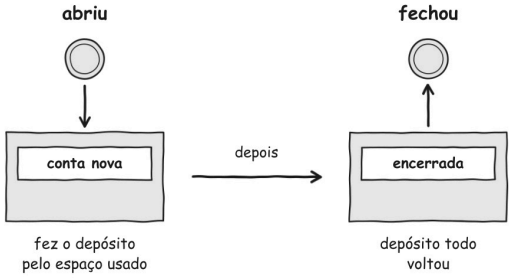
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBalance","params":
  ["BnGihgNUu4X1JdBMNhbWnXUcpib8UQQC154isHTYwDNK"]}'
```

Na primeira resposta, `"uiAmountString":"207637.336362"` é USDT. Na segunda, `"value":2039280` são lamports, apenas 0,00203928 SOL. O método `getBalance` **não enxerga tokens**. É aí que os USDC de Artiom foram "sumir".

Aliás, o dono dessa prateleira é o endereço da própria identidade do USDT. Pelo visto, as pessoas mandaram USDT por engano "para o endereço do USDT", como no capítulo 9.

Depósito pelo espaço

Cada conta ocupa memória em todos os nós. Por isso há nela um **depósito** em SOL (em inglês, rent, "aluguel", embora não seja aluguel).



Uma conta de token tem 165 bytes. De quanto é o depósito para esse tamanho?

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,
  "method":"getMinimumBalanceForRentExemption","params":[165]}'
```

```
{"jsonrpc":"2.0","result":1488440,"id":1}
```

Dá cerca de 0,0015 SOL. Em 2026 o depósito está sendo reduzido por etapas, por isso o programa não grava o número, e sim pergunta. Se o destinatário ainda não tem a prateleira daquele token, o remetente costuma criá-la na mesma transferência e pagar o depósito ele mesmo.

Taxa e prazo de validade

A taxa básica é de **5000 lamports por assinatura**: numa transferência comum, com uma assinatura, são 0,000005 SOL. Metade é queimada. Por cima pode-se acrescentar um pagamento pela urgência, que vai inteiro para o validador. A taxa é cobrada mesmo que a transferência falhe.

Em cada transferência vai um **blockhash**, a impressão digital de um bloco recente. É ele também que define o prazo de validade.




```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getLatestBlockhash",
  "params":[{"commitment":"finalized"}]}'

curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBlockHeight",
  "params":[{"commitment":"finalized"}]}'
```

Na primeira resposta veio `"lastValidBlockHeight":452330099`; na segunda, um segundo depois, a altura 452.329.960. Restavam 139 blocos. Se o prazo venceu e a transferência não está em nenhum bloco, ela não passa mais. Aí se pode assinar uma nova. A exceção são as transferências adiadas especiais (durable nonce), que não têm esse prazo.

2026: ainda mais rápido

O que mudou	Antes	Agora
Duração do slot, isto é, da vez de um bloco (outubro)	0,4 s	0,2 s
Finalidade, nova forma de votação Alpenglow (outubro)	cerca de 13 s	cerca de 0,15 s
Tamanho máximo da transação, novo formato v1 (setembro)	1232 bytes	4096 bytes

Vamos conferir a velocidade nós mesmos. O método devolve quantos slots passaram em um minuto:

```
curl https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getRecentPerformanceSamples","params":[1]}'
```

Para nós veio `numSlots` = 277 em 60 segundos: cerca de 0,22 s por slot. Há slots vazios, por isso o número do slot e a altura do bloco são números diferentes, não os confunda.

O nó pode responder com graus diferentes de rigor: `processed`, o bloco foi visto, mas ainda pode ser pulado; `finalized`, definitivo. Há também a palavra intermediária `confirmed`: depois do Alpenglow ela significa o mesmo que `finalized`. O pedido se despacha com `finalized`.

Cilada: "a transação não abre"

Por causa do novo formato v1, nos métodos `getTransaction` e `getBlock` é preciso acrescentar `maxSupportedTransactionVersion` com o valor 1. Sem isso, as transações novas devolvem erro. O método `getTransaction` está fechado no nó público, por isso isto é um esboço para um endpoint pessoal:

```
curl https://SEU_ENDPOINT \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getTransaction","params":
  ["ASSINATURA_DA_TRANSFERENCIA",{"encoding":"jsonParsed","commitment":"finalized",
  "maxSupportedTransactionVersion":1}]}'
```

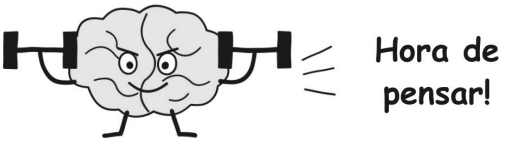
Na Solana, a "assinatura da transferência" serve como número dela, no lugar do hash da transação.

O que a loja confere

- A transação deu certo: `meta.err` é igual a `null`.
- A conta de token **da sua carteira**, com a identidade do token certo, aumentou na quantia certa: compare `preTokenBalances` e `postTokenBalances`.
- É o pagamento justamente deste pedido. Para isso se acrescenta à transferência um **reference**, um endereço descartável que etiqueta o pedido.
- Essa assinatura ainda não foi contabilizada.

E mais cuidado com o SOL comum: uma transferência para um endereço errado é executada mesmo assim. O mais provável é que não haja quem devolva esse dinheiro.

Pense bem



1. A carteira assinou a transferência, mas a rede só a viu dois minutos depois. O que acontece?
2. A loja pergunta o `getBalance` da própria carteira para flagrar um pagamento em USDC. Onde está o erro?
3. Artiom manda USDC a um fornecedor que ainda não tem essa prateleira. O que ele terá de pagar além da taxa?

Respostas: 1) o prazo venceu, a transferência não passa. 2) O `getBalance` mostra só SOL; os USDC ficam na conta de token. 3) O depósito da nova conta de token, cerca de 0,0015 SOL.

O essencial

- Na Solana tudo são contas; os programas e seus dados ficam separados.
- Os tokens ficam em contas de token: uma para cada token, com depósito.
- Leia as transações com `maxSupportedTransactionVersion` e espere o `finalized`.

A seguir: capítulo 19. TON: cartas em vez de telefonemas.

Capítulo 19. TON: cartas em vez de telefonemas

“Mandei, e agora?”

Um cliente mandou USDT na rede TON. No explorador, em vez de um registro, Artiom vê uma cadeia inteira: cinco transações, uns endereços estranhos e um troco miúdo. É um pagamento ou cinco?

Entendendo: todos são contratos, todos escrevem cartas

A **TON** (The Open Network) é uma rede independente. Sua moeda se chama **Gram** (GRAM). Antes se chamava Toncoin, e esse nome ainda aparece muito. Um Gram tem um bilhão de **nanograms**.

Desde abril de 2026 os blocos saem mais ou menos a cada 0,4 segundo, e o registro se torna definitivo em cerca de um segundo.

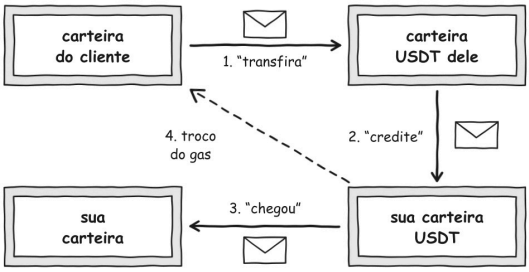
A principal diferença é esta. Aqui, toda conta é um contrato inteligente, e a **carteira também**. Os contratos não chamam uns aos outros diretamente: eles enviam **mensagens**.

Na Ethereum, tudo o que uma transferência disparou acontece dentro de uma única transação: ou por inteiro, ou nada. É como uma conversa por telefone. Na TON é uma troca de cartas: você manda e não fica esperando na linha. Cada carta é tratada pelo destinatário depois, numa transação separada. Esse esquema se chama **assíncrono**.

Toda a correspondência nascida de uma única ordem se chama **trace** (rastro). O sucesso de uma transação dele ainda não significa que o serviço todo deu certo.

Como andam os USDT

Os tokens na TON se chamam **jetton**. Cada um tem um contrato principal (**master**), e cada dono tem a sua própria **carteira de jetton**. Lembra as contas de token do capítulo 18, só que aqui é um contrato independente. No USDT, o contrato principal é `EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs`.



cada carta é tratada numa transação à parte

Não é invenção, e sim uma transferência real de 11,99 USDT, feita na madrugada de 27 de novembro de 2026:

Transação de quem	O que recebeu	O que enviou adiante
carteira do remetente	a ordem assinada	"transfira" e 0,05 Gram para as despesas
a carteira de USDT dele	"transfira"	"credite"
carteira de USDT do destinatário	"credite"	a notificação e o troco
carteira do destinatário	a notificação e 1 nanogram	nada
carteira do remetente	o troco, cerca de 0,0494 Gram	nada

Tudo coube em um segundo. O dinheiro das despesas vai com folga, e a sobra volta.

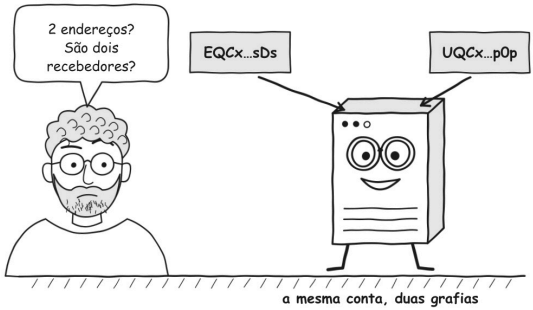
Um detalhe curioso: a quarta transação está marcada como interrompida (`aborted`). Faltou gas ao destinatário para executar o programa: chegou um único nanogram. Mas os USDT já tinham sido creditados na terceira transação. Não se pode julgar por um registro só, nem para um lado nem para o outro.

Dá para ver a cadeia inteira com uma requisição ao indexador. No nó público ele está fechado, por isso isto é um esboço para um endpoint pessoal:

```
curl -G https://SEU_ENDPOINT/api/v3/traces \
  --data-urlencode "trace_id=7rg6DNoAZeR1KPVJHd1FWIc/dobTITAsXqK1Y2kuQlM="
```

Uma conta, dois endereços

O nome verdadeiro de uma conta é o "bruto": o número da parte da rede e 64 caracteres. Para as pessoas se mostra outra forma, de 48 caracteres, com dígito de controle. E uma mesma conta tem várias dessas formas.



Vamos perguntar ao nó sobre o endereço do USDT:

```
curl -s "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs" |
  python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['raw_form'])
print(r['bounceable']['b64url'])
print(r['non_bounceable']['b64url'])"
```

```
0:b113a994b5024a16719f69139328eb759596c38a25f59028b146fecdc3621dfe
EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDs
UQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_p0p
```

- `EQ…` : endereço com retorno (bounceable); é assim que se escrevem os contratos comuns.
- `UQ…` : sem retorno; é a forma recomendada para carteiras.
- Na rede de testes, as formas começam com `kQ` e `0Q` .

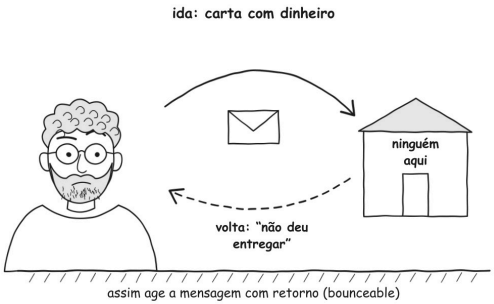
Por isso, no banco de dados da loja, **compara-se a forma bruta**, e não os textos com `EQ` e `UQ` .

O dígito de controle pega erros de digitação. Se no final, em vez de `sDs` , você escrever `sDt` , o nó responde com o código 422:

```
curl -s -o /dev/null -w '%{http_code}\n' \
  "https://public-ton-mainnet.rpcnode.dev/api/v2/detectAddress?address=EQCxE6mUtQJKFnGfaROTKOt1lZbDiiX1kCixRv7Nw2Id_sDt"
```

Carta para uma casa vazia

E se o destinatário não conseguir tratar a carta, ou o contrato dele ainda não existir? Depende de uma marca na mensagem.



A mensagem com retorno volta ao remetente com o dinheiro que sobrou, se ele bastar para o caminho de volta. A sem retorno fica com o destinatário. Uma carteira nova pode ainda não estar "implantada", isto é, o programa dela não foi gravado na rede. Por isso às carteiras se paga no endereço sem retorno.

Na prática: conferir a velocidade

Vamos descobrir o número do bloco principal mais recente e a hora em que saíram ele e o que veio mil antes:

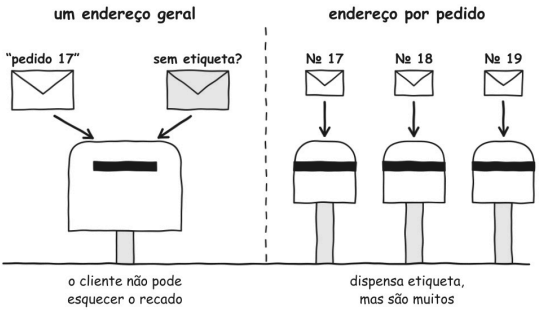
```
B=https://public-ton-mainnet.rpcnode.dev
last=$(curl -s $B/api/v2/getMasterchainInfo |
  python3 -c "import json,sys; print(json.load(sys.stdin)['result']['last']['seqno'])")
for s in $last $((last-1000)); do
  sleep 2
  curl -s "$B/api/v2/getBlockHeader?workchain=-1&shard=-9223372036854775808&seqno=$s" |
    python3 -c "import json,sys
r = json.load(sys.stdin)['result']
print(r['id']['seqno'], r['gen_utime'])"
done
```

```
108011078 1795732480
108010078 1795732076
```

Os segundos números são o tempo em segundos. A diferença é de 404 segundos para 1000 blocos, ou seja, cerca de 0,4 segundo por bloco.

Comentário em vez de endereço separado

A uma transferência na TON dá para anexar um **comentário** em texto. Daí as duas maneiras de saber de quem é o pagamento:



Com um endereço comum há menos trabalho, mas o cliente pode esquecer o comentário ou copiá-lo errado. Além disso, nem todas as carteiras sabem acrescentá-lo. Esses pagamentos são resolvidos na mão.

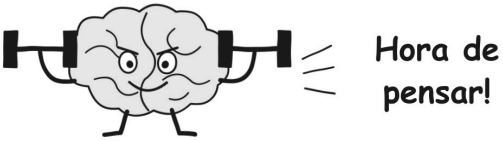
Cilada: "chegou a notificação, vou despachar"

Qualquer um pode escrever um contrato que mande à sua carteira uma notificação igual. Ou lançar um jetton próprio chamado "USDT". Por isso se confere:

- a notificação veio da **sua** carteira de USDT. O endereço dela se pergunta de antemão ao contrato principal verdadeiro (`get_wallet_address`);
- a quantia bate (o USDT tem seis casas decimais);
- o comentário ou o endereço se referem a este pedido, e este pagamento ainda não foi contabilizado;
- o registro entrou na cadeia principal (masterchain), ou seja, é definitivo.

A notificação só chega se o remetente tiver anexado para ela pelo menos um pouco de Gram. É o costume, mas um programa feito em casa pode esquecer. Aí os tokens chegam sem notificação.

Pense bem



1. No banco de dados está um endereço com `UQ`, mas o explorador mostra um com `EQ` e o mesmo miolo. Como saber se é a mesma conta?
2. A primeira transação da cadeia teve sucesso. Então os USDT chegaram?
3. O cliente pagou no endereço comum, mas esqueceu o comentário. O que a loja faz?

Respostas: 1) converter os dois para a forma bruta (`detectAddress`) e comparar. 2) Ainda não: é preciso que a carteira de USDT do destinatário os credite. 3) Não contabilizar automaticamente, e sim analisar na mão: quantia, remetente, número da transferência.

O essencial

- Na TON os contratos se correspondem: uma ordem gera uma cadeia de transações.
- Uma conta tem várias formas de endereço; compare a bruta.
- O jetton se confere pelo contrato principal e pela sua carteira de jetton.

A seguir: capítulo 20. Outras abordagens: Sui e Aptos, XRP Ledger e Stellar, Cardano, Polkadot, Cosmos.

Capítulo 20. Outras abordagens: Sui e Aptos, XRP Ledger e Stellar, Cardano, Polkadot, Cosmos

“E XLM, vocês aceitam?”

Um cliente pergunta se dá para pagar pela Stellar. Outro quer mandar USDC na Sui. Artiom já desvendou o Bitcoin e a Ethereum. Esse conhecimento basta? Nem tanto: aqui os costumes são outros.

Entendendo: cinco perguntas para uma rede nova

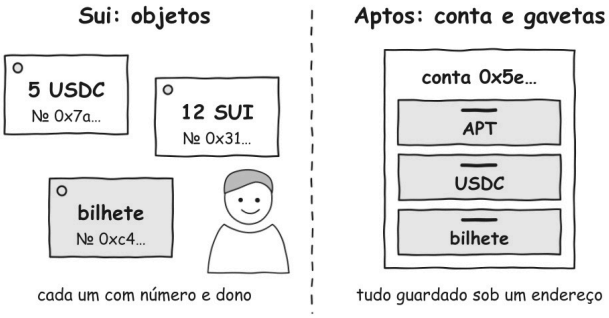
Não é preciso estudar uma rede desconhecida por inteiro. Para começar, bastam cinco respostas:

1. Onde fica o dinheiro: numa conta ou em pedaços separados?
2. O que fazer de antemão para conseguir receber uma transferência?
3. Como saber a que pedido ela se refere?
4. Quando ela é definitiva?
5. Com que pagar a taxa quando for preciso mandar o dinheiro adiante?

Sui e Aptos: mesma língua, construção diferente

Nas duas redes os programas são escritos na linguagem **Move**. Ela cuida para que uma coisa de valor não possa ser simplesmente copiada. Mas o dinheiro é guardado de formas diferentes.

- A **Sui** guarda tudo em **objetos**: são itens, cada um com seu número. O dinheiro pode ficar em objetos-moeda separados, quase como os pedaços do Bitcoin. Desde 2026 um endereço também pode ter um saldo comum. O saldo total é a soma dos dois.
- A **Aptos** é mais próxima da conta de sempre. Debaixo do endereço fica empilhado o patrimônio, chamado de **recursos**. Não é preciso abrir a conta antes.



Nas duas redes, a transferência costuma se tornar definitiva em menos de um segundo. A taxa é paga na moeda própria: SUI ou APT. Na Sui há uma exceção: desde 2026, algumas stablecoins, entre elas o USDC, podem ser simplesmente enviadas sem taxa (a partir de 0,01).

Nas duas, o endereço é `0x` e em geral 64 caracteres. O começo é como na Ethereum, mas o comprimento é outro: lá, depois do `0x`, vêm 40 caracteres.

Na prática: ache o USDC em duas redes

A RpcNode não tem nós públicos gratuitos para as redes deste capítulo. Por isso vamos perguntar nos endereços abertos de quem desenvolve essas próprias redes. Eles são compartilhados e têm limites: servem para conhecer, não para uma loja.

Sui, numa forma chamada GraphQL (você lista os campos que quer):

```
curl -s https://graphql.mainnet.sui.io/graphql \
  -H 'Content-Type: application/json' \
  -d '{"query":"{ coinMetadata(coinType: \"0xdba34672e30cb065b1f93e3ab55318768fd6fef66c15942c9f7cb846e2f900e7::usdc::USDC\") { symbol decimals } }"}'
```

```
{"data":{"coinMetadata":{"symbol":"USDC","decimals":6}}}
```

Aptos, o REST comum do capítulo 11:

```
curl -s https://api.mainnet.aptoslabs.com/v1/accounts/0xbae207659db88bea0cbead6da0ed00aac12edcdda169e591cd41c94180b46f3b/resource/0x1::fungible_asset::Metadata
```

```
{"type":"0x1::fungible_asset::Metadata","data":{"decimals":6,"icon_uri":"https://circle.com/usdc-icon","name":"USDC","project_uri":"https://circle.com/usdc","symbol":"USDC"}}
```

Nos dois casos, o token não se encontra pela palavra “USDC”, e sim pelo endereço comprido da lista do emissor, a empresa Circle. É a regra do capítulo 9. As casas decimais, em ambos, são seis.

Cuidado com exemplos antigos. Antes se consultava a Sui por JSON-RPC. Em julho de 2026 ele foi desligado nos nós da fundação Sui; agora há gRPC e GraphQL. Antes de copiar um exemplo de um artigo, olhe a data dele.

XRP Ledger e Stellar: redes de transferência

Há muito tempo as duas redes sabem não só transferir a própria moeda, mas também controlar tokens alheios. Aqui a página da cadeia se chama **ledger**. O XRP Ledger valida uma nova a cada 3–5 segundos; a Stellar, mais ou menos a cada 5 segundos. O que foi validado não se reescreve.

Mas há duas regras incomuns.

A primeira: um depósito pelo espaço. A conta não existe enquanto não houver nela uma quantia mínima, a **reserva**. E, para receber um token, por exemplo o USDC, o dono da conta abre antes uma **linha de confiança** (trust line): "sim, aceito guardar isto". Cada anotação dessas aumenta a reserva.

	XRP Ledger	Stellar
O endereço começa com	`r`	`G`
Mínimo na conta	1 XRP	1 XLM
Acréscimo por trust line	0,2 XRP (as duas primeiras podem sair sem ele)	0,5 XLM
Taxa mínima	0,00001 XRP	0,00001 XLM por operação
Número do pedido	destination tag	memo ou endereço com `M`

A rede pode mudar esses números por votação, por isso eles se perguntam ao nó:

```
curl -s https://s1.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"server_info","params":[{}]}' | grep -o '"validated_ledger":{[^}]*}'
```

```
"validated_ledger":{"age":10,"hash":"AD65FEE0E665A2CA5D08D3757A9E5A65047ECC2391B5E118F3912F25C76AB64F","seq":10859
3026,"base_fee_xrp":1E-5,"reserve_base_xrp":1E0,"reserve_inc_xrp":2E-1}
```

`1E-5` é 0,00001, `1E0` é 1, `2E-1` é 0,2. Agora a Stellar:

```
curl -s 'https://horizon.stellar.org/ledgers?order=desc&limit=1' \
  | grep -E '"(base_fee|base_reserve)_in_stroops"'
```

```
      "base_fee_in_stroops": 100,
      "base_reserve_in_stroops": 5000000,
```

Um XLM tem dez milhões de **stroops**. Então a taxa é de 0,00001 XLM, e uma cota de reserva, 0,5 XLM (a conta precisa de pelo menos duas).

Um endereço para todos

A segunda regra. Abrir um endereço separado para cada pedido sai caro aqui: em cada um ficaria uma reserva. Por isso a loja mostra um só endereço e pede que se anote um número: **destination tag** no XRP Ledger, **memo** na Stellar.

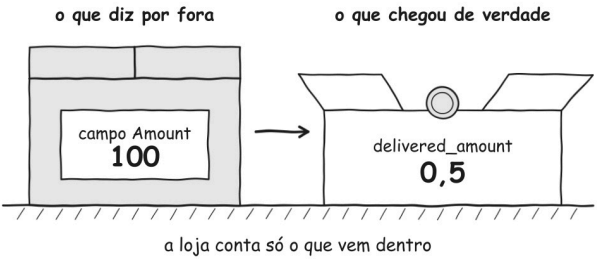


Se o cliente esquece o número, o dinheiro chega, mas não se sabe pelo quê. O que ajuda:

- no XRP Ledger dá para ligar a configuração `RequireDest`: a rede não deixa passar transferência sem número;
- o número pode ir escondido no próprio endereço: o X-address no XRP Ledger, o endereço com `M` na Stellar, se a carteira do cliente os entende.

Cilada: 100 na etiqueta, 0,5 dentro

No XRP Ledger o remetente pode permitir um **pagamento parcial** (partial payment): a transferência é considerada bem-sucedida mesmo que chegue menos. Aí o campo `Amount` não diz "chegou", e sim "no máximo". Quanto chegou de verdade fica anotado em `delivered_amount`.



Eis um registro real da rede; vamos perguntar por ele pelo hash:

```
curl -s https://s2.ripple.com:51234/ -H 'Content-Type: application/json' \
  -d '{"method":"tx","params":[{"transaction":"18F797F64D0255522223B1A404BA9CFF7EAE28E2ABB74CC91DD986E162D42DE
C"}]}' \
  | grep -o '"Amount":{[^}]*}\|"delivered_amount":{[^}]*}\|"TransactionResult":"[^"]*"'
```

```
"Amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"9.552474017069"}
"TransactionResult":"tesSUCCESS"
"delivered_amount":{"currency":"XXX","issuer":"rPXeYRpA9Ne3H7fwcM26ZNe98M3rkzTeg6","value":"4.5344917829"}
```

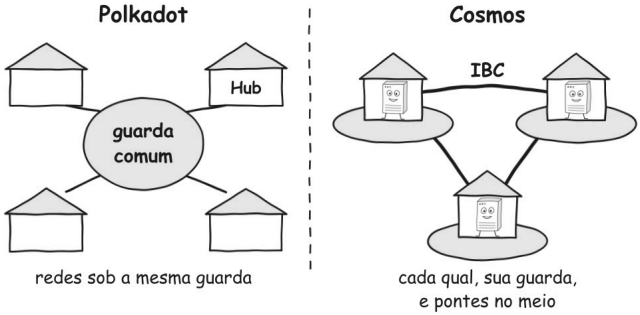
Resultado `tesSUCCESS`, em `Amount` cerca de 9,55, mas chegaram cerca de 4,53. Por si só, um registro assim não é golpe. Mas um golpista pode escrever uma quantia alta, mandar uma migalha e esperar a mercadoria. **Conte só o** `delivered_amount` e confira qual é o token e de qual emissor.

Cardano: pedaços com anexos

A Cardano é construída de um jeito mais próximo do Bitcoin (capítulo 14): não há saldo, e sim pedaços não gastos. Só que o pedaço sabe fazer mais, e o modelo se chama **UTXO estendido** (eUTXO).

- Junto com a moeda ADA, no mesmo pedaço, vão outros tokens. Eles não precisam de contrato próprio.
- O token não viaja sozinho: no mesmo pedaço tem de haver um pouco de ADA. Por isso, com o token chegam também umas moedinhas.
- A taxa depende do tamanho da transferência em bytes.
- A certeza cresce aos poucos: quanto mais blocos por cima, mais seguro.

Polkadot e Cosmos: não uma rede, mas muitas



A **Polkadot** é uma cadeia central que protege as redes ligadas a ela. Uma mudança importante: desde novembro de 2025, os saldos dos usuários já não vivem na cadeia central, e sim na rede **Polkadot Hub** (também chamada de Asset Hub).

- Lá estão o USDT (ativo número 1984) e o USDC (número 1337).
- A conta tem um limiar de sobrevivência, o **existential deposit**: 0,01 DOT. Se não houver nem DOT nem USDT ou USDC na quantidade necessária, a conta é apagada.
- O endereço começa com `1`, como o antigo do Bitcoin. Não confunda.

A **Cosmos** são redes independentes montadas com peças comuns. Cada uma tem sua segurança, sua moeda e seu prefixo de endereço (no Cosmos Hub, `cosmos1…`). Os blocos se tornam definitivos rápido. Na transferência há um campo **memo** para o número do pedido.

As redes são ligadas pelo protocolo **IBC**. Um token que veio de uma rede vizinha não se chama pela palavra de sempre, e sim por `ibc/` seguido de um selo comprido, por exemplo `ibc/7F1D3FCF…`. O selo diz por qual caminho ele chegou.

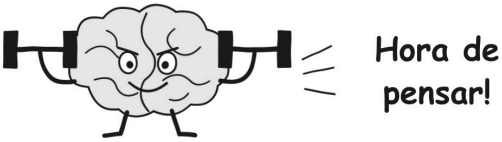
E uma stablecoin também pode ir embora. A Circle anunciou que vai encerrar a emissão de USDC na rede Noble, da família Cosmos: a emissão de novas unidades está desligada desde 13 de outubro de 2026, e em 12 de janeiro de 2027 o USDC será parado ali. Aos detentores se recomenda transferi-lo antes.

Tudo junto

Rede	Endereço	Preciso antes	Definitivo
Sui	`0x…`	nada	menos de um segundo
Aptos	`0x…`	nada	menos de um segundo
XRP Ledger	`r…`	reserva, trust line	3–5 segundos
Stellar	`G…`	reserva, trust line	cerca de 5 segundos
Cardano	`addr1…`	nada	aos poucos
Polkadot Hub	`1…`	saldo acima do limiar	por uma marca
Cosmos Hub	`cosmos1…`	nada	rápido

A trust line só é necessária para tokens. Verificado em 27 de novembro de 2026; as regras mudam.

Pense bem



1. Artiom criou um endereço novo e vazio na Stellar e já emitiu uma cobrança em USDC. O que está errado, e de quantos XLM se precisa no mínimo?
2. Chegou uma transferência: `Amount` de 100 USDC, `delivered_amount` de 0,5, resultado `tesSUCCESS`. Despachar café no valor de 100?
3. Os clientes no XRP Ledger esquecem a destination tag. O que ligar?

Respostas: 1) a conta ainda não existe e a trust line não está aberta; são precisos pelo menos 1,5 XLM de reserva e um pouco para a taxa. 2) Não: chegou 0,5. 3) A configuração RequireDest.

O essencial

- A uma rede desconhecida, faça as cinco perguntas antes de mostrar o endereço ao cliente.
- No XRP Ledger e na Stellar são precisos reserva, trust line e número do pedido; no XRP Ledger, conte só o `delivered_amount`.
- Hábitos da Ethereum e exemplos antigos não se transportam para outra rede.

A seguir: capítulo 21. Qual rede escolher.

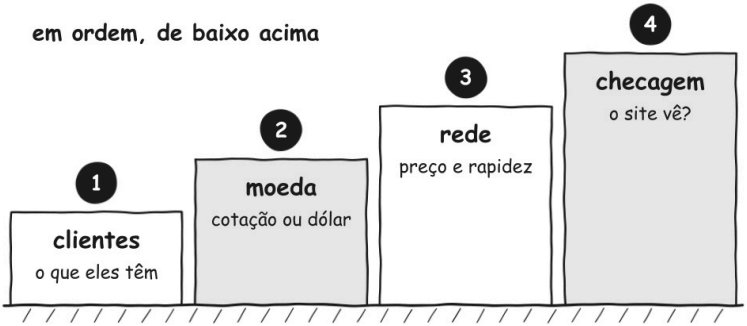
Capítulo 21. Qual rede escolher

“Afinal, qual delas eu pego?”

Em sete capítulos se juntaram muitos nomes: Bitcoin, Ethereum, TRON, Solana, TON e mais uma dezena. Artiom tem uma loja virtual e uma oficina de torra de café. Ligar tudo? Escolher uma? Por qual critério?

Entendendo: quatro passos

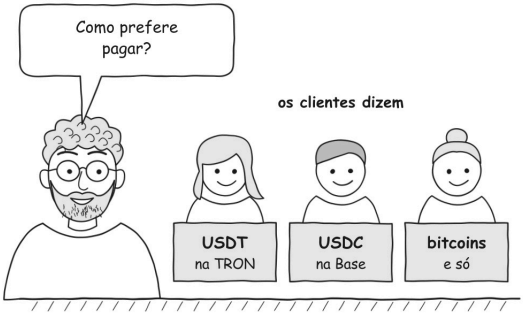
Não existe “a melhor rede”. Existe a rede que serve aos seus clientes e ao seu pedido. O prático é seguir em ordem.



Aqui se trata de com o que é prático receber pagamentos. Não é um conselho sobre onde investir dinheiro.

Passo 1. Pergunte aos clientes

Paga quem já tem as moedas. Se ele as tem numa rede e você aceita em outra, o mais provável é que não haja pagamento.



Pergunte no boletim, no chat, na hora do pedido. Duas perguntas bastam: “O que você tem?” e “Em que rede?”

Passo 2. Qual moeda

	Moeda própria da rede	Stablecoin (USDT, USDC)
Preço em dólares	varia, às vezes bruscamente	procura se manter perto de um dólar
Quem manda	as regras da rede	há um emissor, que pode congelar (capítulo 9)
Com que se paga a taxa	com ela mesma	em geral com a moeda da rede, não com a stablecoin

Se os preços do café de Artiom são em dólar ou atrelados a ele, com stablecoin há menos surpresas. Mas isso também é confiar: na empresa que a emite.

Passo 3. Qual rede

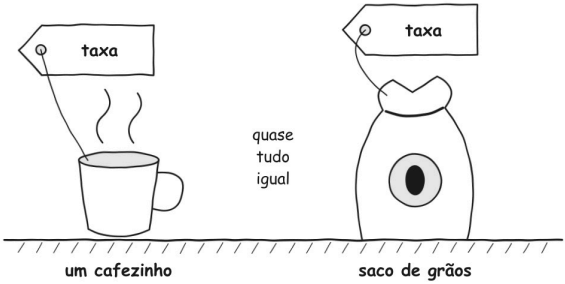
A mesma stablecoin não vive em todo lugar. Aqui estão as redes dos capítulos 14–20. “Sim” quer dizer: no site do emissor consta, para essa rede, o endereço justamente dessa moeda.

Rede	Taxa	USDT da Tether	USDC da Circle	Não esquecer
Bitcoin	BTC	não	não	confirmações
Litecoin, Doge etc.	própria	não	não	redes distintas
Ethereum	ETH	sim	sim	o gas
Base, Arbitrum	ETH	não	sim	finalized
BNB Smart Chain	BNB	não	não	o emissor
TRON	TRX	sim	não	a energia
Solana	SOL	sim	sim	contas de token
TON	Gram	sim	não	as mensagens
Aptos	APT	sim	sim	endereço longo
Sui	SUI	não	sim	gRPC, GraphQL
Polkadot Hub	DOT	sim	sim	limiar da conta
XRP Ledger	XRP	não	sim	reserva, tag
Stellar	XLM	não	sim	reserva, memo
Cardano	ADA	não	não	ADA com o token

Segundo tether.to/en/supported-protocols e developers.circle.com em 27 de novembro de 2026. “Não” não significa que não haja lá um token com esse nome: confira quem o emite (capítulo 9). Na Cosmos, tudo depende da rede específica.

Preço da transferência e tamanho do pedido

A taxa não depende da quantia, e sim do “peso” da transferência e da lotação da rede (capítulo 7).



Para um saco de grãos é uma ninharia. Para uma xícara de café, a taxa pode sair mais cara que a própria xícara. Por isso, para pedidos pequenos, vê-se onde a transferência é mais barata. Os preços mudam, então descubra-os na carteira antes de enviar, e não numa tabela antiga.

E lembre o caminho de volta: para mandar adiante o que recebeu, o endereço precisa ter moeda para a taxa. Cem endereços são cem transferências pagas.

Quanto esperar

Definitivo em	Redes
segundos ou menos	BNB Smart Chain, Solana, TON, Aptos, Sui, XRP Ledger, Stellar
cerca de um minuto	TRON
dezenas de minutos	Ethereum, Base, Arbitrum One
cerca de uma hora	Bitcoin (6 confirmações)

Valores aproximados, conforme os capítulos 8 e 17–20.

Na prática: que idade tem o bloco definitivo

Às redes EVM vamos pedir o bloco `finalized` (capítulo 8) e calcular há quantos segundos ele saiu:

```
for net in ethereum base arb bsc; do
  t=$(curl -s https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber","params":["finalized",false]}' \
    | grep -o '"timestamp":"0x[0-9a-f]*"' | cut -d'"' -f4)
  echo "$net: $(( $(date +%s) - t )) s"
  sleep 1.5
done
```

```
ethereum: 929 s
base: 1343 s
arb: 990 s
bsc: 1 s
```

Na Ethereum são cerca de 15 minutos; na Base, cerca de 22. Na BNB Smart Chain, um segundo. Os seus números serão outros: a marca anda aos saltos. Esperar a marca mais segura por um pedido pequeno não é obrigatório, mas então o risco é seu.

Passo 4. O seu site vai enxergar

Antes de publicar o endereço no site, confira:

- se o provedor tem endpoint para essa rede (capítulo 13);
- se há, justamente para ela, avisos de pagamento. Não é a mesma coisa. Na RpcNode se pode obter endpoint para mais redes do que aquelas que são processadas para eventos (capítulos 26 e 27);
- se há uma rede de testes, para experimentar sem dinheiro de verdade;
- se a sua carteira, e o lugar onde você troca moedas, suportam essa moeda **exatamente nessa rede**.

E dê uma olhada no plano: no Free da RpcNode há endpoints só para Ethereum e Solana (capítulo 22).

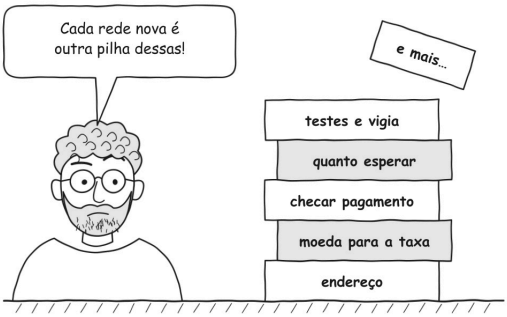
Exemplo: como Artiom decide

Digamos que ele perguntou a 20 clientes. Os números são didáticos; na sua loja serão outros.

Com o que querem pagar	Quantas pessoas	Decisão
USDT na TRON	9	ligar já
USDC na Base	5	ligar já
BTC	3	depois, se pedirem
ETH	2	depois
XLM	1	por ora, não

Duas redes atendem 14 das 20 pessoas. Às outras, Artiom escreve com franqueza que ainda não aceita tudo.

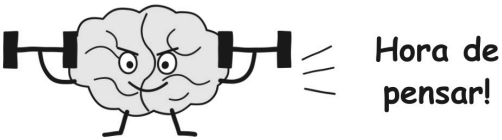
Cilada: “vou ligar tudo de uma vez”



Uma linha a mais na página de pagamento parece de graça. Mas atrás dela vêm um endereço próprio, uma moeda própria para a taxa, regras próprias de verificação e surpresas próprias depois das atualizações. Comece com uma ou duas e acrescente conforme pedirem.

E a segunda cilada: escrever na página de pagamento apenas “USDT”. O cliente vai escolher a rede sozinho e pode escolher a errada (capítulo 16). Escreva a moeda e a rede juntas, em letras grandes: “USDT na rede TRON”.

Pense bem



1. Um pedido de 5 dólares, e a taxa na rede escolhida hoje é de 2 dólares. Para quem isso é ruim e o que fazer?
2. Um cliente pede para aceitar USDT na Base. O que mostra a lista no site da Tether?
3. Artiom quer aceitar oito redes desde o primeiro dia. O que dizer a ele?

Respostas: 1) para o cliente: a taxa é 40% do pedido por cima; propor uma rede onde a transferência esteja mais barata agora. 2) A Base não está lá: antes de aceitar um token com esse nome, descubra quem o emite. 3) São oito pilhas de preocupações; comece com uma ou duas, as que os clientes mais precisam.

O essencial

- A escolha segue uma ordem: clientes, moeda, rede, verificação.
- Compare a taxa com o tamanho do pedido e leve em conta quanto esperar.
- Comece com uma ou duas redes e sempre diga a moeda junto com a rede.

Fim da quarta parte. Artiom sabe em que as redes diferem e decidiu por onde começar. **A seguir:** parte 5, capítulo 22. Os primeiros cinco minutos na RpcNode.

Capítulo 22. Os primeiros cinco minutos na RpcNode

“Quero uma porta só minha”

Ao longo do livro todo, Artiom bateu na porta dos nós públicos. Eles respondiam sem cadastro, mas essa porta é de todo mundo. A loja precisa de uma própria. Demora para fazer? O que é preciso preencher?

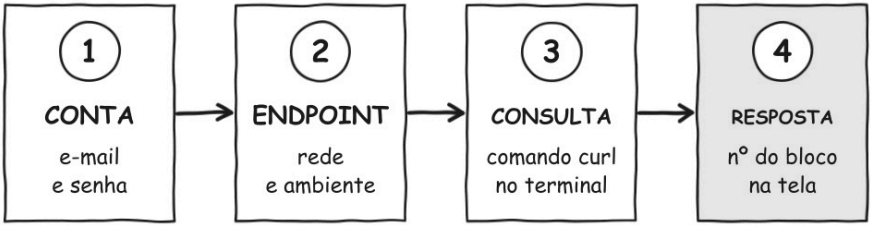
Entendendo: a porta comum e a própria

O endereço pelo qual um programa faz perguntas a um nó se chama **endpoint** (capítulo 3). Na RpcNode há dois tipos.



	Endereço público	Endpoint pessoal
Cadastro	não precisa	precisa
Chave	não há	vai embutida no endereço
Limite	5 requisições por segundo por IP	conforme o plano
Métodos pesados	fechados	abertos
Arquivo (capítulo 24)	fechado	nos planos pagos
WebSocket e gRPC	não	onde a rede suporta

O seu endpoint aparece em quatro passos.



Passo 1. Cadastro

Abra `cabinet.rpcnode.dev/register` ou clique em "Começar" no site rpcnode.dev. O painel é a sua página no serviço: ali ficam os endpoints, o consumo e o plano.

1. Preencha o **E-mail**, invente uma **Senha** e repita-a. O nome é opcional.
2. Leia e marque "Concordo com os termos de serviço". Passe pela verificação "você não é um robô", se ela aparecer.
3. Clique em "Continuar".
4. Chega um código de uso único por e-mail. Digite-o e clique em "Verificar e continuar".

Há também um atalho: entrar com o Google ou o GitHub.

A conta nova recebe o plano gratuito **Free**, sem necessidade de cartão. Ele tem 15 milhões de créditos por mês, 15 requisições por segundo, 3 endpoints e duas redes: Ethereum e Solana. O que são créditos, veremos no capítulo 25.

Passo 2. O endpoint

No menu do painel, escolha "Endpoints" e clique em "Novo endpoint". Abre-se uma janela de três etapas:

Etapa na janela	O que fazer
Selecionar rede	por exemplo, Ethereum
Selecionar ambiente	rede principal ou de testes (sobre elas, mais abaixo)
Confirmar	conferir e clicar em "Criar"

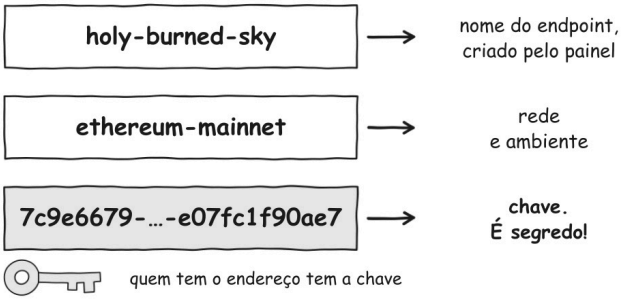
Ambiente é a variante da rede: a de verdade ou a de treino. O nome do endpoint o próprio painel inventa, com três palavras em inglês.

No plano Free, antes de criar, você verá um aviso sobre os nós de arquivo. Não é erro; ele é tema do capítulo 24.

De que é feito o endereço

Eis um exemplo da documentação da RpcNode. É uma linha só, aqui quebrada antes da barra:

```
https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
/7c9e6679-7425-40de-944b-e07fc1f90ae7
```



A cauda comprida depois da barra se chama **UUID**: um número praticamente impossível de adivinhar. Ele é a chave. Não é preciso anexar à requisição nenhuma senha à parte.

Copie o endereço pelo botão do painel, não o digite à mão. Na tela uma parte da chave fica oculta, mas ela é copiada inteira.

No menu há também "Chaves de API". É outra coisa: com elas se assinam comandos ao próprio serviço, por exemplo "crie um pagamento". Para as perguntas comuns ao nó elas não são necessárias.

Passo 3. Na prática: a primeira requisição

Pomos o endereço numa variável e perguntamos o número do último bloco. Isto é um **esboço**: no lugar do texto, coloque o seu endereço.

```
export RPC_URL='SEU_ENDERECO_DO_PAINEL'
curl -s "$RPC_URL" \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

O endereço pessoal só você tem, por isso no livro mostramos a resposta às mesmas linhas com o público: `RPC_URL='https://public-ethereum-mainnet.rpcnode.dev'`.

```
{"jsonrpc":"2.0","id":1,"result":"0x19464e4"}
```

`0x19464e4` é o bloco número 26.502.372. No capítulo 3 o nó informava 26.502.035: a rede teve tempo de somar 337 blocos.

Para a Solana, no lugar de `eth_blockNumber` escreva `getSlot`.

Na página “Visão geral” do painel há a lista “Primeiros passos”. Nela está um comando pronto, já com o seu endereço, e o aviso “Aguardando a primeira requisição…”. E na página do endpoint, na aba “Ao vivo”, aparecem as próprias requisições: hora, método, resposta e latência.

Se não deu certo

Vamos perguntar de propósito com uma chave inventada, só de zeros:

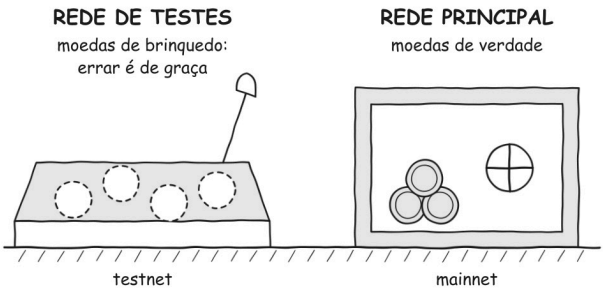
```
HOST=https://holy-burned-sky-ethereum-mainnet.rpcnode.dev
KEY=00000000-0000-0000-0000-000000000000
curl -s -w '  HTTP %{http_code}\n' $HOST/$KEY \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,
"message":"endpoint not found"}}  HTTP 404
```

`endpoint not found` quer dizer “esse endpoint não existe”. Verifique se o endereço foi copiado inteiro. Não repita um endereço errado em laço: quem insiste em tentativas deixa de ser ouvido por um tempo.

Redes de testes: um campo de treino com dinheiro de brinquedo

A **rede de testes** (testnet) funciona pelas mesmas regras da verdadeira, mas suas moedas não valem nada. A verdadeira se chama **mainnet**. Aqui se pode errar de graça.



As moedas de teste se pegam numa “torneira” (faucet): uma página que as envia ao endereço que você indicar.

Rede	Ambientes de teste na RpcNode, por exemplo
Ethereum	Sepolia, Hoodi
Solana	testnet, devnet
Bitcoin	testnet4, signet
TRON	Nile, Shasta

Para aplicações na Ethereum, o site ethereum.org recomenda a Sepolia. A Hoodi é para quem testa a própria rede e os validadores.

Há também redes de testes sem cadastro: em 27 de novembro de 2026, na lista de nós públicos, são a Bitcoin testnet4, a TRON Nile e a TON testnet. Vamos perguntar à TRON, a verdadeira e a de testes, o número da rede (capítulo 16):

```
for host in public-tron-mainnet public-tron-nile; do
  echo -n "$host: "
  curl -s https://$host.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_chainId","params":[]}'
  echo; sleep 2
done
```

```
public-tron-mainnet: {"jsonrpc":"2.0","id":1,"result":"0x2b6653dc"}
public-tron-nile: {"jsonrpc":"2.0","id":1,"result":"0xcd8690dc"}
```

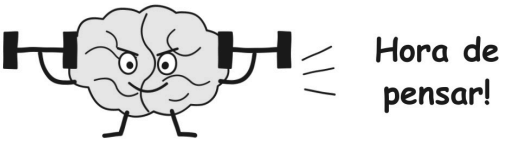
Os números são diferentes: são dois cadernos distintos. O endpoint também é preso a uma rede e a um ambiente. Para passar da rede de testes à verdadeira, cria-se um novo.

Cilada: "vou colar o endereço direto na página do site"

Tudo o que vai para o código da página qualquer visitante vê. Ele pega o seu endereço e passa a gastar o seu limite.

- Consulte o nó a partir do seu **servidor**, e não do navegador do cliente.
- Guarde o endereço nas configurações do servidor, e não no código nem num repositório compartilhado.
- Na página do endpoint, em "Configurações", preencha a "Lista de IPs permitidos": assim um computador estranho recebe uma recusa.
- As "origins permitidas" impedem que o endpoint seja chamado pelo navegador a partir de sites alheios. Mas essa lista não protege contra um programa fora do navegador.
- O endereço vazou? Crie um novo, mude o site para ele e apague o antigo: não será possível trazê-lo de volta.

Pense bem



1. Artiom escreveu o endereço do endpoint num script da página da loja, para mostrar o número do bloco. Quem vai ver a chave?
2. Tudo foi testado na Sepolia. Dá para receber pagamentos de verdade com esse mesmo endpoint?
3. Vem `endpoint not found`. Por onde começar a procurar?

Respostas: 1) qualquer visitante. 2) Não: é preciso um endpoint na mainnet. 3) Comparar o endereço com o que está no painel: copiá-lo de novo, inteiro.

O essencial

- O endpoint pessoal é um endereço com a chave dentro: guarde-o como uma senha.
- O endpoint é preso a uma rede e a um ambiente.
- Primeiro a rede de testes, depois a verdadeira.

A seguir: capítulo 23. O que acontece com a requisição: o gateway, a escolha do nó e as três regiões.

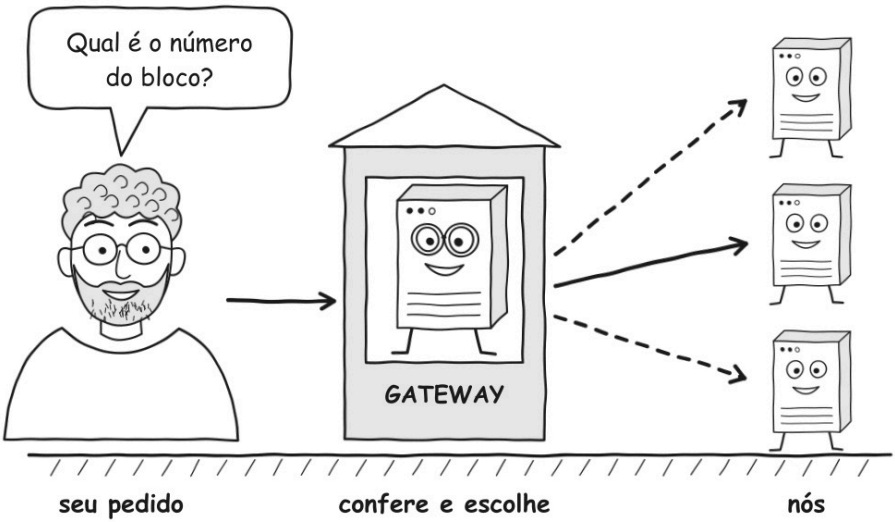
Capítulo 23. O que acontece com a requisição

“E quem foi que me respondeu?”

Artiom mandou uma requisição e recebeu o número do bloco. Mas para onde foi a pergunta? Que nó respondeu? E o que acontece se ele quebrar no auge de uma promoção?

Entendendo: na entrada há um gateway

A sua requisição não vai direto ao nó. Primeiro ela chega ao **gateway**. É um programa despachante: recebe as requisições, confere e as entrega a um nó adequado.



O que o gateway confere no caminho

A ordem lembra uma portaria. Não passou numa verificação, a requisição não segue adiante.

Verificação	Se não passar
Se esse endpoint existe e o endereço está certo	HTTP 404, `-32000`
Se ele não está desativado; se o seu IP e o seu site são permitidos	HTTP 403, `-32000`
Se você não está perguntando com frequência demais	HTTP 429, `-32005`
Se essa rede tem esse método	HTTP 404, `-32601`
Se há créditos suficientes	HTTP 402, `-32006`

Cada rede tem sua lista de métodos. Vamos pedir um que não está nela:

```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_noSuchMethod","params":[]}'
```

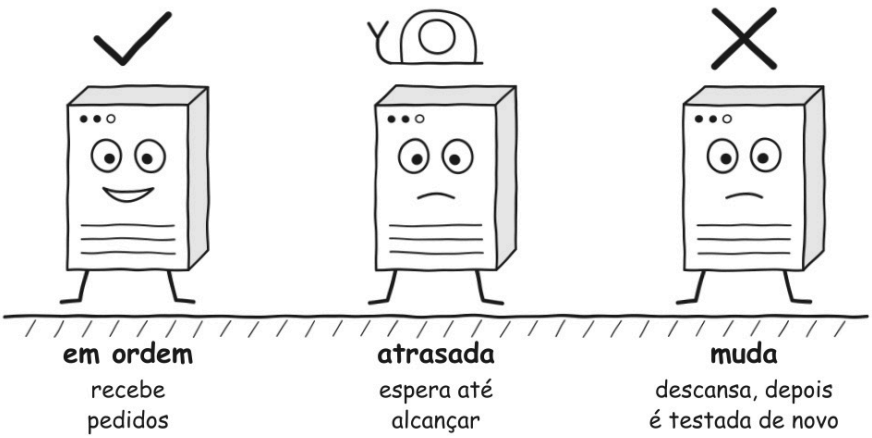
```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,
"message":"method not found: eth_noSuchMethod"}}  HTTP 404
```

Essa pergunta nem chegou ao nó: o gateway respondeu sozinho.

Escolha do nó: balanceamento

Uma rede pode ter vários nós. Distribuir as requisições entre eles se chama **balanceamento**. É como os caixas de uma loja: o próximo cliente vai para onde está mais livre.

Antes, o gateway vê se o nó está em ordem.



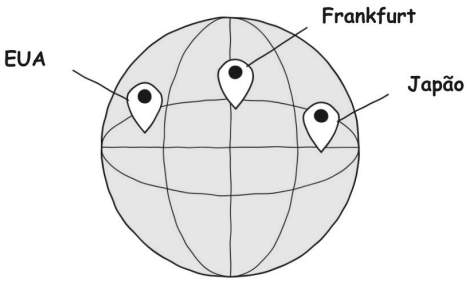
- O nó **atrasado** responde animado, mas com o ontem (capítulo 12). Enquanto houver nós em dia, as requisições vão para eles.

- Entre os nós saudáveis, o ocupado recebe menos e o livre, mais.
- A conexão caiu ou o nó devolveu um erro interno: a mesma requisição vai para o próximo. Você nem percebe.
- Um nó que falhou várias vezes seguidas é deixado em paz por um tempo e depois é testado de novo.

Há uma ressalva. Se não sobrar nenhum nó saudável, o gateway tenta os atrasados mesmo assim: às vezes uma resposta velha é melhor que nenhuma. Por isso, onde o frescor importa, confira você mesmo a hora do bloco (capítulo 12).

Três regiões

A RpcNode funciona em três regiões: Frankfurt, EUA e Japão. Quanto mais perto o nó, menor o caminho de ida e volta. O gateway procura pegar o nó saudável mais próximo dele e mantém os distantes de reserva.



três regiões da RpcNode (desenho, não mapa)

Nem toda rede tem nós nos três lugares. Onde e como responde a rede de que você precisa, mostra a página `rpcnode.dev/status`.

Na prática: ler os carimbos do envelope

Junto com a resposta vêm os **cabeçalhos**: marcas de serviço, como carimbos num envelope. Dois deles dizem em que país estava o gateway e em que país estava o nó:

```
for net in ethereum base arb; do
  echo "== $net"
  curl -s -D - -o /dev/null https://public-$net-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}' |
    grep -iE 'x-gateway-region|x-upstream-country'
  sleep 2
done
```

```
== ethereum
x-gateway-region: DE
x-upstream-country: DE
== base
x-gateway-region: DE
x-upstream-country: JP
== arb
x-gateway-region: DE
x-upstream-country: JP
```

`DE` é Alemanha, `JP` é Japão. Desta vez todas as requisições foram recebidas pelo gateway na Alemanha, que para Base e Arbitrum foi buscar a resposta no Japão. Para você as letras podem ser outras. São marcas de serviço: não construa o funcionamento do site em cima delas.

Erro do nó também é resposta

Vamos perguntar ao contrato do USDT: o que acontece se, de um endereço vazio, se transferir uma quantia enorme? O método `eth_call` só experimenta, sem enviar nada. No campo `data` vão colados o número da função de transferência, o destinatário (`TO`) e a quantia (`SUM`).

```
TO=000000000000000000000000000000000000000000000000000000000000dead
SUM=0000000000000000000000000000000000000000000000000fffffffffffffff
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[{
  "from":"0x0000000000000000000000000000000000000dEaD",
  "to":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "data":"0xa9059cbb'$TO$SUM'"},"latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,
"message":"invalid opcode: INVALID"}}  HTTP 200
```

É assim que esse contrato antigo diz "não vai dar". O nó fez o seu serviço: informou honestamente que o programa parou. Outro nó responderia a mesma coisa, por isso o gateway entrega essa resposta como ela é, com HTTP 200.

Quando ninguém respondeu

Pode ser pior: o gateway tentou os nós um a um e não veio resposta. Aí chegam o código `-32002`, o texto `upstream unavailable` ("nó indisponível") e um HTTP da família dos 500. Os créditos de uma chamada assim são devolvidos.

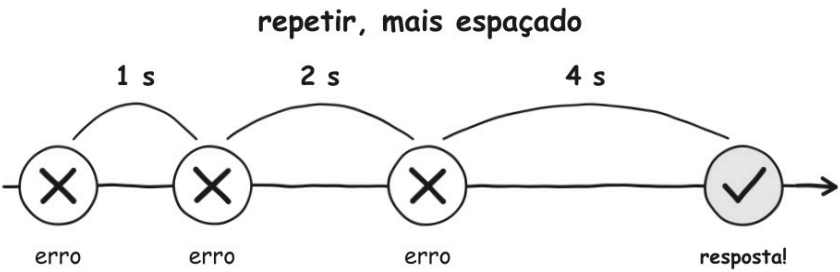
E se for um envio de dinheiro?

Se o nó **recusou** o envio de uma transação, essa é a resposta. Mas se ele **nem deu sinal**, o gateway pode passar a mesma transferência assinada a outro nó.

O dinheiro não sai duas vezes por causa disso: uma transação assinada tem sempre o mesmo hash, e a rede a aceita uma vez só. Agora, se quem não recebeu a resposta foi **você**, não assine uma transação nova no chute. Primeiro procure a antiga pelo hash (capítulo 7).

Na prática: repetir com juízo

O gateway tenta vários nós, mas a conexão entre você e o gateway também pode falhar. Repita você mesmo, com pausas crescentes: 1, 2, 4, 8 segundos. Em inglês isso se chama **backoff**.



Arquivo `ask.js`, execução com `node ask.js` (precisa do Node.js):

```
const url = 'https://public-ethereum-mainnet.rpcnode.dev'
const sleep = (ms) => new Promise((ok) => setTimeout(ok, ms))

async function ask(method, params = []) {
  for (const pause of [1, 2, 4, 8]) {
    try {
      const res = await fetch(url, {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ jsonrpc: '2.0', id: 1, method, params }),
        signal: AbortSignal.timeout(10000),
      })
      if (res.status !== 429 && res.status < 500) return await res.json()
      console.log('HTTP', res.status)
    } catch {
      console.log('sem conexão')
    }
    await sleep(pause * 1000)
  }
  throw new Error('não foi possível perguntar')
}

ask('eth_blockNumber').then(console.log)
```

```
{ jsonrpc: '2.0', id: 1, result: '0x19464ee' }
```

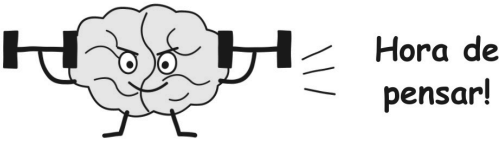
Só se repete o que pode dar certo na segunda vez: "frequência alta demais", queda de conexão, erros 500. Um erro na própria requisição não adianta repetir. E não esqueça de conferir o campo `error` depois da resposta (capítulo 11).

Cilada: "sem resposta" não quer dizer "sem pagamento"

Se não deu para perguntar, o site **não sabe** se o cliente pagou. Escreva "verificando o pagamento", e não "não pago", e pergunte mais tarde.

E lembre: um provedor com vários nós cai menos que um nó sozinho, mas não é “nunca” (capítulo 3).

Pense bem



1. Veio HTTP 200 com `execution reverted`. Adianta repetir a requisição cinco vezes?
2. O programa pediu `eth_blokNumber`. A requisição chegou ao nó?
3. Artiom mandou uma transferência e não recebeu resposta. O que fazer primeiro?

Respostas: 1) não: é uma resposta de verdade, não uma falha. 2) Não, o gateway recusou o método desconhecido. 3) Procurar a transação pelo hash.

O essencial

- O gateway confere a requisição e escolhe um nó saudável; se um não responde, tenta outro.
- O erro do próprio nó sobre a sua requisição chega até você como ele é.
- Mesmo assim, o seu programa deve repetir as requisições com pausas.

A seguir: capítulo 24. Histórico e dados de arquivo.

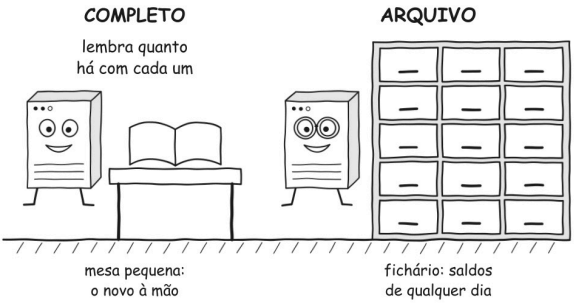
Capítulo 24. Histórico e dados de arquivo

“Quanto havia em primeiro de janeiro?”

O contador pede a Artiom: “Me diga quanto havia no endereço da loja no começo do ano”. Sobre hoje o nó responde na hora. E sobre aquele dia?

Entendendo: registros e saldos

Vamos relembrar o capítulo 10. **Histórico** são os blocos e as transferências. **Estado** é quem possuía o quê num momento específico.



O **nó completo** mantém à mão os saldos recentes. O **de arquivo** lembra os saldos em qualquer bloco, mas precisa de muito mais disco.

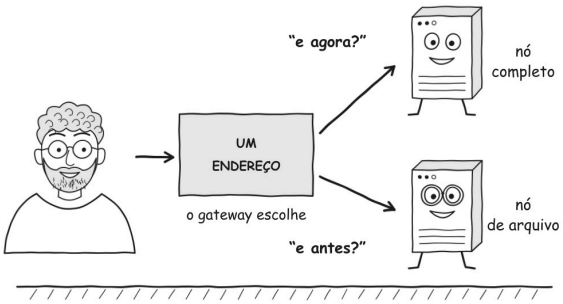
Que perguntas pedem arquivo

Pergunta	Exemplo
Saldo ou resposta de contrato num bloco antigo	`eth_getBalance`, `eth_call` com o número do bloco
Eventos de blocos passados	`eth_getLogs`
“Rebobinar” uma transação passo a passo (tracing)	`trace_…`, `debug_…`
Prova sobre o estado	`eth_getProof` num bloco antigo
Slots antigos da Solana	`getBlock` com um número antigo

Já “quanto há na conta agora” e um pagamento recente comum não exigem arquivo. No dia a dia, a loja em geral não precisa dele.

Como isso funciona na RpcNode

O endereço do endpoint é um só. Você não precisa escolher o nó: o gateway olha a pergunta e decide a quem entregá-la.



O arquivo está incluído nos planos pagos; no Free ele não existe. Ao criar um endpoint no Free, você verá no painel:

> "Os nós de arquivo não estão disponíveis no plano gratuito. Suas requisições são atendidas por nós comuns. Eles respondem tudo o que armazenam. O que só um nó de arquivo guarda (estado antigo, histórico distante, alguns traces) não está disponível: o nó retorna o próprio erro."

- Não há preço à parte para pergunta "antiga": paga-se pelo método (capítulo 25).
- A profundidade do arquivo varia de rede para rede. Antes de construir algo importante em cima dele, teste a sua pergunta mais antiga na sua rede.
- Os endereços públicos não atendem perguntas de arquivo.

Na prática: o nó público é rigoroso

Vamos perguntar o saldo não "agora", e sim pelo número de um bloco de apenas dez blocos atrás, cerca de dois minutos:

```
E=https://public-ethereum-mainnet.rpcnode.dev
H='Content-Type: application/json'
N=$(curl -s $E -H "$H" \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}' |
  grep -o '0x[0-9a-f]*')
OLD=$(printf '0x%x' $((N - 10)))
sleep 2
curl -s -w '  HTTP %{http_code}\n' $E -H "$H" \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBalance",
  "params":["0xdac17f958d2ee523a2206206994597c13d831ec7","'$OLD'"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,"message":
"archive data is not available on the public endpoint,
use a personal endpoint"}}  HTTP 403
```

Até um passado tão recente o endereço público considera arquivo. Pergunte a ele com as palavras do capítulo 8: `latest`, `safe` ou `finalized`.

Segundo teste: o nó pode ter jogado fora

Um bloco é histórico, não estado. Mas nem o histórico todos guardam. Desde julho de 2025, os programas de nó da Ethereum conseguem não manter o conteúdo dos blocos surgidos antes da passagem para o staking (capítulo 10). Vamos pedir o bloco 46.147 (`0xb443`), de agosto de 2015:

```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockByNumber",
  "params":["0xb443",false]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":4444,
"message":"pruned history unavailable"}}  HTTP 200
```

"Histórico podado indisponível." O nó diz com franqueza: isso eu não tenho. No plano pago, quando recebe de um nó comum um "isso eu não tenho", o gateway sabe repetir a pergunta a um de arquivo, se a rede tiver um.

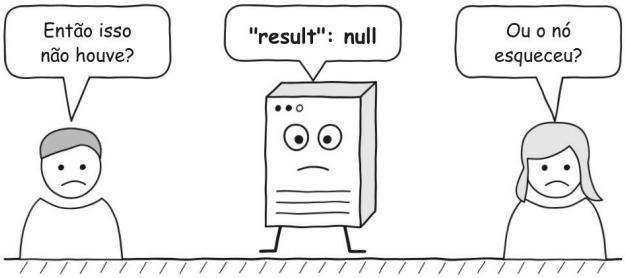
Cilada: "null quer dizer que não existiu"

Naquele mesmo bloco há uma transferência. Vamos perguntar por ela pelo hash e, em seguida, por um hash que simplesmente inventamos:

```
for hash in \
  0x5c504ed432cb51138bcf09aa5e8a410dd4a1e204ef84bfed1be16dfba1b22060 \
  0x0000000000000000000000000000000000000000000000000000000000000001
do
  curl -s https://public-ethereum-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionByHash",
    "params":["'$hash'"]}'
  echo; sleep 2
done
```

```
{"jsonrpc":"2.0","id":1,"result":null}
{"jsonrpc":"2.0","id":1,"result":null}
```

As respostas são iguais! A primeira transferência existiu de verdade; a segunda nunca existiu.



Não tire conclusões de um `null` isolado, sobretudo sobre registros antigos. Às vezes ele quer dizer "não existiu"; às vezes, "não guardo"; e, para uma transferência recente, "ainda não está em bloco".

Quanto um nó da Solana lembra

A Solana faz blocos várias vezes por segundo, e os registros se acumulam em montanhas. Vamos perguntar o número do slot mais recente e o bloco mais antigo que o nó ainda guarda:

```
for method in getSlot getFirstAvailableBlock; do
  echo -n "$method: "
  curl -s https://public-solana-mainnet.rpcnode.dev \
    -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"'$method'"}'
  echo; sleep 2
done
```

```
getSlot: {"jsonrpc":"2.0","result":474316453,"id":1}
getFirstAvailableBlock: {"jsonrpc":"2.0","result":473916707,"id":1}
```

A diferença é de cerca de 400.000 slots. Na verificação, isso era apenas um dia. E o bloco número 1000 já foi removido há muito tempo:

```
curl -s https://public-solana-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"getBlockTime","params":[1000]}'
```

```
{"jsonrpc":"2.0","error":{"code":-32001,"message":"Block 1000 cleaned up,
does not exist on node. First available block: 473916707"},"id":1}
```

De volta ao contador

O saldo "no começo do ano" é o saldo depois do último bloco do ano anterior (contamos a hora universal, UTC). O nó não informa de imediato o número do bloco a partir da data. Mas os

blocos vêm em ordem, e dá para procurar por **busca binária**: pegamos o bloco do meio, olhamos a hora dele e descartamos a metade que sobra.

Vamos achar o primeiro bloco de 2026. O script leva cerca de um minuto:

```
E=https://public-ethereum-mainnet.rpcnode.dev
ask() {  # method, params
  curl -s $E -H 'Content-Type: application/json' \
    -d '{"jsonrpc":"2.0","id":1,"method":"'$1'","params":'$2'}'
  sleep 2
}
WHEN=1767225600   # 1º de janeiro de 2026, 00:00 UTC
low=20000000
high=$(( $(ask eth_blockNumber '[]' | grep -o '0x[0-9a-f]*') ))
steps=0
while [ $low -lt $high ]; do
  mid=$(( (low + high) / 2 ))
  hex=$(printf '0x%x' $mid)
  t=$(ask eth_getBlockByNumber '["'$hex'",false]' |
    grep -o '"timestamp":"0x[0-9a-f]*"' | grep -o '0x[0-9a-f]*')
  if [ -z "$t" ]; then echo "sem resposta"; exit 1; fi
  if [ $(( t )) -lt $WHEN ]; then low=$(( mid + 1 )); else high=$mid; fi
  steps=$(( steps + 1 ))
done
echo "bloco $low, perguntas: $steps"
```

```
bloco 24136053, perguntas: 23
```

Entre milhões de blocos, o certo apareceu em 23 perguntas (no seu caso a contagem pode diferir em uma). Então o último bloco de 2025 é o 24.136.052 (`0x1704974`). O saldo nele se pergunta como no primeiro teste, só que com esse número. O endereço público vai recusar: é preciso um endpoint pessoal num plano pago. Para um token, no lugar de `eth_getBalance` usa-se `eth_call` perguntando `balanceOf` ao contrato.

Um trecho longo, aos pedaços

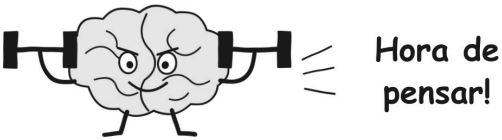
Um pedido de "todos os eventos do ano" o nó muito provavelmente não aguenta: a resposta é grande demais. O trecho de blocos é cortado em partes, que se perguntam uma de cada vez, juntando as respostas.

Isto é um **esboço**: `rpc` é uma função sua que consulta o endpoint pessoal, parecida com a `ask` do capítulo 23.

```
async function logsByChunks(rpc, filter, from, to, size = 1000) {
  const all = []
  for (let a = from; a <= to; a += size) {
    const b = Math.min(a + size - 1, to)
    const answer = await rpc('eth_getLogs', [{
      ...filter,
      fromBlock: '0x' + a.toString(16),
      toBlock: '0x' + b.toString(16),
    }])
    if (answer.error) throw new Error(answer.error.message)
    all.push(...answer.result)
  }
  return all
}
```

Não passa? Diminua o pedaço; cada um é uma chamada paga à parte. E o arquivo mais barato de uma loja é o **próprio**: anote os pagamentos no banco de dados assim que os vir (capítulo 26).

Pense bem



1. O nó devolveu `null` para o hash de uma transferência antiga. Isso prova que o cliente não pagou?
2. É preciso o saldo de USDC em 1º de julho. O plano Free basta?
3. É preciso examinar os eventos de 5000 blocos em pedaços de 1000. Quantas chamadas serão?

Respostas: 1) não: o nó podia não guardar esse registro. 2) O arquivo não está no Free, não se pode contar com ele. 3) Cinco.

O essencial

- Saldos antigos, eventos antigos e tracing são trabalho para um nó de arquivo.
- O arquivo está nos planos pagos; o endereço do endpoint continua o mesmo.
- `null` e resposta vazia ainda não provam que não houve nada.

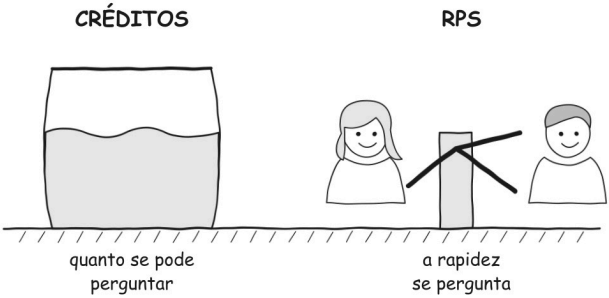
A seguir: capítulo 25. Planos, preço da chamada e limites.

Capítulo 25. Planos, preço da chamada e limites

“E quanto vai dar por mês?”

O endpoint funciona. Agora Artiom quer saber quanto a loja vai custar. Na página de preços há dois números principais: créditos e requisições por segundo. O que é o quê?

Entendendo: duas medidas diferentes



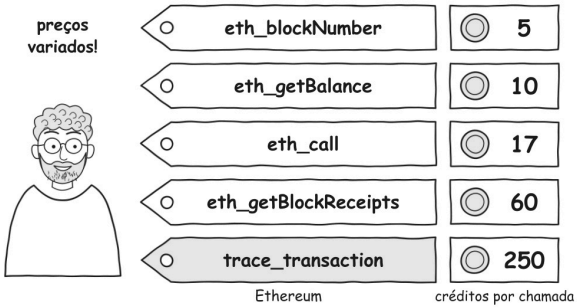
Créditos são o “dinheiro” interno do serviço: quanto se pode perguntar ao todo no mês. **RPS** são requisições por segundo: com que rapidez se pode perguntar. Os dois precisam ser suficientes.

Na prática: descobrir o preço da chamada

Perguntas diferentes custam diferente: a uma o nó responde de memória; por causa de outra, ele revira um bloco inteiro. Os preços são abertos e podem ser consultados sem cadastro. Vamos pegar alguns métodos da Ethereum (você vai precisar do `python3`):

```
curl -s https://rpcnode.dev/api/v1/public/networks/ethereum/methods |
python3 -c 'import json, sys
want = """eth_blockNumber eth_getBalance eth_call
eth_getLogs eth_getBlockReceipts trace_transaction""".split()
for m in json.load(sys.stdin)["methods"]:
    if m["method"] in want and m["variant"] == "default":
        print(m["method"], m["credits"])'
```

```
eth_blockNumber 5
eth_getBalance 10
eth_call 17
eth_getLogs 17
eth_getBlockReceipts 60
trace_transaction 250
```



- Um `trace_transaction` custa o mesmo que 50 perguntas sobre o número do bloco.
- Cada rede tem sua tabela de preços. No endereço, troque `ethereum` por `solana` ou `tron` e mude os nomes dos métodos.
- Um lote (capítulo 11) custa a soma das chamadas.
- As assinaturas por WebSocket também são pagas. Na Ethereum, abrir uma assinatura de blocos novos custa 10 créditos, e cada evento enviado, 2.

No painel, os preços estão reunidos na seção "Redes".

Os planos

Dados de 27 de novembro de 2026. Os valores atuais estão em `rpcnode.dev/pricing`.

Plano	US$ por mês	US$ por ano	Créditos por mês	RPS	Endpoints
Free	0	—	15 mi	15	3
Starter	19	190	50 mi	40	5
Build	36	360	100 mi	75	12
Grow	99	990	300 mi	150	25
Scale	219	2.190	700 mi	300	50
Business	449	4.490	1,5 bi	500	100

O plano não é só RPC. Veja o que mais ele traz (capítulos 26 e 27):

Plano	Endereços monitorados	Pagamentos: abertos / por mês	Transações monitoradas	Arquivo e todas as redes
Free	—	—	—	não: só Ethereum e Solana
Starter	15	10 / 200	15	sim
Build	50	25 / 1.000	40	sim
Grow	150	50 / 5.000	80	sim
Scale	400	100 / 20.000	150	sim
Business	1.000	250 / 100.000	400	sim

- Uma conta nova pode experimentar o Starter por 7 dias, uma única vez, de graça e sem cartão.
- Se os planos prontos não servirem, na página de preços há a opção "Monte o seu".
- Para comprar um plano, clique em "Comprar" e confirme. Aparece uma fatura válida por 15 minutos, que pode ser paga em criptomoeda.
- Venceu o prazo e não houve novo pagamento: a conta passa para o Free. Ponha um lembrete.

Vamos calcular a loja de Artiom

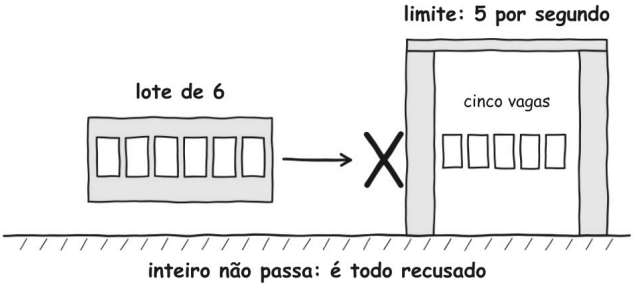
Vinte pedidos esperam pagamento, cada um com seu endereço na Ethereum. Tomamos 30 dias e consideramos que sempre há pedidos esperando: é uma estimativa por cima.

Como o site pergunta	Requisições	Créditos	Plano suficiente
saldo de cada endereço a cada 10 s	5.184.000	51.840.000	Build
o mesmo a cada 30 s	1.728.000	17.280.000	Starter
número do bloco e um `eth_getLogs` a cada 12 s	432.000	4.752.000	Free

A terceira forma é mais de dez vezes mais barata que a primeira: uma só pergunta sobre todos os endereços de uma vez. Mas a comparação tem uma ressalva: as duas primeiras formas percebem a chegada de ETH, e a terceira, transferências de tokens (capítulo 15). E no Free há só duas redes, Ethereum e Solana.

O limite por segundo

O contador de RPS zera a cada segundo e, segundo a documentação da RpcNode, cada endpoint tem o seu. A requisição excedente recebe HTTP 429 e o código `-32005`. **Um lote de N chamadas são N requisições**, e ele passa por inteiro ou não passa.



Vamos testar no endereço público, onde o limite é 5. Basta um lote de seis perguntas:

```
q='{"jsonrpc":"2.0","id":1,"method":"eth_blockNumber","params":[]}'
curl -s -i https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d "[$q,$q,$q,$q,$q,$q]" | grep -iE '^(HTTP|retry-after)|error'
echo
```

```
HTTP/2 429
retry-after: 1
{"jsonrpc":"2.0","id":null,"error":{"code":-32005,"message":
"rps limit exceeded: a batch of 6 calls counts as 6 requests
(the limit is 5 a second)"}}
```

`retry-after: 1` quer dizer "repita daqui a um segundo". O endereço público também protege os métodos caros:

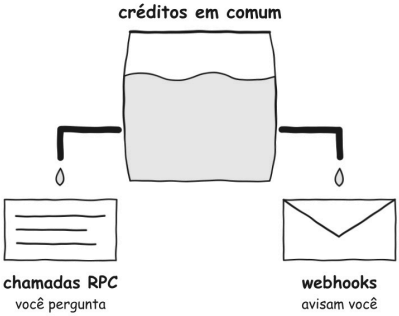
```
curl -s -w '  HTTP %{http_code}\n' https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getBlockReceipts","params":["latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"error":{"code":-32601,"message":
"method eth_getBlockReceipts is not available on the public endpoint,
use a personal endpoint"}}  HTTP 403
```

O que é cobrado e o que não é

O que aconteceu	Créditos
O nó respondeu com um resultado	são cobrados
O nó respondeu com um erro sobre a sua requisição, por exemplo `execution reverted`	são cobrados: isso também é resposta
Recusa pelo limite por segundo	não são cobrados
O método não está na lista da rede	não são cobrados
Nenhum nó respondeu	são devolvidos
Requisição ao endereço público	de graça

Uma reserva para tudo



Da mesma reserva se pagam os **webhooks**, mensagens que o próprio serviço envia ao seu site (capítulo 26). Na rede principal da Ethereum, um evento de endereço monitorado custa 450 créditos, e um pagamento, 750. Nas redes de teste da Ethereum sai pela metade.

O saldo de créditos se vê no painel, na seção "Uso": lá também estão os métodos mais frequentes.

Quando os créditos acabam

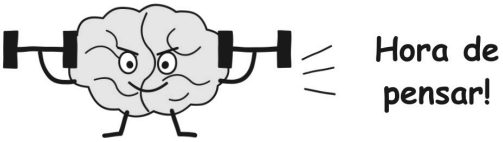
Não haverá conta de excedente: não existe cobrança além do pacote. Mas não conte com o serviço seguindo como se nada fosse. Preveja no programa a recusa do capítulo 11 (`-32006`) e acompanhe o saldo com antecedência.

Cilada: escolher o plano pela etiqueta de preço

Primeiro calcule: **quantas vezes × preço do método**. Umas poucas chamadas pesadas podem comer mais que milhares de leves.

Os preços dos métodos e as condições podem mudar. Pelos termos do serviço, o pagamento do plano não é reembolsado, salvo se a lei exigir o contrário, e o funcionamento ininterrupto não é garantido: em caso de falha grave num plano pago, o período pago é prorrogado. Leia os termos antes de pagar.

Pense bem



1. O site chama `eth_call` (17 créditos) a cada 10 segundos o mês inteiro, 30 dias. Quantos créditos são?
2. No plano Free se enviou um lote de 20 chamadas. O que acontece?
3. Uma requisição recebeu 429. Os créditos dela foram cobrados?

Respostas: 1) 259.200 × 17 = 4.406.400. 2) É recusado por inteiro: 20 é mais que 15. 3) Não.

O essencial

- Os créditos limitam o volume no mês; o RPS, a velocidade.
- Cada método tem seu preço; o lote conta pelo número de chamadas.
- Perguntar com mais esperteza costuma compensar mais que pegar um plano maior.

A seguir: capítulo 26. Eventos por endereço: webhooks e sockets no lugar da consulta constante.

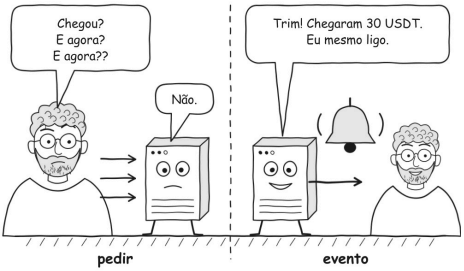
Capítulo 26. Eventos por endereço: webhooks e sockets no lugar da consulta constante

"Vinte pedidos, vinte perguntas por segundo?"

No capítulo 3 fizemos a conta: vinte pedidos em espera, cada um conferido uma vez por segundo, e já é mais do que permitem as 15 requisições por segundo do plano Free. E quase todas as respostas são iguais: "não chegou".

Entendendo: perguntar ou esperar o telefone tocar

Consulta periódica (polling) é quando o seu site pergunta ele mesmo, sem parar: "Chegou?" **Evento** é quando avisam você por conta própria, assim que algo acontece.



Na prática: uma pergunta sobre todos os endereços

Primeiro vamos perguntar com mais esperteza. Não é preciso perguntar por cada pedido separadamente. Ao método `eth_getLogs`, do capítulo 15, dá para informar de uma vez uma lista de destinatários:

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xdac17f958d2ee523a2206206994597c13d831ec7",
  "topics":["0xddf252ad1be2c89b69c2b068fc378daa952ba7f163c4a11628f55a4df523b3ef",
  null,
   ["0x000000000000000000000000000000000000000000000000000000000000dead",
    "0x0000000000000000000000000000000000000000000000000000000000000001",
    "0x000000000000000000000000dac17f958d2ee523a2206206994597c13d831ec7"]],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

A pergunta é esta: "No bloco mais recente há transferências de USDT para algum destes três endereços, venham de quem vierem?" Os endereços foram completados com zeros à esquerda até 64 caracteres. A resposta que recebemos:

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

Lista vazia: ninguém pagou neste bloco. Os endereços podem ser vinte, e a requisição continua sendo uma só. Mas será preciso perguntar sobre **cada** bloco novo, contar as confirmações por conta própria e pegar à parte as transferências do próprio ETH (capítulo 15).

Address Watch: deixe que vigiem por você

O **Address Watch**, monitoramento de endereços, assume esse trabalho. A RpcNode examina os blocos e, quando uma transação envolve o seu endereço, avisa o seu servidor.

O endereço se adiciona no painel: "Notificações" → "Endereços" → "Adicionar endereço". Um programa faz o mesmo com a requisição `POST /v1/address-watches` e o corpo `{"network_slug": "tron", "address": "SEU_ENDERECO"}`.

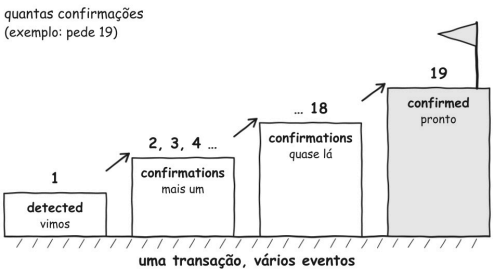
Na resposta vem o `watch_id`, o número do monitoramento. Um endereço numa rede ocupa um **slot** do plano. Removeu o monitoramento, o slot fica livre, mas há limite também para adições e remoções. No Starter são 15 slots. No Free não há nenhum nas redes principais, só um "sandbox": alguns endereços nas redes de teste.

Pode-se obter endpoint para mais redes do que aquelas que são processadas para eventos. Em 27 de novembro de 2026, essas são Ethereum, Bitcoin, Solana, TRON, BNB Smart Chain, Base, Arbitrum One, TON, Litecoin, Dogecoin, Bitcoin Cash, Dash, Mantle e Monad, e, das de teste, só a TRON Nile.

Chegada de dinheiro é um evento do tipo `payment`; saída, `payout`; todo o resto (digamos, uma autorização approve), `address`. Blocos novos vêm com o tipo `block`, e sobre `tx` e `payment_order` veja o capítulo 27.

Uma transação, vários toques da campainha

A transferência não é avisada uma vez só, e sim à medida que vai "endurecendo" (capítulo 8). O que exatamente aconteceu, diz o campo `event`:



event	Significa	O que a loja faz
`detected`	a transação foi vista num bloco	mostrar "estamos vendo a transferência"
`confirmations`	mais uma confirmação	atualizar o contador
`confirmed`	confirmações suficientes, sucesso	considerar que o dinheiro chegou
`failed`	confirmações suficientes, a transação falhou	não despachar

Quantas confirmações esperar se define em "Configurações" → "Confirmações"; se você não definir, vale o valor da rede.

No Bitcoin, na Litecoin e no Dash, antes de todos pode chegar o `pending`: a transferência ainda está no mempool (capítulo 7). O tipo dele é sempre `address`, e não `payment`, e isso não é dinheiro: uma transferência assim pode ser substituída.

Webhook: uma carta na sua caixa

Webhook é uma requisição que o próprio serviço envia ao seu endereço na internet. O endereço e as assinaturas por rede se definem em "Notificações" → "Configurações de entrega". O endereço precisa ser acessível pela internet: `localhost` não serve.

Se você ligar o webhook sem escolher nada, chegarão as transferências de entrada e de saída de todas as redes, mas os blocos, não.

Eis parte de um exemplo da documentação: o primeiro toque sobre 30 USDT na TRON:

```
{ "delivery_id": "3f9c…6b3d90",
  "type": "payment", "event": "detected", "network": "tron",
  "tx_hash": "bb08…c320c", "address": "TToL…QRzRG",
  "direction": "in", "currency": "USDT",
  "amount": "30", "amount_raw": "30000000",
  "confirmations": 1, "required_confirmations": 19,
  "confirmed": false }
```

As quantias vêm como texto, para que nada seja arredondado; `amount_raw` é o mesmo sem a vírgula (capítulo 9). Um token que não está no catálogo do serviço terá apenas `amount_raw`. Nos cabeçalhos vêm `X-RpcNode-Delivery` (número da entrega), `X-RpcNode-Event` (tipo) e `X-RpcNode-Signature` (assinatura).

"É você mesmo?"

O endereço do webhook é aberto: qualquer um pode bater e escrever "pagaram você". Por isso cada entrega é assinada. Pegam-se a hora, um ponto e o corpo, e deles se calcula uma impressão digital com o segredo do webhook. Esse método se chama **HMAC-SHA256**. O segredo fica no bloco "Segredo e IPs"; não é o segredo da chave de API.

Na prática: vamos assinar nós mesmos

O segredo e o corpo aqui são inventados:

```
printf '%s' '1787347206.{"type":"test"}' |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}'
```

```
e17efb57778b0e0c758a9bf2c5705049c4c383d2af3a2bd2783c8b665c151cde
```

Mude um caractere que seja no corpo ou no segredo e sairá uma sequência completamente diferente. O seu servidor calcula a mesma coisa e compara com o cabeçalho no formato `t=…,v1=…`.

Eis um **esboço** de receptor em Node.js. Nós o testamos com requisições feitas à mão, mas não com entregas de verdade: para elas é preciso um painel pessoal. A verificação dos 300 segundos impede que reenviem a entrega de ontem.

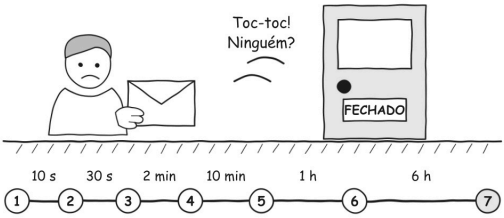
```
import { createServer } from 'node:http'
import { createHmac, timingSafeEqual } from 'node:crypto'

const SECRET = process.env.RPCNODE_WEBHOOK_SECRET
const seen = new Set()      // numa loja de verdade, é um banco de dados
createServer(async (req, res) => {
  const parts = []
  for await (const part of req) parts.push(part)
  const body = Buffer.concat(parts)   // os bytes como chegaram
  const [, t, v1] = /^t=(\d+),v1=([0-9a-f]{64})$/
    .exec(req.headers['x-rpcnode-signature'] ?? '') ?? []
  const ours = createHmac('sha256', SECRET)
    .update(`${t}.`).update(body).digest()
  const ok = v1 && Math.abs(Date.now() / 1000 - t) < 300 &&
    timingSafeEqual(ours, Buffer.from(v1, 'hex'))
  if (!ok) return res.writeHead(401).end()
  const id = req.headers['x-rpcnode-delivery']
  if (!seen.has(id)) {
    seen.add(id)
    console.log(id, body.toString())  // aqui entra o seu trabalho
  }
  res.writeHead(200).end()
}).listen(3000)
```

Se você não estava em casa

A entrega deu certo se o servidor respondeu com um código da faixa dos 200 em até 10 segundos. Caso contrário, ela é repetida, e depois da sétima falha é abandonada. Por isso:

- Um evento pode chegar duas vezes. O número da entrega é o mesmo: filtre por ele.
- A ordem não é garantida: `confirmations` pode ultrapassar `detected`.
- Faça o trabalho demorado **depois** de responder.



pausas crescem; sete tentativas, cerca de sete horas

Teste o seu endereço antes com uma entrega de teste: ela é gratuita. O registro de entregas no painel guarda os dados por um dia.

Socket: uma linha aberta

A segunda forma é o **WebSocket**: o seu programa abre a conexão e escuta, sem precisar de um endereço próprio na internet. Não é a assinatura `eth_subscribe` do capítulo 11: lá, o nó fala sobre a rede; aqui, o serviço fala sobre os seus endereços e pedidos. O corpo é o mesmo do webhook, mas o que se perdeu enquanto você estava fora **não é repetido**.

O protocolo é compatível com o Pusher (serve a biblioteca `pusher-js`), e a chave é o número da sua chave de API. O socket se liga à parte, pelo botão "Entregar via WebSocket". Se ligar as duas formas, cada aviso é entregue e cobrado duas vezes.

Quanto custa

Os eventos gastam os mesmos créditos das requisições. Um evento de endereço na rede principal da TRON ou da Ethereum, em 27 de novembro de 2026, custa 450 créditos, **uma vez por transação**: as confirmações e o resultado são de graça. Paga-se pela entrega, mesmo que depois o seu servidor não a aceite.

Vamos estimar um mês de 30 dias na Ethereum, onde cada requisição da tabela custa 17 créditos:

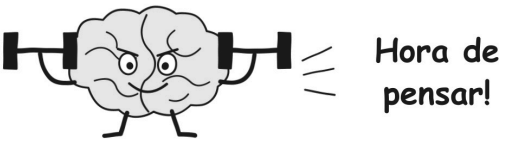
Como ficamos sabendo do pagamento	Requisições ou eventos	Créditos
20 pedidos, `eth_call` a cada 10 segundos (capítulo 13)	5.184.000	88.128.000
um `eth_getLogs` por bloco, a cada 12 segundos	216.000	3.672.000
Address Watch, 200 transferências no mês	200	90.000

O Starter tem 50 milhões de créditos: a primeira forma não cabe neles. É só um exemplo. Quanto mais transferências, mais caros saem os eventos, e as transferências miúdas de estranhos ao seu endereço também contam.

Cilada: "chegou o evento, vou despachar"

- **Nem todo evento é pagamento:** `detected` ainda não é definitivo; `payout` é uma despesa sua.
- **O token se reconhece pelo endereço do contrato** no campo `token`, e não pelo nome (capítulo 9).
- **Teste de antemão as transferências especiais,** por exemplo ETH enviado de dentro de um contrato (capítulo 15).
- **Os eventos de endereço não são um arquivo:** pela API ficam disponíveis por um dia.

Pense bem



1. O servidor da loja ficou desligado por uma hora. O que acontece com as entregas por webhook e o que acontece com os avisos no socket?
2. Chegaram duas entregas com o mesmo `X-RpcNode-Delivery`. Quantas vezes creditar o dinheiro?
3. No mês houve 1000 transferências na TRON para os endereços monitorados, com entrega só por webhook. Quantos créditos?

Respostas: 1) os webhooks são repetidos até acabarem as sete tentativas; o socket não repete o que se perdeu. 2) Uma. 3) 450.000.

O essencial

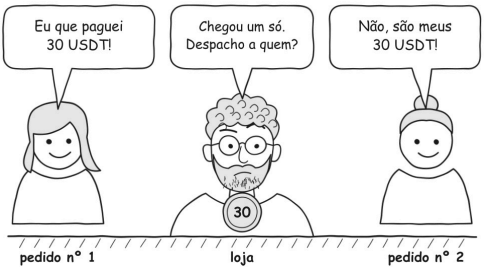
- Não pergunte sem parar: deixe que avisem você sobre a transferência.
- Confira a assinatura, filtre as repetições, responda rápido.
- O dinheiro chegou no `confirmed`, e não no primeiro toque.

A seguir: capítulo 27. Pagamentos e monitoramento de transações.

Capítulo 27. Pagamentos e monitoramento de transações

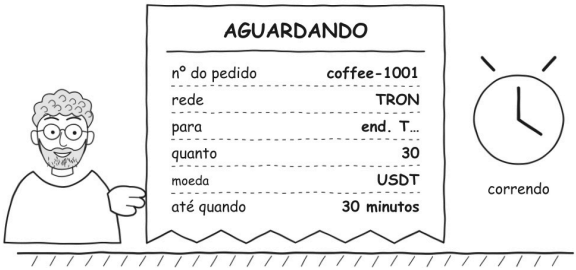
“O dinheiro chegou. Mas de quem é?”

Artiom recebeu um evento: chegaram 30 USDT ao endereço. Abre a lista de pedidos e lá há dois de trinta. O evento sabe o endereço e a quantia, mas não sabe nada sobre o carrinho do cliente.



Entendendo: o pagamento é uma espera

Um **pagamento** (seção “Pagamentos” do painel) é uma espera registrada de antemão: o pedido tal aguarda tanto, no endereço tal, até tal hora. Chegou a transferência que combina, o próprio serviço informa o número do pedido.



O endereço é seu, você o fornece. O dinheiro vai do cliente direto para você; o serviço não precisa da frase secreta (capítulo 6).

O pagamento se cria pelo botão “Novo pagamento” ou com a requisição `POST /v1/payments`:

Campo	O que é
`order_id`	o seu número de pedido, sem repetição
`network_slug` , `address`	a rede e o seu endereço nela
`amount`	a quantia **como texto**: `"30"` , `"12.5"`
`token_contract`	o contrato do token; sem ele, espera-se a moeda da rede
`ttl_seconds`	o prazo em segundos; sem ele, uma hora; no máximo 30 dias

Na prática: vamos bater sem chave

```
curl https://api.rpcnode.dev/v1/payments
```

```
{"ok":false,"error_code":"public_api.key_missing",
 "error":"Missing X-Api-Key header."}
```

É uma resposta com o código 401: "e você, quem é?". Cada requisição a essa API é assinada com o segredo da chave de API (seção "Chaves de API"). Assinam-se quatro linhas: a hora, o método, o caminho e a impressão digital do corpo.

Eis um **esboço** em linguagem de shell. Conferimos a sintaxe e o cálculo da assinatura, mas não o enviamos: é preciso uma chave pessoal.

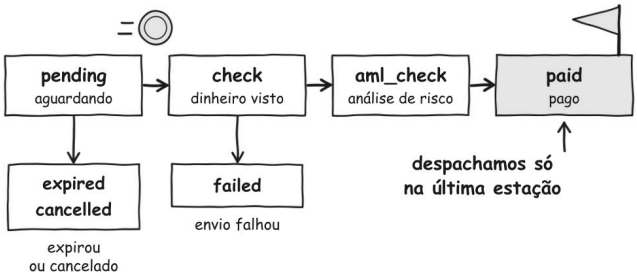
```
BODY='{"order_id":"coffee-1001","network_slug":"tron",
"address":"SEU_ENDERECO","amount":"30","ttl_seconds":1800,
"token_contract":"TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t"}'
TS=$(date +%s)
HASH=$(printf '%s' "$BODY" | openssl dgst -sha256 | awk '{print $NF}')
SIG=$(printf '%s\nPOST\n/v1/payments\n%s' "$TS" "$HASH" |
  openssl dgst -sha256 -hmac "$SECRET" | awk '{print $NF}')

curl https://api.rpcnode.dev/v1/payments \
  -H 'Content-Type: application/json' \
  -H "X-Api-Key: $KEY_ID" -H "X-Timestamp: $TS" \
  -H "X-Signature: $SIG" -d "$BODY"
```

Na resposta vêm o `payment_id` , o status `pending` e o prazo `expires_at` . Se você repetir a requisição com o mesmo `order_id` e a mesma rede, endereço, token e quantia, volta o mesmo pagamento. Com dados diferentes, o erro `payment.order_exists` .

A estrada do pagamento

Cada passo chega como um evento `payment_order`, por webhook ou no socket (capítulo 26).



`event`	**O que aconteceu**
`check`	o dinheiro foi visto, aguardando confirmações
`confirmations`	aumentou o número de confirmações
`aml_check`	há confirmações suficientes, a análise de risco está em andamento
`paid`	pago: pode despachar
`expired`, `cancelled`	o prazo venceu ou você cancelou o pagamento que aguardava
`failed`	a transação falhou na rede (não construa tudo em cima desse status: o pagamento pode simplesmente expirar)

AML é a verificação de se o dinheiro enviado está ligado a um histórico ruim. Onde ela existe, o objeto `aml` no evento `paid` traz uma pontuação `score` de 0 a 100 (quanto maior, mais arriscado) e um nível `risk_level`. Não há cobrança à parte. Ela não existe em todas as redes: sem ela, o pagamento vira `paid` imediatamente.

É uma dica, não uma sentença nem um parecer jurídico: o que fazer com uma transferência arriscada, quem decide é você. Se um pagamento empacar em `aml_check`, ele ainda não é pagamento: escreva ao suporte.

Parte de um exemplo da documentação:

```
{ "type": "payment_order", "event": "paid", "order_id": "ord_123",
  "network": "tron", "status": "paid", "tx_hash": "bb08…c320c",
  "received_amount_raw": "30000000",
  "confirmations": 19, "required_confirmations": 19,
  "aml": { "score": 0, "risk_level": "low" } }
```

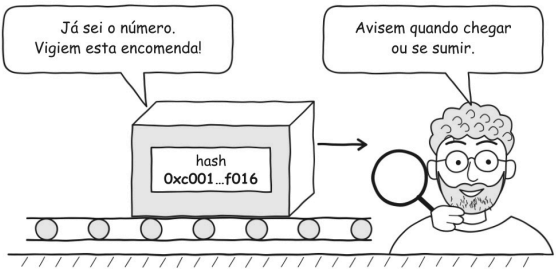
As letras miúdas

- **Menos que o necessário não conta.** O pagamento continua esperando e vai expirar. A documentação tem o status `underpaid`, mas não construa a loja em cima dele: perceba o pagamento a menor pelo evento de endereço.
- **Mais que o necessário conta.** Quanto chegou de fato, diz o `received_amount_raw`.
- **Transferência atrasada não paga.** Já a que foi vista a tempo acumula confirmações tranquilamente mesmo depois do prazo.
- **Um endereço para cada pedido.** Assim não há como confundir dois "de trinta" (capítulo 28).
- **O slot do endereço não se libera sozinho** depois de `paid` ou `expired`: remova o monitoramento. Depois de cancelar um pagamento, o endereço que foi adicionado só para ele é retirado automaticamente.

Os pagamentos estão nos planos pagos: no Starter, 10 abertos ao mesmo tempo e 200 por mês. Um pagamento na rede principal da TRON ou da Ethereum custa 750 créditos, e eles só são cobrados quando o dinheiro é visto. No Free há um sandbox: 3 pagamentos abertos nas redes de teste; hoje, na TRON Nile. A análise de risco ali não é de verdade e vem marcada com `"sample": true`.

Monitoramento de transações

Pode ser o contrário: o hash você já tem. Artiom devolveu dinheiro a um cliente ou pagou um fornecedor e quer saber se chegou.



"Notificações" → "Transações" → "Adicionar hash", ou `POST /v1/tx-watches` com o corpo `{"network_slug": "tron", "tx_hash": "HASH"}`. O hash pode ser adicionado antes de aparecer na rede.

Chegam eventos `tx`: `detected`, depois `confirmations` e, no fim, `confirmed` ou `failed`. Se o hash nunca for encontrado, `cancelled`. Eles chegam se, para essa rede, estiverem ligadas as transferências de entrada ou de saída.

Um monitoramento aberto ocupa um slot e o libera quando termina. No Starter são 15 slots; no Free esse serviço não existe. A transação se paga uma vez: na TRON e na Ethereum são 150 créditos.

Na prática: contar as confirmações por conta própria

É isto que se faz debaixo do capô de qualquer monitoramento desses. Vamos pegar a transferência de USDT do capítulo 15:

```
URL=https://public-ethereum-mainnet.rpcnode.dev
H=0xc0015ff5178d72a68831b133187c3f9f1fbee2e440d3af311d0b32110e2cf016
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionReceipt",
  "params":["'$H'"]}' | grep -oE '"(status|blockNumber)":"0x[0-9a-f]+"' | sort -u
sleep 1
curl -s $URL -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":2,"method":"eth_blockNumber","params":[]}'
```

```
"blockNumber":"0x194646a"
"status":"0x1"
{"jsonrpc":"2.0","id":2,"result":"0x19464c5"}
```

Status `0x1` é sucesso. O bloco da transferência conta como a primeira confirmação, portanto `echo $((0x19464c5 - 0x194646a + 1))` dá **92**. O seu número será maior.

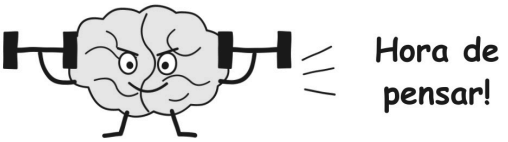
Cilada: dois avisos sobre o mesmo dinheiro

O pagamento põe o endereço sob monitoramento. Por isso, sobre uma mesma transferência, podem chegar tanto o `payment` (evento de endereço) quanto o `payment_order` (evento de pagamento). Cada um é cobrado pelo seu preço.

> **Regra:** só fecha o pedido um `payment_order` com `event` = `paid`. E confira o número do pedido, o `payment_id`, o token e a quantia.

E não pergunte sobre o pagamento sem parar: a API tem limite próprio, 100 requisições por minuto por conta. Pergunta-se depois de uma parada, para conferir: `GET /v1/payments/orders/{orderId}`.

Pense bem



1. Um pagamento de 30 USDT, e chegaram 29,5. O que acontece com o pagamento?
2. O prazo venceu e a transferência foi vista um minuto depois. Onde está o dinheiro e o que acontece com o status?
3. Chegou o evento `check`. Pode despachar o café?

Respostas: 1) ele não está pago e vai expirar no prazo. 2) O dinheiro está no seu endereço, o pagamento continua `expired`*: você resolve na mão. 3) Não, espere o* `paid`.

O essencial

- O pagamento liga a transferência ao pedido; o monitoramento de hash, ao destino da transação.
- Despacha-se com `paid`, e não no primeiro aviso.
- Pagamentos a menor, atrasos e slots ocupados são preocupação sua.

Fim da quinta parte. Artiom sabe obter um endpoint, entende o caminho da requisição e os planos, e fica sabendo do dinheiro sem precisar perguntar.

A seguir: parte 6, capítulo 28. Montando a loja: cobrança, endereço, espera do pagamento, confirmações.

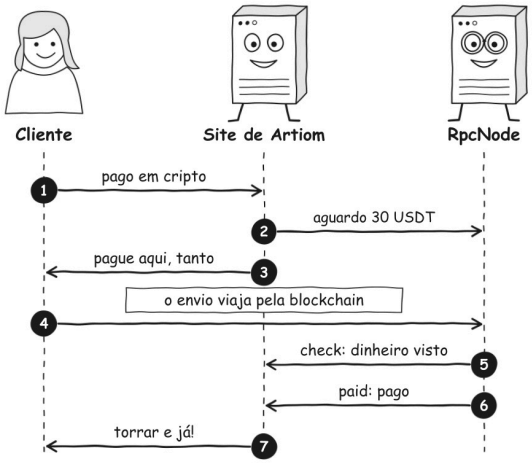
Capítulo 28. Montando a loja: cobrança, endereço, espera do pagamento, confirmações

“Separado, entendi tudo. E junto?”

Artiom já sabe sobre endereços, confirmações, webhooks e pagamentos. E eis que um cliente clica em “Pagar com criptomoeda”. O que o site deve fazer a partir daí, passo a passo?

Entendendo: quem participa

São quatro participantes. O **cliente**, com a carteira dele. O **site da loja**, com o banco de dados de pedidos. A **RpcNode**, que vigia a rede. E a **carteira de Artiom**, onde ficam as chaves. Na conversa entre o site e o serviço, a carteira não entra.

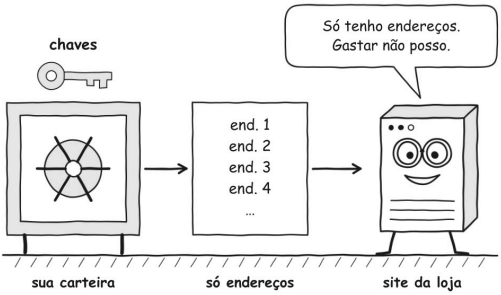


Olhe o passo 4. O dinheiro não vai para o site nem para a RpcNode. Ele cai direto no endereço de Artiom, e a RpcNode só o percebe no bloco.

De onde tirar endereços sem chaves no servidor

No capítulo 6 prometemos: um endereço para cada pedido. E lá mesmo dissemos: o servidor da loja não é lugar para chave privada. Como conciliar as duas coisas?

Dá para preparar os endereços separados das chaves.



Há duas maneiras.

	Lista de endereços	Chave pública estendida
O que fica no servidor	endereços prontos, um por pedido	uma chave (muitas vezes começa com `xpub`), a partir da qual o programa calcula os endereços
Onde serve	em qualquer rede	Bitcoin, Ethereum, TRON e parentes; em Solana e TON assim não dá
Se invadirem o servidor	veem os endereços da lista	veem todos os seus endereços e tudo o que entra neles
Trabalho	reabastecer a tempo	configurar uma vez

Com uma chave dessas não se gasta dinheiro. Mas o padrão BIP-32 avisa: essa chave, **junto com uma única** chave privada do mesmo ramo, permite calcular todas as chaves privadas do ramo. Por isso não exporte avulsas as chaves de uma carteira assim. Para um iniciante é mais simples começar pela lista.

Tire a lista de uma carteira cujas chaves não estejam num servidor na internet. É prática a **carteira de hardware**: um aparelho separado que guarda as chaves dentro de si. Antes de carregar a lista, confira alguns endereços na tela dele. E, na TON, mostre ao cliente o endereço na forma `UQ` (capítulo 19).

Na prática: o banco de dados não deixa confundir

Distribuir endereços é serviço para um banco de dados. Eis um de brinquedo: dois endereços livres e três pedidos. Você precisa do programa `sqlite3`, que muitos sistemas já têm.

```
sqlite3 :memory: <<'SQL'
CREATE TABLE addresses(address TEXT PRIMARY KEY, order_id TEXT UNIQUE);
INSERT INTO addresses(address) VALUES ('endereco-1'), ('endereco-2');

UPDATE addresses SET order_id = 'coffee-1001' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1002' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
UPDATE addresses SET order_id = 'coffee-1003' WHERE address =
  (SELECT address FROM addresses WHERE order_id IS NULL ORDER BY address LIMIT 1)
  RETURNING order_id, address;
SELECT 'faltou endereço para o terceiro' WHERE changes() = 0;

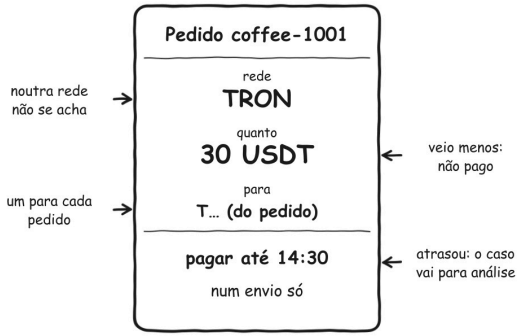
UPDATE addresses SET order_id = 'coffee-1001' WHERE address = 'endereco-2';
SQL
```

```
coffee-1001|endereco-1
coffee-1002|endereco-2
faltou endereço para o terceiro
Runtime error near line 15: UNIQUE constraint failed: addresses.order_id (19)
```

Não há com que emitir a terceira cobrança: é hora de reabastecer a lista. E o último comando queria dar ao pedido 1001 um segundo endereço, e o banco recusou. Quem monta guarda são as palavras `PRIMARY KEY` e `UNIQUE`: o endereço é registrado uma vez, e o pedido não tem mais que um.

E não distribua endereços antigos numa segunda rodada. Uma transferência atrasada de um pedido passado pagaria o novo.

Emitindo a cobrança



O cliente precisa de cinco coisas: rede, moeda, quantia exata, endereço e prazo. E de um pedido: mandar tudo **numa única transferência**, porque duas metades não se somam. Avise: se a carteira ou a corretora descontar a taxa da própria quantia, vai chegar menos.

Preços em rublos ou dólares, converta para a moeda no momento do pedido e mantenha a cotação até o fim do prazo. Por isso o prazo é curto, meia hora por exemplo.

Agora é preciso o pagamento do capítulo 27. Abaixo vai um **esboço** em Python. Nele, a função `rpcnode` envia uma requisição assinada: escreva-a seguindo o capítulo 27 ou use o cliente oficial da RpcNode (há para Python, TypeScript, PHP, Go e Kotlin). O banco de dados é o mesmo, de duas tabelas.

```
USDT = 'TR7NHqjeKQxGTCi8q8ZY4pL8otSzgjLj6t'    # contrato do USDT na TRON
# addresses(address, order_id)
# orders(id, payment_id, amount_raw, status — primeiro 'waiting')

def take_address(db, order_id):
    with db:                                    # uma transação do banco
        db.execute('INSERT OR IGNORE INTO orders(id) VALUES (?)',
                   (order_id,))
        mine = db.execute('SELECT address FROM addresses'
                          ' WHERE order_id=?', (order_id,)).fetchone()
        if mine:
            return mine[0]                      # de novo: o mesmo
        free = db.execute(
            'UPDATE addresses SET order_id=? WHERE address='
            ' (SELECT address FROM addresses WHERE order_id IS NULL'
            '  ORDER BY address LIMIT 1) RETURNING address',
            (order_id,)).fetchall()
        if not free:
            raise RuntimeError('acabaram os endereços')
        return free[0][0]
```

```
def checkout(db, rpcnode, order_id, amount):   # amount é um texto
    address = take_address(db, order_id)
    reply = rpcnode('POST', '/v1/payments', {
        'order_id': order_id, 'network_slug': 'tron',
        'address': address, 'amount': amount,
        'token_contract': USDT, 'ttl_seconds': 1800})
    if not reply['ok']:
        raise RuntimeError(reply['error_code'])
    payment = reply['payment']
    with db:
        db.execute('UPDATE orders SET payment_id=?, amount_raw=?'
                   ' WHERE id=?', (payment['payment_id'],
                   payment['amount_raw'], order_id))
    return payment        # ao cliente: address, amount, expires_at
```

O principal aqui é a ordem: o endereço é fixado ao pedido **antes** de se chamar o serviço. Caiu a conexão: a nova chamada pega o mesmo endereço e o mesmo número de pedido, e o serviço devolve o mesmo pagamento. Testamos isso com uma função `rpcnode` inventada; com o serviço de verdade o esboço não foi executado.

Crie o pagamento quando o cliente escolher pagar com criptomoeda, e não a cada carrinho: o limite mensal do plano conta os pagamentos criados, os não pagos também.

A RpcNode reconhece o pagamento pelo endereço, pela moeda e pela quantia. O comentário do capítulo 19 e a etiqueta `reference` do capítulo 18 não participam aqui: o pagamento não tem esse campo. O XRP Ledger e a Stellar, do capítulo 20, não estão na lista de redes com pagamentos (capítulo 26).

Esperando o pagamento

A página do pedido pergunta a cada poucos segundos ao **seu servidor**, e o servidor responde a partir do próprio banco de dados. O banco é atualizado pelos webhooks. Não se pode consultar a RpcNode direto do navegador: o segredo da chave iria parar nas mãos do cliente, e além disso o limite da API é de 100 requisições por minuto por conta.

Status	O que o cliente vê	O que a loja faz
`pending`	"Aguardando a transferência até 14:30"	nada
`check`	"Transferência vista. Confirmações: 3 de 19"	não despacha
`aml_check`	"Verificando a transferência"	espera
`paid`	"Pago, preparando o pedido"	despacha
`expired`	"Prazo encerrado. Não pague esta cobrança"	fecha a cobrança
`failed`	"A transferência não passou"	propõe recomeçar
`cancelled`	"Cobrança cancelada"	fecha a cobrança

Enquanto a transferência não entra num bloco, o pagamento fica calado. Por isso escreva "aguardando", e não "não pago".

Quantas confirmações esperar

Isso se define em "Configurações" → "Confirmações", separadamente para cada rede. Se não definir, vale o valor da rede. O número necessário aparece no pagamento, no campo `required_confirmations`.

Colocar menos que o padrão é arriscado: blocos recentes às vezes são substituídos (capítulo 8). E o número é um por rede, não por pedido. Se quiser esperar mais por um saco de grãos do que por uma xícara, espere mais um pouco depois do `paid` e confira você mesmo, como no capítulo 27.

Libere o lugar

Há dois tipos de lugar. O **slot de pagamento** se libera sozinho quando o pagamento vira `paid`, `expired`, `failed` ou `cancelled`. O **slot de endereço** não se libera sozinho (salvo no cancelamento, capítulo 27).

No Starter são 10 pagamentos abertos, 15 endereços monitorados e 200 pagamentos por mês. O décimo primeiro pagamento aberto recebe a recusa `payment.slot_limit`. E, quando todos os endereços estão ocupados, vem `watch.address_limit`.

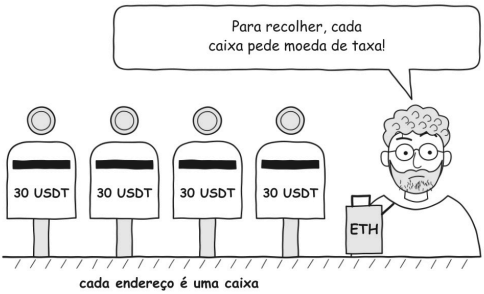
Deixe o pedido fechado descansar um pouco, caso a transferência esteja atrasada. Depois retire o monitoramento: ache o `watch_id` dele com a requisição `GET /v1/address-watches?`

`network_slug=tron&address=…` e envie `DELETE /v1/address-watches/{watch_id}`. Depois disso não haverá mais avisos sobre o endereço, embora o dinheiro ainda possa chegar a ele.

As adições e remoções de endereços também têm limite: por dia, por semana e por mês, e quanto mais slots, maior a cota. Um endereço novo a cada pedido gasta essa cota. Num dia de promoção, dá para receber `watch.quota` mesmo com slots livres. Se espera muitos pedidos, descubra o teto antes.

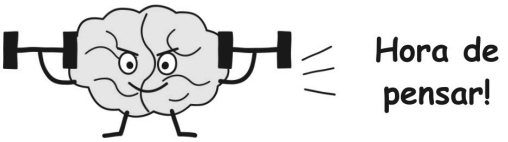
Cilada: dinheiro espalhado em cem endereços

Um endereço por pedido facilita a contabilidade, mas depois o dinheiro fica separado. Para mandar USDT de um endereço, é preciso que nele mesmo haja moeda para a taxa (capítulos 9 e 17): ETH na Ethereum, TRX ou energia na TRON. Cem pedidos são cem transferências separadas. No Bitcoin, muitos "pedaços" pequenos também encarecem o envio (capítulo 14). Calcule isso antes de escolher a rede para valores miúdos.



E mais uma coisa. Carteiras que seguem o padrão BIP-44 param de procurar depois de 20 endereços vazios seguidos. Se houver muitos pedidos abandonados, a carteira pode não mostrar um pagamento num endereço distante. O dinheiro está a salvo, mas verifique se ela sabe procurar mais adiante.

Pense bem



1. O cliente recarregou a página de pagamento. São precisos um endereço novo e um pagamento novo?
2. Artiom, no Starter, já tem 10 pagamentos abertos. Chega o décimo primeiro cliente. O que acontece?
3. A página de pagamento consulta a RpcNode direto do navegador. Cite dois problemas.

Respostas: 1) não: o mesmo número de pedido e o mesmo endereço. 2) Recusa `payment.slot_limit`*: peça para esperar ou ofereça outra forma de pagamento. 3) O segredo da chave fica visível ao cliente, e o limite de requisições acaba rápido.*

O essencial

- As chaves ficam na carteira e no site só há endereços: um pedido, um endereço, sem repetição.
- O cliente pergunta ao seu site, e o site fica sabendo das novidades pelos webhooks.
- Calcule de antemão os slots e o custo de recolher o dinheiro.

A seguir: capítulo 29. Quando tudo dá errado: erros, repetições, transações travadas, crédito em dobro.

Capítulo 29. Quando tudo dá errado: erros, repetições, transações travadas, crédito em dobro

"O café foi duas vezes!"

De manhã, Artiom viu: para um único pedido saíram dois pacotes. E outro cliente escreve: "Eu paguei, e aí continua tudo 'aguardando'". Não é milagre. Acontece quando o programa foi escrito só para tempo bom.

Entendendo: três tipos de problema

Problema	Exemplo	Remédio
A sua requisição ficou sem resposta	a conexão caiu ao criar o pagamento	repetir sem causar dano
O aviso chegou torto	duas vezes, fora de ordem ou nunca	tranca no banco e reconciliação
A transferência saiu errado	travou, atrasou, veio a menos	regras e análise manual

Quando repetir uma requisição à API

As pausas crescentes vimos no capítulo 23. Mas só se pode repetir sem medo o que é **idempotente**: faça quantas vezes fizer, o resultado é o de uma vez só. A criação de pagamento é assim graças ao `order_id`. Repita com o mesmo número e os mesmos dados. Assine cada repetição de novo: a assinatura contém a hora.

Os clientes oficiais da RpcNode, depois de uma queda ou de um erro 500, não repetem o `POST` por conta própria. Portanto, repetir a criação do pagamento é tarefa sua.

O que fazer com as diferentes respostas da API:

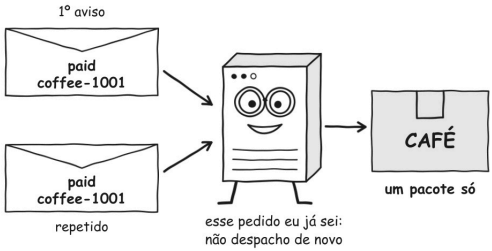
O que veio	O que fazer
nada; 502, 503, 504	repetir com pausas
429 com o cabeçalho `Retry-After`	esperar o que ele mandar
429 `watch.quota` (capítulo 28)	não adianta repetir já: a cota acabou
400	erro na requisição: em `errors` estão os campos
401	conferir a chave, a assinatura e o relógio do servidor
403	plano, limite de slots ou lista de IPs permitidos
409 `payment.order_exists`	esse número já pertence a um pagamento com outros dados: investigar, e não trocar o número no chute

Outra coisa é quando o pagamento expirou e o cliente quer tentar de novo. O `order_id` antigo devolve o mesmo pagamento expirado. É preciso um número novo, digamos `coffee-1001-2`.

Um aviso, duas vezes

Duplicatas vêm de todo lado:

- o webhook foi repetido porque você não respondeu em 10 segundos;
- estão ligados tanto o webhook quanto o socket;
- sobre o mesmo dinheiro chegaram `payment` e `payment_order` (capítulo 27);
- o webhook e a sua verificação agendada dispararam ao mesmo tempo;
- um funcionário clicou em "despachar" à mão.



O número da entrega (capítulo 26) filtra as repetições de uma mesma entrega. Mas não salva de um segundo aviso sobre o mesmo dinheiro vindo por outro caminho. A tranca principal é o estado do pedido: só se pode passar de "aguardando" para "pago" uma vez.

Na prática: um webhook duas vezes

Eis a tranca inteira. Os avisos fomos nós que inventamos. Salve os dois pedaços num único arquivo `once.py` e execute `python3 once.py`.

```
import sqlite3
db = sqlite3.connect(':memory:')
db.executescript("""
CREATE TABLE orders(id TEXT PRIMARY KEY, payment_id TEXT,
                    amount_raw TEXT, status TEXT);
CREATE TABLE to_ship(order_id TEXT PRIMARY KEY);
INSERT INTO orders VALUES('coffee-1001','p1','30000000','waiting');
""")

def on_news(n):
    if n['type'] != 'payment_order' or n['event'] != 'paid':
        return 'ainda não é pagamento'
    with db:              # tudo aqui dentro é uma transação do banco
        cur = db.execute(
            "UPDATE orders SET status='paid' WHERE id=? AND"
            " payment_id=? AND amount_raw=? AND status='waiting'",
            (n['order_id'], n['payment_id'], n['amount_raw']))
        if cur.rowcount != 1:
            return 'já contabilizado ou não é nosso'
        db.execute('INSERT INTO to_ship VALUES (?)', (n['order_id'],))
    return 'despachar'
```

Depois, quatro avisos seguidos e a resposta do programa:

```
paid = {'type': 'payment_order', 'event': 'paid',
        'order_id': 'coffee-1001', 'payment_id': 'p1',
        'amount_raw': '30000000'}
print(1, on_news({**paid, 'event': 'check'}))
print(2, on_news(paid))
print(3, on_news(paid))                     # o mesmo webhook outra vez
print(4, on_news({**paid, 'event': 'check'}))  # aviso atrasado
print('pacotes:', db.execute('SELECT count(*) FROM to_ship').fetchone()[0])
```

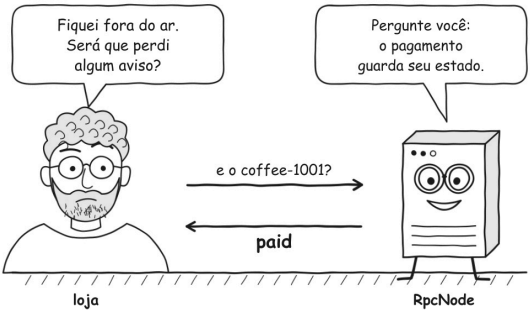
```
1 ainda não é pagamento
2 despachar
3 já contabilizado ou não é nosso
4 ainda não é pagamento
pacotes: 1
```

Um pacote só. Tudo se apoia num único comando: "marque `paid` **somente se** o pedido ainda está aguardando e o pagamento e a quantia são aqueles mesmos". A marca e a tarefa de despacho são gravadas juntas: ou as duas, ou nenhuma. Isso se chama **transação do banco de dados** (a palavra é a mesma da blockchain, mas a coisa é outra). Testamos também oito avisos iguais de uma vez: saiu um despacho só.

No esboço do capítulo 26, os números de entrega viviam na memória e eram anotados antes do trabalho. Com dinheiro não pode ser assim: o programa cai no meio e embaralha tudo. Primeiro grave no banco de dados, depois responda 200.

O aviso que não chegou nunca

O webhook é abandonado após sete tentativas, o que dá cerca de sete horas. O socket não repete de jeito nenhum o que se perdeu. Ou seja, parte das notícias pode se perder. O remédio é a **reconciliação**: perguntar você mesmo sobre os pedidos que, para você, continuam abertos.



A requisição `GET /v1/payments/orders/{orderId}` devolve o pagamento. Os nomes ali são um pouco diferentes: no lugar de `event`, olhe o `status`; a rede se chama `network_slug`; o contrato do token, `token_contract`. Monte com a resposta um "aviso" igual e passe-o pela mesma tranca.

Reconcilie depois de uma parada do servidor, depois de reconectar o socket e de forma agendada. Dez pedidos abertos uma vez por minuto são 10 requisições das 100 permitidas.

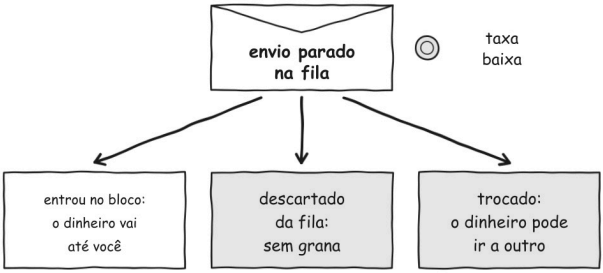
A transferência travou

O cliente mandou o hash: "Enviei há uma hora". O pagamento continua calado. Vamos perguntar ao nó se ele conhece essa transação. O hash aqui é inventado:

```
curl -s https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getTransactionByHash","params":
  ["0x0000000000000000000000000000000000000000000000000000000000000001"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":null}
```

`null` quer dizer "este nó não conhece essa transação". As causas variam: erro de digitação, outra rede, a transferência ainda não chegou ou já foi descartada. E, se a transação foi encontrada mas o `blockNumber` dela é `null`, ela está esperando na fila.



fora de um bloco, o dinheiro ainda não é seu

Rede	O que acontece com quem espera muito
Ethereum e parecidas	segura todas as transferências seguintes do remetente (capítulo 15). Pode ser substituída por uma transferência com o mesmo `nonce` e taxa maior: o programa de nó geth, por padrão, exige pelo menos 10% a mais
Bitcoin	o remetente pode substituí-la por uma transferência com taxa maior. O Bitcoin Core, por padrão, joga fora da fila o que ficou duas semanas parado
TRON	a transação tem prazo de validade, por padrão um minuto. Não entrou em bloco a tempo, não entra mais
Solana	a transação comum vale enquanto estiver fresco o bloco a que ela se refere: 150 blocos, algumas dezenas de segundos

Em regra, a loja não consegue acelerar a transferência de outra pessoa: é preciso a chave do remetente. Espere o bloco e não despache. Se nesse meio tempo o pagamento expirou e mesmo assim a transferência passou, o dinheiro cai no seu endereço, mas o pagamento continua `expired`.

Travou um envio **seu**, um reembolso por exemplo? A mesma transferência assinada pode ser enviada outra vez: o hash não muda. Já uma transferência nova é um segundo dinheiro saindo, a menos que ela substitua a primeira. Se o endereço tem fila na Ethereum, vê-se assim:

```
URL=https://public-ethereum-mainnet.rpcnode.dev
A=0x8f8d1206d1bce12ff892731f8a140ed18b93884e
for tag in latest pending; do
  curl -s $URL -H 'Content-Type: application/json' -d '{"jsonrpc":"2.0",
    "id":1,"method":"eth_getTransactionCount","params":["'$A'","'$tag'"]}'
  echo; sleep 2
done
```

```
{"jsonrpc":"2.0","id":1,"result":"0x67641"}
{"jsonrpc":"2.0","id":1,"result":"0x67641"}
```

O primeiro número é quantas transferências do endereço já estão em blocos; o segundo, contando as que esperam. São iguais: este nó não vê fila. É o endereço do remetente do capítulo 15.

E cuidado com o monitoramento de hash (capítulo 27): a resposta `cancelled` significa que o serviço não encontrou a transação, e não que ela jamais vai aparecer.

O dinheiro chegou, mas não como devia

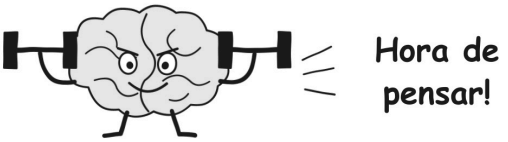
O que aconteceu	O que acontece com o pagamento
chegou menos	não está pago, expira no prazo; o dinheiro já está com você
em duas partes	as partes não se somam: mesma coisa
chegou mais	`paid` ; o valor real está em `received_amount_raw`
mandaram outra vez depois do `paid`	o pagamento já não espera: no máximo se descobre pelo evento de endereço
depois do prazo	`expired`
token errado ou rede errada	não é contabilizado (capítulo 16)
a transação falhou	não há dinheiro, não despache: o pagamento vira `failed` ou simplesmente expira

Decida de antemão o que fazer em cada caso e escreva isso ao cliente na página de pagamento.

Cilada: "devolvo para onde veio"

O cliente pagou a mais, e Artiom quer mandar a sobra ao endereço do remetente. Mas podia ser o endereço comum de uma corretora: quem recebe o dinheiro é ela, não o cliente. Pergunte ao cliente o endereço e a rede para o reembolso. Envie da carteira, com as próprias mãos, e não a partir do site.

Pense bem



1. A resposta à criação do pagamento não chegou. Pode repetir? Com qual `order_id`?
2. O mesmo `paid` chegou por webhook e pelo socket. Quantos pacotes o programa do exemplo envia?
3. No lugar de 30 USDT chegaram duas transferências de 15. O que acontece com o pagamento de 30?

Respostas: 1) sim, com o mesmo e com os mesmos dados. 2) Um. 3) Não está pago: transferências não se somam.

O essencial

- Repita só o que é seguro repetir.
- O despacho é protegido por uma tranca no banco de dados, e os avisos perdidos são achados pela reconciliação.
- Enquanto a transferência não está num bloco, não há dinheiro; para pagamentos esquisitos, as regras têm de existir antes.

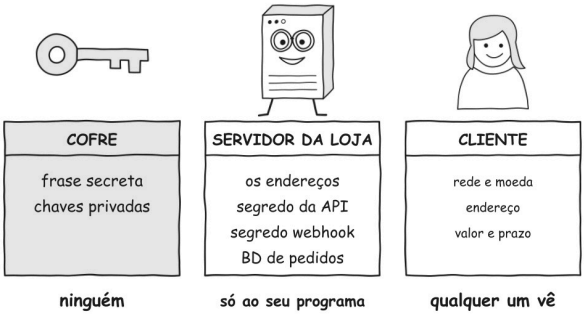
A seguir: capítulo 30. Segurança e a lista de conferência antes do lançamento.

Capítulo 30. Segurança e a lista de conferência antes do lançamento

“Amanhã abrimos. O que eu esqueci?”

Na rede de testes tudo funciona. Artiom já escreve aos clientes: “Aceitamos cripto!” Mas dorme mal. O que pode dar errado quando o dinheiro for de verdade?

Entendendo: o que fica onde



O segredo da chave de API não permite gastar as suas moedas: a RpcNode não tem chaves privadas. Mas estrago ele pode fazer. Preencha os “IPs permitidos” da chave: assim um segredo roubado não funciona a partir de outro computador.

A **autenticação em duas etapas** é quando, além da senha, é preciso um código de um aplicativo. No painel ela está em “Configurações”. Ligue a mesma coisa no e-mail: muitas vezes é pelo e-mail que se recupera o acesso a todo o resto.

O que acontece se algo disso vazar:

O que vazou	O que um estranho consegue	O que fazer
frase secreta, chave privada	levar o dinheiro sem volta	transferir o saldo com urgência para uma carteira nova
segredo da chave de API	mexer em pagamentos, monitoramentos, endpoints e no webhook, e ler o segredo dele	excluir a chave, emitir outra, trocar o segredo do webhook, conferir as configurações
segredo do webhook	falsificar um aviso `paid`	"Gerar novo segredo"
endereço do endpoint	gastar os seus créditos	criar um novo e apagar o antigo (capítulo 22)
acesso ao painel	quase tudo desta tabela, menos as suas moedas	trocar a senha, ligar a autenticação em duas etapas

Segredos não moram no código

Um segredo no código vai parar no repositório compartilhado, no arquivo enviado a um prestador, numa captura de tela. Guarde os segredos nas configurações do servidor (variáveis de ambiente) e proteja de estranhos o arquivo que as contém.

Na prática: vamos procurar em casa

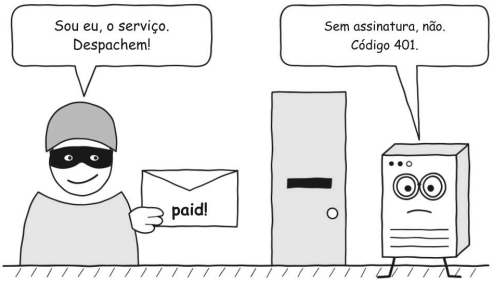
O segredo da chave de API começa com `sk_live_`. Vamos criar uma pasta de treino com um erro e achá-lo:

```
mkdir demo
echo "const SECRET = 'sk_live_exemplo'" > demo/pay.js
echo "const SECRET = process.env.RPCNODE_API_SECRET" > demo/ok.js
grep -rnE 'sk_live_|PRIVATE KEY|xprv' demo
```

```
demo/pay.js:1:const SECRET = 'sk_live_exemplo'
```

A busca achou o primeiro arquivo e não mexeu no segundo. Troque `demo` pela pasta do seu projeto. `xprv` é o começo de uma chave **privada** estendida. Se achou algo, não basta apagar a linha: ela pode ter ficado no histórico de alterações. Considere o segredo vazado e emita outro. Já a frase secreta não se pega assim, são palavras comuns: simplesmente não a guarde no computador.

Não acredite no que chega



Vamos testar a nossa porta. Salve o esboço de receptor do capítulo 26 em `receiver.mjs` e execute:

```
RPCNODE_WEBHOOK_SECRET=demo-secret node receiver.mjs
```

Em outra janela do terminal, bata à porta de três maneiras estranhas:

```
URL=http://localhost:3000
BODY='{"type":"payment_order","event":"paid"}'
# 1) sem assinatura
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY"
# 2) assinatura no chute
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY" \
  -H "X-RpcNode-Signature: t=$(date +%s),v1=$(printf 'a%.0s' {1..64})"
# 3) assinatura verdadeira, mas de uma hora atrás
T=$(( $(date +%s) - 3600 ))
SIG=$(printf '%s.%s' $T "$BODY" |
  openssl dgst -sha256 -hmac 'demo-secret' | awk '{print $NF}')
curl -s -o /dev/null -w '%{http_code}\n' $URL -d "$BODY" \
  -H "X-RpcNode-Signature: t=$T,v1=$SIG"
```

```
401
401
401
```

Três vezes "não entra". Se em algum caso veio 200, a porta está aberta. Troque `- 3600` por `- 0`, e a terceira batida recebe 200: a assinatura está fresca e correta.

Para pedidos caros, ponha uma segunda tranca. Ao receber `paid`, consulte você mesmo o pagamento (`GET /v1/payments/orders/{orderId}`) e confira `payment_id`, endereço, token, quantia e status. Assim, nem com o segredo do webhook roubado se leva café de graça.

Além disso, receba os webhooks por `https` e limite o tamanho da requisição de entrada: o esboço do capítulo 26 não faz isso. Pode-se ainda deixar entrar só os endereços da seção "Segredo e IPs". É um complemento da assinatura, não um substituto.

A placa trocada



O ladrão não precisa necessariamente conseguir as chaves. Basta invadir o site e mostrar ao cliente o endereço dele. O que ajuda:

- a administração do site não pode permitir trocar a lista de endereços sem alarde;
- de tempos em tempos, compare os endereços distribuídos com os da sua carteira, a partir de outro computador;
- de vez em quando, compre de você mesmo e veja para onde mandam pagar;
- atualize os programas do servidor e entre na administração com autenticação em duas etapas.

Os ladrões também escrevem para você pessoalmente: “suporte”, “devolva o pagamento travado”, uma transferência minúscula de um endereço parecido (capítulo 6). O suporte de verdade não pede a frase secreta.

O ensaio geral



Primeiro, tudo na rede de testes. Pagamentos e eventos em rede de testes funcionam na TRON Nile (capítulo 26). No Free há um sandbox: por padrão, 3 pagamentos abertos e 3 endereços; a análise de risco ali não é de verdade. No sandbox os endereços também não se liberam sozinhos.

O que encenar	O que deve sair
pagar o valor exato	um pacote
pagar menos	nada despachado, pagamento expirado
mandar a si mesmo o mesmo aviso duas vezes	um pacote
desligar o seu servidor, pagar, ligar	pagamento encontrado: pela repetição do webhook ou pela reconciliação
bater sem assinatura	401

Depois, uma pequena compra de verdade de você mesmo, na rede em que vai vender. Aproveite para ver se a sua carteira enxerga o dinheiro e se você consegue mandá-lo adiante: há moeda suficiente para a taxa?

Lei e impostos

Em muitos países, aceitar criptomoeda é restrito ou tributado. Antes de lançar, verifique as leis do seu país. Isto não é um conselho jurídico. Pergunte também ao contador que registros de pagamento ele precisa e guarde-os desde o primeiro dia.

Cilada: "conferi uma vez e esqueci"

A loja funciona, mas tudo muda em volta. Venceu o prazo do plano: a conta passa para o Free, onde não há pagamentos na rede principal (capítulo 25). Os créditos são gastos também pelos eventos. A lista de endereços está acabando. Ponha lembretes.

Lista de conferência antes do lançamento

Chaves e acesso

- □ a frase secreta está anotada e não está em nenhum servidor nem computador;
- □ no site há só endereços, e alguns foram conferidos com a carteira;
- □ não há segredos no código, a busca no projeto não acha nada;
- □ a chave de API tem os IPs permitidos preenchidos;
- □ a autenticação em duas etapas está ligada no painel, no e-mail e na administração do site.

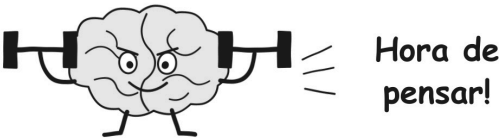
Recebimento do dinheiro

- □ um pedido, um endereço, sem repetições;
- □ a cobrança tem rede, moeda, quantia, endereço, prazo e "numa única transferência";
- □ o contrato do token foi tirado de `GET /v1/network-tokens` e conferido no site do emissor;
- □ o número de confirmações foi escolhido conforme as suas quantias;
- □ a assinatura do webhook é verificada, e o despacho é protegido por uma tranca no banco;
- □ a reconciliação dos pedidos abertos roda de forma agendada.

O que fica em volta

- □ há regras para pagamento a menor, a maior, atraso e reembolso;
- □ slots, créditos, prazo do plano e estoque de endereços estão sob vigilância;
- □ está claro com que pagar a taxa para recolher o dinheiro;
- □ faz-se cópia do banco de pedidos, e já se restaurou a partir dela ao menos uma vez;
- □ o ensaio foi feito, e a sua própria compra de verdade também;
- □ as leis e os impostos do seu país foram verificados.

Pense bem



1. Vazou o segredo da chave de API. Conseguem tirar moedas da sua carteira? O que fazer mesmo assim, imediatamente?
2. O seu receptor respondeu 200 a uma mensagem sem assinatura. Qual é o perigo?
3. Todos os testes na TRON Nile passaram. Dá para abrir o recebimento de pagamentos na Ethereum sem testar de novo?

Respostas: 1) não, as chaves da carteira não estão lá; mas é preciso excluir a chave de API, emitir outra, trocar o segredo do webhook e conferir as configurações. 2) Qualquer um "paga" um pedido sem dinheiro. 3) Não: outra rede, outros endereços, taxas e confirmações.

O essencial

- Mantenha as chaves do dinheiro longe do site, e os segredos longe do código.
- Não acredite de cara em mensagem que chega, captura de tela nem carta.
- Primeiro o ensaio, depois uma pequena compra de verdade, depois a abertura.

Fim da sexta parte. Artiom montou a loja, sabe o que fazer quando ela quebra e como protegê-la.

A seguir: capítulo 31. DeFi: finanças sem intermediário.

Capítulo 31. DeFi: finanças sem intermediário

"E as moedas podem não ficar só paradas?"

Artiom juntou uma parte do faturamento em USDC. Um conhecido aconselha: "Põe no DeFi, lá o dinheiro faz dinheiro sozinho". Que lugar é esse, sem guichê nem caixa?

> Isto **não é um conselho financeiro**, e sim uma explicação. Os nomes de protocolos são apenas exemplos. No seu país esses serviços podem ser restritos ou tributados: verifique as leis.

Entendendo: o banco e o programa

DeFi, "finanças descentralizadas", são trocas, empréstimos e outros serviços de dinheiro executados por contratos inteligentes (capítulo 9). Um conjunto de contratos com regras comuns é chamado de **protocolo**.

	Banco	DeFi
Onde está o dinheiro	no banco	num endereço ou num contrato
Quem executa	funcionários e seus sistemas	um programa na rede
Quando funciona	no horário dele	24 horas por dia
Onde estão as regras	no contrato assinado	no código
Errou	dá para recorrer ao banco	em geral não há quem desfaça
Perdeu o acesso	o documento de identidade resolve	sem a chave não há volta

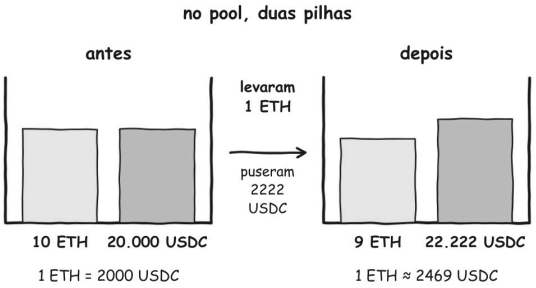
"Sem intermediário" não quer dizer "sem confiança". Você confia no código, em quem pode alterá-lo, nos fornecedores de preços e no emissor das moedas.

Troca sem corretora: o pool de liquidez

Numa corretora comum, procura-se um vendedor para o comprador. No DeFi, muitas vezes se cria um **pool de liquidez**: um contrato onde ficam duas moedas ao mesmo tempo. Você põe uma no pool e leva a outra. Não precisa esperar uma segunda pessoa.

Quanto você recebe quem decide é uma fórmula. Nos pools simples ela é: $\mathbf{x \times y = k}$, em que x e y são os saldos das duas moedas. A troca não pode diminuir esse produto.

O preço aqui é a proporção entre os saldos. Digamos que o pool tenha 10 ETH e 20.000 USDC. Então 1 ETH vale cerca de 2000 USDC, e k = 200.000. Artiom quer retirar 1 ETH (por ora sem contar a taxa). Vão restar 9 ETH, logo os USDC têm de passar a ser 200.000 : 9 ≈ 22.222.



Ele vai pagar 2222 USDC, e não 2000: **a própria compra dele subiu o preço**. Isso se chama **impacto no preço**. Num pool cem vezes maior (1000 ETH e 2.000.000 USDC), o mesmo 1 ETH sairia por uns 2002 USDC.

Se no pool o ETH ficou mais caro que em outros lugares, alguém vai trazer ETH para lá e levar USDC até os preços se aproximarem. Isso é **arbitragem**.

Quem põe moedas no pool são os **provedores de liquidez**, que recebem uma parte das taxas. Mas, quando os preços se afastam, o pool entrega a moeda que se valorizou e acumula a outra. Sem contar as taxas, se o preço dobra, o aporte vale cerca de 5,7% menos do que se as moedas tivessem simplesmente ficado paradas. Isso se chama **perda impermanente**: retire o aporte nesse momento e ela fica bem permanente.

Na prática: perguntar a um pool de verdade

Vamos pegar o pool de USDC e WETH do protocolo Uniswap v2 na Ethereum. **WETH** é o ETH em forma de token, um para um.

O endereço de um pool não se tira de uma busca qualquer. Pegamos o endereço da "fábrica" de pools na documentação da Uniswap, o do USDC no emissor dele, a Circle, e o do WETH no ethereum.org. Depois fizemos à fábrica a pergunta `getPair`: "onde está o pool deste par?" Resposta: `0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc`.

Agora a pergunta `getReserves`: "quais são os saldos?" É um `eth_call` do capítulo 9: ele só lê.

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc",
   "data":"0x0902f1ac"},"latest"]}'
```

Na madrugada de 27 de novembro de 2026, o nó respondeu:

```
{"jsonrpc":"2.0","id":1,"result":"0x000000000000000000000000000000000000000000000000000079ecc8266bc000000000000000000000000000000000000000000000000000010b762ae141ddef4f5000000000000000000000000000000000000000000000000000006b08b4af"}
```

Depois do `0x` vêm três números de 64 caracteres:

Número (sem os zeros à esquerda)	O que é
`79ecc8266bc`	saldo de USDC, 6 casas decimais: 8.378.617 USDC
`10b762ae141ddef4f50`	saldo de WETH, 18 casas: 4933,80 WETH
`6b08b4af`	quando os saldos mudaram pela última vez, em segundos desde 1970

Qual moeda vem primeiro dizem as perguntas `token0` e `token1`: neste pool, a primeira é o USDC. Resta dividir:

```
python3 -c 'u = 0x79ecc8266bc / 10**6; w = 0x10b762ae141ddef4f50 / 10**18; print(u, w, u / w)'
```

Será impresso `8378617.325244 4933.79553339896 1698.2092728662094`.

Um ETH custava neste pool cerca de **1698 USDC**. Os seus números serão outros.

E quanto se recebe por um lote grande? A taxa de troca aqui é de 0,3%: desses, 0,25% vão para os provedores e 0,05% para o protocolo. Com os mesmos saldos, sai assim:

Entregamos	Recebemos em USDC	Por 1 ETH	Pior que o preço 1698,21
1 ETH	1692,77	1692,77	em 0,3%
100 ETH	165.957,86	1659,58	em 2,3%
1000 ETH	1.408.492,58	1408,49	em 17%

O cálculo bateu até a última casa com as respostas do contrato auxiliar da Uniswap (router) à pergunta `getAmountsOut`.

Importante: dividir os saldos serve para pools do tipo v2. Na Uniswap v3 e v4, os aportes estão distribuídos por faixas de preço, e ali o preço se calcula de outra forma.

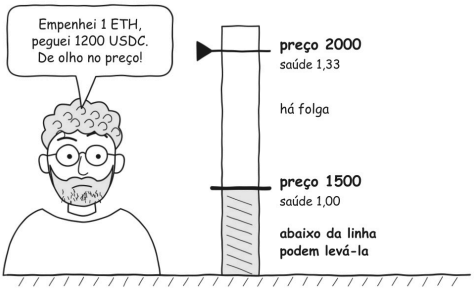
Empréstimos com garantia e liquidação

Num protocolo de empréstimos (por exemplo, a Aave), deposita-se uma **garantia** e se toma emprestado menos do que ela vale. Para quê? Digamos que se precise de dólares, mas não se queira vender ETH.

O empréstimo tem uma "saúde" (health factor):

saúde = valor da garantia × limiar de liquidação : dívida

Exemplo inventado: garantia de 1 ETH a 2000 USDC, limiar de 80%, dívida de 1200 USDC. Saúde: 2000 × 0,8 : 1200 ≈ 1,33. Se o preço cair para 1500, fica exatamente 1.



Se a saúde fica abaixo de 1, qualquer participante pode quitar parte da dívida, às vezes toda, e levar por isso a garantia com um bônus. Isso é a **liquidação**. Não vai haver telefonema pedindo "complete o dinheiro": o programa faz tudo, mesmo no meio da noite. E a dívida também cresce sozinha, pelos juros.

Stablecoins: no que se apoia "um dólar"

Capítulo 9: a stablecoin apenas **procura** valer cerca de um dólar. Cada uma se segura nele de um jeito. **Reservas no emissor**: dólares e outros ativos (USDT, USDC). **Garantia em cripto com folga** em contratos (DAI, USDS; na garantia pode haver também outras stablecoins). Um **algoritmo** que muda a quantidade de moedas, sem garantia.

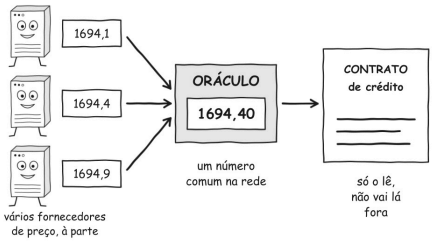
Nenhum método é uma certeza. Se uma moeda dessas se desvalorizar num pool, é justamente ela que vai se acumular com os provedores.

Staking em poucas palavras

O validador do capítulo 8 precisa de 32 ETH. Com menos, participa-se por meio de um **pool de staking coletivo** (não é um pool de troca). Em troca, muitas vezes se recebe um token-recibo. Riscos: multas aos validadores, erro no contrato, confiança no operador. Ninguém promete o tamanho do rendimento.

Oráculos: como o contrato fica sabendo o preço

Um contrato de empréstimo precisa do preço do ETH. Mas ele não vai sozinho à internet: todos os nós precisam chegar ao mesmo resultado (capítulo 15). Quem grava o preço na rede é um **oráculo**.



Um exemplo é o oráculo Chainlink. O endereço do feed "ETH / USD" foi tirado do catálogo da própria Chainlink. A pergunta `latestRoundData` pede o registro mais recente:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_call","params":[
  {"to":"0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419",
   "data":"0xfeaf968c"},"latest"]}'
```

```
{"jsonrpc":"2.0","id":1,"result":"0x0000000000000000000000000000000000000000000000000000000700000000000008e4f00000000000000
0000000000000000000000000000000000000000000002773693800000000000000000000000000000000000000000000000000000000000006b08aefa
0000000000000000000000000000000000000000000000000000006b08af0f0000000000000000000000000000000000000000000000000000000700
00000000008e4f"}
```

O segundo dos cinco números, `2773693800`, é o preço com 8 casas decimais: **1694,40 dólares**. O pool mostrava 1698,21 USDC, uma diferença de cerca de 0,2%. O quarto número é a hora do registro: 25 minutos antes da nossa pergunta. Esse feed se atualiza quando o preço se afasta 0,5% do registrado e também de forma agendada, mais ou menos uma vez por hora.

Por que não pegar o preço direto do pool? Uma operação grande pode deslocá-lo por um instante. Mas o oráculo também pode errar ou atrasar, e a liquidação seguirá o número dele.

Como um aplicativo DeFi lê a rede

O site de trocas ou de empréstimos é uma vitrine. Os dados ele pega de um nó:

- `eth_call`: o que há agora, como saldos do pool, saldos de contas, preço do oráculo, saúde do empréstimo.
- **Eventos** (capítulo 15): o que mudou. O pool v2, depois de cada mudança, anuncia o evento `Sync` com os novos saldos. Eles são procurados com `eth_getLogs` ou escutados por WebSocket (capítulo 11).
- **Envio**: uma troca altera o caderno. A carteira a assina, o nó a repassa (capítulo 7), e isso custa gas.

Eis a pergunta sobre os eventos `Sync` do nosso pool no bloco mais recente:

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"eth_getLogs","params":[{
  "address":"0xB4e16d0168e52d35CaCD2c6185b44281Ec28C9Dc",
  "topics":["0x1c411e9a96e071241c2f21f7726b17ae89e3cab4c78be50e062b03a9fffbbad1"],
  "fromBlock":"latest","toBlock":"latest"}]}'
```

```
{"jsonrpc":"2.0","id":1,"result":[]}
```

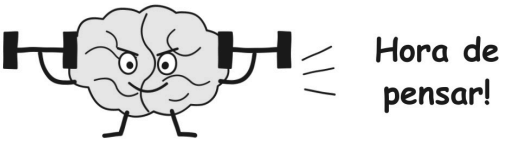
Lista vazia também é resposta: neste bloco o pool não mudou. O terceiro número do `getReserves` confirma: a última mudança tinha sido uns 10 minutos antes da nossa pergunta. Mudanças passadas, para um gráfico de preço por exemplo, já são arquivo ou um banco de dados próprio (capítulo 10).

Cilada: o que pode dar errado

Perigo	O que acontece	O que reduz o risco
Erro no código	o dinheiro é roubado ou fica preso	auditoria do código, mas não é garantia
Token falso	nome conhecido, endereço estranho	conferir o endereço (capítulo 9)
Autorização sem limite (`approve`)	os tokens são levados depois	autorizar pouco, zerar o que sobra
Slippage	você recebe menos do que esperava	definir a quantia mínima
Liquidação	a garantia se vai, com bônus	folga grande e vigilância
Site clonado	uma cópia do site pede assinatura	conferir o endereço; não digitar a frase secreta
“Rendimento garantido”	prometem e somem	não participar

Slippage (derrapagem) é a diferença entre o preço esperado e o real. Enquanto a troca espera na fila, operações alheias mexem no preço, às vezes de propósito. A quantia mínima protege: abaixo dela a troca não passa, embora o gas seja gasto.

Pense bem



1. Um pool tem 100 ETH e 200.000 USDC. Alguém retirou 20 ETH. Quantos USDC essa pessoa colocou (sem taxa)?
2. Garantia de 2 ETH a 2000 USDC, limiar de 80%, dívida de 2400 USDC. Qual é a saúde? A que preço do ETH ela fica igual a 1?
3. Um site promete 3% ao dia sem risco e pede autorização para gastar USDT sem limite. O que você faz?

Respostas: 1) restam 80 ETH, então os USDC passam a ser 250.000: colocou 50.000, a 2500 por ETH. 2) Cerca de 1,33; ao preço de 1500. 3) Não assinar nada e fechar o site.

O essencial

- No DeFi os serviços são executados por contratos, e o risco e a responsabilidade recaem em grande parte sobre você.
- Num pool do tipo v2, o preço sai dos dois saldos, e uma operação grande o desloca.
- O aplicativo lê a rede com as mesmas requisições que você: `eth_call` e eventos.

A seguir: conclusão. Da mensagem de um cliente à própria loja.

Verificado em 27 de novembro de 2026: developers.uniswap.org, developers.circle.com/stablecoins/usdc-contract-addresses, ethereum.org (wrapped-eth, defi, stablecoins, staking, developers/docs/oracles), aave.com/help/borrowing/liquidations, docs.chain.link.

Conclusão. Da mensagem de um cliente à própria loja

Tudo começou com uma linha numa mensagem: “Posso pagar com criptomoeda?” Artiom não sabia então o que era um nó e conferia a carteira na mão. Veja o caminho que você percorreu junto com ele.

O que Artiom sabe fazer agora

Parte	O que aprendemos
1. Por que um provedor	Calcular o custo de servidores próprios e escolher entre manter o nó ou contratar um provedor
2. Blockchain do zero	Entender blocos, chaves, assinaturas, taxas, confirmações e tokens
3. Nós e RPC	Fazer perguntas a um nó, distinguir os tipos de nó e verificar se ele ficou para trás
4. Como as redes funcionam	Ver em que as redes diferem e escolher a que serve aos clientes
5. RpcNode	Obter um endpoint, calcular o preço das chamadas e saber dos pagamentos sem consultar sem parar
6. Mão na massa	Montar o recebimento de pagamentos, sobreviver às falhas e proteger a loja
7. E depois: DeFi	Ler um pool direto da rede e avaliar os riscos com honestidade

Sete regras para guardar

1. O endereço se mostra a todos; a chave privada e a frase secreta, a ninguém.
2. Confere-se tudo de uma vez: rede, endereço do contrato do token, destinatário e quantia.
3. O dinheiro chegou quando a transação teve sucesso e juntou confirmações, e não quando veio uma captura de tela.
4. O código 200 ainda não é sucesso: leia o que há dentro da resposta.
5. Um pagamento, um despacho, não importa quantas vezes chegue o aviso.
6. Guarde o endpoint e os segredos como senhas.
7. Não acredite de cara: pergunte à rede e confira a resposta.

Por onde começar amanhã

1. Repita a requisição do capítulo 3 a um nó público.
2. Crie o seu endpoint, numa rede de testes (capítulo 22).
3. Encene um pagamento a si mesmo com dinheiro de brinquedo (capítulos 28 e 30).
4. Percorra a lista de conferência do capítulo 30.

5. Verifique as leis e os impostos do seu país.

Se algo estiver desatualizado

O livro descreve as redes e o serviço como eram em 27 de novembro de 2026. As redes se atualizam, os preços e os planos mudam. Se a resposta do seu nó não bater com o livro, acredite no nó e na documentação mais recente. Os planos e os recursos da RpcNode estão no site rpcnode.dev.

Até logo

Se vai pôr parte do faturamento em DeFi, Artiom ainda está pensando. Em compensação, sabe que perguntas fazer. Talvez seja essa a principal coisa que ele tirou de toda a história.

Parte dos números deste livro vai envelhecer, mas o hábito de verificar, não. Boa sorte para você e para o seu negócio!



Apêndice A. Glossário

Entre parênteses, o capítulo em que a palavra é explicada com mais detalhe.

AML — verificação de se o dinheiro está ligado a um histórico ruim (27).

API — guichê pelo qual um programa se dirige a outro (11).

Assinatura — prova de que o dono da chave autorizou a transferência (6).

Balanceamento — distribuição das requisições entre vários nós (23).

Bloco — a "página do caderno": transações, número, hora e o selo do bloco anterior (5).

Blockchain — lista comum de registros, mantida em conjunto segundo regras comuns (4).

Carteira — programa ou aparelho onde as chaves ficam guardadas (6).

Chain ID — número de uma rede EVM (16).

Chave privada — aquilo com que se manda no dinheiro; não se mostra a ninguém (6).

Confirmações — blocos empilhados sobre a sua transação, contando o próprio bloco dela (5).

Consenso — regras pelas quais os participantes escolhem uma única cadeia (8).

Contrato inteligente — programa gravado na blockchain (9).

Créditos — unidades em que o serviço conta o consumo de requisições (3, 25).

DeFi — serviços de dinheiro executados por contratos inteligentes (31).

Endereço — o "número da caixa" para onde se manda dinheiro (6).

Endpoint — endereço pelo qual um programa faz perguntas a um nó (3).

Evento (log) — anúncio de um contrato sobre o que aconteceu (15).

EVM — máquina virtual da Ethereum, que executa os contratos (15).

Finalidade — momento a partir do qual o bloco não será mais substituído (8).

Frase secreta — 12 ou 24 palavras das quais se recuperam as chaves (6).

Gas — medida do trabalho pelo qual se cobra taxa na Ethereum e parentes (15).

Gasto duplo — tentativa de pagar duas vezes com a mesma moeda (4).

gRPC — forma de os programas conversarem em mensagens compactas num esquema rígido (11).

Hash, "selo" — impressão digital curta de quaisquer dados (5).

Idempotência — repita quantas vezes for, o resultado é o de uma vez só (29).

Indexador — programa que organiza num "fichário" os registros da blockchain (10).

L2, "segundo andar" — rede que executa as transferências no próprio espaço e envia os registros delas à Ethereum (16).

Limite de requisições (RPS) — quantas vezes por segundo se pode perguntar (3, 25).

Liquidação — venda da garantia quando ela deixa de cobrir o empréstimo (31).

Mempool — fila das transferências que esperam um bloco (7).

Mineração — resolver um problema na base da tentativa pelo direito de acrescentar um bloco (8).

Nó — computador com o programa da rede: guarda, confere e responde (1).

Nó de arquivo — lembra o estado em qualquer bloco do passado (10).

Nó podado — conferiu tudo e apagou os blocos antigos (10).

Nonce — número de ordem da transação de uma conta (15).

Oráculo — quem grava na rede informações de fora, por exemplo um preço (31).

Ponte — meio de levar valor de uma rede a outra (16).

Pool de liquidez — contrato com duas moedas, onde elas são trocadas conforme uma fórmula (31).

Provedor — serviço que mantém nós e dá acesso a eles (1, 3).

Rede de testes — campo de treino com moedas sem valor (22).

Reorganização — substituição de parte dos blocos recentes por outro ramo (8).

REST — forma de perguntar em que cada pergunta tem seu caminho no endereço (11).

RPC, JSON-RPC — forma de pedir a um programa em outro computador que faça algo (11).

Sincronização — o processo pelo qual o nó alcança a rede (12).

Stablecoin — token que procura se manter em torno de um dólar (9).

Staking — participação na proteção da rede com uma garantia em moedas (8).

Taxa — pagamento à rede por um lugar no bloco (7).

Token — registros mantidos por um contrato inteligente (9).

UTXO — um "pedaço de dinheiro" não gasto no Bitcoin e parentes (14).

Validador — participante que deposita uma garantia e toma parte na criação dos blocos (8).

Webhook — requisição que o próprio serviço envia ao seu endereço quando algo acontece (26).

WebSocket — conexão que não se fecha, para receber as novidades (11).

Wei — a menor fração do ETH (1).

Apêndice B. As redes do livro em uma página

Informações dos capítulos 6, 8 e 14–20, em 27 de novembro de 2026. Elas mudam com as atualizações das redes.

Bitcoin e parentes (capítulo 14)

Rede	Moeda	Bloco, aprox.	O endereço começa com
Bitcoin	BTC	10 min	`1`, `3`, `bc1`
Litecoin	LTC	2,5 min	`L`, `M`, `ltc1`
Dogecoin	DOGE	1 min	`D`
Bitcoin Cash	BCH	10 min	`bitcoincash:q…`
Dash	DASH	2,5 min	`X`
Zcash	ZEC	75 s	`t1`, `zs`, `u1`

Ethereum e redes EVM (capítulos 15 e 16)

Em todas, o endereço começa com `0x`.

Rede	Chain ID	Taxa paga em	Bloco, aprox.
Ethereum	1	ETH	12 s
Base	8453	ETH	0,2 s
Arbitrum One	42161	ETH	0,3 s
BNB Smart Chain	56	BNB	0,45 s
Mantle	5000	MNT	2 s
Monad	143	MON	0,3 s

As demais (capítulos 17–20)

Rede	Moeda	O endereço começa com	Definitivo em
TRON	TRX	`T`	cerca de um minuto
Solana	SOL	não há começo especial	frações de segundo
TON	Gram	`EQ`, `UQ`	cerca de um segundo
Sui	SUI	`0x`	menos de um segundo
Aptos	APT	`0x`	menos de um segundo
XRP Ledger	XRP	`r`	3–5 segundos
Stellar	XLM	`G`	cerca de 5 segundos
Cardano	ADA	`addr1`	aos poucos
Polkadot Hub	DOT	`1`	por uma marca
Cosmos Hub	ATOM	`cosmos1`	rápido

Apêndice C. Cola de requisições

Todos os exemplos funcionam nos nós públicos da RpcNode: sem cadastro e sem chave, com limite de frequência. No seu projeto, troque o endereço pelo de um endpoint pessoal.

JSON-RPC: um comando para todos os casos

```
curl https://public-ethereum-mainnet.rpcnode.dev \
  -H 'Content-Type: application/json' \
  -d '{"jsonrpc":"2.0","id":1,"method":"METODO","params":[PARAMETROS]}'
```

Coloque o método e os parâmetros das tabelas. Para outra rede, troque `ethereum` no endereço.

Ethereum e redes EVM

O que descobrir	Método	Parâmetros
número do último bloco	`eth_blockNumber`	vazio
número da rede	`eth_chainId`	vazio
saldo da moeda	`eth_getBalance`	`"0xENDERECO","latest"`
recibo da transação	`eth_getTransactionReceipt`	`"0xHASH"`
preço do gas	`eth_gasPrice`	vazio
perguntar a um contrato	`eth_call`	`{"to":"0x…","data":"0x…"},"latest"`

Bitcoin e Solana

Endereço	Método	O que descobrir
`public-bitcoin-mainnet`	`getblockcount`	altura da cadeia
`public-bitcoin-mainnet`	`getblockchaininfo`	estado do nó
`public-solana-mainnet`	`getSlot`	número do slot
`public-solana-mainnet`	`getBalance`	saldo em lamports; o parâmetro é o endereço

TRON e TON: a pergunta no próprio endereço

```
curl https://public-tron-mainnet.rpcnode.dev/wallet/getnowblock
```

```
curl https://public-ton-mainnet.rpcnode.dev/api/v2/getMasterchainInfo
```

Códigos de erro

Código	O que houve
`-32700`	não dá para interpretar a requisição
`-32601`	método não encontrado ou indisponível
`-32602`	parâmetros inválidos
`-32005`, HTTP 429	frequência alta demais
`-32006`, HTTP 402	créditos insuficientes

Documentação completa e planos atualizados

https://rpcnode.dev

Estamos no Telegram

@rpcnode_dev

Autor do livro: rpcnode.dev

Fim do livro.

Sem complicação

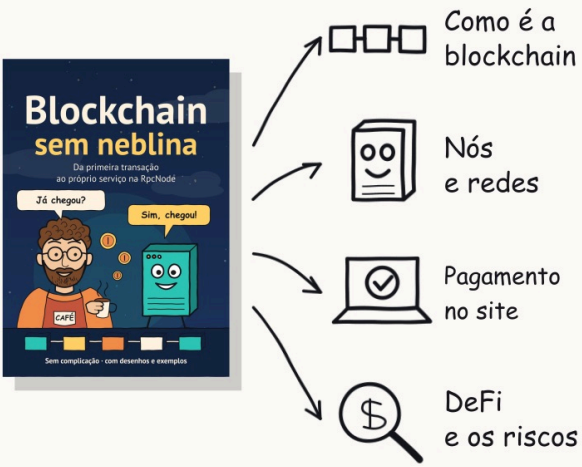
Blockchain sem neblina

O que este livro ensina?

Um cliente pergunta: “Posso pagar com cripto?” O que dizer, como saber se o dinheiro chegou, sem perdê-lo no caminho?

Dentro: blocos, chaves, taxas e tokens, nós e suas consultas, Bitcoin, Ethereum, TRON, Solana, TON e outras redes. Depois, prática: endpoint RpcNode, pagamentos, webhooks e a defesa da loja. No fim, um papo franco sobre DeFi.

> “Meu site sabe somar o carrinho e mandar e-mails. Como ele sabe o que se passa na blockchain?”
>
> **— Artiom,** o herói



O que há de especial aqui?

Não é preciso saber nada. Cada capítulo: uma dúvida, resposta, arte e desafios. Dá para rodar as requisições e ver a resposta real.

rpcnode.dev
docs.rpcnode.dev
Telegram: @rpcnode_dev
Autor: rpcnode.dev